

High Performance Embedded Solutions for Memory and Compute Intense Applications

BY

UMER I. CHEEMA

B.Sc. Electrical Engineering, University of Engineering & Technology, Lahore, Pakistan, 2008
M.S. Electrical & Computer Engineering, University of Illinois at Chicago, 2014

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2016

Chicago, Illinois

Defense Committee:

Ashfaq Khokhar, Advisor
Rashid Ansari, Chair & Co-advisor
Wenjing Rao
Hulya Seferoglu
John Lillis, Computer Science

Copyright by
Umer I. Cheema
2016

Dedicated to my parents: Sajida and Iftikhar

ACKNOWLEDGMENTS

My time at UIC has been a tremendous learning experience not only in academics but also in all other aspects of life. I owe my success to all my teachers throughout my academics. First of all, I would like to thank my PhD advisor, Professor Ashfaq Khokhar, for his consistent feedback and support throughout my PhD. He let me chose an independent path which was tougher but I got to work on something that I really enjoyed and learned much more during the process than I would have otherwise. I would also like to thank my co-adviser, Professor Rashid Ansari, for his consistent support especially after Professor Khokhar left UIC.

I would also like to thank my thesis Committee members Prof. Wenjing Rao, Prof. Hulya Seferoglu and Prof. John Lillis, for their feedback and encouraging comments. I got to work with Prof. Wenjing Rao as a Teaching Assistant as well as a student. I took a lot of inspiration from her teaching methods as well as the work ethics.

I would like to thank Prof. Vahe Caliskan, for entrusting me to teach key undergraduate courses and his consistent help and support during my time as course instructor. Thanks to Prof. Robert Becker for the wonderful learning experience I had as his Teaching Assistant. I would also like to thank all the professors I took my PhD course work with: Prof. Grygory Turon, Prof. Masud Chowdhury, Prof. Ajay Kshemkalyani, Prof. Prasad Sistla, Prof. Mitchell Theys, Prof. Tadao Murata, Prof. Shantanu Dutt and Prof. Michael Stroschio. Throughout my stay at UIC, I had the opportunity to interact with some amazing people. I would like to acknowledge Gregory Nash for the discussions about the trends in industry related to my

ACKNOWLEDGMENTS (Continued)

thesis. His knowledge in the field helped me chose the right tools for my research. I had a great interaction with Mohamed Hefeida - the senior. Observing him go through the PhD process, gave me inspiration and a better understanding of the process. I regret the accident we had in racket ball game that gave him a lot of pain. Mohamed Ali was my lab mate as well as apartment mate for the longest. I didnt get to see him a lot because of his off-campus commitments but I learned a lot from him in multiple domains. To Kamran Lodhi, we had some great road trips together and his spicy cooking was always enjoyable. To Ansab Ali, who always had quality high-end games that I could enjoy on his computer. To Fahad Saeed, for being a great example and inspiration throughout my PhD. To Fadi Almasalha, for the help and support when I first came to Chicago. I would also like to thank my lab-mates: Xi Xu, Yasaman Keshtkarjahromi, Nazanin Makkinejad and Shafagh Kamkar for the time we spent together. The ECE department staff - Erica Plys, Agustina Alvarado, Ala Wroblewski, Beverly Miller, Mona Hurt, Evelyn Reyes, Mihai Bora, Martha Salinas, George Ashman and Harold Sosa - were very helpful in the administrative matters. I would also like to thank my fellow graduate students - Pitamber Shukla, Jian Xu, Mehak Gill, Narueporn Nartasilpa and Soumik Sen - who did a great job assisting me in teaching so I could focus well on my research. Some other friends I had good time with were Ahmad Khalil, Muzammil Rafique, Farukh Ijaz, Talal Haider, Ahsan Shahid, Sumra Bari and Rasheed Abdulkader.

Coming to Chicago from the other side of the globe was initially an overwhelming experience. My uncle, Zafar Singhera and his family, Arusa and Fakhar, made this transition relatively easier. The supply of tasty food from California was a life saver during the early days. My

ACKNOWLEDGMENTS (Continued)

uncle has been an inspiration for me right from my child hood and his consistent guidance and support played a vital part in my academics. Next, I would like to thank my family. It is really hard to find the words to thank them for their contribution in my life. To my sisters: Aafia, whose motivational quotes kept my morale up throughout the course of this work. Sobia, the best help and source of feedback I had all along my academics. Qudsia, for her endless stories and jokes whenever I felt down. To Shahid and my nephews, Abdullah and Rayan, for the refreshing talks. To my parents Sajida and Iftikhar, the best human beings in my life. Their unconditional love, support and hard-work is the reason behind any success that I had and I dedicate this thesis to them.

UIC

ACKNOWLEDGMENTS (Continued)

Contribution of Authors

I would like to acknowledge Prof. Ashfaq Khokhar for his feedback related to the selection of the compute intense problems addressed in this work. His feedback helped improve arbitration aspect of Re-gridding architecture discussed in chapter 2 and scalability aspect of matrix inversion architecture in chapter 3. Prof. Rashid Ansari and Prof. Ashfaq Khokhar gave consistent feedback on improving the presentation of the proposed solutions. Gregory Nash helped in selecting the tools for the experimentation. His feedback helped improve the architecture of FpMA units as well as the openCL implementation for Re-gridding architecture in chapter 2 .

TABLE OF CONTENTS

<u>CHAPTER</u>	<u>PAGE</u>
1 INTRODUCTION	1
1.1 Heterogeneous Computing Platforms	1
1.1.1 Design Space for Heterogeneous Computing Platforms	2
1.1.2 Heterogeneous Computing Architectures in Embedded Systems	4
1.2 FPGAs as a High-performance Computing Platform	6
1.2.1 Programmability	7
1.2.2 Application domains	8
1.3 Contributions	11
1.4 Thesis organization	13
 2 RE-GRIDDING ARCHITECTURE FOR ACCELERATING NON-UNIFORM FAST FOURIER TRANSFORM	 15
2.1 Introduction	15
2.1.1 Spectral Sampling Trajectories	17
2.1.2 Re-Gridding or Data-Translation	23
2.2 Related Work	26
2.3 Proposed Re-Gridding Architecture	27
2.3.1 Points Representation	28
2.3.2 Data-translator Components	31
2.3.3 Data-flow through the System	37
2.4 Hardware Implementation	38
2.4.1 Optimized Verilog Based Implementation	38
2.4.2 OpenCL Implementation	42
2.4.3 Comparison of Implementations	53
2.5 Comparison with Related Work	54
2.5.1 Verilog Implementation	55
2.5.2 OpenCL Implementation	56
2.6 Conclusions and Future Work	62
 3 HARDWARE EFFICIENT ARCHITECTURE FOR MATRIX INVERSION	 63
3.1 Introduction	63
3.1.1 Gauss-Jordan Elimination Algorithm	64
3.2 Related Work	66
3.3 Hardware Optimization using Modified Gauss-Jordan Algorithm	67
3.3.1 Pipeline Utilization	71

TABLE OF CONTENTS (Continued)

<u>CHAPTER</u>		<u>PAGE</u>
3.4	Proposed Architecture and Data-flow	71
3.4.1	Data-flow	72
3.4.2	Scalability	73
3.5	Performance Analysis	74
3.5.1	Performance comparison with related work	75
3.6	Hardware Complexity Analysis	76
3.7	Experimental Setup and Results	78
3.8	Conclusions and Future Work	80
4	A HIGH PERFORMANCE ARCHITECTURE FOR COMPUT- ING BURROWS-WHEELER TRANSFORM ON FPGAS	89
4.1	Background	89
4.2	Related Work	92
4.3	Flowthrough FIFO based technique for computing BWT in Hardware	94
4.4	Proposed architecture and data-flow	95
4.4.1	Suffix Block and Address Generator	95
4.4.2	Parallel Suffix Sorter	95
4.4.3	FIFO Network	97
4.4.4	BWT index generator	97
4.5	Complexity Analysis	98
4.5.1	Hardware Complexity Analysis	98
4.5.2	Time Complexity Analysis	99
4.6	Experimental Setup & Results	100
4.7	Summary	106
5	AN FPGA BASED HIGHLY PIPELINED AND SCALABLE ARCHITECTURE FOR MEDIAN FILTERING	107
5.1	Introduction	107
5.2	Related Work	109
5.3	Proposed Architecture and Data flow	111
5.3.1	Row Sorter	111
5.3.2	Sorted-Row Buffers	111
5.3.3	Merger block	114
5.3.4	Data-flow	115
5.4	Results and Discussion	115
5.5	Conclusion and Future Work	120
6	CONCLUSION AND FUTURE DIRECTIONS	123
	CITED LITERATURE	126

TABLE OF CONTENTS (Continued)

<u>CHAPTER</u>	<u>PAGE</u>
APPENDIX	138
VITA	142

LIST OF TABLES

<u>TABLE</u>		<u>PAGE</u>
I	COMPARISON BETWEEN DEDICATED HARDWARE AND CPU BASED SOLUTIONS	3
II	THROUGHPUTS FOR VARIOUS SIZES OF TILE, CONVOLU- TION WINDOW AND TARGET SET T	41
III	POWER AND PERFORMANCE FOR DIFFERENT TRAJECTO- RIES	46
IV	PERFORMANCE FOR LARGE MATRIX SIZES IN TERMS OF NUMBER OF CLOCK CYCLES	77
V	PERCENTAGE REDUCTION IN HARDWARE RESOURCES FOR FLOATING-POINT COMPUTATION UNITS COMPARED TO PIPELINED IMPLEMENTATION OF GJ	78
VI	PERFORMANCE FOR VARIOUS MATRIX SIZES	79
VII	COMPARISON OF HARDWARE COMPLEXITY BETWEEN PAR- ALLEL SORTER IN AND PARALLEL SUFFIX SORTER USED IN OUR PROPOSED ARCHITECTURE FOR BWT	100
VIII	PERFORMANCE COMPARISON OF PROPOSED BWT ARCHI- TECTURE WITH WEAVESORTER AND PARALLEL SORTER FOR A STRING OF SIZE 128 AND $LCP = 8$	105
IX	THROUGHPUT IN TERMS OF IMAGE SLICE RATE FOR VAR- IOUS NUMBER OF <i>MEDIANPIPES</i>	121

LIST OF FIGURES

<u>FIGURE</u>		<u>PAGE</u>
1	Heterogeneous Computing platform	2
2	Design Space for Heterogeneous Computing platforms	5
3	Complexity of Embedded Software. Source: (1)	6
4	Example of heterogeneous SoC architecture for mobile devices	7
5	NuFFT algorithm flow	16
6	Radial Trajectory. (Generated using (2))	19
7	Spiral Trajectory. Generated using (2)	20
8	Polar/Curvilinear trajectory. Generated using (3)	21
9	Cartesian Trajectory	22
10	Visual Representation of the Re-gridding process	22
11	Pseudo code for Re-gridding processes in 2-dimension. The values of s_i and t_k are denoted by $v(s_i)$ and $v(t_k)$ respectively. (s_{i_x}, s_{i_y}) and (t_{k_x}, t_{k_y}) denote the respective co-ordinates.	24
12	Mapping between Tiles and point FIFO. The boundaries of the tiles are shown by dotted blue line and tiles are numbered from 1 - 12. Grey dots indicates the target points, crosses indicates the source points and dotted red box indicates the convolution widow for each source point. The FIFOs at the bottom are shown filled with the source points and have the same index as the tile they correspond to.	29
13	The proposed Re-Gridding (Data Translator) architecture depicting the data-flow through its components. The solid lines indicate the data flow and dotted lines indicate control lines.	30

LIST OF FIGURES (Continued)

<u>FIGURE</u>		<u>PAGE</u>
14	The convolution operation of the source point with the interpolation kernel function to update the target point values. The red box on the tile indicates the convolution window.	34
15	Logical connection of Floating point Multiplier and Adder (FpMA) units to target points in fetched tile indicated by color coding: (a) Array of FpMAs units, each box represents an FpMA unit (cell). The co-ordinates in each box indicates the index of the FpMA unit in a 2-d array format (b) A tile of size 8 x 8. Each box represents a target point of the tile. The co-ordinates in each box indicate the co-ordinate of target point in the 2-d tile. For clarity, logical connection of (0,0) FpMA unit is indicated with dotted line as well	35
16	Architecture of the proposed Pipelined Floating Multiplier and Adder (FpMA) unit	37
17	OpenCL based Experimental Setup	43
18	Reading Order of Point-FIFOs could be decided a priori based on the known Trajectory. The figure shows the target frame divided into 16 tiles.	44
19	Comparison of Hardware Resources for TSA and GA based Architectures for target set size of 1024×1024 , tile size of 256×256 , $\alpha = 4$ and $\sigma = 2$	47
20	Throughput comparison for trajectory specific architectures	48
21	Throughput comparison for trajectory specific architectures	49
22	Comparison of Power Consumption and Hardware Resources for tile size of 256×256 and 128×128 when the size of target point set is 512×512	50
23	Scalability of Hardware Resources, MFLOPS and throughput with the target set size and tile size for GA based architecture	51
24	Scalability of Hardware Resources, MFLOPS with the target set size for TSA based architecture. The vertical axis on left indicates the count for resources and the vertical axis on right is for MFLOPS.	52
25	Comparison of OpenCL implementation with Optimized Verilog based implementation for a target set of size 256×256 , $\alpha = 4$ and $\sigma = 2$	53

LIST OF FIGURES (Continued)

<u>FIGURE</u>		<u>PAGE</u>
26	Performance comparison of the verilog implementation of the proposed architecture with existing FPGA (4; 5) and GPU (6) based techniques for a target point set of 256×256	56
27	Throughput comparison of Spiral Trajectory with related work	57
28	Comparison of Power efficiency for Spiral Trajectory with related work	58
29	Throughput comparison of Radial Trajectory with related work	59
30	Comparison of Power efficiency for Radial Trajectory with related work	60
31	Comparison of throughput and power efficiency for Random trajectory and a target set of size 1024×1024	61
32	An $N \times N$ matrix at the i^{th} iteration	68
33	Pseudo-code Modified Gauss-Jordan Algorithm	70
34	Steps to find invert of a sample 3×3 matrix. The augmented matrix is shown along with the contents of the memory. The reciprocal is found over three steps. The values in green indicate the values that required for further computation and are stored in the memory. Values in red indicate the values that are no longer required in computation	82
35	Simplified architecture of pipelined Implementation of Gauss-Jordan Algorithm. Two floating-point multiplication elements are required per row.	83
36	Simplified data-flow of Pipelined Implementation for Modified Gauss-Jordan Algorithm. A single floating-point multiplication is required per row as compared to two for the pipelined implementation of original Gauss Jordan algorithm (Figure 35)	84
37	Computational flow scalability	85
38	Block diagram of the data path	85
39	Comparison of proposed architecture (InvArch) ($k = 8$) against Duarte et al. (7) ($k = 8$), Matos et al (8) ($b = 8$) and pipelined GJ implementation with same hardware resources as our proposed architecture ($k = 8$) for a 1024×1024 matrix.	86

LIST OF FIGURES (Continued)

<u>FIGURE</u>		<u>PAGE</u>
40	Comparison of the hardware resources required for implementing floating point blocks for different values of k . (a) gives the number of Adaptive Look-Up Tables (ALUT)s and (b) gives the number of DSP blocks. The numbers are for the area-optimized implementation of Altera Floating point units on Startix IV FPGAs	87
41	Comparison of pipeline-utilization for modified-gauss jordan algorithm (left) with Gauss-jordan algorithm utilizing same number of floating point multiplication units (right). Iterations are indicated with distinct colors. Note that we assume number of NE blocks to be 8 in this figure i.e., 8 multiplication and subtraction units. During the i^{th} iteration, for forward-elimination, result of normalized pivot ($m_{ik} = a_{ik}/a_{ii}$) is buffered to be eliminate the elimination rows. For back-substitution, normalized pivot row element is available in time for the first 7 iterations but needs a single cycle stall for the last iteration. For same hardware resources, the pipeline on right needs stall cycles every iteration.	88
42	Visual illustration of BWT: input string s along with Matrix M , matrix Q and the Burrows-Wheeler Transform (BWT)	90
43	Limited sized suffices for a Single Block	96
44	Connectivity of FIFOs between two stages	98
45	Dataflow through the FIFO pipeline	99
46	Effect of the size of Longest Common Prefix (LCP) on the total memory used on pipeline of FIFOs for different string lengths.	101
47	Effect of size of Longest Common Prefix (LCP) on the number of Register consumed	102
48	Effect of the length of String on the total memory used on pipeline of FIFOs for different different sizes of LCP	103
49	Effect of LCP on Throughput using a fixed string of size two thousand characters. Longer LCP corresponds to more hardware complexity but higher throughput.	103
50	Performance (in terms of number of clock cycles) for various length of strings having fixed LCP = 16	104

LIST OF FIGURES (Continued)

<u>FIGURE</u>		<u>PAGE</u>
51	Block Diagram of three <i>medianPipes</i>	112
52	First stage sorting using a single comparator	113
53	Merger Block using three Comparators	114
54	Parallel computation of median values in column C_2 using two <i>medianPipes</i> . The ordered pairs (R, C) represent a pixel, where R and C represents row and column respectively. Rows $R_1 - R_6$ are sorted in parallel. Merging sorted rows at R_1, R_2 and R_3 compute the median value at $(2, 2)$. R_2, R_3 and R_4 compute median at $(3, 2)$	116
55	Computation for median values across the columns. Since a single column of image slice is read from the external memory, if the computation of median values for C_2 starts at i^{th} cycle, the computation for C_3 will start at $(i + 1)^{th}$, C_4 at $(i + 2)^{th}$ and so on. The <i>medianPipes</i> dedicated to the columns are re-used once the median values for the columns are computed	117
56	Resource usage for various number of <i>medianPipes</i>	118
57	Resource usage for pixel sizes of 8, 16 and 32 bits	119
58	(a) Trend in pixel rate with respect to pixel size using 128 median pipes for 768 pixels(b) Trend in pixel rate with respect to image slice size (using 8-bit pixel)	120

LIST OF ABBREVIATIONS

ALUT	Adaptive Look-up Tables
BWT	Burrows Wheeler Transform
CS	Compressed Sensing
CT	Computed Tomography
DDR	Double Data Rate
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FIFO	First-In-First-Out
FPD	Floating-point Division
FPM	Floating-point Multiplication
FPS	Floating-point Subtraction
FPGA	Field Programmable Gate Array
FpMA	Floating-point Multiplier and Adder
GA	Generic Arbiter
GJ	Gauss Jordan
GPU	Graphics Processing Unit
IoT	Internet of Things

LIST OF ABBREVIATIONS (Continued)

LCP	Longest Common Prefix
MAC	Multiply and Accumulate
MFLOPS	Mega Floating-point Operations per Second
MRI	Magnetic Resonance Imaging
MUX	Multiplexer
NE	Normalization and Elimination
NuDFT	Non-uniform Discrete Fourier Transform
NuFFT	Non-uniform Fast Fourier Transform
RAM	Random Access Memory
SAR	Synthetic Aperture Radar
SoC	System-on-Chip
TSA	Trajectory Specific Arbiter
VLSI	Very Large Scale Integration

SUMMARY

The use of application-specific architectures has gained popularity in implementing solutions to many compute intense tasks in recent years due mainly to the performance and power limitations associated with high-end processor based systems. Considering their superior performance at lower energy requirements, such targeted solutions have found applications in a number of domains including Embedded Systems, Big-Data processing in Data centers, Computer Network Security, Medical Imaging and Internet of Things (IoT). These application specific hardware solutions could be used as stand-alone computation devices or as part of a heterogeneous computing system. Heterogeneous Embedded Systems, in the form of System-on-Chip (SoC), are used in applications ranging from smart hand-held devices, cars, drones to numerous other battery-operated electronic devices. Their higher performance-to-power ratio for compute intense tasks, like video processing, make them an ideal fit for battery-operated devices. In computer networks domain, specialized architectures are designed for real-time Deep Packet Inspection at high speed for network security. These hardware solutions also power the energy-efficient data centers for big data processing and storage. With the growing interest in the Cloud-centric IoT, there is a high potential for using power-efficient heterogeneous systems for sensing, analysis and visualization of tremendous amount of data associated with IoT. At the IoT node level, design space exploration of the hybrid (CPU-FPGA-GPU) nodes are already being explored.

SUMMARY (Continued)

Considering the tremendous potential of application specific hardware solutions in diverse range of applications, this thesis targets development of efficient solutions for various memory and compute-intense applications. We specifically target FPGAs as the computing platform considering the power-efficiency and the enormous design space associated with the platform due to its inherent flexibility. The hardware solutions proposed as part of this thesis have shown improved performance at reduced power compared with existing techniques.

The first computationally complex task we consider is the re-gridding process in Non-uniform Fast Fourier Transform (NuFFT). Some applications of NuFFT include Synthetic Aperture Radar (SAR), Computed Tomography (CT), and Magnetic Resonance Imaging (MRI). At the heart of the NuFFT task is a Gridding algorithm that maps non-equispaced data points onto a uniform grid using an interpolation function. This interpolation step, also referred to as re-gridding or data-translation, is known to consume 88 – 90% of the overall computation time of NuFFT. The dissertation proposes a novel FPGA based architecture for the memory and compute intense re-gridding process. The proposed architecture is based on the novel use of customizable hardware components such as FPGA block memory in First-In-First-Out (FIFO) configuration, fill-status based arbiter, distributed storage of grid-points and an array of pipelined single precision floating point multipliers and adders. The proposed solution exhibits high performance over a wide range of configurations, data sizes and spectral sampling trajectories. The architecture targets a generic solution for unknown trajectories and a trajectory-specific variant that targets well known trajectories in MRI and SAR. Compared with existing FPGA-based solutions, throughput is improved by over 9.6 times whereas compu-

SUMMARY (Continued)

tational power efficiency (in terms of MFLOPS/Watt) is improved by over 15 times. Compared with GPU-based techniques, over 9.6 times higher MFLOPS/watts was achieved at a comparable performance.

Second, the thesis proposes an efficient architecture (*InvArch*) for computing matrix inversion using Gauss-Jordan Elimination method. The proposed architecture exploits parallelism through pipelined floating-point computational units and reduces the number of floating-point multiplication units required compared with the existing pipelined implementations. The reduction in multiplication units results in over 80% reduction in hardware for floating point computation units. The architecture performs in-place inversion and provides scalability across the rows and columns. Hardware efficiency is achieved by reaping benefit from regularity in computation and better utilization of pipelined computational resources. Multiple rows are normalized within an iteration of Gauss-Jordan algorithm that allows reduction in number of floating-point multiplication units in the elimination step. In addition to implementing the architecture, an analytical performance model is also developed for *InvArch* and some related works. *InvArch* achieves performance comparable to reference architectures in terms of clock cycles and throughput while using significantly less hardware resources.

Third, an FPGA based architecture for Burrows Wheeler transform (BWT) has been proposed. BWT has applications in diverse areas such as compressed string matching, biological sequence analysis, error correction, and channel encoding. Numerous efforts have been made to improve the performance of BWT in software and hardware. Its use in real-time applications such as deep packet inspection and channel coding requires efficient hardware implementations

SUMMARY (Continued)

that must yield high throughput. We explore novel hardware techniques to compute BWT. Our techniques are based on using a limited length of suffixes, a parallel suffix sorter, and an efficient First-In-First-Out (FIFO) memory pipeline to sort these suffixes. The hardware complexity analysis shows that our technique scales linearly with the length of string and the claim is verified by the hardware synthesis results. In terms of performance, our technique outperforms the existing hardware-based techniques by over four times.

Fourth, a novel, FPGA-based, highly pipelined and scalable architecture (*MedianPipe*) for median filtering has been proposed. Median filters and its variants are widely used for noise suppression in image processing. All variants of median filter depend on the computation of median values. *MedianPipe* is a highly pipelined architecture and hence an ideal fit for FPGAs. Multiple *MedianPipe* modules are used depending on the size of image slice. The overall hardware complexity of proposed architecture scales linearly with image-slice size. The architecture for *MedianPipe* is based on the principle of merge sort and uses a median window of size 3 x 3. Without loss of generality, the pixels of an image slice are assumed to be read in a column major format. All the median values within the column of the image slice can be computed in parallel using multiple *MedianPipes*. The computation of median values in the following column is delayed by a clock cycle. Hardware resources scale linearly by varying the number and size of pixels. The pixel rate achieved for various pixel sizes is well above 124 MHz which is the standard for 1080p High-Definition.

Finally, the thesis is concluded by suggesting future directions for research based on the proposed solutions.

CHAPTER 1

INTRODUCTION

Processors had an enormous performance gain in 1990s and early 2000s due to the steady increase in clock frequency. Technology issues like power consumption, heat dissipation, leakage currents and signal propagation delays limit the increase in clock frequency beyond certain limits. Apart from clock frequency, another front for improving the performance of processing systems was to introduce parallelism at instruction level. But due to high data and control dependency of instructions in a program, instruction-level parallelism cannot solely be relied upon to achieve high performance demands in future. By exploiting parallelism at the thread- and data-level, and employing multiple low frequency processing cores with higher power efficiency enabled the processing systems achieve high-performance demands. In the multi/many core era, the performance improvement is achieved using multiple, relatively simple cores instead of single complex processor and by expressing the parallelism in the code itself. Although the performance of the processing systems is increasing, computational demands of research, software development and scientific computing are also becoming more complex.

1.1 Heterogeneous Computing Platforms

Due to the performance and power limitations of the high-end processors, heterogeneous computing platforms have gained popularity in recent years. Since no processing core is an ideal fit for all computational tasks, the focus has shifted towards designing systems to achieve better

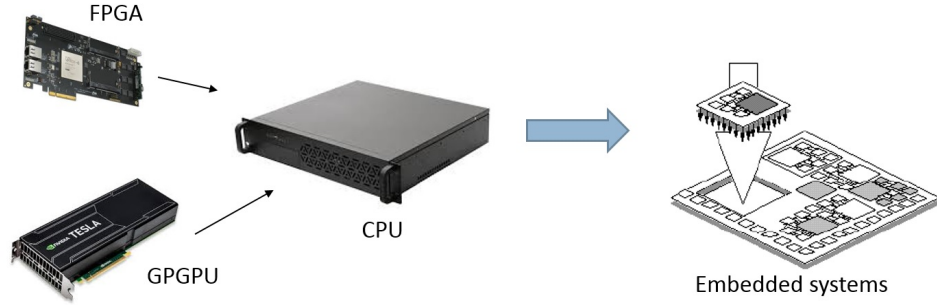


Figure 1. Heterogeneous Computing platform

performance for certain work loads. Computationally intense applications are mapped on to dedicated hardware solutions based on platforms like ASICs, GPUs and FPGAs. Owing to the ability of these solutions to provide orders of magnitude higher performance compared with the optimized implementations on general purpose processor cores at a lower power consumption (9), the use of these dedicated hardware accelerators is increasing in all fronts of complex computing. The performance, power and cost benefits at running complex tasks are outweighing the inherent disadvantage of additional programming effort required. A summary of comparison between general CPU-based and dedicated hardware solutions is given in Table I.

1.1.1 Design Space for Heterogeneous Computing Platforms

Design space for heterogeneous computing platforms can be roughly represented as shown in Figure 2. General purpose processor (or CPU) is the most flexible computing platform capable of running software for any application. But this flexibility comes at the cost of efficiency. Application software is executed in the form instructions and performance is improved

TABLE I

COMPARISON BETWEEN DEDICATED HARDWARE AND CPU BASED SOLUTIONS

Metric	Dedicated hardware solution	CPU based
Performance	<ul style="list-style-type: none"> - Memory bandwidth dedicated for data - No control latencies due to jump and branch 	Limitations: <ul style="list-style-type: none"> - Dark Silicon - Clock frequency Scaling
Ease and Flexibility of Programming	Improved in recent years	Better
Energy-efficiency	Better in term os total energy	
Cost	GPUs and FPGAs have bridged the gap	Better

by adding parallelism at Instruction, Data, Thread and Core level. As we move towards the right in Figure 2, computing platforms become less flexible but more domain-specific. FPGAs are reconfigurable devices that are capable of implementing high-performance, power-efficient application-specific custom implementations for the application at hand. The added flexibility of reconfigurability in these platforms give them an additional benefit compared to all other computing platforms. GPUs, in general, have greater computing power compared to FPGAs and CPUs. Compared with CPUs, instead of having extensive logic for flow-control and data caching, a bulk of processing nodes are added that makes them an ideal fit for computationally intense parallel tasks. Digital Signal Processors (DSPs) have domain-specific instruction set and specialized associated hardware. Flexibility is lower compared with CPUs and GPUs because writing application software requires detailed knowledge about the underlying hardware. Application-Specific Instruction set Processor (ASIP) has customizable instruction set where ASIP designer not only defines a part of instruction set but also the hardware associated for these custom instructions. Application Specific Integrated Circuits (ASIC) on the extreme are application-specific, fixed and non-flexible solutions that have the best power, form-factor and performance efficiency compared with other computing platforms. Since the architecture is fixed any change in application at hand requires redesign of ASIC. Cost and time of the overall design cycle for ASICs is one major concern for solutions based on ASICs.

1.1.2 Heterogeneous Computing Architectures in Embedded Systems

Complexity of embedded software has been increasing tremendously in recent years (Figure 3) and hence the performance expectations from embedded systems are also increasing. For

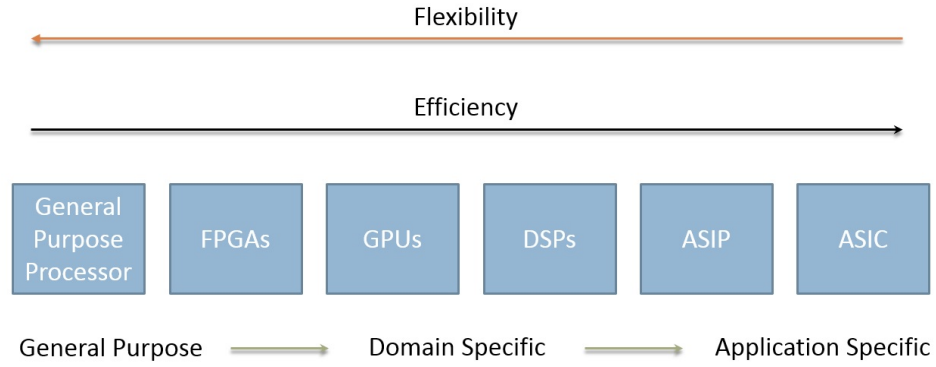


Figure 2. Design Space for Heterogeneous Computing platforms

example, in mobile devices, a processing unit has to serve an increasing number of varied applications like computational photography, High-Definition (HD) video capture and playback, interface with touch-screen and other wide range of sensors. As most of these applications are expected to run in parallel, a single Central Processing Unit (CPU) cannot serve all these applications efficiently.

In recent years, mobile devices employ System-on-Chip (SoC) based processing units that have application-specific solutions for various classes of applications. An example of one such SoC is *snapDragon* from Qualcomm (10) that utilizes dedicated processing cores for graphics processing, sensor interfacing, computational photography and video playback as shown in Figure 4.

The growth of embedded systems is much higher compared with other computing platforms like desktop computers (11). As the future of computing is shifting towards embedded systems

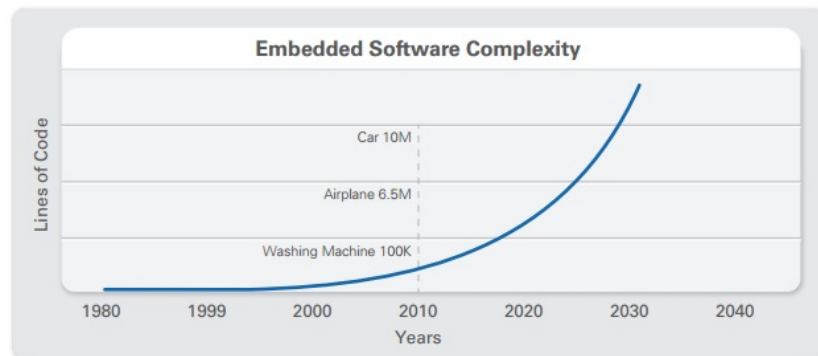


Figure 3. Complexity of Embedded Software. Source: (1)

and the increased reliance of embedded systems on heterogeneous SoCs indicate the great importance of heterogeneous architectures in computing in future.

1.2 FPGAs as a High-performance Computing Platform

FPGAs were traditionally used for prototyping. Due to the improvement in VLSI technology coupled with FPGAs inherent configurable nature attracted their use in acceleration of compute and data intense applications. Reconfiguration allows the architecture to adapt according to the change in computation demands. Another advantage of FPGAs is their lower power consumption compared to GPUs.

Some of the major factors that allow FPGAs to achieve hardware acceleration compared to general-processor based software solutions are (12):

- Compared with traditional processor based system there is no need for fetching instructions from the memory and hence memory bandwidth is completely allocated for data

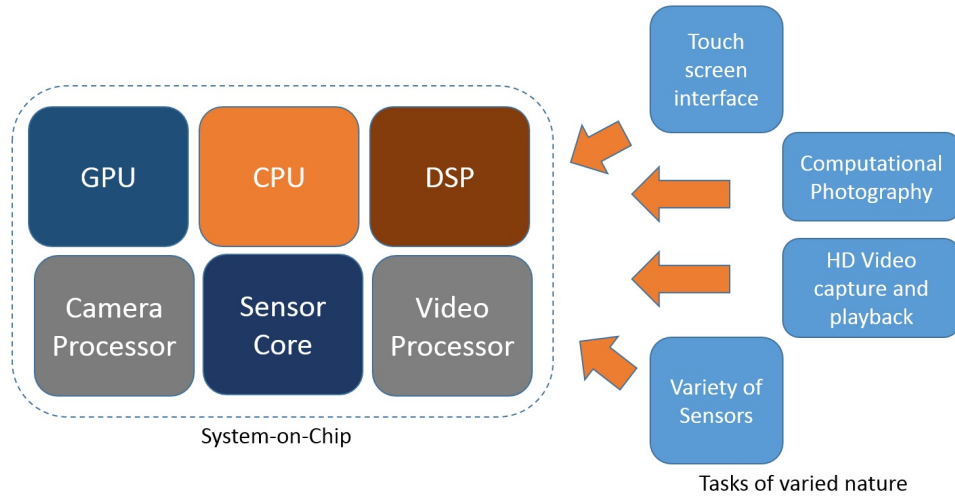


Figure 4. Example of heterogeneous SoC architecture for mobile devices

- Parallelism can be achieved by generating multiple copies of processing units. Compared to software technique, this can be viewed as loop unrolling.
- Straight-forward implementation of basic blocks of codes incurs no additional delays
- There are no latencies related to the control instructions like jumps and branches. Data-paths and control units are implemented in hardware that select the output based on control logic

1.2.1 Programmability

Compute Unified Device Architecture (CUDA) and OpenCL are the two major programming frameworks for GPUs. These programming frameworks provide abstraction from the underlying hardware. For FPGAs, Hardware Description Languages (HDLs) Verilog and VHDL, are the

two main tools for describing custom hardware. In recent years, FPGA vendors have provided support for high level languages such as Impulse C, Handel C and Mitrion C (13). Lately, FPGA vendors have also provided support to program FPGA platforms using selected features of the OpenCL programming framework. Efforts have also been made to standardize various aspects, like interfaces, in reconfigurable computing and various communities like OpenFPGA (14) and Center for High-Performance Reconfigurable Computing (15) have been developed for this purpose. The standardized interfaces would allow easy interfacing of FPGA solutions to any system.

Considering the advances in hardware and the programmability of FPGAs along with promise of high performance-to-power ratio motivates energy-efficient and high-throughput hardware solutions for a wide range of complex computational tasks on FPGAs. We look at some of the areas of applications for the hardware accelerators.

1.2.2 Application domains

This section provides overview of some domains that benefit from the hardware accelerators.

Big-Data Processing in Data centers

FPGAs have been successfully used in data-centers to accelerate Big-data services in data centers. Microsoft’s Catapult project (16) explored the use of FPGAs to improve performance and power-efficiency of various tasks in the data centers. Since GPUs are also capable of providing massive parallelism, their use was also explored but the high power consumption of current high-end GPUs made them unfit for conventional data-centers. Also, latency-sensitive ranking tasks mapped better on FPGAs compared to GPUs. A medium scale FPGA deployment

resulted in 95% increase in throughput of ranking workload compared with software solution and added only 10% overhead in power consumption and less than 30% overhead to cost of individual server. The study concluded FPGAs to be a viable path forward towards continued improvement in cost and capability of servers especially considering the power and technology barriers in high-end processors.

Internet of Things

Deployment of IoT implies massive increase in the number of devices communicating with each other as well as with the data centers through network communication infra-structure. This will result in massive increase in the computing performance and power requirements in the data centers. As discussed earlier, FPGA based solutions have already proved to be viable path in improving the performance of data centers. At the network communication level, there will be a tremendous increase in bandwidth requirements and hence higher need of energy-efficient network communication and security. FPGAs being energy-efficient and high-performance computational device, is capable of playing a significant part in improving the network communication and security. At the device and IoT gateways level, power-efficient compute solutions will be required especially for the battery operated devices. Considering the reconfigurable nature and low-power requirements of FPGAs, they are capable of playing significant role at this level also. The promise of FPGAs at the node level is well-recognized by tech industry giants. Intel, in 2014, announced an Atom-FPGA hybrid to target the IoT applications (17).

Computer Network Security

The scale of computer networks have been increasing continuously and rapidly. The future with IoT, due to the addition of high number of heterogeneous devices, protocols and network traffic, would necessitate the use of fast, real-time network security techniques. FPGAs have already been found useful in real-time security of Computer Networks using Deep Packet Inspection for intrusion detection (18). Their use in network security is expected to increase manyfold.

Digital Signal Processing

Advances in hardware accelerator technology have extensively expanded the application domain in Digital Signals Processing (DSP). The functional efficiency of hardware and programmability of software renders configurable platforms like FPGAs attractive to be used in wide range of DSP applications including video, audio, speech and control. Due to the abundance of programmable logic, various functional units of DSP systems can be directly realized in hardware. Fine-grained parallelism, which is often required for high-sampling rates and distributed computing in DSP applications, can also be extracted utilizing the abundant configurable logic available. FPGA based designs should be highly pipelined in order to achieve better clock frequency. Many operations in DSP applications, like image and speech processing, are capable of being highly parallelized and pipelined. This renders FPGAs a naturally suited platform for such applications (19).

Recent advancements in FPGA architecture to include special hardware for floating point arithmetic (20) has helped FPGAs overcome the earlier disadvantage of handling the floating

point arithmetic. The added hardware is highly pipelined that allows FPGAs to operate at high clock frequencies.

Computational Biology

The volume of available genetic data is doubling every six months (21) and hence the conventional computing systems have not been able to keep up. Sequence alignment is one such computationally intense application in computational biology that has benefited from hardware accelerators. Smith Waterman algorithm is one fundamental sequence alignment algorithm that is implemented using dynamic programming and has a quadratic computational complexity in software. When implemented in hardware, it achieves linear computational complexity due to the massive parallelism provided by the FPGAs (21).

Sensor Systems and Sensor Networks

In sensor systems, the sensors having capabilities such as self-diagnostics and decision-making are called smart sensors. Smart sensors are sometimes implemented using an embedded FPGA-based device and benefit from the small size, low power consumption and high computational accuracy of FPGA based devices (22). Other applications of FPGA include Wireless Sensor Networks (WSNs) where they are used for reducing the amount of data-transmissions and providing flexibility at the sensor node (22).

1.3 Contributions

This thesis reaffirms the tremendous potential of targeted hardware accelerators to provide high-performance and energy-efficient solutions to a wide range of data, memory and compute intense tasks in various application domains. Some key contributions are listed below:

1. *A novel memory-optimized and power-efficient architecture to accelerate the memory and compute intense re-gridding process in NuFFT (23; 24):* Re-gridding step is known to be the most time-consuming task (over 90 % of the whole process) in the computation of NuFFT. Re-gridding involves multiplication of each source point with an interpolation kernel function. Various practical trajectories were considered for the sampling of source points and high throughputs were achieved for wide range of configurations. Compared with GPU implementations, comparable throughputs were achieved at much lower power consumption. In addition to targeting arbitrary trajectory, proposed architecture was extended to target known trajectories in MRI and SAR applications. Compared with existing FPGA based techniques, up to 15 times better power efficiency was achieved in terms of MFLOPS/Watt at up to 7.35 times the throughput. Compared with GPU based technique, up to 9.6 times better power efficiency was achieved at comparable throughput.
2. *A novel and hardware efficient architecture for matrix inversion (25):* The architecture targets Gauss-Jordan elimination method for matrix inversion and is based on normalizing multiple rows in an iteration benefiting from the pipelined nature of floating point blocks and thereby reducing the number of multiplication units required. This results in 80% reduction in hardware resources for floating point computational logic compared to the existing pipelined implementation. The proposed architecture also provides better scalability and hardware efficiency at comparable performance. An analytical model was also developed for the proposed architecture as well as a scalable version for the architecture proposed in (7).

3. *A high-performance hardware technique for the computationally intense Burrows-wheeler-Transform (26)*: The use of Burrows-Wheeler Transform (BWT) in real time applications requires efficient hardware implementations that must yield high throughput. A high performance hardware technique to compute BWT on an FPGA has been proposed. The hardware and time complexity of the proposed technique scale linearly with the length of input string.
4. *A highly pipelined FPGA based architecture for median filtering (27)*: Various variants of median filters have been a key component for noise reduction in image processing. Median filter is the core process in all these variants. Efficient hardware implementations of median filter is hence highly desirable. A highly pipelined architecture for median filtering has been proposed as part of this thesis. The proposed architecture scales linearly with hardware resources and pixel sizes and achieves Pixel rates higher than those required for standard 1080p HD video.

1.4 Thesis organization

Chapter 2 introduces NuFFT and its applications followed by the description of the grid-ing algorithm used in the computation of NuFFT. The proposed architecture for known and arbitrary trajectories, is described by explaining various functional components and the data-flow. Verilog and OpenCL based Experimental setup is described. At the end of the chapter, comparison between optimized Verilog and OpenCL based solution is made and the proposed architecture is compared with GPU and other FPGA based solutions.

Chapter 3 describes hardware-efficient architecture for Gauss-Jordan based Matrix Inversion. It also gives an analytical model for the proposed architecture and some related work for comparison purposes.

Chapter 4 explains the proposed high-performance architecture for BWT. It briefly describes BWT and existing hardware based techniques to compute BWT. The proposed architecture and data-flow is explained in section 4.4 followed by a detailed analysis of hardware and time complexity analysis in 4.5. Analysis is followed by experimental setup, results and comparison with related work.

Chapter 5 starts with the description of Median filter and its applications followed by brief overview of other hardware based median filtering. Next the proposed highly-pipelined architecture for Median filter is presented followed by the experimental setup and results.

Chapter 6 concludes the thesis and points out the directions and avenues for future work.

CHAPTER 2

RE-GRIDDING ARCHITECTURE FOR ACCELERATING NON-UNIFORM FAST FOURIER TRANSFORM

Part of this chapter is based on our work that has appeared in the proceedings of 22nd IEEE International Symposium on Field-Programmable Custom Computing Machines (23) and 24th IEEE International Conference on Field Programmable Logic and Applications [Copyright © 2014 IEEE] (28).

2.1 Introduction

The Discrete Fourier Transform (DFT) can be viewed as the Fourier Transform of a periodic and regularly sampled signal. Non-Uniform Discrete Fourier Transform (NuDFT) is a generalization of the DFT for data that may not be regularly sampled in spatial or temporal domain. This flexibility allows for benefits in situations where sensor placement cannot be guaranteed to be regular or irregular sampling patterns allow more efficient information analysis. Some of the salient applications of NuFFT include Synthetic Aperture Radar (SAR) (29), Computed Tomography (CT) (30) and Magnetic Resonance Imaging (MRI) (31).

Consider a set S of size M consisting of non-equispaced source samples x . Let N be the number of equispaced Cartesian grid cells of dimension d belonging to set I_N and let f_j be the

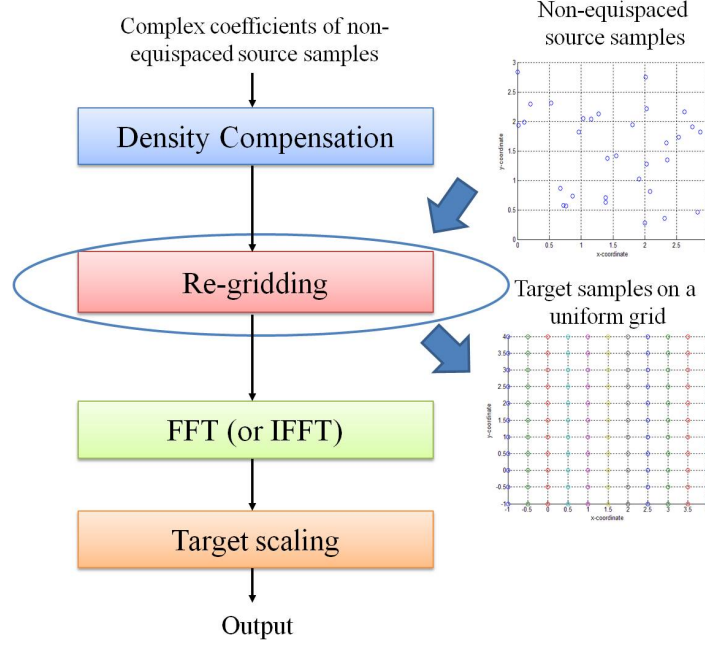


Figure 5. NuFFT algorithm flow

complex Fourier co-efficient corresponding to the grid cell j . NuDFT can be formally defined as in equation Equation 2.1 (32).

$$f_k = \sum_{j \in I_N} f_j e^{-2\pi i j x_k} \quad (2.1)$$

Direct computation of NDFT requires $O(MN)$ arithmetic operations. Non-Uniform Fast Fourier Transform (NuFFT) helps computing NuDFT with some approximation in $O(M + N \log N)$ complexity (4). It uses the Fast Fourier Transform (FFT) in combination with an approximation scheme.

The main steps of the NuFFT algorithm are shown in Figure 5. The first step is density compensation of the non-uniform samples in the source domain. Density compensation is followed by translating non-uniformly distributed data to a uniform grid using interpolation kernel functions. This re-gridding is performed in the source domain and is also referred to as data-translation, gridding or re-sampling. Re-gridding is followed by domain transformation using FFT (or IFFT). Finally, the transformed data is scaled in the target domain. For some applications, the trajectory of the source samples is known a priori whereas in others the trajectory may be arbitrary and the samples are available in a random order (33).

2.1.1 Spectral Sampling Trajectories

Performance of the re-gridding process greatly depends on the sampling trajectory of the source points. Solutions tailor made for one particular sampling trajectory may perform poorly for the other. In this work, we propose memory and power efficient generic solution for any potential sampling trajectory. We also target some of the most common trajectories used in the applications of NuFFT. MRI, which is one major application of NuFFT for image reconstruction, employs radial (Figure 6) and spiral (Figure 7) trajectories in addition to general Cartesian (Figure 9) trajectory. Another important application of re-gridding is in SAR processing for image reconstruction using the Polar/curvilinear trajectory (Figure 8).

Random trajectory

A number of works in literature ((31; 4; 33; 5; 24)) target random trajectories to give a generic solution for arbitrary sampling. Random trajectory is of growing interest in Compressive

sensing (31) where the aim is to reconstruct a signal by sampling at a rate much lower than the Nyquist rate. An application for random sampling could be in Wireless Sensor Networks.

Radial trajectory

Radial trajectory is based on equispaced sampling along straight lines. These straight lines are distributed equiangularly through out the spectrum and all pass through the origin. Radial trajectory has applications in Magnetic Resonance Imaging (MRI) (34), Multi-dimensional Wavelet analysis (35) and parallel-beam tomographic applications (36).

Radial trajectory has two major advantages compared to the general Cartesian trajectory.

1. Since all the straight lines pass through the origin, the repeated sampling gives signal average over image space and hence are more tolerable to motion effects (37).
2. Under-sampling has a lower impact on Signal-to-Noise-Ratio (SNR) of a radial trajectory based reconstructed image compared to Cartesian trajectory. Therefore, lesser samples in radial trajectory give same image quality at a reduced scan time (38).

Sampling function for the radial trajectory is given in equation Equation 2.2. The data is sampled equidistantly in radial h_r and angular h_θ direction. The spacing in angular direction is decided based on C_θ parameter which is the equiangular projections that cover 180° with initial projection at 0° . Δh_r is the spacing in radial direction and $2C_r$ are the number of samples on each radial projection.

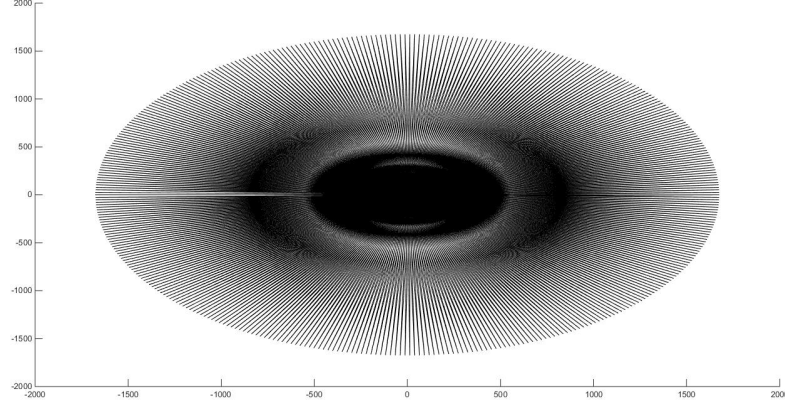


Figure 6. Radial Trajectory. (Generated using (2))

$$s(h_r, h_\theta) = \left(\frac{\pi}{C_\theta}\right) \Delta h_r \sum_{m=0}^{2C_\theta-1} \sum_{n=0}^{C_r-1} [\delta(h_\theta - m(\frac{\pi}{C_\theta})) \delta(h_r - n\Delta h_r)] \quad (2.2)$$

Spiral trajectory

Spiral trajectory is useful in MRI applications. It allows oversampling near the spectrum origin, which is desired for most MRI images, while meeting the fast-imaging requirements (39). The use spiral trajectory for MRI is compelling for dynamic studies in cardiovascular, spectroscopy, breast, renal and functional neuroimaging (40). The effectiveness is mainly due to its efficient use of gradient power and moment-nulling motion compensation (40).

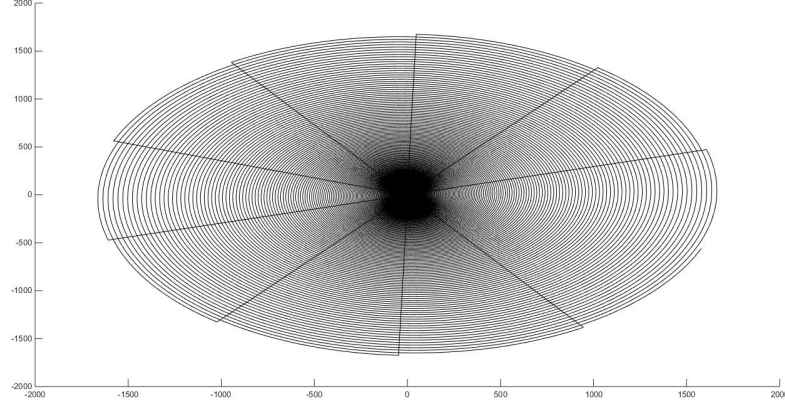


Figure 7. Spiral Trajectory. Generated using (2)

Spiral trajectory, $\mathbf{h}(t)$, in the (h_x, h_y) -plane can be formally defined by equation Equation 2.3. (41).

$$\mathbf{h}(t) = A\omega(t)e^{i[\omega(t)+\psi_0]} \quad (2.3)$$

where $\omega(t)$ is a function of time t and an important parameter in design of trajectory, $\psi_0 = 2\pi(p-1)/C_p$, C_p is the number of interlaced stacked spirals and p is the spiral index within the stack. This work does not target real-time generation of trajectory and utilizes the stack-of-spiral trajectory generated by (2).

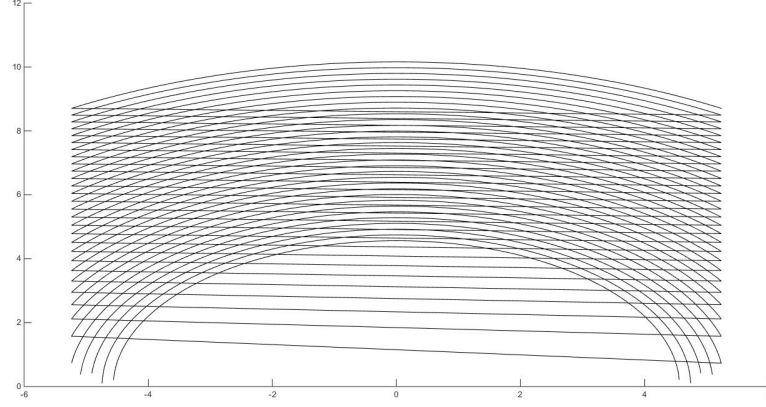


Figure 8. Polar/Curvilinear trajectory. Generated using (3)

Curvilinear trajectory

Curvilinear sampling trajectory (Figure 8) is encountered in Synthetic Aperture Radar (SAR) applications. The challenge in SAR applications is in real-time and energy-efficient image reconstruction which makes fast hardware implementations of this trajectory important (42).

In this chapter, we propose a generic architecture to cater arbitrary trajectories as well as a variant of the proposed architecture that is targeted towards known trajectories. Performance comparison demonstrates that prior knowledge of the trajectory can be used to improve the performance of NuFFT computation. The trajectory specific architecture targets Radial, Spiral, Curvilinear and Cartesian trajectories.

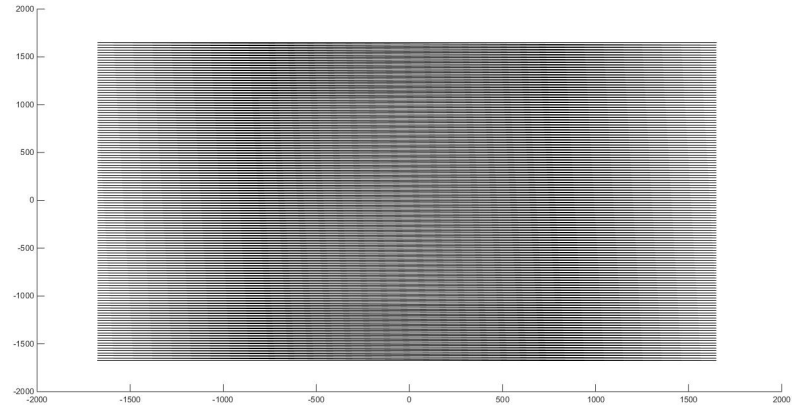


Figure 9. Cartesian Trajectory

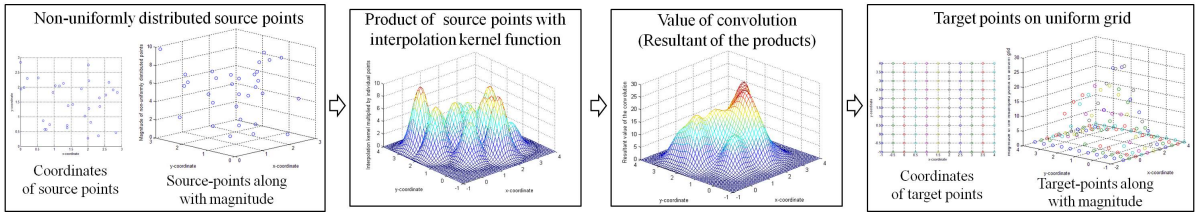


Figure 10. Visual Representation of the Re-gridding process

2.1.2 Re-Gridding or Data-Translation

Several re-gridding methods have been proposed in literature that vary in terms of accuracy and associated complexity (43), (44). In this thesis, we focus on methods that are based on convolving the non-uniformly spaced source points S with a convolution kernel function Φ . The output of the convolution is a uniform grid of target points T .

$$T = \Phi(T, S) * S \quad (2.4)$$

The spacing and range of the uniform grid can be expressed in terms of an oversampling factor α (45). The relationship between the number of source $|S|$ and target points $|T|$ can be expressed in terms of α .

$$|T| = \alpha |S| \quad (2.5)$$

Figure 10 shows the graphical representation of the re-gridding process for non-equispaced source points. Each source point is multiplied by interpolation kernel function and resultant of these products is computed. The resultant curve is sampled on a uniform grid to give the set of target points T .

The basic procedure for computing the target point array is shown as pseudo-code in Figure 11. It is based on updating all the target points within a specified distance of the source point using an interpolation kernel function Φ . In this thesis, we refer to this specified distance as interpolation threshold σ . The process of updating the target points is repeated for all the

```

for each source point  $s_i \in S$  do
  | for each target  $t_k \in T$  such that  $|s_{i\mathbf{x}} - t_{k\mathbf{x}}| \leq \sigma$ 
  | and  $|s_{i\mathbf{y}} - t_{k\mathbf{y}}| \leq \sigma$  do
  | |  $v(t_k) = v(t_k) + \Phi(s_i, t_k) * v(s_i)$ 
  | end
end

```

Figure 11. Pseudo code for Re-gridding processes in 2-dimension. The values of s_i and t_k are denoted by $v(s_i)$ and $v(t_k)$ respectively. (s_{ix}, s_{iy}) and (t_{kx}, t_{ky}) denote the respective co-ordinates.

source points in S . The final target array T is the translated version of S . S and T are assumed to be available in external memory as they cannot fit in on-chip memory for large problem sizes. Depending on the trajectory of source points, accesses to T for updating corresponding target points lack data-locality for most the applications of NuFFT. This incurs significant delays due to repeated memory accesses and hence re-gridding is known to be the most time consuming step (over 90%) of the whole NUFFT computation (4).

Other works like (33), (46), (4) also use the same gridding algorithm for re-gridding but our proposed architecture improves memory bandwidth utilization and hence improves on performance and power consumption.

Challenges of Implementing Re-gridding in Hardware

Limited on-chip Memory

For large number of target points, it is usually not possible to fit all the target points in the on-chip FPGA memory. The target set is available in a memory external to the device. Updating the points, requires reading these points from the external memory and writing back.

Data-locality

Ordering of source point dictates the order in which the target points will be updated. If the source point have poor geometric data-locality, the corresponding memory accesses to the external memory also lack the data-locality.

Utilizing Memory Bandwidth Efficiency

Efficient utilization of memory bandwidth is the key to improve the performance of the re-gridding process. Minimizing re-reads of source and target points from external memory improves the efficiency of memory bandwidth.

In this chapter, we propose a novel on-chip buffering scheme to group the source points together and hence improve the data-locality of accessing the target points in external memory. This results in better performance and power consumption. The novelty of the proposed architecture is in the efficient use of FPGA on-device block memory for storing temporarily the subsets of S and T , and the efficient usage of other customizable hardware components such as fill-status based arbiter, decentralized memory control logic and an array of pipelined single precision floating point multipliers and adders.

2.2 Related Work

Considering the growing gap between the on-chip throughput capacity and memory bandwidth, efficient memory bandwidth utilization is most important to truly benefit from the parallel computation capability of GPUs and FPGAs. Several contributions ((47; 48)) have been reported to improve the memory bandwidth efficiency for Fast Fourier Transform. Considering the wide range of possible trajectories for the case of NuFFT, efficient memory utilization is even more challenging. High performance implementations for computing NuFFT have been pursued on various platforms including multi-core CPU's, (31), (29), (30), Graphics Processing Units (GPU) (31) and FPGAs (4),(5). Since the lack of locality in memory accesses is the major bottleneck in the re-gridding process, all these work target efficient utilization of memory bandwidth between the processing unit and the external memory.

Sorensen et al. (6) targeted the re-gridding process for NuFFT on GPU and multi-core CPU. Like our approach, it extracts "target driven" parallelism but has an additional overhead of preprocessing to decide the processor each source point is going to map to. It divides the Target Cartesian grid into rectangular regions and process each region on a distinct processor. They specifically targeted Spiral and Radial trajectories that mainly have applications in MRI. Zhang et al. (46) also proposed multi-core processor based approaches for better cache utilization.

Kestur et al. (4) proposed an FPGA based implementation employing a linked-list based approach to improve the locality of memory accesses and hence the memory bandwidth efficiency. Later they improved the performance of their framework in (5) and also targeted known

trajectories. To the best of our knowledge these two are the only FPGA based work that are aimed at accelerating the re-gridding process of NuFFT other than our work published in (24).

Many efforts have been put to accelerate various applications of NuFFT. Some of the recent works related to re-gridding process in SAR image reconstruction include (49), (42). Nash et al. (49) proposed an FPGA based pipelined and implementation for Range Migration Algorithm based SAR. Sadi et al. (42) demonstrated a specialized platform for regridding process in SAR processing using 3D-stacked logic in memory. Image reconstruction in MRI is another important application for NuFFT. Huang et al.(50) proposed a CPU and GPU based approach for image reconstruction in 3-D MRI. Kalamkar et al. (31) also proposed a multi-core CPU based technique for accelerating 3D NuFFT. In this work, we focus re-gridding process for 2D NuFFT and its applications.

2.3 Proposed Re-Gridding Architecture

The proposed Re-gridding architecture, called Data-translator, is based on mapping uniform grid of target points to a number of on-chip, block memory based FIFOs. The 2-Dimensional (2D) uniform grid, that consists of an array of target points T , is subdivided into smaller 2D sub-arrays, referred to as tiles in this thesis. The size of tile is chosen based on the resources available on device. Each tile has a corresponding FIFO buffer that is used to group the source points that affect any target point within the tile. The FIFO-to-tile mapping is illustrated in Figure 12. The source points are read from the external memory and pushed into the FIFO(s) that correspond to the tile(s) they affect. The idea is to group all the source points that affect the same tile onto a single FIFO. One FIFO is read at a time and the corresponding tile is

loaded into the device from the memory. Once all the source points in the selected FIFO update the corresponding target points, the fetched tile is written back and another FIFO-tile pair is selected to be processed. The process is repeated till the target points corresponding to all source point in S are updated.

As shown in Figure 15, the proposed Re-gridding architecture consists of memory interfaces, multiple block memory based FIFO buffers, decentralized control logic, read arbiter, address generator and an array of floating-point multipliers/adders (FpMAs). Rest of the section explains the architectural details of individual components and data-flow through the system.

2.3.1 Points Representation

Source Points

Each source point s_i is stored as a string of 128-bits. s_{i_x} and s_{i_y} are the x and y co-ordinate in single precision floating point format as defined in (51). The co-ordinates may have any value within the specified range. Both real and imaginary part of s_i are also stored in single precision floating point format. The set of source points S is available in external memory in an arbitrary or application-specific order. It has to be noted that source points stored at consecutive locations in external memory may not be contiguous in terms x and y coordinates.

Target Points

Set of target points T is also stored in external memory. Since target points are on uniform grid, by storing them in order, the coordinates are no longer required to be stored as part of the string. Target point is a string of 64-bits consisting of real and imaginary part expressed as single precision floating point values.

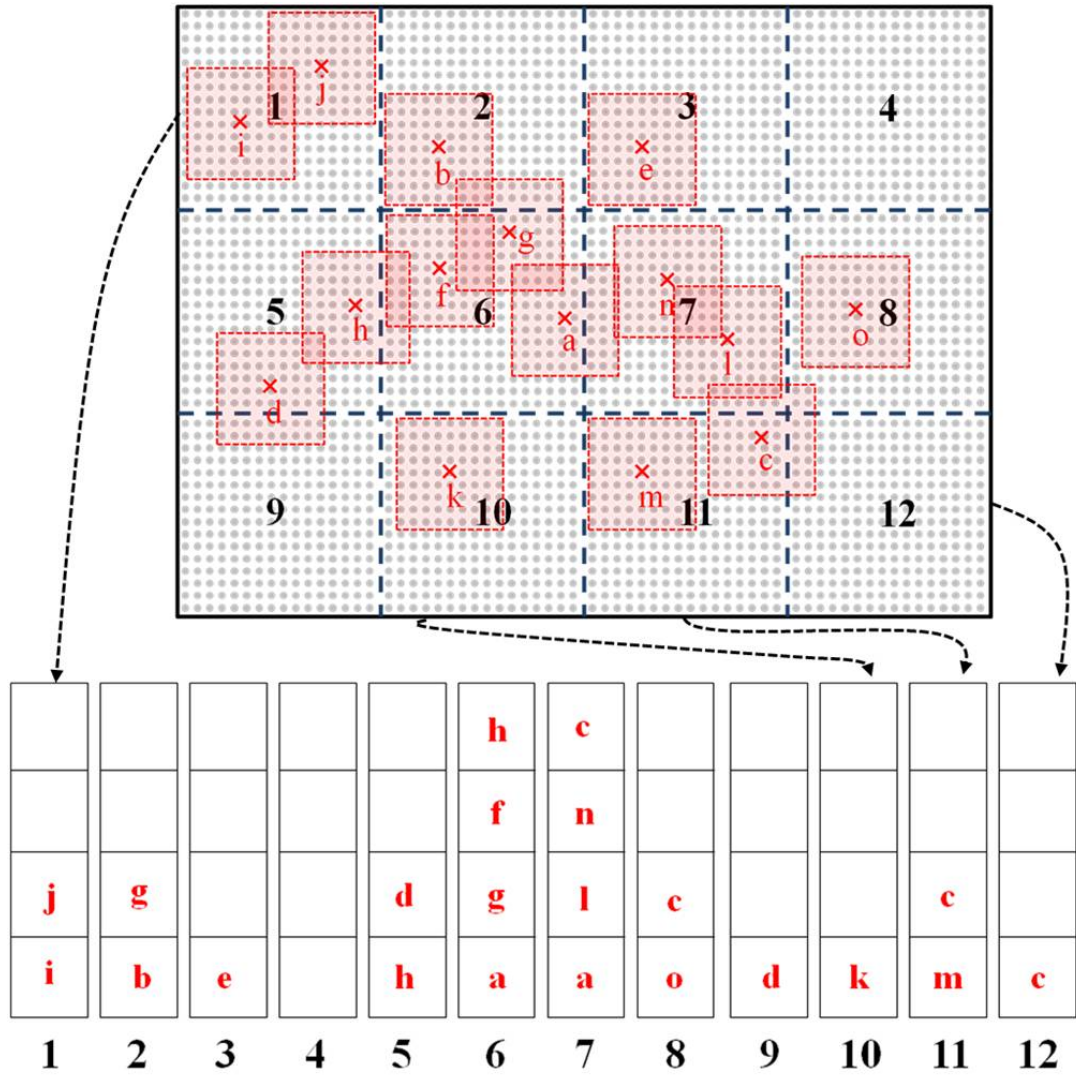


Figure 12. Mapping between Tiles and point FIFO. The boundaries of the tiles are shown by dotted blue line and tiles are numbered from 1 - 12. Grey dots indicates the target points, crosses indicates the source points and dotted red box indicates the convolution widow for each source point. The FIFOs at the bottom are shown filled with the source points and have the same index as the tile they correspond to.

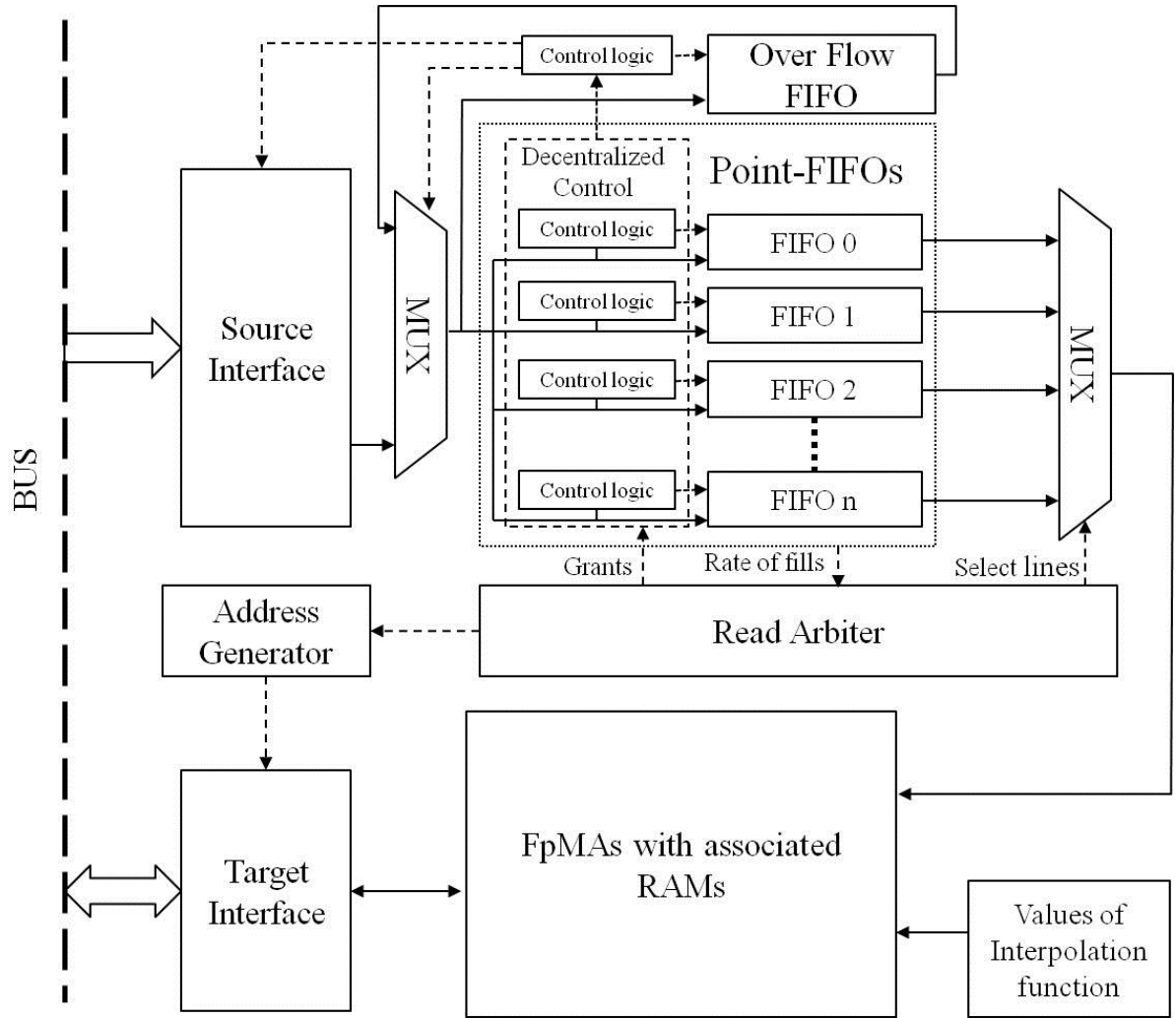


Figure 13. The proposed Re-Gridding (Data Translator) architecture depicting the data-flow through its components. The solid lines indicate the data flow and dotted lines indicate control lines.

2.3.2 Data-translator Components

Memory Interface

The proposed architecture has two memory interfaces: Source Interface (SI), that reads the source points from the external memory and Target Tile Interface (TTI), used to read and write target tiles from the external memory.

Point-FIFOs and Overflow-FIFO

There is a FIFO f_m corresponding to each tile e_m in the target set T as depicted in Figure 12. The set of FIFOs corresponding to tiles are referred as point-FIFOs. The decision to push an incoming source point s_i into f_m is made using a decentralized control logic. s_i may be pushed into multiple FIFOs and if any FIFO corresponding to s_i is full, s_i is pushed into an extra FIFO called the overflow FIFO. If the number of empty slots in the overflow FIFO become less than the read burst size used by SI, read operation of source points from the external memory is stalled. Overflow FIFO is emptied by repeatedly trying to write the source points contained into the point-FIFOs. If any FIFO corresponding to the source point is still full, the point is written back to the overflow-FIFO.

The total number of point-FIFOs depend on the size of the target frame T and the tile size. If the target set has $|T|$ points and a tile of size $z \times z$ is used, the total number of point-FIFOs will be $|T|/z^2$.

Decentralized control logic for the point FIFOs

The source points read from the memory through SI are pushed into the point-FIFOs depending on their co-ordinates and interpolation threshold σ . Consider the m^{th} tile e_m spanning

the Cartesian region (x, y) where $X_{m_l} < x \leq X_{m_h}$ and $Y_{m_l} < y \leq Y_{m_h}$. m is called the tile index of tile e_m . A source point s_i is pushed into f_m , if the following conditions are satisfied:

$$X_{m_l} - \sigma < s_{i_x} \leq X_{m_h} + \sigma \quad (2.6)$$

$$Y_{m_l} - \sigma < s_{i_y} \leq Y_{m_h} + \sigma \quad (2.7)$$

The control logic corresponding to each FIFO also keeps track of the Fill Status (FS), i.e., the number of points in the FIFO.

Read Arbiter

A single point-FIFO is read at a time. The selection of a particular point-FIFO to be read is made by the read arbiter. The arbiter also controls the select line for the multiplexer at the output of the point FIFOs. Since the arbiter is a centralized control, the hardware complexity greatly depends on the number of tiles.

We develop General Arbiter (GA) based on arbitration scheme to target arbitrary or unknown trajectories. This arbitration is based on Fill Status of FIFOs. FIFO that are full is given the highest priority. In case multiple FIFOs are full, the FIFO corresponding to greater tile index is given priority. When none of the FIFO is full, the decision is based on number of points in the FIFO. The FIFO with highest point count is read first. Only the FIFO having the highest count requests for read. When multiple FIFOs have the same highest count, the tile with the greatest tile index is given highest priority.

To have better performance for known trajectories, we propose Trajectory Specific Arbiter (TSA). Since the ordering of source points is known, the ordering in which the FIFOs are selected can be decided a priori. This results in lower hardware complexity but the hardware is specific to a particular trajectory. As part of this work, we target well-known trajectories that are discussed in Section 2.1.1.

Tile Address Generator

If the read arbiter decides to read data from a FIFO f_m , address generator generates the address of the corresponding tile e_m in the memory and requests the TTI to read e_m from the memory.

Floating point Multipliers and Adders (FpMAs) with associated Random Access Memory (RAM)

Since for each source point, an array of $(2\sigma) \times (2\sigma)$ target points in the convolution window need to be updated, we use an array of $(2\sigma) \times (2\sigma)$ FpMA units to compute all these values in parallel as shown in Figure 14. An array of pre-computed values of interpolation kernel function is stored locally in each FpMA unit in the form of a Look-Up-Table (LUT).

Mapping of Target tile points to FpMA units

Storing the contents of tile in a single RAM on the device restricts the number of simultaneous accesses to the target points. This severely hinders the extraction of parallelism in the computation of target point values. Another way to store the tile on the device is to use registers. This solves the problem of parallel access by using numerous multiplexers but the approach is not efficient with respect to interconnect resources required for large tile sizes.

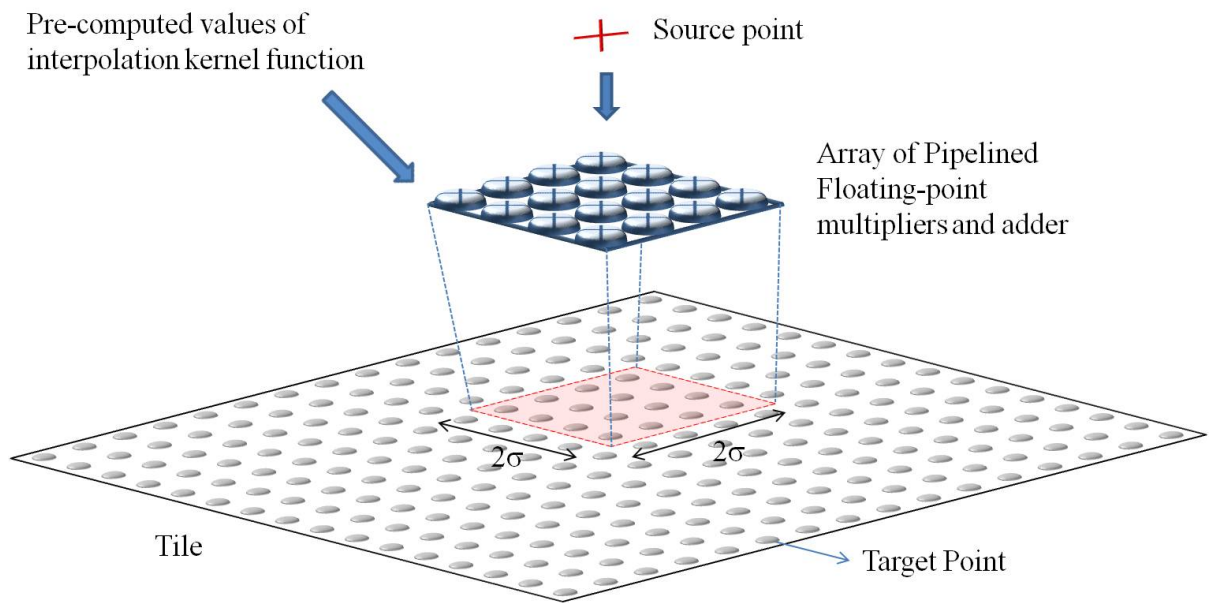


Figure 14. The convolution operation of the source point with the interpolation kernel function to update the target point values. The red box on the tile indicates the convolution window.

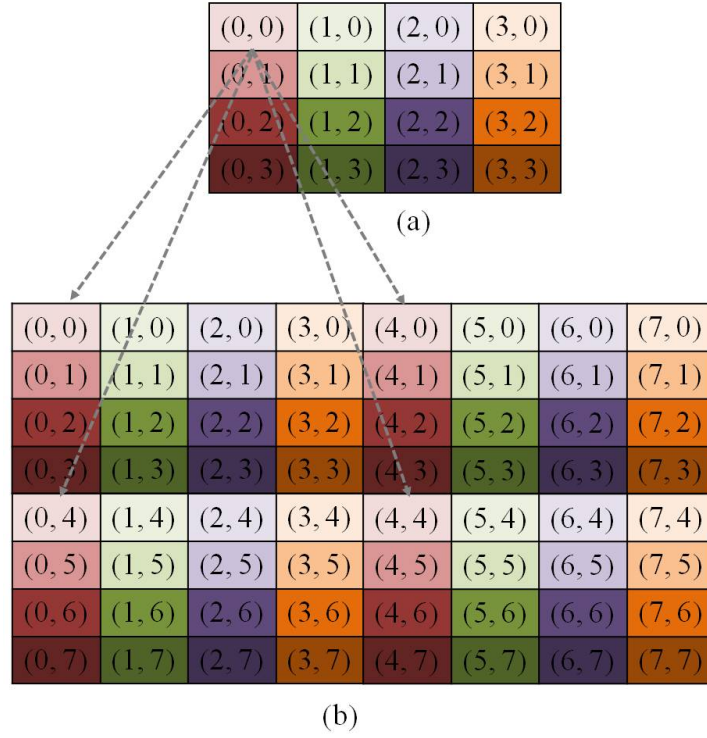


Figure 15. Logical connection of Floating point Multiplier and Adder (FpMA) units to target points in fetched tile indicated by color coding: (a) Array of FpMAs units, each box represents an FpMA unit (cell). The co-ordinates in each box indicates the index of the FpMA unit in a 2-d array format (b) A tile of size 8 x 8. Each box represents a target point of the tile. The co-ordinates in each box indicate the co-ordinate of target point in the 2-d tile. For clarity, logical connection of (0,0) FpMA unit is indicated with dotted line as well

A distributed approach is proposed to store the values of fetched target tile on the device. Each target point is mapped to a single FpMA unit and all target points mapped to the same FpMA unit are stored in a single RAM. The mapping of an 8×8 tile onto an array of 4×4 FpMAs is shown in Figure 15. FpMA units are also indexed as a two dimensional array to make mapping easier and more intuitive. If a target point t_k is at index (x_{t_k}, y_{t_k}) in the tile, the index (q_x, q_y) of the FpMA associated with it is given by the following expressions:

$$q_x = x_{t_k} \bmod (2 \times \sigma) \quad (2.8)$$

$$q_y = y_{t_k} \bmod (2 \times \sigma) \quad (2.9)$$

Architecture of FpMA

Each FpMA unit has a pipelined floating point multiplier along with a floating point adder to perform the multiply and accumulate operation. The values of interpolation kernel functions are addressed from the LUT based on the distance between the target point being updated and the source point. The proposed architecture is based on pipelined floating point multiplier and adder by Altera (20). In Figure 16, the depth of pipeline for floating-point multiplier is 5 whereas the floating point adder is single stage. A register pipeline is used to synchronize the timing of source co-ordinates with multiplier output. A new source point is available every clock cycle and the whole process of multiplication and accumulation is pipelined. An interface with TTI is provided to read a new tile from memory and write back the fetched tile. For

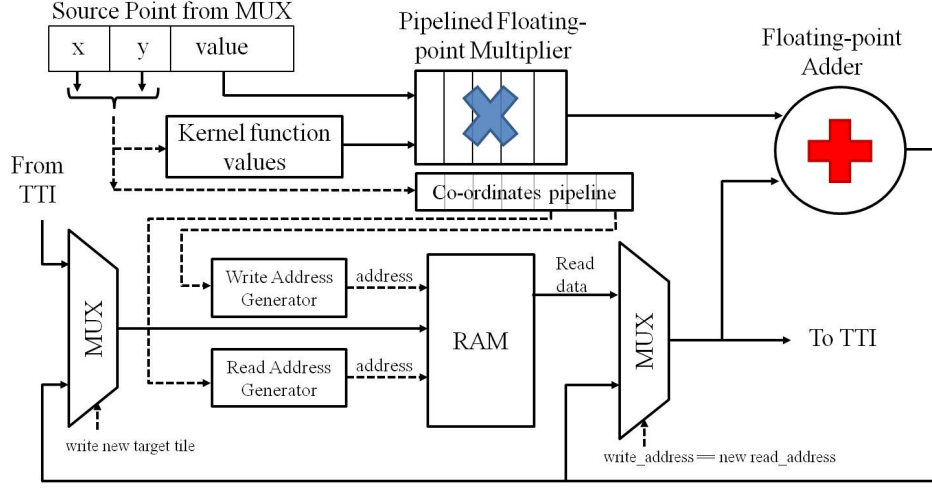


Figure 16. Architecture of the proposed Pipelined Floating Multiplier and Adder (FpMA) unit

simplicity, Figure 16 does not show separate multipliers and adders for the real and complex part of the values.

The hardware complexity of an FpMA cell depends on the size of the tile and σ . For a tile of size $z \times z$, the number of memory locations in the RAM of each FpMA would be $(z^2)/(4 \times \sigma^2)$.

2.3.3 Data-flow through the System

The source points are written to external memory by an initializing bus master in the format described earlier. All the target points are initialized to zero. SI reads source points from external memory in a burst memory access mode. During each cycle of burst, a new source point s_i is read and pushed into appropriate FIFO(s) by the decentralized control logic. If any point-FIFO corresponding to s_i is full, s_i is pushed into the Overflow-FIFO. For the case of GA,

based on Fill-Status of FIFO buffers, GA selects the FIFO to be read. For TSA, the tile index of next FIFO to be read is read from the memory. Address generator generates the address of the selected tile. TTI requests the bus to read the specified tile from the memory. The contents of the fetched tiles are distributed over the RAMs associated with the FpMA units. Each target point has a single FpMA unit associated with it and all the target points associated with an FpMA unit are stored in the RAM local to FpMA. Source point are read from the selected FIFO at each clock cycle and all the target points within the fetched tile that are σ distance from the source point are updated in parallel. SI and TTI access the memory through a shared bus with the bursts from both interfaces interleaved. This implies that the source points are read out from the FIFOs in between the FIFO-fill operation. This minimizes the chance of any FIFO getting full. The arbitration share for each memory interface on the system bus can be modified to give preference to a particular one. If SI is given a higher preference, the overall FIFO fill-rate is greater than the rate of popping points out of FIFOs. This means grouping more points together at the expense of higher probability of any FIFO getting filled.

2.4 Hardware Implementation

2.4.1 Optimized Verilog Based Implementation

An optimized implementation of the proposed architecture was done in Verilog for relatively small target set size. To target larger target set sizes, an openCL implementation was also done that is discussed in Section 2.4.2.

Experimental Setup

For verilog based implementation, synthesis and fitting is done using Altera Quartus II 13.1 for Stratix IV EP4SGX230KF40C2 device. Simulations are performed on Modelsim Altera 10.0d. The proposed architecture is integrated to a DDR2 memory controller using two standard Avalon Bus Master Interfaces (52): SI and TTI. SI is 128-bit wide and reads source points from the memory in the form of bursts. The upper limit on burst size is imposed by Avalon Specifications (52). A maximum of 1024 points are read as a single burst and for larger number of points, multiple burst transactions are done. The Target interface is 2048-bit wide and is used to read/write the target points from/to the DDR2 memory in the form of bursts. Floating point multiplier and adder in the FpMA units are generated using Altera Mega-wizard (20). The initializing bus master, SI and TTI communicate with memory through the Avalon Bus Interconnect that has built-in bus arbitration support. The arbitration shares amongst the masters are programmable. For our experiments, equal arbitration share is given to the three bus masters in the system. The optimized Verilog implementation is targeted only random trajectories. The random trajectory for source points is generated using a MATLAB script. The co-ordinates for the source points are generated randomly within the specified range. This mimics the arbitrary sampling order. The real and imaginary parts of source point values are also generated randomly. Each co-ordinate as well as real and imaginary parts are represented in single precision floating point representation. The power analysis of the proposed architecture is done using Altera's Powerplay power analysis tool (53).

Results

The system was simulated using Modelsim Altera 10.0d. The simulation models for the Avalon bus, memory controller and DDR2 memory were generated using Altera Qsys software. The simulation gives highly precise results since the latency in the simulation models may differ at most by two clock cycles compared to the design on board (54). Performance of the proposed architecture is evaluated by simulating and computing the number of clock cycles required. Using a clock frequency of 50 MHz, which is chosen to be less than the achieved maximum frequency of 62.9 MHz, computation time for the translation process is calculated. Based on this computed time throughput is calculated in terms of frames per second (fps). For all the experiments, over-sampling factor α is taken to be equal to 4. This implies that for target point set T of size 256×256 , the sizes of corresponding S will be 128×128 . Table II shows the throughput for various sizes of tile, convolution window and set T .

Taking larger tile size meant larger part of T was available on the device at a time and more number of source points are grouped together in a single FIFO. This improves the locality of memory access and hence results in a higher throughput. Taking larger tile size for a fixed $|T|$ results in less number of point-FIFOs and hence reduction in the complexity of Multiplexer and the centralized arbiter. Because of fewer number of FIFOs, size of an individual point-FIFO can be increased to accommodate more points.

Larger convolution window (greater σ) results in more points pushed into multiple FIFOs and hence higher computation time. For a four times bigger convolution window, throughput

was only reduced by 5 %. But it has to be noted that bigger convolution window results in more number of FpMA units and hence higher hardware complexity.

TABLE II

THROUGHPUTS FOR VARIOUS SIZES OF TILE, CONVOLUTION WINDOW AND

TARGET SET T

Size of Target set	σ	Tile Size	Throughput
256 x 256	2	16 x 16	586.38 fps
256 x 256	2	32 x 32	691.33 fps
256 x 256	4	32 x 32	654.57 fps
256 x 256	2	64 x 64	715.66 fps
128 x 128	2	64 x 64	3081.98 fps

Other parameters like memory bandwidth utilization, power consumption, Mega Floating point Operations per Second (MFLOPS) and MFLOPS per watt were also computed. Memory bandwidth utilization was computed by taking the ratio of time the memory interface was active, to the overall computation time for the re-gridding process. For a target set T of size

256×256 and a tile size of 64×64 , we have achieved 65.37% memory bandwidth utilization, while spending 27.02 watts to reap 750.42 MFLOPS.

2.4.2 OpenCL Implementation

OpenCL is an "open standard for parallel programming of heterogeneous systems" (55). An OpenCL program has a host code that runs on the host machine and a device code.

Experimental Setup

To target bigger problem sizes and better experimentation of various design parameters, the data-translator was also implemented using OpenCL solution on Altera Stratix V GX 5SGXEA7N2F45C2 FPGA. Data-translator is implemented in the form of multiple kernels in the device code. These kernel functions are translated into architecture by Altera OpenCL solution (56) which is implemented on FPGA. The FPGA is mounted on DE5 net (57) board from Terasic that has two independent banks of on-board DDR3 RAM. The arrays of source and target points are generated in the host code and transferred to the on-board DDR3 memory. After the values of target points have been computed, the target array is copied back from DDR3 to the host machine. The overall OpenCL setup is shown in Figure 17.

The host program generates source and target points sets. For the case of random trajectory, the coordinates are generated randomly at run time of host program. For specified trajectories, the coordinates, real and imaginary values are read from an external file. Spiral, Radial and Cartesian trajectories were generated using the MATLAB framework provided by (2). Curvilinear trajectories were generated using MATLAB script based on (3).

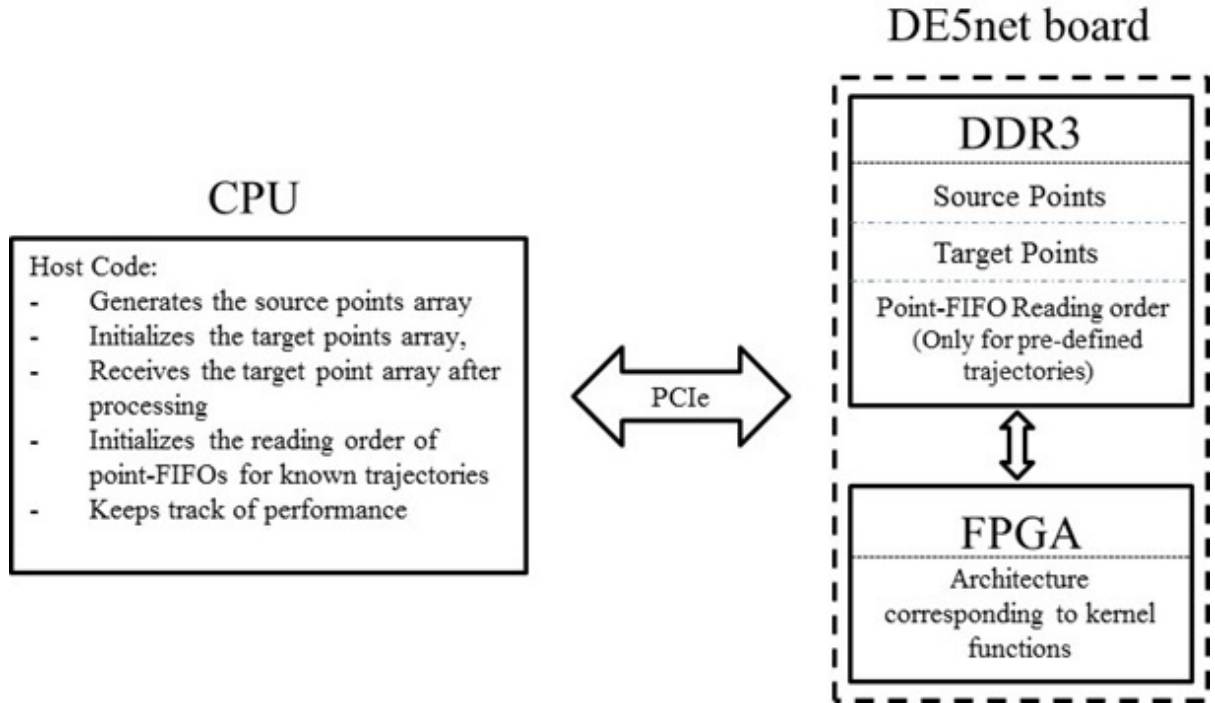


Figure 17. OpenCL based Experimental Setup

Performance of the proposed architecture is computed in host program by calculating the time lapse of kernel execution. The time is measured in milliseconds using the standard openCL libraries. Power consumption is estimated for the whole FPGA design using Powerplay Power Analyzer tool (58).

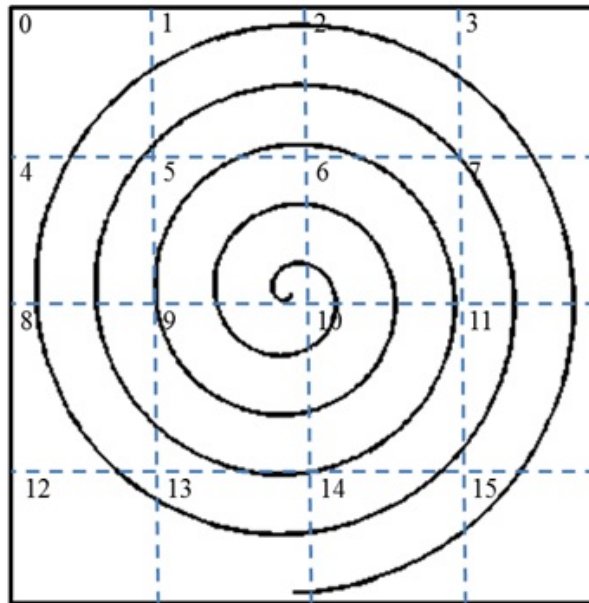


Figure 18. Reading Order of Point-FIFOs could be decided a priori based on the known Trajectory. The figure shows the target frame divided into 16 tiles.

The point-FIFOs are implemented using Altera OpenCL Channel extension (56). The architectural components are implemented in the form of kernel functions. The data and control exchange amongst the kernels is also done via channels.

Trajectory Specific Architecture

Since for known trajectories, the reading order for point-FIFOs could be decided a-priori, an array having the index of point-FIFOs, in the order in which they have to be read, is loaded by the host program to the on-board DDR3 memory. Since the ordering is predetermined, trajectory specific arbiter is much simpler compared to generic arbiter.

Results for the Proposed Architectures

Performance and power characteristics for various values of design parameters are given in table Table III. Throughput is reported in terms of frames per second (fps) and power efficiency in terms of Mega Floating Point Operations per Second (MFLOPS) per watt. For all the results reported in Table III, $\sigma = 2$ which implies a convolution window of size 4×4 .

Hardware Resource Utilization

Figure 19 shows a comparison of Hardware resources for GA and TSA based architectures for target set size of 1024×1024 , tile size of 256×256 , $\alpha = 4$ and $\sigma = 2$. It can be seen that the number of DSP blocks is the same for both architectures since the window size and hence the number of FpMA units are same. Block memory utilization is slightly higher for trajectory specific architecture since buffer reading order, specified by host, is also stored in the on-chip block memory. The hardware complexity associated with selecting the buffer to read in architecture for arbitrary sampled data is reflected in higher number of Logic Elements (16.7 % higher) and Flip-flops (27 % higher) for GA compared to TSA.

Performance

Figure 20 shows the throughput comparison of various trajectories on GA and TSA based architectures for a target sets of size 1024×1024 , $\alpha = 4$ and $\sigma = 2$. For TSA based architecture, prior knowledge about the trajectory helps achieve better memory bandwidth utilization and hence higher throughput compared to GA based architecture that relies on real-time decision of the tile to be updated. For random trajectory, the chance of any FIFO buffer getting filled is much lower compared to other specified trajectories as the points are uniformly distributed

TABLE III

POWER AND PERFORMANCE FOR DIFFERENT TRAJECTORIES

Trajectory	Arbiter	Size of Target set	α	Tile Size	Throughput	MFLOPS	MFLOPS/Watt
Cartesian	TSA	512 x 512	4	256 x 256	153.76 fps	1127.98	37.37
Cartesian	TSA	1024 x 1024	4	256 x 256	67.23 fps	644.9	64.85
Spiral	TSA	512 x 512	4	256 x 256	150.86 fps	632.8	45.61
Spiral	TSA	512 x 512	4	128 x 128	113.46 fps	475.91	24.06
Spiral	TSA	1024 x 1024	4	256 x 256	55.83 fps	936.66	51.38
Curvilinear	TSA	512 x 512	4	256 x 256	153.61 fps	1128.94	37.33
Curvilinear	TSA	1024 x 1024	4	256 x 256	67.29 fps	644.3	64.91
Radial	TSA	512 x 512	4	256 x 256	142.31 fps	596.9	43.02
Radial	TSA	512 x 512	4	128 x 128	116.55 fps	488.83	24.71
Radial	TSA	1024 x 1024	4	256 x 256	51.27 fps	860.14	47.18
Random	GA	512 x 512	4	256 x 256	113.22 fps	474.05	27.47
Random	GA	1024 x 1024	4	256 x 256	21.94 fps	368.05	18.21
Random	GA	512 x 512	1	128 x 128	41.82 fps	701.57	50.567

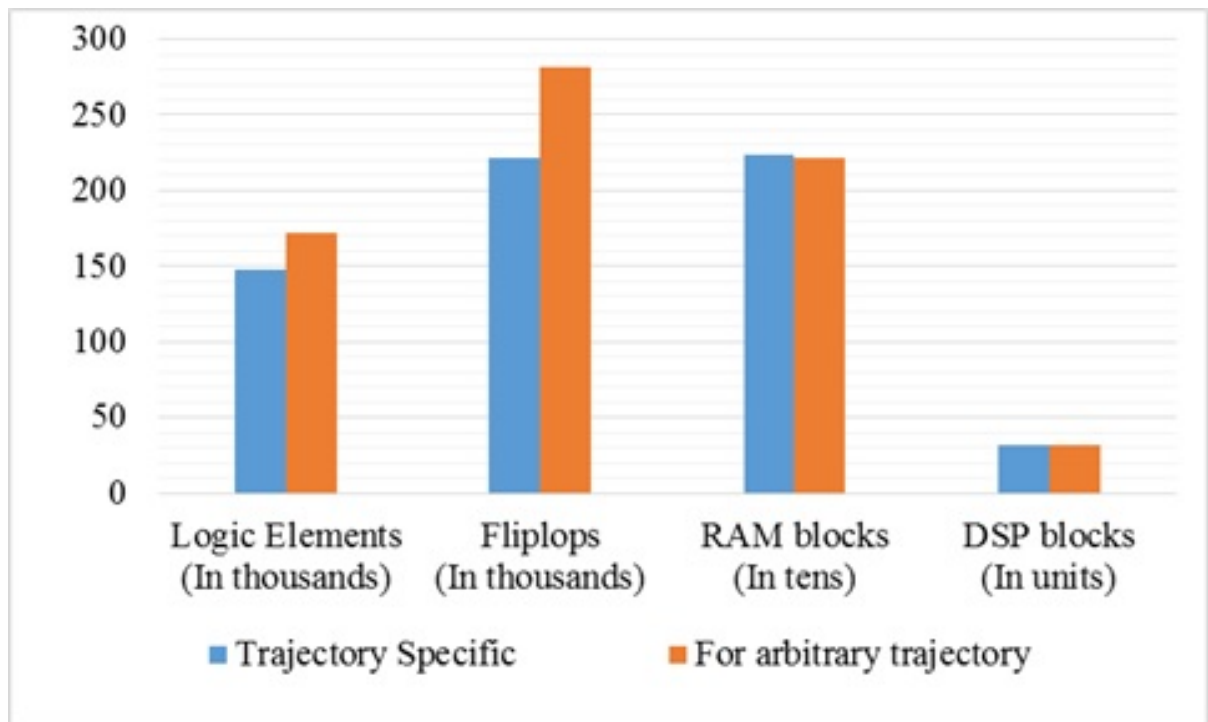


Figure 19. Comparison of Hardware Resources for TSA and GA based Architectures for target set size of 1024×1024 , tile size of 256×256 , $\alpha = 4$ and $\sigma = 2$

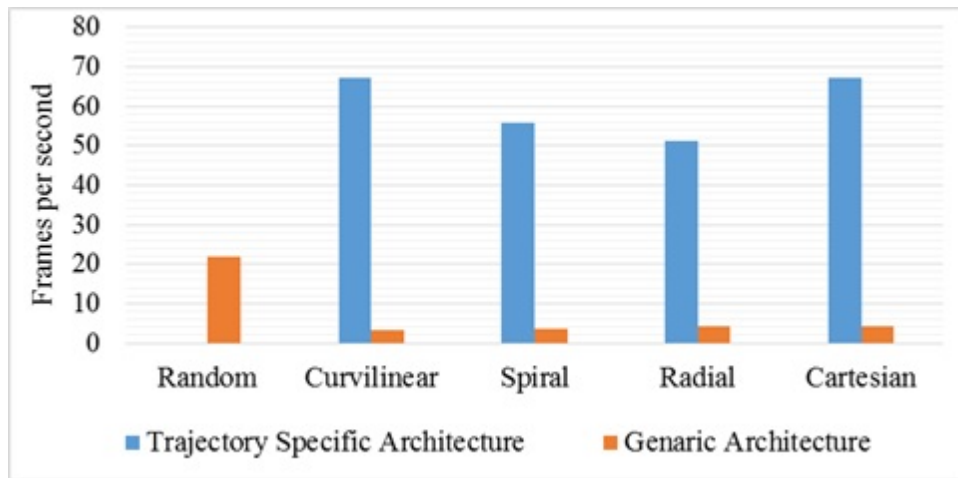


Figure 20. Throughput comparison for trajectory specific architectures

and available in random order. Hence the performance of GA based architecture is the best for random trajectory.

For TSA based architecture, since curvilinear and Cartesian are the most "regular" trajectories in terms of co-ordinate ordering, they achieve the highest throughput. Compared to performance for GA on random trajectory, the performance of TSA for curvilinear and Cartesian trajectory is 3 times, for spiral trajectory is 2.55 times and 2.34 times for Radial.

Power Consumption

Since the arbiter is more complex for the architecture targeting random trajectories, it not only results in higher hardware complexity (as shown in Figure 19) but also higher in higher power consumption. The power consumption of the architecture targeted at arbitrary trajectories and trajectory specific architecture as estimated by Altera Power Play Power analyzer

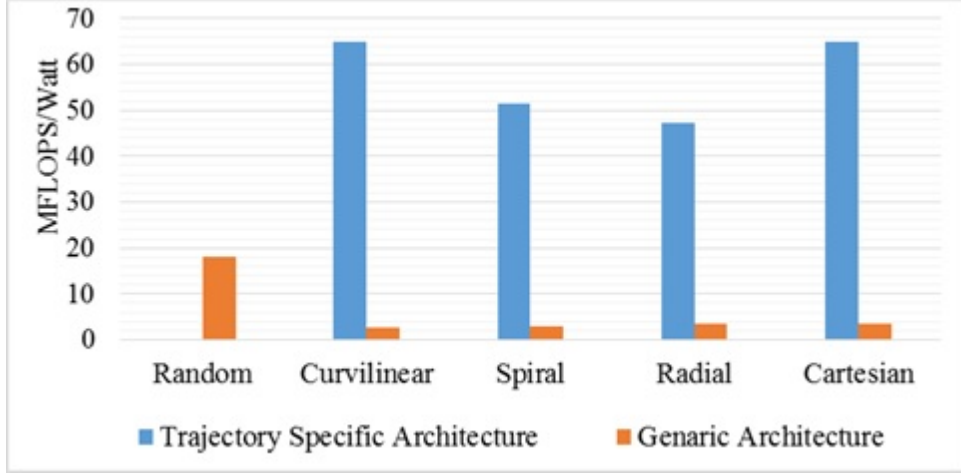


Figure 21. Throughput comparison for trajectory specific architectures

tool are 20.22 Watts and 18.23 Watts (for a target set of size 1024×1024 , tile size of 256×256 , $\sigma = 2$ and $\alpha = 4$).

Effect of Tile Size

As mentioned earlier, bigger tile size allow bigger portion of target set to be available on-chip and hence better grouping of points. This results in better memory bandwidth efficiency and hence higher throughput as indicated in table Table III. It also results in lower hardware complexity for multiplexer and arbiter but higher memory usage specially for FpMA units. Fig Figure 22 shows the comparison of power consumption and hardware resources for tile size of 256×256 and 128×128 when the size of target point set is 512×512 . Block RAMs are higher for the greater tile size, as more points are stored in onchip block RAM per FpMA unit. The number of block RAMs also include the resources allocated to Altera channel based

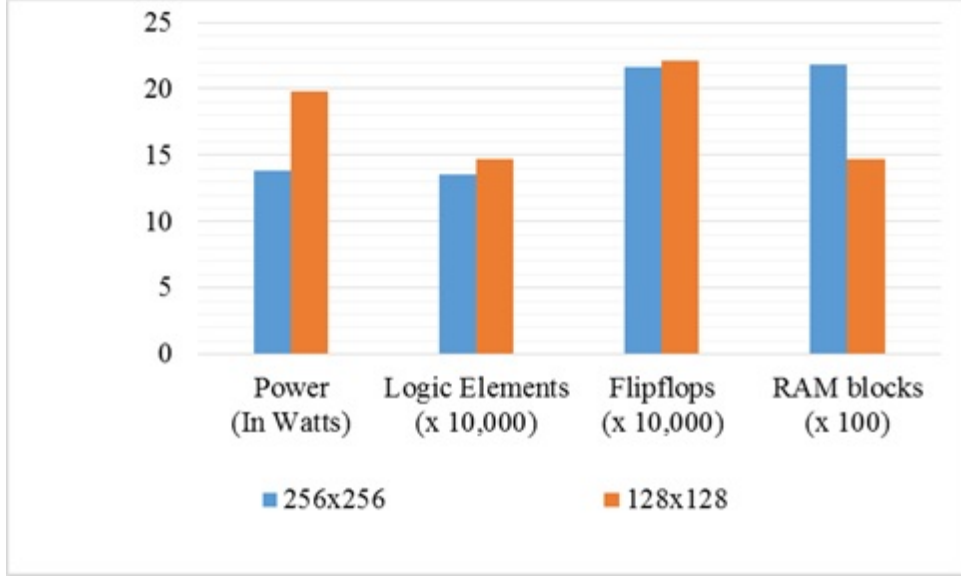


Figure 22. Comparison of Power Consumption and Hardware Resources for tile size of

256×256 and 128×128 when the size of target point set is 512×512

FIFO buffers. The number of Logic Elements and Flipflops are lower for greater tile size as the complexity of multiplexer and arbiter are reduced due to fewer number of FIFO buffers. Larger tile size results in lower power consumption mainly due to lower hardware complexity and less number of external memory accesses.

Scalability

We study the scalability of hardware resources and throughput against target set size for GA and TSA based architecture. Figure 23 is for GA based architecture. It shows that the throughput (in terms of frames per second) decreases as the target set size increases. MFLOPS

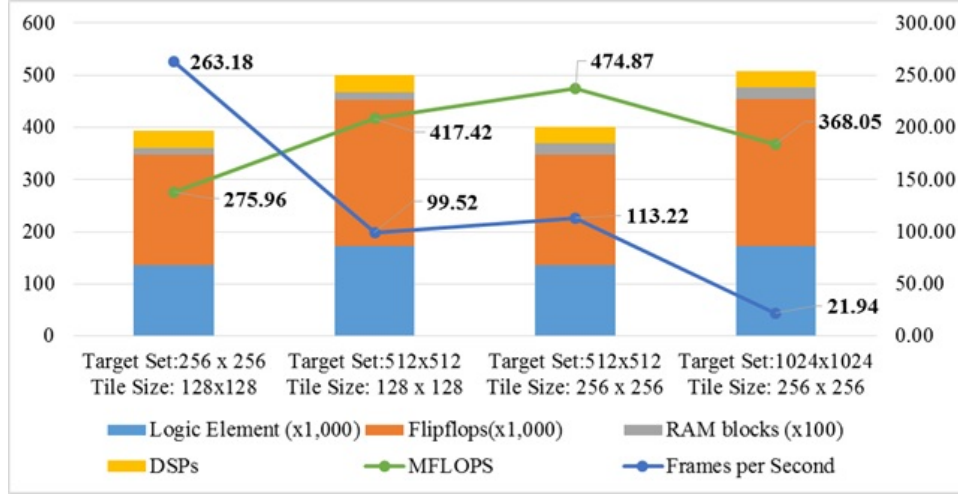


Figure 23. Scalability of Hardware Resources, MFLOPS and throughput with the target set size and tile size for GA based architecture

increase as the frame size increases from 256×256 to 512×512 but for frame size of 1024×1024 , MFLOPS are lower compared to 512×512 which is understandable considering the overhead incurred when the FIFO buffers are full and the reading operation of source points from external memory has to be stalled. As expected, hardware resources also increase with the increase in frame size when the tile size is constant. Effect of tile size on hardware resources has already been discussed. Figure 24 shows the hardware resources and MFLOPS for known trajectories on TSA based architecture. MFLOPS for 1024×1024 frame are greater than 512×512 for TSA based architecture as opposed GA based architecture since trajectories are known a-priori.

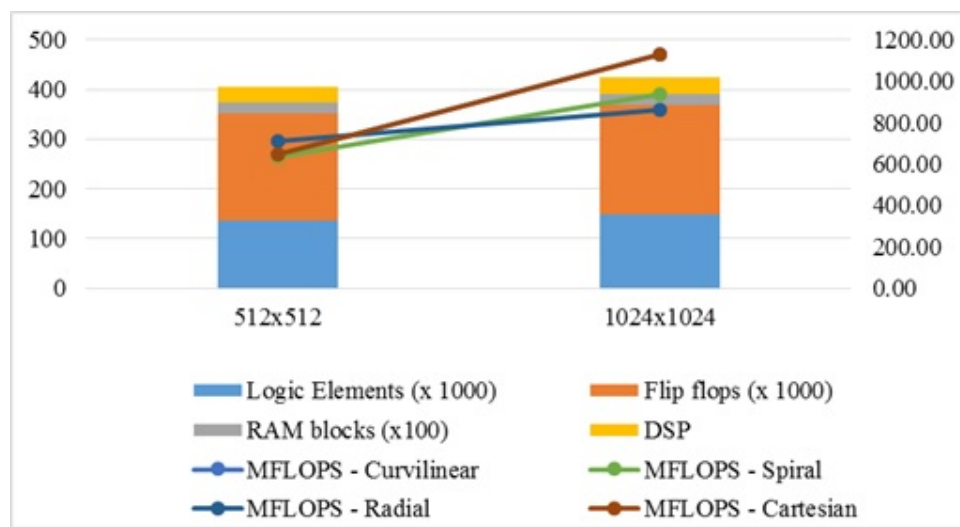


Figure 24. Scalability of Hardware Resources, MFLOPS with the target set size for TSA based architecture. The vertical axis on left indicates the count for resources and the vertical axis on right is for MFLOPS.

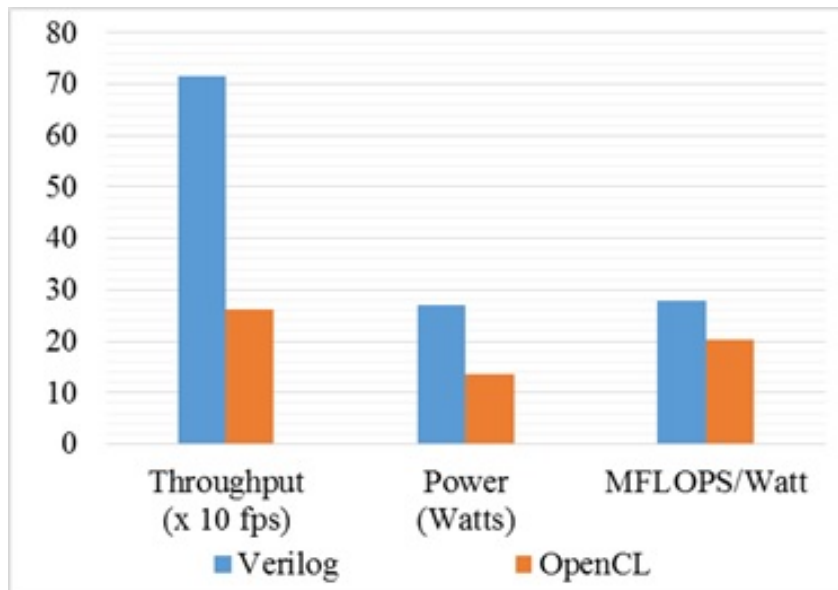


Figure 25. Comparison of OpenCL implementation with Optimized Verilog based implementation for a target set of size 256×256 , $\alpha = 4$ and $\sigma = 2$

2.4.3 Comparison of Implementations

Comparison of the optimized Verilog based implementation with the openCL implementation shows that the throughput of Verilog implementation is 2.7 times higher for a target set of size 256×256 , $\alpha = 4$ and $\sigma = 2$ (Fig Figure 25). Power consumption is twice but it still achieves higher (1.36 times) MFLOPS/Watts. It has to be noted that Verilog based implementation is based on the simulation and synthesis results on Stratix IV EP4SGX230KF40C2 device and OpenCL implementation is based on Stratix V GX 5SGXEA7N2F45C2.

One of the reasons Verilog based implementation out performs openCL implementation is the optimized implementation of Multiply and ACcumulate (MAC) operation. The floating-point adder block in each FpMA unit in Verilog implementation takes a single cycle for computation. This allows pipelined implementation of MAC operations in verilog implementation at the expense of lower clock frequency. Since openCL is a higher level language, it does not allow this level of flexibility. MAC operations are implemented using loop constructs. OpenCL allows pipelined and parallel implementation of loop constructs. If the loop is not parallelizable, the compiler attempts to pipeline. Pipelining across multiple iterations of loops is only possible if inputs to the following iterations do not depend on the output of previous iterations. If it is dependent, like in case of MAC operations, successive iteration are issued when computation of previous iteration is done. This incurs stall cycles and hence degrades the performance. OpenCL implementation does not have the flexibility of controlling the relative reading rate of source and target tiles from external memory. In Verilog, this could be controlled by changing the arbitration share of SI and TTI interface. Overall, openCL allowed easy experimentation but lower flexibility compared to Verilog based implementation.

2.5 Comparison with Related Work

Sorensen et al. (6) reported performance for re-gridding process for MRI trajectories on CPU and GPU. Their proposed algorithm was run on ATI FireStream 2U GPU with 1 GB of memory. A 64-bit Linux machine running Fedora Core 6 on Intel Xeon 2.33 dual core processor with 4GB Random Access Memory was used to run the reference design.

Kestur et al. (4) proposed an FPGA based implementation of the re-gridding process in 2010. They implemented the gridding method and used a linked-list based technique to map the arbitrary samples dynamically. The results show that the FPGA based implementation achieved superior per-Watt performance compared to CPU and GPU but the throughput was quite low compared to the GPU version (6). Later, they reported improved framework and throughput for various trajectories in 2011 (5). Both the implementations were targeted on BEE3 board using two Virtex5 FPGAs.

2.5.1 Verilog Implementation

First, we compare the throughput and power consumption of the optimized verilog based implementation of our proposed technique against FPGA implementations in (4; 5) and GPU(6) based approach for a target set of size 256×256 and random trajectory. All these results are for the same size of convolution window ($\sigma = 2$).

It can be seen from Figure 26 that compared to the throughput of FPGA based technique proposed in (4), Verilog based optimized architecture achieves a 9.6 times speed up but has 1.35 times higher power consumption. Compared to (5), a speed up of 4.3 times is achieved at 1.1 times lower power consumption. When compared in terms of MFLOPS per watt, it achieves 7 times more MFLOPS per watt of power. Compared to the GPU based technique (6), the throughput of our proposed architecture is marginally higher but the power consumption is 5.9 times lower.

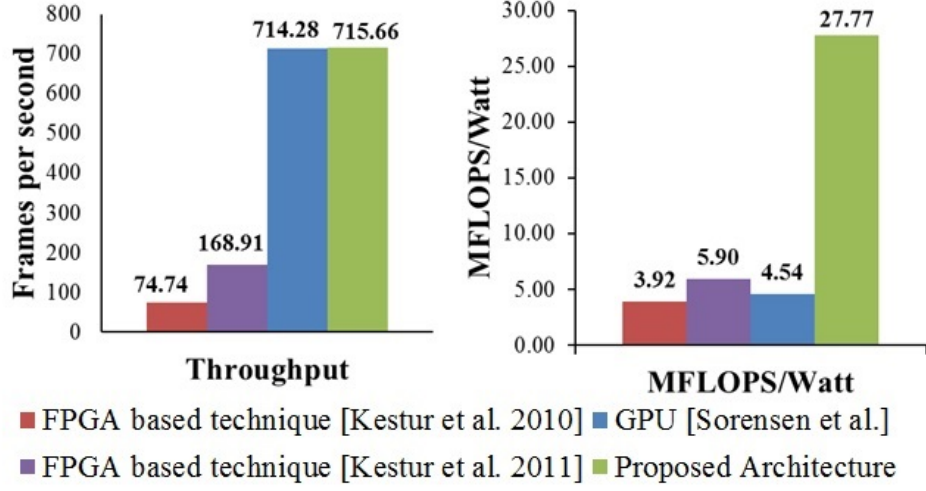


Figure 26. Performance comparison of the verilog implementation of the proposed architecture with existing FPGA (4; 5) and GPU (6) based techniques for a target point set of 256×256

2.5.2 OpenCL Implementation

OpenCL implementation was mainly targeted at bigger frame sizes and greater number of source points. It was also used to implement the trajectory specific architectures. We compare the performance and power efficiency for the following trajectories with related work.

Spiral

Figure 27 and Figure 28 compares TSA based architecture for spiral trajectory in terms of throughput and power-efficiency (MFLOPS/Watt), respectively, with FPGA (4; 5), CPU (6) and GPU (6) based solutions. For a target set of 1024×1024 , our proposed architecture has a marginally higher (6% higher) throughput compared to GPU implementation but at a much

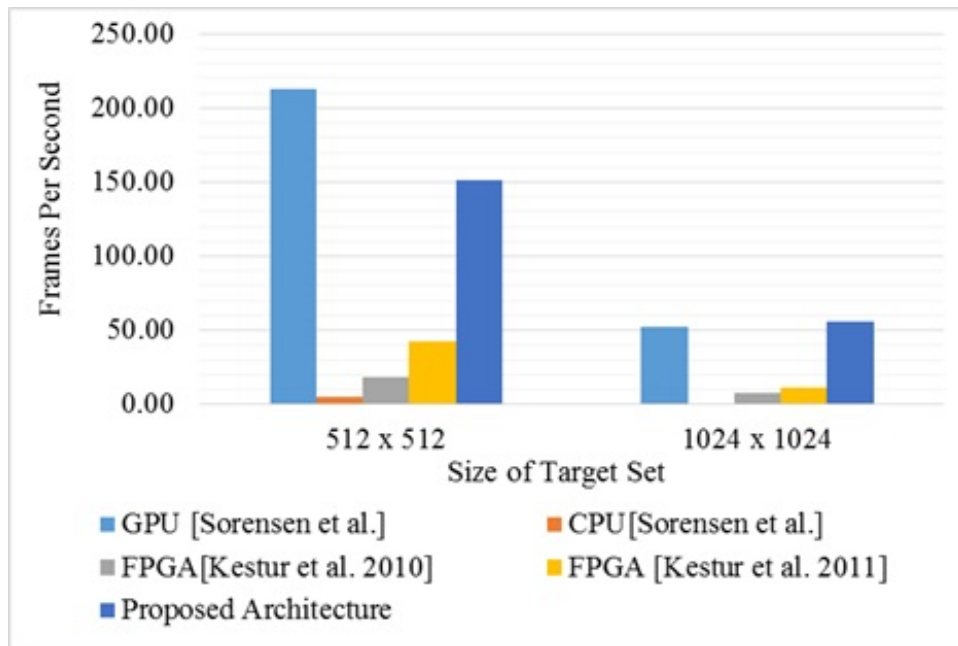


Figure 27. Throughput comparison of Spiral Trajectory with related work

higher power efficiency (9.6 times). For smaller target set of size 512×512 , the GPU has a higher (1.4 times) throughput but power efficiency is better (8.45 times) for our FPGA based architecture. Compared to FPGA based solutions (4; 5), proposed architecture achieves higher throughput (7.35 and 4.77 times respectively) and better power efficiency (15.22 and 7.9 times) for a target set of 1024×1024 . For a target set of 512×512 , the improvement in throughput is 8.12 and 3.56 times whereas power efficiency is better by 11.7 and 7.72 times respectively.

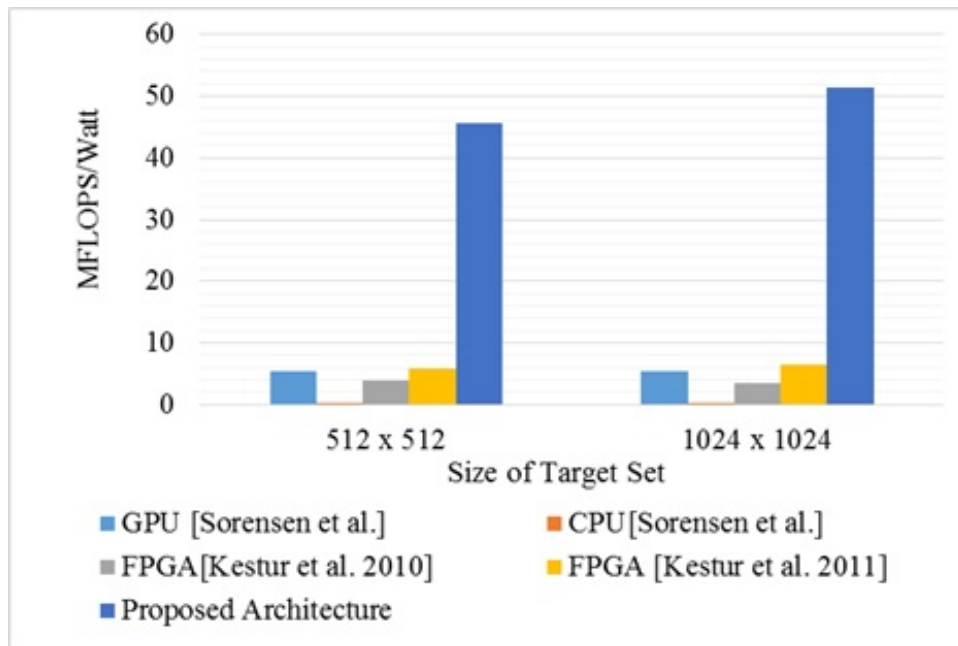


Figure 28. Comparison of Power efficiency for Spiral Trajectory with related work

Radial

Figure 29 and Figure 30 compares the throughput and power-efficiency (MFLOPS/Watt), respectively, of the proposed architecture for radial trajectory. For a target set of 1024×1024 , GPU implementation has 1.5 times higher throughput than our proposed architecture but power efficiency of our proposed architecture is still 6 times higher. For smaller target set of size 512×512 , the GPU again has a higher (1.76 times) throughput but power efficiency is better (6.77 times) for our FPGA based architecture. Compared to FPGA based solutions (4; 5), proposed architecture achieves higher throughput (6.74 and 4.38 times respectively) and

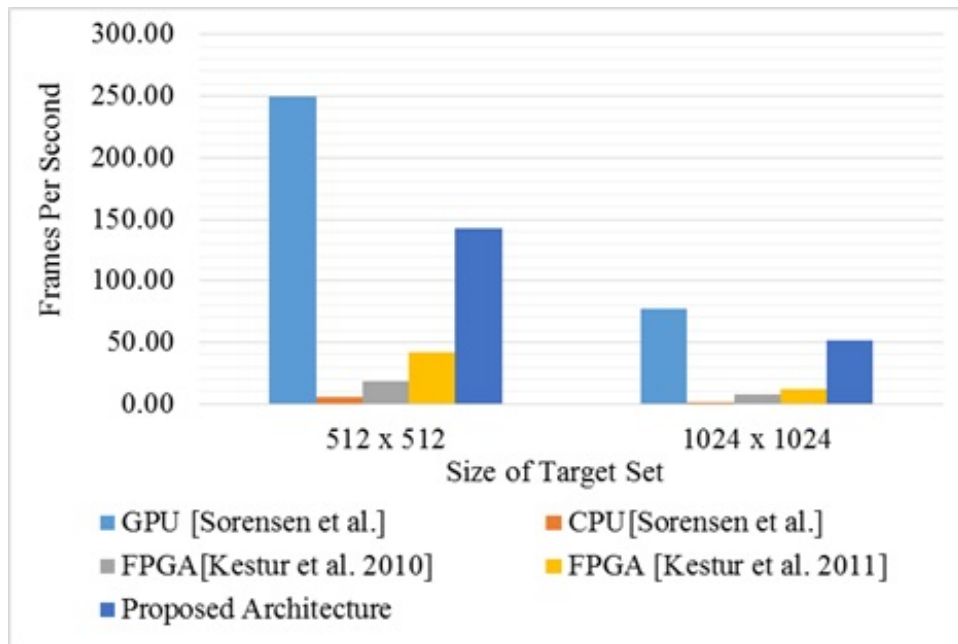


Figure 29. Throughput comparison of Radial Trajectory with related work

better power efficiency (13.97 and 7.21 times) for a target set of 1024×1024 . For a target set of 512×512 , the improvement in throughput is 7.66 and 3.37 times whereas power efficiency is better by 11 and 7.28 times respectively.

Random

Kestur et al. targeted random trajectories in (4). Figure 31 shows the throughput and MFLOPS/Watt comparison for a target set of size 1024×1024 . The major reason for improvement is better memory efficiency as source points are read only once and no preprocessing is required to generate the ordering of source points.

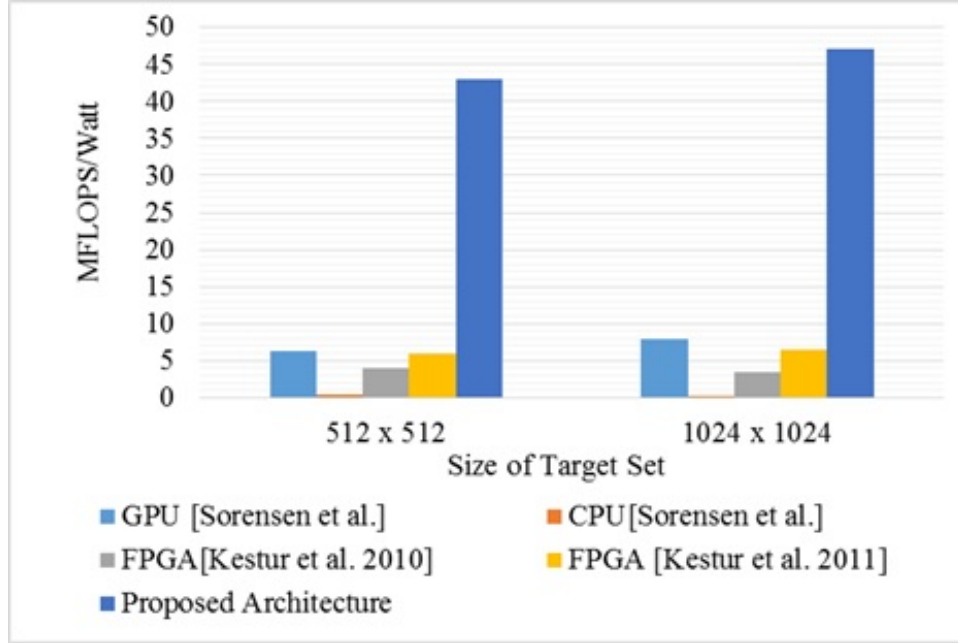


Figure 30. Comparison of Power efficiency for Radial Trajectory with related work

Curvilinear

The major application of curvilinear trajectory is in SAR image reconstruction. A number of efforts (for example, (42; 49; 59)) have been made to accelerate the SAR image reconstruction on a variety of platforms but most of the them report the overall throughput of the reconstruction process. In our work related to SAR image reconstruction (49), we reported a pipelined NuFFT based architecture for Range Migration Algorithm based SAR image reconstruction. The architecture achieved 570 frames per second (8.47 times the proposed architecture in this work) for the SAR image reconstruction because of higher (1.6 times) clock frequency achieved

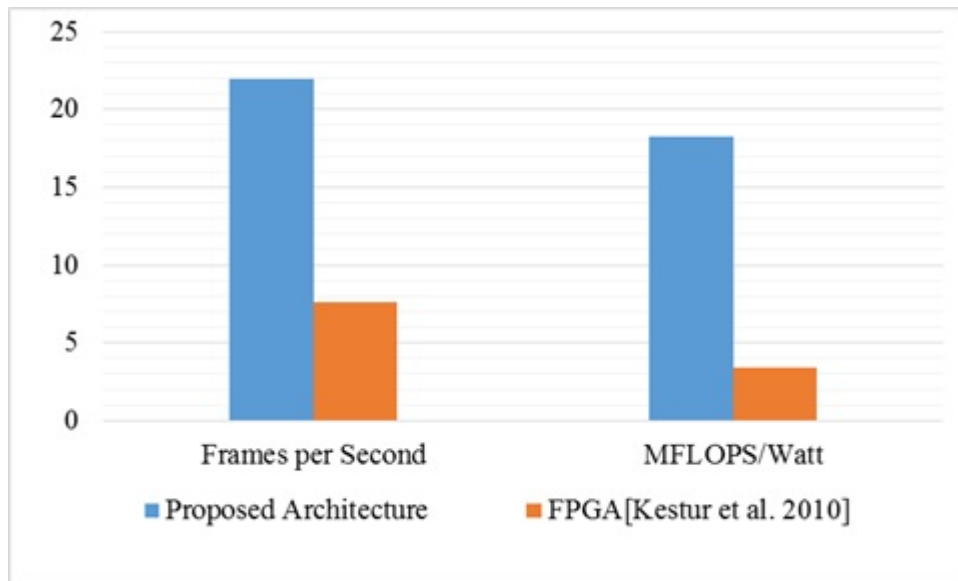


Figure 31. Comparison of throughput and power efficiency for Random trajectory and a target set of size 1024×1024

and one dimensional interpolation window. The architecture in this work, however, targets only the regridding processes but is based on 2D interpolation window. Recently, (42) targeted the regridding process in SAR imaging and demonstrated 1985.4 GFLOPS at 68.3 GFLOPS/Watt for the same frame size on a 3D-stacking technology based Application Specific Integrated Circuit (ASIC) platform. As the results are based on ASIC simulation tools and target the 3D-stacking technology it is not fair to compare it with platforms like FPGAs and GPUs.

2.6 Conclusions and Future Work

In this chapter, we proposed a novel power efficient architecture to accelerate the memory and compute intense re-gridding process in NuFFT. Various practical trajectories were considered for the sampling of source points and high throughputs were achieved for wide range of configurations at low power consumption. This was achieved by accessing the memory efficiently using novel hardware components such as block memory based FIFO buffers, fill-status based point-arbiter, decentralized memory control logic and an array of FpMA units. When compared against GPU implementations, comparable throughputs were achieved at much lower power consumption. In addition to targeting arbitrary trajectory, proposed architecture was extended to target known trajectories in MRI and SAR applications. Compared to existing FPGA based technique, up to 15 times better power efficiency was achieved in terms of MFLOPS/Watt at upto 7.35 times the throughput. Compared to GPU based technique, upto 9.6 times better power efficiency was achieved at comparable throughput. As part of future work, optimized Verilog based implementation targeting bigger frame sizes could be pursued. Since OpenCL SDK are still new to FPGA domain, as these solutions improve, better optimizations and hence throughput and power efficiency could be achieved.

CHAPTER 3

HARDWARE EFFICIENT ARCHITECTURE FOR MATRIX INVERSION

The contents of this chapter have been published in the proceedings of the 33rd IEEE International Conference on Computer Design [Copyright © 2015 IEEE] (25)

3.1 Introduction

Matrix inversion is a fundamental operation in many real-time applications. Some of these applications include Orthogonal Matching Pursuit (OMP) in Compressive Sensing based signal reconstruction (60), Multiple Input Multiple Output-Orthogonal Frequency Division Multiplexing (MIMO-OFDM) systems (61) and Cryptography(62). The computational complexity of most matrix inversion algorithms for a $N \times N$ matrix is $O(N^3)$. Considering the real-time applications of matrix inversion, efficient hardware solutions are needed that scale well with the input size. Configurable hardware, like Field Programmable Gate Arrays (FPGAs), have shown promise in high-performance computing, especially with the emergence of efficient floating-point computation solutions on FPGAs (20). Lately, with the emergence of hard floating-point blocks in FPGAs (63), the use of FPGAs as computational devices is expected to increase further. In this work, we devise an efficient architecture for computing Gauss-Jordan based Matrix Inversion. The proposed architecture employs pipelined floating point units and reordering of operations. The reordering includes normalizing multiple rows within an iteration and saving

on the multiplication operation during the elimination of rows. We use Altera's FPGA platform to quantify the performance and hardware utilization of the proposed architecture.

3.1.1 Gauss-Jordan Elimination Algorithm

Gauss-Jordan (GJ) Elimination algorithm computes the inverse of matrix using multiple elementary row operations. In order to compute inverse of a matrix A of size $N \times N$, the algorithm defines an $N \times 2N$ augmented matrix that has matrix A on the left augmented and an $N \times N$ identity matrix to the right. A series of elementary row operations are performed on the augmented matrix till the first N columns of augmented matrix are transformed to identity. At this instant, the transformed identity matrix is the inverse of A .

$$[A|I] \implies [I|A^{-1}] \quad (3.1)$$

The algorithm is iterative in nature with the number of iterations being equal to N . The algorithm consists of following steps (8):

1. *Partial Pivoting*: Pivot is defined as the largest element in column i from rows i till N .
Row i is interchanged with the row having pivot element.
2. *Normalization*: The i^{th} row, called the pivot row, is normalized by the pivot element.
3. *Forward Elimination*: For all the rows below the pivot row, perform elimination by subtracting a multiple of pivot row from each row. Increment i , and repeat the steps starting from step 1, till the left matrix is transformed to an upper triangular matrix.

4. *Back Substitution*: All elements that are above the diagonal in column i , are eliminated by subtracting a multiple of pivot row from each row. i is decremented and this step is repeated till all elements above the diagonal become zero.

In this thesis, we modify the GJ algorithm to suit the pipelined computational resources. We propose a scalable and hardware-efficient architecture for Gauss-Jordan Elimination based matrix inversion. The elements of the matrix are assumed to be available in single-precision floating-point format. An architecture for double precision is a straightforward extension by replacing the single precision blocks with double precision floating-point blocks. The proposed architecture has the following salient features:

- *Hardware-efficiency* is improved by minimizing the number Floating point multiplication units used.
- *Parallelism* is exploited by benefiting from the pipelined nature of floating-point computation blocks and the reordering of operations.
- *Scalability*: The performance of the proposed architecture is scalable with respect to hardware resources. Depending on hardware resources available in the device, number of floating point compute units can be increased and hence the performance can be scaled.

The rest of the chapter is organized as follows: Section 5.2 describes other GJ algorithm based architectures followed by description of hardware optimized architecture based on modified Gauss-Jordan algorithm in Section 3.3. Architectural details are given in Section 3.4 followed by detailed performance analysis in Section 3.5 and hardware complexity analysis in

Section 3.6. Experimental setup and results are given in Section 4.6 followed by conclusion and future work in Section 3.8.

3.2 Related Work

Matos et al., in (64), presented an analysis and straightforward implementation of Gauss-Jordan elimination on FPGAs. The proposed architecture had the memory complexity of $2N \times N$. Later they presented an improved version of their architecture in (8) where they improved the memory efficiency by a factor of 2. The improved architecture, though memory-optimized, assumes that enough computational resources are available and the performance is modeled in terms of number of memory banks. For the case of floating point arithmetic, to achieve better performance, pipelined floating point computational units are preferred (20; 65), but to the best of our knowledge, (8) assumes single cycle computational units.

Moussa et al. (61) has implemented matrix inversion for floating-point complex matrix inversion using GJ based method for small matrix sizes. Duarte et al. (7) and Garcia et al. (66) also proposed pipelined floating point units based architectures for the Gauss-Jordan based matrix inversion. In (7), the authors used a row processing unit that consists of two multiplication and one subtraction floating point unit. This row processing unit computes all the values of row in parallel. The paper does not talk about large matrices for which the whole row cannot be computed in parallel due to the lack of computational resources. Our architecture is similar in computing the elements row by row but compared to (7), it improves the hardware efficiency by reducing a Floating point multiplier units. Scalability is improved by providing scalability across columns using k processing units instead N to process a row.

The pipelined architecture by (66) uses a fixed number of floating point computation units and benchmarks their parallel architecture against software approaches. This architecture performs normalization of rows as the last step to reduce the number of multiplication operations.

A number of matrix inversion architectures based on other inversion algorithms like QR decomposition ((67)), LDL decomposition ((68)) and Rank based (69) have also been proposed.

3.3 Hardware Optimization using Modified Gauss-Jordan Algorithm

To achieve better hardware efficiency and resource utilization of the pipelined floating-point computation units, GJ algorithm is modified. The main difference is in normalizing multiple rows instead of just the pivot row in a single iteration.

Before further explanation, we define some of the terms used throughout this chapter using Figure 32. Modified Gauss-Jordan algorithm is iterative in nature. During the i^{th} iteration, we refer to the i^{th} row as the pivot row and the element at i^{th} column of pivot row as pivot element. Rows above the pivot row are called substitution rows and the ones below are called elimination rows.

Following steps are performed during the i^{th} iteration of modified GJ algorithm:

1. *Normalization*: Normalize pivot row with pivot element and each elimination row with the element in i^{th} column of the elimination row.
2. *Forward Elimination*: Eliminate all rows below the pivot by straightforward subtraction (No multiplication required).
3. *Back Substitution*: Eliminate all rows above the i^{th} row by

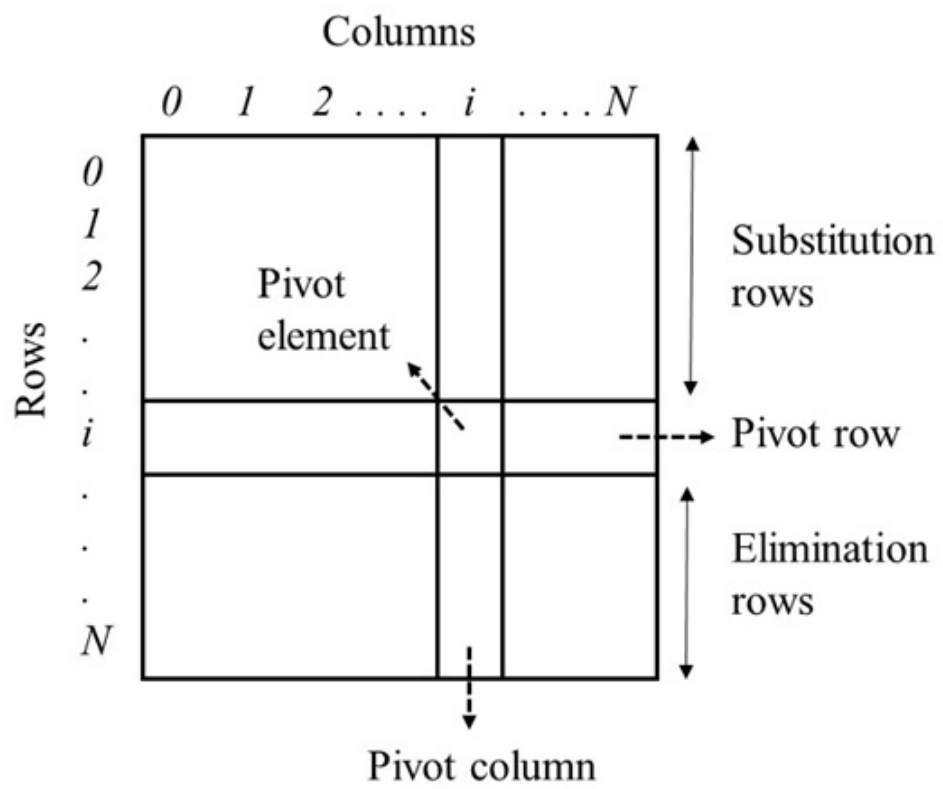


Figure 32. An $N \times N$ matrix at the i^{th} iteration

- (a) Multiplying element in i^{th} column of substitution row with the pivot row
- (b) Subtracting the product from the substitution row.

.

Pseudo-code for the algorithm is given in Figure 33 and the algorithm is applied to 3×3 matrix in Figure 34. It can be seen from Figure 34 that the algorithm performs in-place inversion, i.e., no extra store $N \times N$ storage is needed for the identity matrix rather $(N - 1) \times 1$ memory is needed to keep the normalized values of diagonal elements of the initial identity matrix.

The general GJ algorithm requires normalizing the pivot row with the pivot element. A parallel and hardware efficient-way of doing this to use a single Floating-Point Division (FPD) block in combination with multiple Floating Point Multiplication (FPM) blocks. Hardware-efficiency is achieved due to the higher hardware complexity of FPD blocks compared to FPM blocks (63). A single reciprocal is computed per iteration of the algorithm. Figure 35 shows the simplified architecture for straightforward pipelined implementation of GJ algorithm. It should be noticed that multiple (equal to N) FPM and Floating-point Subtraction (FPS) blocks are used to compute row elements in parallel. This implementation has been done in (7).

We propose an architecture based on Modified GJ algorithm that achieves further hardware efficiency by utilizing the pipelined nature of FPD block. During the i^{th} iteration, reciprocals of multiple values in i^{th} column are computed using a pipelined FPD block. Compared with general GJ algorithm, this modification saves a multiplication step in eliminating the elimination rows since all the elimination rows are normalized. Figure 36 shows the simplified architecture

```

A  $\implies$  An  $N \times N$  matrix for inversion
reciprocals  $\implies$   $1 \times N$  matrix initialized to all 1
for each iteration  $i$  do
    for each row  $m$  from  $i$  to  $N$  do
        reciprocals( $m$ ) = reciprocals  $\times$  ( $1/A(m, i)$ )
         $A(m, :) = A(m, :) \times (1/A(m, i))$ 
        //Normalize  $m^{th}$  row //A(m,:) denotes  $m^{th}$  row
    end
    if not first iteration
        for  $m$  from 1 to  $i - 1$  do
            temp =  $A(i, :) \times A(m, i)$ 
             $A(m, i) = 0$ 
             $A(m, :) = A(m, :) - temp$ 
        end
        if not last iteration
            for  $m$  from  $i + 1$  to  $N$  do
                 $A(m, i) = 0$ 
                 $A(m, :) = A(m, :) - A(i, :)$ 
            end
        end
    end
end

```

Figure 33. Pseudo-code Modified Gauss-Jordan Algorithm

for modified GJ algorithm. It should be noted that compared with Figure 35, only a single FPM is used instead of two.

3.3.1 Pipeline Utilization

The FPD, FPM and FPS blocks are cascaded together to form a pipeline. If depth of division, multiplication and subtraction pipelines are l_d , l_m and l_s respectively, the overall length, l_t , of pipeline turns out to be $l_t = l_d + l_m + l_s + 1$. A buffer is used to store the value of the pivot row elements during the forward elimination step.

For better explanation of pipeline utilization of the proposed architecture, consider the example of an 8×8 matrix shown in Figure 41. l_d , l_m and l_s are assumed to be 3, 2 and 2 respectively. It is assumed that the number of FPM and FPS blocks are equal to 8, i.e., all the elements of a row are computed in parallel. The iterations are shown in distinct colors. During each iteration, normalization of pivot row is followed by forward elimination and then back-substitution. No back-substitution is required during the first step. In the subsequent iterations, since the pipeline is filled with forward-elimination rows, no stalls are required to wait for the normalized pivot row elements except during the last iteration. For comparison purposes, pipeline utilization of general GJ algorithm is also shown. In this case, pipeline has to be stalled every iteration to wait for the normalized pivot row elements.

3.4 Proposed Architecture and Data-flow

The data-path of the proposed architecture is shown in 5.3. It consists of k Normalization and Elimination (NE) blocks. Each NE block consists of a FPM and a FPS block. We define their pipelined depth as l_m and l_s respectively. The parameter k is selected based on the

available resources on the device. All k NE blocks are synchronized and perform normalization and elimination operation on elements from k different columns. Each NE block has a local Block memory based Random Access Memory (RAM) called Matrix RAM. The contents of the matrix to be inverted are distributed over these RAMs. Each column of matrix is associated with a unique NE block. This helps minimize the routing resource utilization of the design and allows parallel access of data in a particular row.

A single pipelined FPD block is used since elements of a single row are processed in one cycle. A separate RAM, called R-RAM, is used to store the values of diagonal elements in the initial identity matrix present in augmented matrix. This memory is required since multiple rows are normalized at once and hence the updated value of diagonal elements need to be stored. The control of data-path is handled through a dispatch state-machine along with multiple counters and pipelines for the control signals. The counters keep track of the iteration number, row index and the range of columns being processed and help in achieving the overall synchronization.

3.4.1 Data-flow

Initialization

The matrix A to be inverted is stored in k Matrix RAMs in single precision floating point format. All the values in R-RAM are initialized to one. The in-place matrix-inversion is done in an iterative manner.

Normalization

During the i^{th} iteration, reciprocal of all the elements in the pivot column and elimination rows are computed using FPD in a pipelined fashion. First element fed to FPD is at i^{th} row

and i^{th} column followed by element in $(i + 1)^{st}$ row and i^{th} column in following cycle and so on. Reciprocals from FPD are fed to FPMs in all k NE blocks in order to normalize k values in pivot and elimination rows.

If $k = N$, the complete row is processed in parallel but if $k < N$, only k values in a row can be normalized in parallel. Once k columns in all pivot and elimination rows are normalized, elements in next k columns are normalized in a second pass. When the last $k - 1$ values of the i^{th} row are being computed, the value in R-RAM corresponding to elimination row is also normalized and stored in the pivot column of the elimination row. No extra FPM would be required for this since element at i^{th} column in current didn't need any multiplication and FPM on one NE block will be available (N is perfectly divisible by k).

Forward-Elimination

The values from FPM are available one value per cycle. Elements of pivot row are saved in a row buffer and fed to FPS for subtraction from the elimination rows.

Back-Substitution

Back-substitution follows forward-elimination in a pipelined manner as shown in Fig Figure 41. It involves multiplication of values in pivot column of substitution row with the pivot row and subtracting the result from the substitution row.

The repetition of these iteration N times results in an in-place inversion of matrix.

3.4.2 Scalability

Scalability across the rows is achieved by computing elements of one row at a time in a pipelined manner. In order to add scalability across the columns, values across k (instead

of N) columns are computed in parallel, where $k < N$ and is chosen based on the hardware resources available. In i^{th} iteration, other than the dependence of elements in i^{th} column, all the computational dependencies are within the same column. Hence computation of k columns for a given iteration is completed before moving on to the next k columns. This also helps in minimizing the memory interface traffic. This is illustrated in the Figure 37. If the on-chip memory cannot fit elements in k columns of N rows, l rows can be saved, where $l < N$ and the rest of the rows can be read from the external memory. Depending on memory latency, stalling the pipeline may be required in this case. However, in this work, we choose $l = N$, i.e., we assume all the rows of a sub-matrix of size $N \times k$ can fit on the on-chip memory of the device.

3.5 Performance Analysis

The total number of clock cycles, c , required to compute inverse for a matrix of size $N \times N$ using k NE blocks is given by Equation 3.2. In the analysis, we assume that $N > (l_t - 1)$. It has to be noted that during last $l_m - 1$ iterations, the pipeline has to be stalled to wait for the normalized pivot row elements. At the $(N - j)^{th}$ iteration, where $0 \leq j < l_m - 1$, the pipeline has to be stalled for $l_m - j - 1$ clock cycles. In Figure 41, a stall of single cycle can be noticed during the last iteration.

$$c = l_d + l_m + l_s + [N^2 + l_m(l_m - 1)/2 - 1] \times (N/k) \quad (3.2)$$

3.5.1 Performance comparison with related work

For performance reference, we compare the performance of our proposed architecture with that of a pipelined implementation of simple GJ algorithm in (7). Although (7) uses N NE blocks, we develop a model for this pipelined approach assuming k NE blocks for better comparison. In terms of notations used in this chapter, the number of clock cycles c_d required to compute the invert of a matrix is given by equation Equation 3.3. Compared to our proposed architecture, it utilizes an extra FPM per NE block but does not require the stall cycles during the last few iterations.

$$c_d = l_d + 2 \times l_m + l_s + N^2 \times (N/k) \quad (3.3)$$

The pipelined implementation of simple GJ algorithm utilizing the same amount of hardware resources would require additional stall cycles per iteration. The number of clock cycles c_m required for simple GJ algorithm utilizing the same hardware resources is given by equation Equation 3.4.

$$c_m = l_d + l_m + l_s + [(N + (lm - 1)) \times N] \times (N/k) - 1 \quad (3.4)$$

By comparing Equation 3.2 and Equation 3.3, it can be seen that there is no significant drop in performance using our proposed architecture compared to the pipelined implementation of GJ algorithm in (7). If performance comparison is made using the same hardware resources

(comparing Equation 3.2 with Equation 3.4), our proposed architecture performs significantly better, specially for large matrix sizes due to the additional stall cycles incurred every iteration.

In (8), memory optimized architecture for GJ algorithm has been proposed. It assumes single cycle computation blocks and report performance in terms of memory banks. We compare the performance when the number of memory banks (b) are equal to k . Figure 39 shows the performance of the architectures discussed above for a matrix size of 1024×1024 .

Another GJ algorithm based architecture is proposed in (66). The authors report results for relatively small matrix sizes only. For a 36×36 matrix, the number of clock cycles required are 12,500 compared to 4,089 required in the our proposed architecture using $k = 12$.

Performance of the *InvArch* for various large matrix sizes is shown in the table Table IV in terms of number of clock cycles. The size of matrices and the value of k are chosen based on resources available on Altera Straix IV FPGA.

3.6 Hardware Complexity Analysis

Performance comparison reveals that pipelined implementation of GJ algorithm (7) performs only marginally better compared to *InvArch* but *InvArch* require much lower hardware resources for floating point computational units. Since there are k NE blocks and our architecture reduces one FPM per NE block, hardware complexity of *InvArch* is much lower than the pipelined implementation of Gauss Jordan algorithm (7). Figure 40 shows a comparison of the hardware resources of *InvArch* with (7). The percentage reduction in hardware resources for floating-point computation units is given in table Table V for various values of k . Even for small number of NE

TABLE IV

PERFORMANCE FOR LARGE MATRIX SIZES IN TERMS OF NUMBER OF CLOCK
CYCLES

Matrix Size	k	Clock cycles (Thousands)	Projected Computation Times (100MHz)	Projected Computation Times (200MHz)
256 x 256	128	131.2	1.31 ms	0.66 ms
512 x 512	128	1048.8	10.48 ms	5.24 ms
1024 x 1024	128	8389.1	83.89 ms	41.94 ms
2048 x 2048	128	67109.8	671.1 ms	335.5 ms

TABLE V

PERCENTAGE REDUCTION IN HARDWARE RESOURCES FOR FLOATING-POINT
COMPUTATION UNITS COMPARED TO PIPELINED IMPLEMENTATION OF GJ

k	% reduction in ALUTs	% reduction in DSP blocks
8	80.21	66.67
16	86.12	80
32	89.4	88.89
64	91.15	94.11
128	92.5	96.97
256	92.5	98.46

units, the reduction in hardware resources is over 80%. Other GJ algorithm based architectures do not give sufficient architecture information to deduce the hardware complexity.

3.7 Experimental Setup and Results

The design is described in Verilog Hardware Description Language using Altera Quartus 14.1 on Stratix IV FPGA. The floating point compute units used were generated using Altera's Megawizard. (20). The simulations were performed on Modelsim Altera 10.0d. For the ease

TABLE VI

PERFORMANCE FOR VARIOUS MATRIX SIZES

Matrix Size	k	Clock cycles	Computation Time (f = 200 MHz)
32 x 32	32	2,327	11.63 μ s
64 x 64	64	6,647	33.23 μ s
128 x 128	128	21,431	107.1 μ s

of implementation, following relaxations were made in the implemented architecture. The implementation will be extended as part of future work to eliminate these relaxations:

1. Although two consecutive iterations of algorithm can be in the pipeline during the computation as shown in Figure 41, it is assumed that an iteration is completed before the start of next.
2. We assume $k = N$. This implies N NE blocks and hence the architecture did not fit in FPGAs for larger matrices with DSP blocks being the bottleneck.

The computation times for various matrix sizes for the implemented architecture are given in table Table VI. The clock counts are higher than the analytical model (equation Equation 3.2) because the implemented architecture processes one iteration in pipeline at a time.

The value of k and N are chosen to be powers of 2 as it helps in better parameterization of the architecture. Based on the available DSP resources on Stratix IV FPGA used, the maximum value of k (that is power of 2) could be 128.

Since the number of NE blocks is k and these are the main source of hardware complexity, the data-path complexity mainly depends on k . The operating frequency achieved for maximum possible value of $k = 128$ is $237.02MHz$. We project the results for bigger matrix sizes using a clock frequency that is much lower than the one achieved for maximum k . The frequency is chosen conservatively since the added complexity in control would be to cater for non-disjoint iterations and $k < N$. But hardware complexity of the data-path would not change. Based on the analytical model of the proposed technique in equation Equation 3.2 and operating frequencies of $100MHz$ and $200MHz$, the projected performance for higher matrix sizes are computed in table Table IV.

Considering the higher number of normalization operations compared to the original GJ algorithm, the precision of floating-point numbers may be effected. Precision analysis of the proposed architecture is part of the future work.

3.8 Conclusions and Future Work

In this chapter, we proposed a novel and hardware efficient architecture for matrix inversion. The architecture is based on normalizing multiple rows in an iteration benefiting from the pipelined nature of floating point blocks and thereby reducing the number of multiplication units required. This results in 80% reduction in hardware resources for floating point computational logic compared to the existing pipelined implementation. The proposed architecture also

provides better scalability and hardware efficiency at comparable performance. Better memory complexities have been claimed in the GJ based architectures ($N \times N$ compared with our architecture $(N + 1) \times N$) but either they are based on unreasonable assumption of computing floating-point values in single cycle (8) or having higher hardware complexities (7). We also present an analytical model of the proposed architecture as well as a scalable version for the architecture proposed in (7). A precision comparison of the proposed architecture against the general GJ based implementation would be done as part of future work. The FPGA implementation has to be improved using $k < N$ NE blocks to compute for larger matrix sizes and also improve the implementation to support processing elements from two successive iterations in the pipeline simultaneously.

Augmented Matrix						
	7	2	1	1	0	0
	5	3	-1	0	1	0
	-3	4	-2	0	0	1

First Iteration						
1	1	0.28571	0.14286	0.14286	0	0
	1	0.6	-0.2	0	0.2	0
	1	-1.33333	0.66667	0	0	-0.33333
2	1	0.28571	0.14286	0.14286	0	0
	0	0.31429	-0.34286	-0.14286	0.2	0
	0	-1.61905	0.52381	-0.14286	0	-0.33333

Second Iteration						
1	1	0.28571	0.14286	0.14286	0	0
	0	1	-1.09091	-0.45455	0.63636	0
	0	1	-0.32353	0.08824	0	0.20588
2	1	0.28571	0.14286	0.14286	0	0
	0	1	-1.09091	-0.45455	0.63636	0
	0	0	0.76738	0.54278	-0.63636	0.20588
3	1	0	0.45455	0.27273	-0.18182	0
	0	1	-1.09091	-0.45455	0.63636	0
	0	0	0.76738	0.54278	-0.63636	0.20588

Third Iteration						
1	1	0	0.45455	0.27273	-0.18182	0
	0	1	-1.09091	-0.45455	0.63636	0
	0	0	1	0.70732	-0.82927	0.26829
2	1	0	0	-0.04878	0.19512	-0.12195
	0	1	0	0.31707	-0.26829	0.29268
	0	0	1	0.70732	-0.82927	0.26829

Memory						
Matrix				Inverses		
7	2	1				1
5	3	-1				1
-3	4	-2				1

0.14286	0.28571	0.14286	0.142857
0	0.6	-0.2	0.2
0	-1.33333	0.66667	-0.33333
0.14286	0.28571	0.14286	0.142857
-0.14286	0.31429	-0.34286	0.2
-0.14286	-1.61905	0.52381	-0.33333

0.14286	0.28571	0.14286	0.142857
-0.45455	0.63636	-1.09091	0.636364
0.08824	0	-0.32353	0.205882
0.14286	0	0.14286	0.142857
-0.45455	0.63636	-1.09091	0.636364
0.54278	-0.63636	0.76738	0.205882
0.27273	-0.18182	0.45455	0.142857
-0.45455	0.63636	-1.09091	0.636364
0.54278	-0.63636	0.76738	0.205882

0.27273	-0.18182	0.45455	0.142857
-0.45455	0.63636	-1.09091	0.636364
0.70732	-0.82927	0.26829	0.20588
-0.04878	0.19512	-0.12195	0.142857
0.31707	-0.26829	0.29268	0.63636
0.70732	-0.82927	0.26829	0.268293

Values required for further computation

Not required

Figure 34. Steps to find invert of a sample 3×3 matrix. The augmented matrix is shown along with the contents of the memory. The reciprocal is found over three steps. The values in green indicate the values that required for further computation and are stored in the memory. Values in red indicate the values that are no longer required in computation

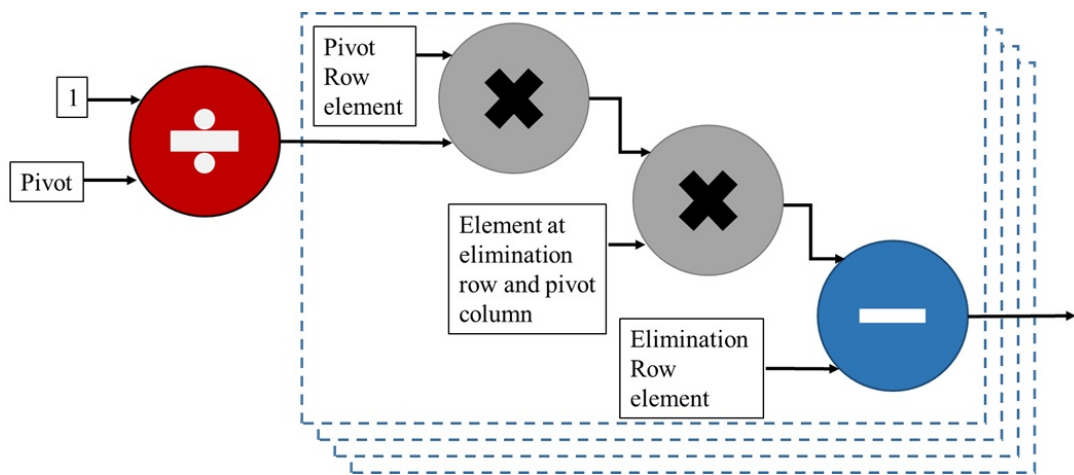


Figure 35. Simplified architecture of pipelined Implementation of Gauss-Jordan Algorithm.

Two floating-point multiplication elements are required per row.

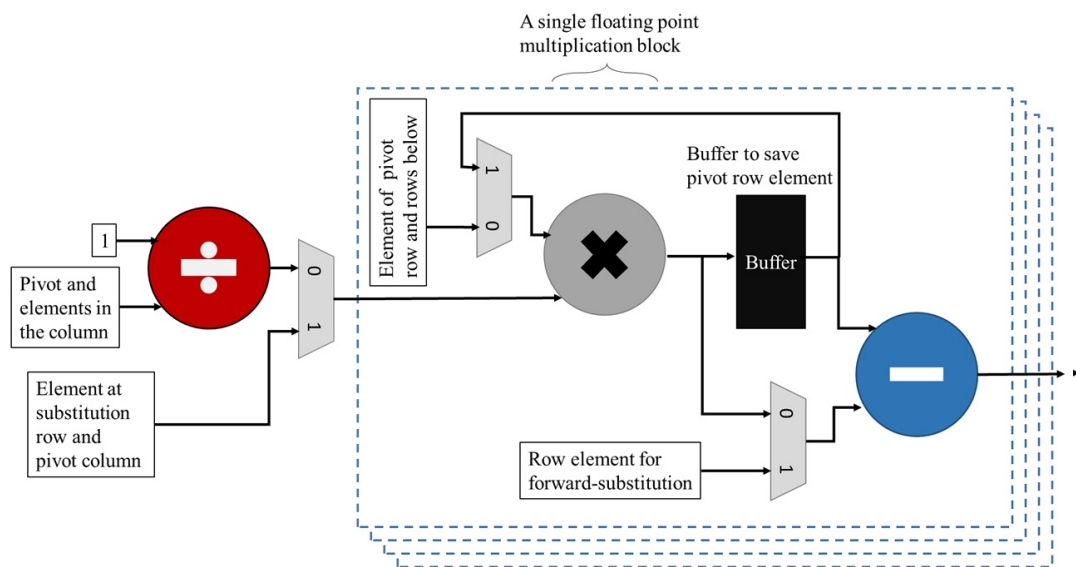


Figure 36. Simplified data-flow of Pipelined Implementation for Modified Gauss-Jordan Algorithm. A single floating-point multiplication is required per row as compared to two for the pipelined implementation of original Gauss Jordan algorithm (Figure 35)

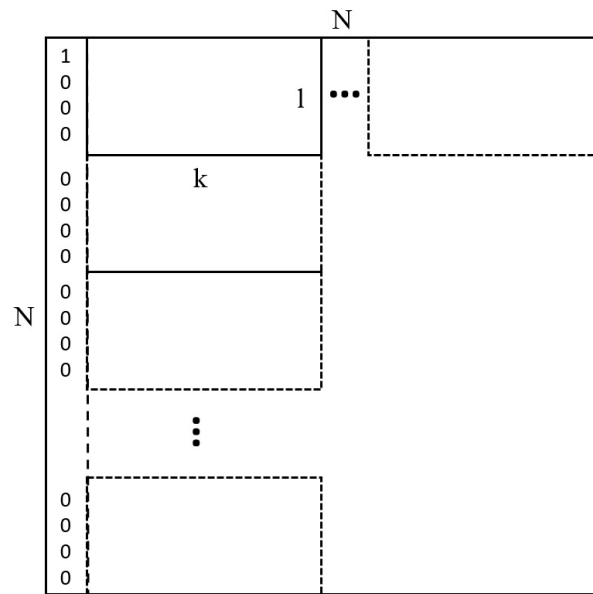


Figure 37. Computational flow scalability

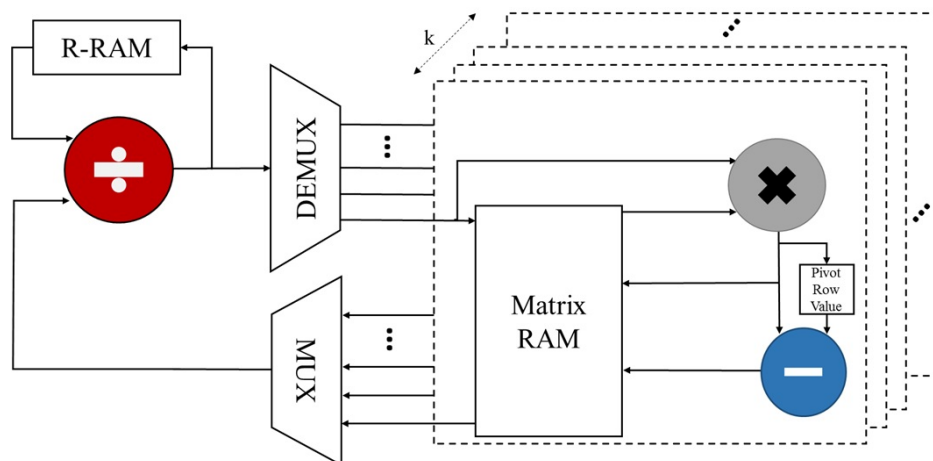


Figure 38. Block diagram of the data path

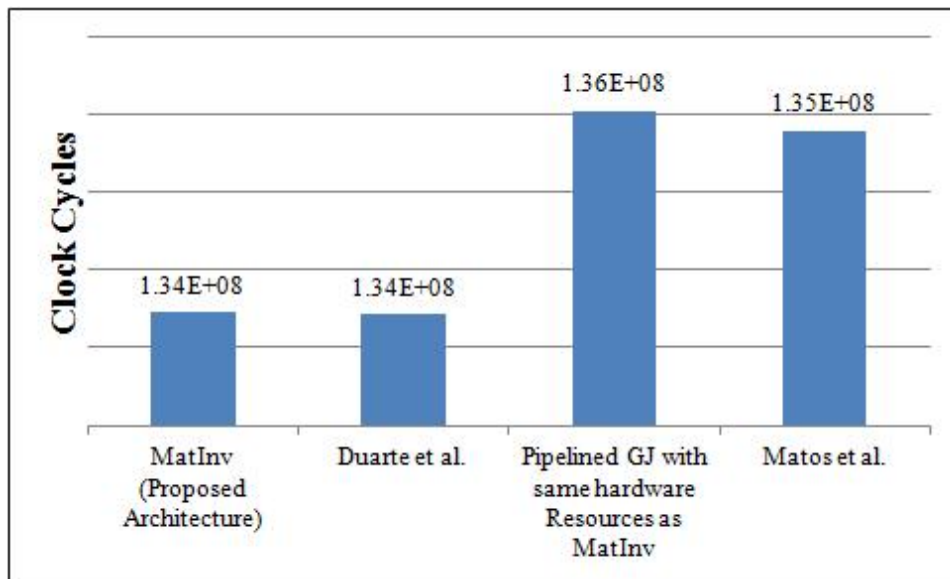


Figure 39. Comparison of proposed architecture (InvArch) ($k = 8$) against Duarte et al. (7) ($k = 8$), Matos et al (8) ($b = 8$) and pipelined GJ implementation with same hardware resources as our proposed architecture ($k = 8$) for a 1024×1024 matrix.

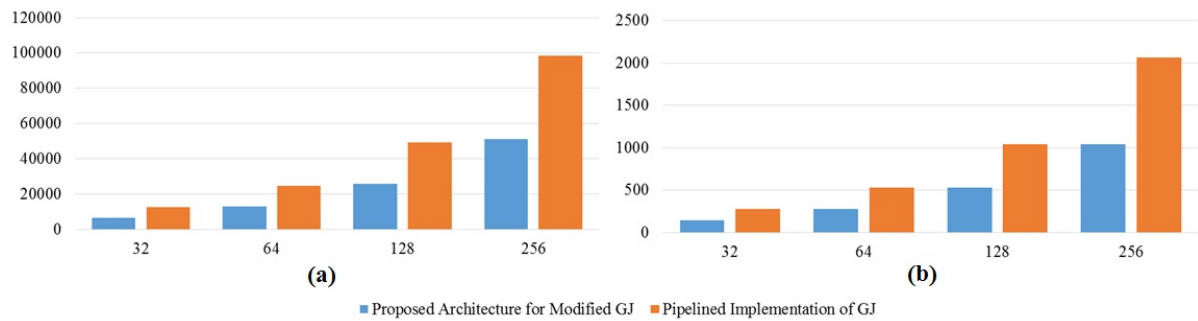


Figure 40. Comparison of the hardware resources required for implementing floating point blocks for different values of k . (a) gives the number of Adaptive Look-Up Tables (ALUT)s and (b) gives the number of DSP blocks. The numbers are for the area-optimized implementation of Altera Floating point units on Startix IV FPGAs

Clock Cycle	Pipeline utilization for Modified Gauss Jordan Algorithm				Pipeline utilization for Gauss-Jordan Algorithm using single Multiplication block			
	DIVISION Pipeline Depth = 3	MULTIPLICATION Pipeline Depth = 2	SUBTRACTION Pipeline Depth = 2	BUFFER	DIVISION Pipeline Depth = 3	MULTIPLICATION Pipeline Depth = 2	SUBTRACTION Pipeline Depth = 2	
1	1/a11				1/a11			
2	1/a21	1/a11			1/a11	1/a11		
3	1/a31	1/a21	1/a11			1/a11		
4	1/a41	1/a31	1/a21	a11c11		a11c11		
5	1/a51	1/a41	1/a31	a21c21	a11c11	a11c11		
6	1/a61	1/a51	1/a41	a31c31	a21c21	a11c11	a21* m1k	
7	1/a71	1/a61	1/a51	a41c41	a31c31	a11c11	a31* m1k a21* m1k	
8	1/a81	1/a71	1/a61	a51c51	a41c41	a11c11	a41* m1k a31* m1k	a21c m2k
9	1/a22	1/a81	1/a71	a61c61	a51c51	a11c11	a51* m1k a41* m1k	a31c m3k a21c m2k
10	1/a32	1/a22	1/a81	a71c71	a61c61	a11c11	a61* m1k a51* m1k	a41c m4k a31c m3k
11	1/a42	1/a32	1/a22	a81c81	a71c71	a11c11	a71* m1k a61* m1k	a51c m5k a41c m4k
12	1/a52	1/a42	1/a32	a21c a22	a81c81	a11c11	a81* m1k a71* m1k	a61c m6k a51c m5k
13	1/a62	1/a52	1/a42	a31c a32 a21c a22	a21c a22	a81c m1k	a21c a22 a81* m1k	a71c m7k a61c m6k
14	1/a72	1/a62	1/a52	a41c a42 a31c a32	a21c a22	a81c m1k	a21c a22	a81c m8k a71c m7k
15	1/a82	1/a72	1/a62	a51c a52 a41c a42	a21c a22	a31c m2k	a31* m2k	a81c m9k
16	1/a82	1/a72	a61c a62 a51c a52	a21c a22	a41c a22 a41c m2k a31c m2k	a31* m2k	a31* m2k a11* m2k	
17	1/a33	1/a82	a71c a72 a61c a62	a21c a22	a51c a22 a51c m2k a41c m2k	a41* m2k	a41* m2k a31* m2k	a11c m1k
18	1/a43	1/a43	a81c a82 a71c a72	a21c a22	a61c a22 a61c m2k a51c m2k	a51* m2k	a51* m2k a41* m2k	a31c m3k a11c m1k
19	1/a53	1/a43	m31c a11 a81c a82	a21c a22	a71c m2k a61c m2k	a61* m2k	a61* m2k a51* m2k	a41c m4k a31c m3k
20	1/a63	1/a53	a31c a33 m21c a11	a81c m2k a71c m2k	a71* m2k a61* m2k	a51* m2k	a51* m2k a41* m2k	a31c m3k a21c m2k
21	1/a73	1/a63	a41c a43 a31c a33	a31c a33	a61* m2k a51* m2k	a41* m2k	a41* m2k a31* m2k	a21c m2k a11c m1k
22	1/a83	1/a73	a51c a53 a41c a43	a31c a33	a31c a33	a31c a33	a31c a33	a11c m1k
23	1/a83	1/a73	a61c a63 a51c a53	a31c a33	a41c a33	a41c m3k	a41* m3k	a31c m3k
24	1/a83	1/a73	a71c a73 a61c a63	a31c a33	a51c a33	a51c m3k a41c m3k	a51* m3k	a41* m3k
25	1/a44	1/a44	a81c a83 a71c a73	a31c a33	a61c a33	a61c m3k a51c m3k	a61* m3k	a51* m3k
26	1/a54	1/a54	m31c a11 a81c a83	a31c a33	a71c m3k a61c m3k	a61* m3k	a61* m3k a51* m3k	a41c m4k a31c m3k
27	1/a64	1/a64	a31c a34 m31c a11	a81c m3k a71c m3k	a71* m3k a61* m3k	a51* m3k	a51* m3k a41* m3k	a31c m3k a21c m2k
28	1/a74	1/a64	a41c a44 m31c a11	a11c m1k a81c m3k	a61* m3k a51* m3k	a41* m3k	a41* m3k a31* m3k	a21c m2k a11c m1k
29	1/a84	1/a74	a51c a54 m31c a11	a21c m2k a11c m1k	a51* m3k a41* m3k	a31* m3k	a31* m3k a21* m3k	a11c m1k
30	1/a84	1/a74	a61c a64 a51c a54	a41c a44	a61* m3k a51* m3k	a41* m3k	a41* m3k a31* m3k	a21c m2k a11c m1k
31	1/a84	1/a74	a71c a74 a61c a64	a41c a44	a71c m3k a61* m3k	a51* m3k	a51* m3k a41* m3k	a31c m3k a21c m2k
32			a81c a84 a71c a74	a41c a44	a81c m3k a71* m3k	a61* m3k	a61* m3k a51* m3k	a41c m4k a31c m3k
33	1/a55		m31c a11 a81c a84	a41c a44	a71c m3k a61* m3k	a51* m3k	a51* m3k a41* m3k	a31c m3k a21c m2k
34	1/a65	1/a55	m31c a21 m31c a11	a81c m3k a71c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
35	1/a75	1/a65	m31c a31 m31c a21	a11c m1k a81c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k	a41c m4k a31c m3k
36	1/a85	1/a75	a51c a55 m31c a21	a21c m2k a11c m1k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
37	1/a85	1/a75	a61c a65 a51c a55	a31c m3k a41c m3k	a51* m3k	a51* m3k a41* m3k	a41* m3k	a31c m3k a21c m2k
38	1/a85	1/a75	a71c a75 a61c a65	a41c a55	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
39			a81c a85 a71c a75	a41c a55	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
40			m31c a21 a61c a85	a51c a55	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
41	1/a66		m31c a31 m31c a21	a81c m3k a71c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
42	1/a76	1/a66	m31c a41 m31c a31	a11c m1k a81c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k	a41c m4k a31c m3k
43	1/a86	1/a76	a51c a56 m31c a31	a21c m2k a11c m1k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
44	1/a86	1/a76	a61c a66 m31c a31	a31c m3k a41c m3k	a51* m3k	a51* m3k a41* m3k	a41* m3k	a31c m3k a21c m2k
45			a71c a76 a61c a66	a41c m3k a51c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
46			a81c a86 a71c a76	a41c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
47			m31c a21 a61c a86	a51c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
48			m31c a31 m31c a21	a81c m3k a71c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
49	1/a77		m31c a41 m31c a31	a11c m1k a81c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k	a41c m4k a31c m3k
50	1/a87	1/a77	m31c a51 m31c a41	a21c m2k a11c m1k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
51	1/a87	1/a77	m31c a61 m31c a51	a31c m3k a41c m3k	a51* m3k	a51* m3k a41* m3k	a41* m3k	a31c m3k a21c m2k
52			a71c a77 a61c a61	a41c m3k a51c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
53			a81c a87 a71c a77	a41c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
54			m31c a21 a61c a87	a51c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
55			m31c a31 m31c a21	a81c m3k a71c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
56			m31c a41 m31c a31	a11c m1k a81c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k	a41c m4k a31c m3k
57	1/a88	1/a88	m31c a51 m31c a41	a21c m2k a11c m1k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
58			m31c a61 m31c a51	a31c m3k a41c m3k	a51* m3k	a51* m3k a41* m3k	a41* m3k	a31c m3k a21c m2k
59			a71c a78 a61c a61	a41c m3k a51c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
60			a81c a88 a71c a78	a41c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
61			m31c a21 a61c a88	a51c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
62			m31c a31 m31c a21	a81c m3k a71c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
63			m31c a41 m31c a31	a11c m1k a81c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k	a41c m4k a31c m3k
64			m31c a51 m31c a41	a21c m2k a11c m1k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
65			m31c a61 m31c a51	a31c m3k a41c m3k	a51* m3k	a51* m3k a41* m3k	a41* m3k	a31c m3k a21c m2k
66			a71c a79 a61c a61	a41c m3k a51c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
67			a81c a89 a71c a79	a41c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
68			m31c a21 a61c a89	a51c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
69			m31c a31 m31c a21	a81c m3k a71c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
70			m31c a41 m31c a31	a11c m1k a81c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k	a41c m4k a31c m3k
71			m31c a51 m31c a41	a21c m2k a11c m1k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
72			m31c a61 m31c a51	a31c m3k a41c m3k	a51* m3k	a51* m3k a41* m3k	a41* m3k	a31c m3k a21c m2k
73			a71c a80 a61c a61	a41c m3k a51c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
74			a81c a90 a71c a80	a41c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
75			m31c a21 a61c a90	a51c a66	a71c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k a41c m3k
76			m31c a31 m31c a21	a81c m3k a71c m3k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k
77			m31c a41 m31c a31	a11c m1k a81c m3k	a71* m3k	a71* m3k a61* m3k	a51* m3k	a41c m4k a31c m3k
78			m31c a51 m31c a41	a21c m2k a11c m1k	a61* m3k	a61* m3k a51* m3k	a41* m3k	a31c m3k a21c m2k

Figure 41. Comparison of pipeline-utilization for modified-gauss jordan algorithm (left) with Gauss-jordan algorithm utilizing same number of floating point multiplication units (right).

Iterations are indicated with distinct colors. Note that we assume number of NE blocks to be

8 in this figure i.e., 8 multiplication and subtraction units. During the i^{th} iteration, for

forward-elimination, result of normalized pivot ($m_{ik} = a_{ik}/a_{ii}$) is buffered to be eliminate the elimination rows. For back-substitution, normalized pivot row element is available in time for

the first 7 iterations but needs a single cycle stall for the last iteration. For same hardware

resources, the pipeline on right needs stall cycles every iteration.

CHAPTER 4

A HIGH PERFORMANCE ARCHITECTURE FOR COMPUTING BURROWS-WHEELER TRANSFORM ON FPGAS

The contents of this chapter have been published in the proceedings of 2013 International Conference on Reconfigurable Computing and FPGAs [Copyright © 2013 IEEE] (26)

4.1 Background

Burrows-Wheeler Transform (BWT), published in 1994 (70), revolutionized the data compression world. BWT transforms the data into a form that has long sequences of the same characters and hence it is easier to compress. Due to the similarity between BWT and suffix arrays (71), FM-index (72) was developed which led to the use of BWT in string matching as well. It has also found applications in Bioinformatics (73), Computational Biology (74), Image processing (75), computer vision (76), Test Data Compression (77) and Communications (78). In Bioinformatics, it is used for whole-genome comparisons, genome annotation and measuring distances between two sequences. In computer vision, it is used for image compression, machine translation and shape matching. In the communications domain, it is used in channel coding. Due to the real time nature of most of the applications, it is highly desirable to realize fast and hardware efficient implementations of the BWT that yield high throughput. An intuitive way to understand BWT is to view all the cyclic rotations for a string s of length n in the form of a matrix M , as shown in Figure 42. Assume that all the rows of M are sorted lexicographically

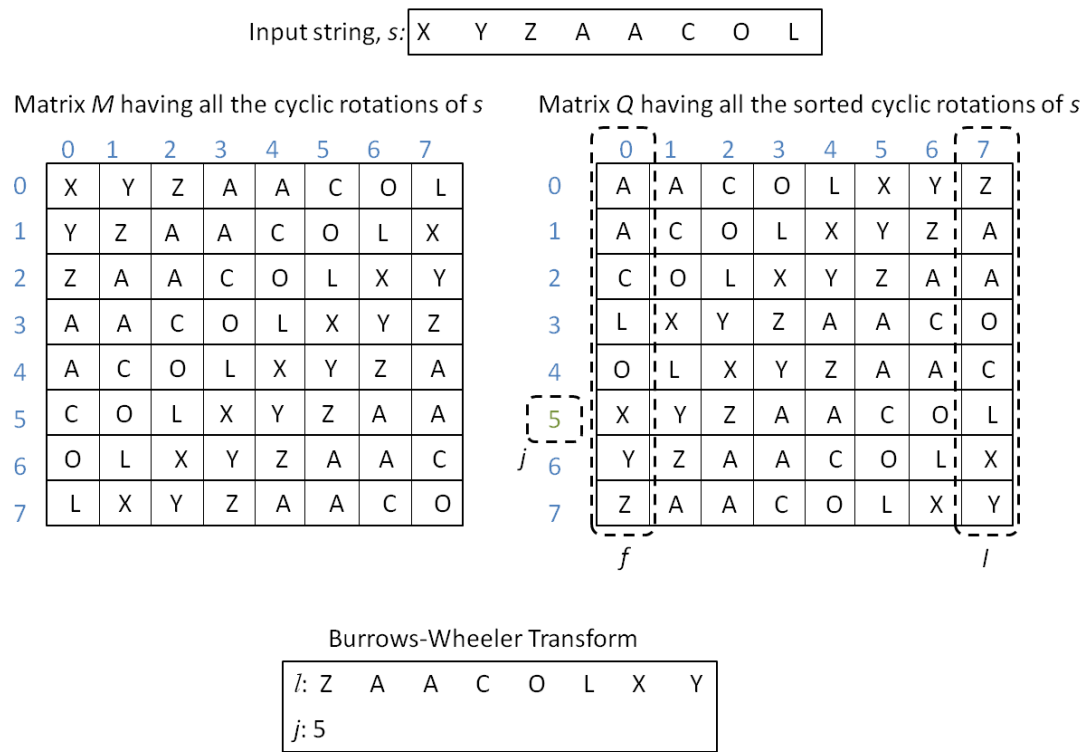


Figure 42. Visual illustration of BWT: input string s along with Matrix M , matrix Q and the Burrows-Wheeler Transform (BWT)

and stored in a matrix Q . The last column l of matrix Q along with j , where j represents the position of original string s in Q , is the BWT of the string. So BWT computation is basically sorting of the all the cyclic rotations of BWT. Once we have the BWT, original string s can be reconstructed efficiently from l and j by retrieving matrix Q column by column. For additional details of BWT we refer to (78).

Computing the BWT in hardware by saving all the sorted rotations of s in the form of matrices Q and M is not efficient, especially when the string size is large. The existing hardware techniques to compute BWT are based on the observation that each character in column l of Q is the prefix of character in the corresponding row of f . Also, since l and f are just permutations of string s , they can be represented in terms of indexes of characters in the s . Consider a character u in a particular row of column f and character v in column l of the same row. If i and t are the index of u and v in s respectively, t can be determined from u using Equation 4.1.

$$t = (i - 1) \bmod n \quad (4.1)$$

In other words, BWT can be computed from f in terms of indexes of characters in s . Since, f is a sorted version of s , it can be determined by a single sorting iteration if there are no repeated characters in s . When there are repeated characters in s , in order to have correct index location of repeated characters in f , the ordering of repeated characters is determined based on characters that precede repeating characters in s . Once column f is computed, column l and hence BWT can be computed using Equation 4.1 by the procedure mentioned above. In the hardware techniques proposed earlier ((79) and (80)), the ordering of repeated characters is usually done by repeated sorting iterations. In this work, we propose an innovative technique to compute BWT which is based on the observation that in general, we have common prefixes amongst the cyclic rotations of string. These prefixes are usually not long but they

reduce the overall throughput due to multiple sorting iterations. During each iteration, multiple suffix characters have to be loaded and compared, which incurs delays and hence reduction in throughput. Instead of having multiple iterations to cater for these common prefixes, we save a limited context for each character of string s and perform the sorting operation on these contexts. We call these contexts blocks. A block of size k represents the k^{th} order context, where k is the size of the Longest Common Prefix (LCP) in the cyclic rotations of string s . In terms of matrix M , we use only the first k characters of each of the n cyclic rotations of string s . Based on these blocks, BWT is computed using a suffix block generator, a tree structured pipeline of flow through First-In-First-Out (FIFO) memories, and a parallel sorter. A variant of BWT called the Sort Transform (ST) (81) also uses limited context to sort the string rotations. Therefore, our implementation can be used for the ST computation as well.

4.2 Related Work

To the best of our knowledge, the first effort aimed at implementing BWT in hardware was reported by Mukherjee et al. in (79) and it is based on saving the original index i of characters in string s and sorting the string. For a group of same characters, comparison of the following characters is made. This technique is based on Weavesorter algorithm. The basic weavesorter operation starts by shifting the string s right character by character and performing compare/swap operation at each step. When the string is completely inserted, the left-most character is the smallest character in the string s . Next, the direction of shift operation is changed and the string is shifted out character by character and compare swap/ operation is performed at each step. The smallest character, amongst all the characters in Weavesorter, is

shifted out in each step. So, the output is a sorted version of the input string. To determine the order of repeated characters, the following characters in the original string are inserted by alternating the direction of insertion to the weavesorter but the original indices are maintained. The process is repeated till the correct order is determined.

To improve on the weavesorter approach, (80) proposed parallel sorting strategy for sorting. A single sorting iteration takes $n/2$ cycles at most. Index i of each character u in the original string is saved in the form of (i, u) . Note that for a particular row in Q , the last character (that is in column l) will have an index that is one less than the index of the character in f . Hence the problem of computing BWT is simplified to finding the order of characters in the first column f in Q . As first column is simply the sorted version of s given there are no repeated characters in s , the problem is reduced to a simple string sorting problem. But if there is character repetition in s , the correct order of characters cannot be determined by single sorting iteration. The relative order of the repeated characters is determined by sorting the suffix characters in the same row. The following characters are loaded by shifting values amongst the registers. In the worst case, it will take n sorting iterations to find the correct order of characters in first column of Q . Once the correct order is determined, the index t of each character in BWT is calculated using Equation 4.1.

For the case of common prefixes, shifting registers to load the following characters becomes challenging and time consuming. Reading multiple suffix character from the memory itself would reduce the throughput considerably since the memory accesses are slow and most memory interfaces are single or dual port. The problem would even be worse for long strings, since there

will be a possibility of greater number of common prefixes and hence more characters have to be shifted around the registers that would require more number of clock cycles. Our approach minimizes the above mentioned common prefix problem by saving the k -order context for each character where k is the size of LCP and uses parallel suffix sorter and pipeline of flow-through queue structures to compute BWT. Further details are discussed in the next section.

4.3 Flowthrough FIFO based technique for computing BWT in Hardware

As mentioned earlier, our approach of computing BWT in hardware uses limited string context to sort the sorted rotations of the input string. Hardware complexity is much less compared to the trivial way of computing transform by sorting the complete sorted rotations. If we look at it from the perspective of matrix M in Figure 42, only the first k characters are used to sort rotations. Another way of looking at it would be to save k^{th} order context corresponding to each character of string s and sort these contexts. We call these contexts blocks. Our technique, like the other two techniques described in the previous section, saves the index of the characters in the original string along with the block. Once the blocks are sorted, these indices are used to find indices for BWT using Equation 4.1. But our technique improves the overall throughput and is highly suitable for BWT computation of long strings. Some key innovations of our work are:

- Since the block size is equal to LCP and for most type of inputs, LCP is not long, these blocks provide an excellent compromise between saving/sorting complete cyclic rotations of string and doing multiple iterations of single character sorting operation.

- Since memory FIFOs, which are available in fair amount on modern FPGAs, are known to give high throughputs for merge based sorting operations (82), we use FIFO memory centric pipelined architecture to compute BWT for long strings at a high throughput.
- Since it is a single iteration, flow through approach for BWT computation, our technique gives a speed-up of over four when compared with the existing techniques using comparable hardware.

4.4 Proposed architecture and data-flow

The architecture is a pipeline of various components. These components include Suffix Block Generator, Parallel Suffix Sorter, flow through FIFO network and BWT index generator. Each component is described in detail below.

4.4.1 Suffix Block and Address Generator

Since the address of each character in the original string need to be stored along with suffices to find the BWT. The input string s is shifted in character by character and suffix of length k corresponding to each character is registered. Address generator generates the address corresponding to each block and saves it along with k characters. k such suffices along with addresses are fed to the parallel suffix sorter for sorting. Initial sorting of blocks is required before the FIFO based merge network since merge sorting is based on having sorted subsequences.

4.4.2 Parallel Suffix Sorter

The basic architecture of the parallel suffix sorter is similar to the parallel sorter explained in (80), the difference being the width of register stage and the number of comparators. In

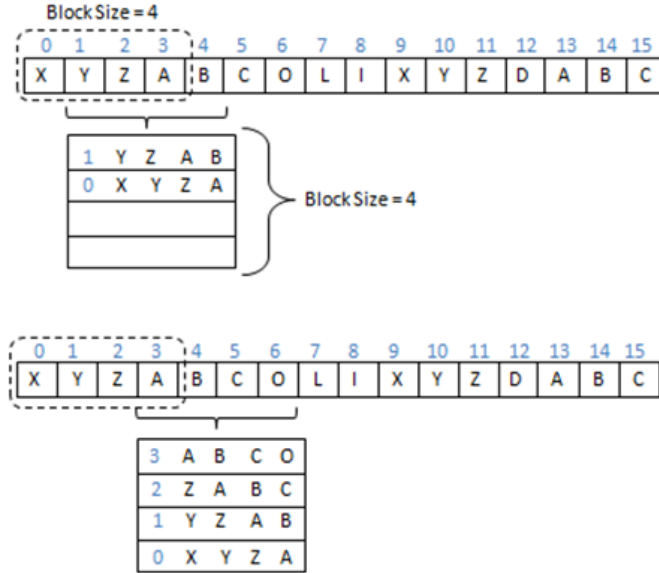


Figure 43. Limited sized suffices for a Single Block

parallel sorter, the register stage saves a single character along with the address but in case of parallel suffix sorter the whole suffix is saved. The comparison between contents of two registers is also made over the whole block instead of just a single character. In other words, the width of data-path is k . Although the number of comparators and the register sizes are increased, but since we use only a single parallel suffix sorter and k is much less than the length of string n , the overall hardware complexity of parallel suffix sorter is much less than the simple parallel sorter corresponding to string of length n .

4.4.3 FIFO Network

The next stage in pipeline is the network of First-in-First-Out (FIFO) memory. A number of FIFO based sorting techniques have been proposed (82; 83; 84; 85). The FIFO network has a number of stages depending on the length of string n and the number of characters in one block k . Each stage of network has a couple of FIFOs and a comparator. Two consecutive stages are connected together as shown in figure Figure 44. The first stage gets input from parallel sorter and the output of last stage goes to BWT index generator. At each stage, block from the output of each FIFO is compared and the one that is lexicographically smaller is fed to one of the FIFOs in the next stage. The FIFOs are filled using FIFO based merge sorting technique proposed in (82). The two FIFOs in each stage are filled alternately. Once the FIFO A of a particular stage d is filled, blocks from stage $d-1$ are directed towards FIFO B and at the same time blocks are read from both A and B. Blocks from A and B are compared, and the lexicographically smaller block is fed to the next stage $d+1$. Hence, read and write operations at each stage are parallelized with blocks being continuously read from previous stage and written to next stage. Figure Figure 45 depicts the data flow through the first couple of stages in the FIFO network.

4.4.4 BWT index generator

Once the blocks are sorted, index t for BWT is calculated by using index i saved as part of the each block using equation Equation 4.1.

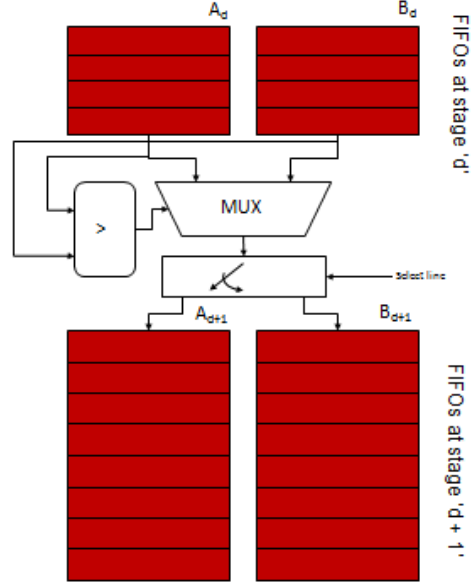


Figure 44. Connectivity of FIFOs between two stages

4.5 Complexity Analysis

4.5.1 Hardware Complexity Analysis

A major portion of the hardware cost is the FIFO memory blocks. There are two FIFOs at each stage d and the size of FIFO at each stage is $2^d \times k$. The total number of FIFO stages would be $\log_2(n/k)$. The total complexity for the FIFO part of hardware turns out to be $O(n)$. For the parallel sorter the hardware complexity would be $O(k^2)$ which would not contribute much to the overall hardware cost since $n \gg k$. The results of the analysis are verified by the experimental results explained in section 4.6. The hardware complexity of BWT computation

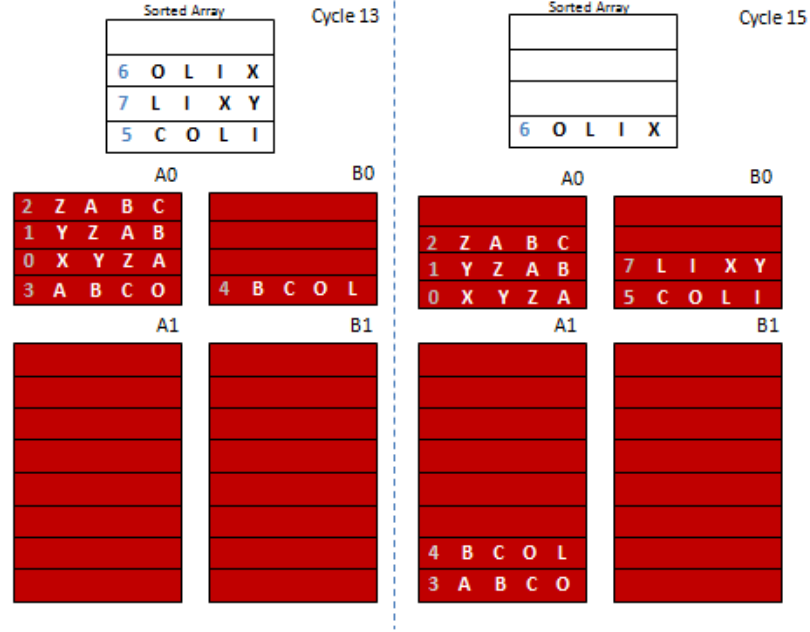


Figure 45. Dataflow through the FIFO pipeline

in (80) and (79) is also $O(n)$ in terms of the number of registers required. So from hardware complexity perspective our technique is as efficient as those reported in (80) and (79).

4.5.2 Time Complexity Analysis

Time complexity for finding BWT using our approach is also $O(n)$ since for each FIFO at stage d , half of the time for the stage is overlapped with the next FIFO stage $d + 1$. The time complexity for the case of (80) would be $O(n + |LCP|)$. For (79), the time complexity depends on the number of common prefixes as well and hence it is even worst compared to (80). However, it may be misleading just to consider the asymptotic time complexity as the hardware is implemented on FPGA and the final throughput not only depends on algorithmic

TABLE VII

COMPARISON OF HARDWARE COMPLEXITY BETWEEN PARALLEL SORTER IN
AND PARALLEL SUFFIX SORTER USED IN OUR PROPOSED ARCHITECTURE FOR

BWT	
Architecture	Complexity
Parallel Sorter for sorting string of length n (80)	$O(n)$
Parallel Suffix Sorter for sorting k blocks each of length k	$O(k^2)$

time complexity but also the hardware resources that are used in FPGAs. FIFOs are known to give high throughputs for large amount of data (82) and hence our flow through FIFO based architecture also gives high throughput for BWT computation.

4.6 Experimental Setup & Results

The design was synthesized for Stratix IV EP4SGX230KF-40C2 FPGA using Altera Quartus II 12.0. The Register Transfer Logic (RTL) code was parameterized in terms of length of string n and the block size k . The maximum clock frequency achieved was 131.34 MHz. The design was simulated using Modelsim Altera 10.0 d. The number of clock cycles required to compute the BWT depend on the number of stages in the FIFO network. Since at each clock cycle, a block of data is forwarded from one stage of FIFO network to the next, string contents have no effect on number of clock cycles. Only the length of string and block size affect the total

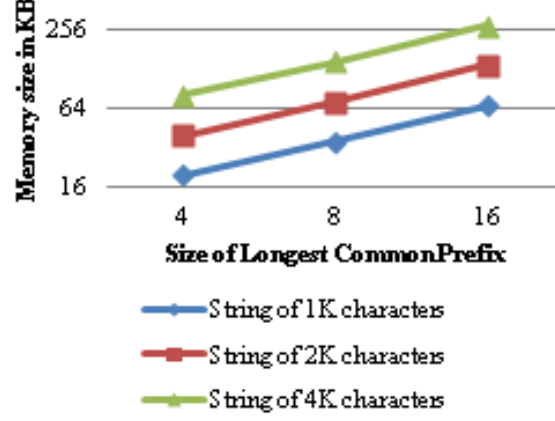


Figure 46. Effect of the size of Longest Common Prefix (LCP) on the total memory used on pipeline of FIFOs for different string lengths.

number of clock cycles. Hence, random strings of various lengths and block sizes, generated using (86), were used for our experiments. Throughput was calculated theoretically using the number of clock cycles.

To study the impact of LCP on hardware complexity, two parameters were recorded: memory blocks used and the number of registers used. Memory blocks are the direct indication of the amount of memory allocated to the FIFO pipeline. The number of registers gives an indication of the amount of hardware used to implement parallel suffix sorter, multiplexers, buffer registers and control state machines. It can be seen in figure Figure 46 that the FIFO memory increases linearly with the LCP size k which backs our analysis that the size of FIFOs vary linearly with block size (FIFO size at depth $d = 2^d \times k$. Total size would be sum over d but still

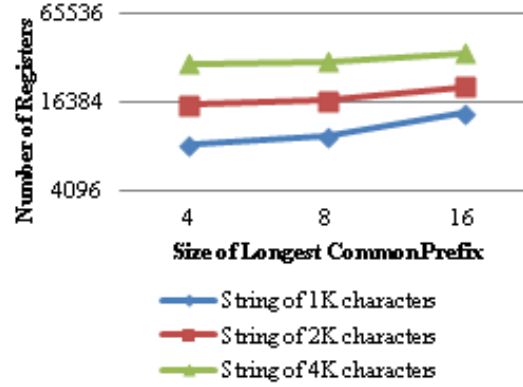


Figure 47. Effect of size of Longest Common Prefix (LCP) on the number of Register consumed

linear with respect to k). The number of registers in parallel suffix sorter scale quadratically with size of LCP. This is evident from figure Figure 47 for the number of registers corresponding to string of length of one thousand(K) characters. As the length of the string is increased, the effect becomes much less pronounced since for longer string k is very small compared to n . The impact of the length of string n on hardware complexity was also studied. Figure Figure 48 shows that for a given LCP size, memory size scales linearly with n as predicted by our analysis.

Throughput was also recorded against various LCP sizes keeping the length of string constant. As expected the through put is greater for longer LCP (at the expense of higher hardware complexity) since the number of FIFO stages ($= \log_2(n/k)$) is reduced for longer LCP, greater

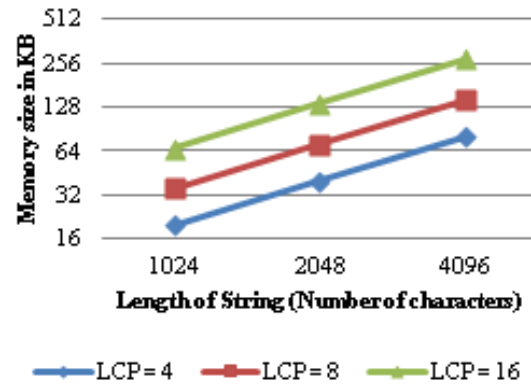


Figure 48. Effect of the length of String on the total memory used on pipeline of FIFOs for different different sizes of LCP

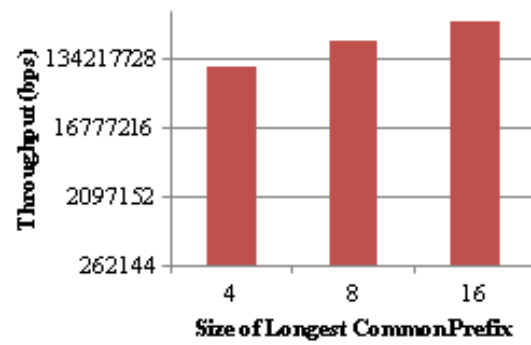


Figure 49. Effect of LCP on Throughput using a fixed string of size two thousand characters.

Longer LCP corresponds to more hardware complexity but higher throughput.

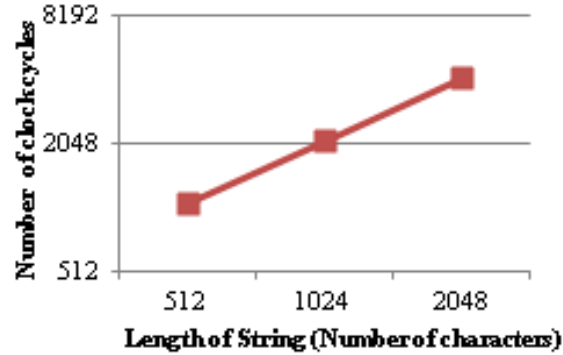


Figure 50. Performance (in terms of number of clock cycles) for various length of strings having fixed LCP = 16

LCP size also means that more suffixes of greater length are sorted in the parallel sorter and then fed to the FIFO network. As a confirmation to our analysis, Figure 50 depicts that the time required for computation of BWT scales linearly with the length of string.

In (80), Parallel Sorter technique is compared to Weavesorter approach (79) using a relatively small string size of 128 characters on a Virtex 2 xv2v2000. The throughput is reported in terms of clock cycles and time in mille-seconds (ms). For a fair comparison with our technique, we synthesize our design using a comparable Altera device: Stratix EP1S10B672C6 and use the same string size. We compare the throughput in terms of number of clock cycles required to compute BWT for a string of length 128 for the case where the length of LCP is 8. For parallel sorter and weavesorter based technique, this would result in 8 sorting iterations. With block

size equal to 8 in our technique, we get the following improvement over the two mentioned techniques.

Comparison in Table VIII suggests that our proposed technique achieves a speed up of 4.3 over the parallel sorter (77% reduction in clock cycles) and a speed up of 8.1 over the weavesorter approach (87.6% reduction in number of clock cycles) for a string size of 128 characters on a comparable device.

TABLE VIII

PERFORMANCE COMPARISON OF PROPOSED BWT ARCHITECTURE WITH
WEAVESORTER AND PARALLEL SORTER FOR A STRING OF SIZE 128 AND LCP = 8

Architecture	Device	Max Frequency	Number of Clock cycles	Time (μsec)
Weavesorter Based	Virtex xcv300	45 MHz	2304	51.2
Parallel Sorter Based	Virtex 2 xv2v2000	51.67 MHz	1231.5	23.8
Proposed Architecture	Stratix EP1S10B672C6	51.4 MHz	285	5.54

4.7 Summary

In this chapter, a novel high performance method for computing BWT on FPGAs was proposed. The trade offs for various design parameters such as hardware complexity and throughput were studied assuming different lengths of Longest Common Prefixes (LCP). In our implementations, both hardware complexity and throughput scale linearly with LCP. Over four times improvement in terms of clock cycles was achieved compared to the existing state of the art hardware techniques in the published literature using a comparable device. Maximum clock frequency of 131.34 MHz was achieved on Stratix IV EP4SGX230KF-40C2 FPGA which may be further improved by applying various optimizations at the code and placement level. Due to the dependence of block size on LCP of string, determination of block size for arbitrary data is an open problem and will be addressed in our future work. The performance will also be computed for standardized datasets used in various application domains e.g., Data compression and DNA string matching.

CHAPTER 5

AN FPGA BASED HIGHLY PIPELINED AND SCALABLE ARCHITECTURE FOR MEDIAN FILTERING

The outline of this work is published as a poster abstract in proceedings of 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA2015) (27)

5.1 Introduction

Median filtering (87) has been widely used for noise suppression in image processing. Several types of noise may be added to the image in the image capturing and transmission process. Some of the noise types include Gaussian noise, multiplicative noise, impulse noise etc. These added noises might hinder the accuracy of subsequent image processing steps. Impulse noise is usually divided into two main categories: random-valued-shot and salt-and-pepper noise. Median filter is a nonlinear filter that is used for the suppression of salt-and-pepper noise. The main principle of median filter is to consider a window of odd number of pixels around each pixel p , sort all the pixels in the window and take the median (middle) value as the output value for the pixel p . Median filter is a special case of rank order filter. The r^{th} rank order filter takes r^{th} value in a sorted sequence of pixels in the window around the pixel p . Image dilation and erosion operations (also called morphological filters) take the maximum and minimum value in the sorted sequence of pixels in the window respectively (88). Our implementation can be easily extended for rank order and morphological filters.

A median filter with a window of 3×3 or 5×5 pixels is sufficient if the noise intensity is lower than 10 - 20% (89). Employing large window size can have an adverse effect on image clarity and hence generally a window of size 3×3 is used (90). We refer to this window as median window and we select a size of 3×3 for the median window. For larger noise intensities multiple variants of median filters are proposed. Some of these variants include Adaptive Median Filters (91), weighted median filters (92), switching median filter (92) and weighted order statistics filters (93). All these variants have the basic median filtering technique at the core. Many hardware based techniques are proposed for median filters and its variants due to its high demand in real time image processing. An overview of some various techniques is given in 5.2.

Considering the growing size of images on displays and capturing devices, the performance demands for performing real-time operations like median filtering is increasing. Our work is part of the effort to improve the performance of real time median filtering of large images that do not necessarily fit in the FPGAs. The key features of the proposed design are described below.

- The proposed architecture is scalable to the image size and available hardware resources.

Depending on the available resources on the device, the performance can be scaled. Since reading pixels from the external memory effect performance, our technique minimizes the re-reading of pixels from external memory.

- Block memory in First-In-First-Out (FIFO) configuration have shown promise in high performance sorting applications (82),(26). Our proposed architecture is also based on the novel configuration of block memory to compute median filtering.
- The number of comparators required to compute median for a single window are reduced compared to the famous FPGA based sliding window technique proposed by (94).
- The proposed technique is highly pipelined which makes it an ideal fit for FPGAs.

5.2 Related Work

A number of hardware techniques have been proposed for the median filters. Some of these techniques are based on threshold decomposition (95), bit serial approach (96), histogram (97), sorting network (89); (88); (90) and insert-delete (98).

Threshold decomposition is based on representing an n bit number into $2^n - 1$ threshold values. The architecture allows high degree of parallelism but is poor at scalability. Bit-serial approach is based on determining equivalent bit level representation of pixel values that preserve the rank instead of representing them in threshold values. Insert delete approach is based on keeping an ordered list of data. The list is continuously updated as the new data is read continuously by comparing with the existing data in the ordered list and inserting at the proper location. The size of the ordered list is maintained by removing the old data every time the new data arrives.

Histogram based technique is based on principle similar to classical counting sort algorithm. There is a separate counter c_i corresponding to each possible pixel value v_i . For 8-bit resolution, the value of i ranges from 0 - 255 and hence there are 256 counters. These counters keep track

of the occurrence of pixel values in the window of interest. The values of counts are added in an ascending order of i . For a window of size $k \times k$, where k is an odd number greater than 1, the value of i for which the sum of counts becomes equal to $\lfloor k^2/2 \rfloor + 1$, represents the median value of the window. Hardware implementation of histogram based technique (97) needs 256 counters, 127 adders and 128 comparators for 8-bit resolution.

Sorting network based median filtering technique is one of the most widely used technique in hardware implementations. Sorting networks are based on a fixed structure of comparators and hence they are free from scenario based control dependencies. This renders them suitable for highly parallel processing architecture. Scalability is one important concern in design of sorting network based architecture. Bubble sort and odd-even transpose network are two network based sorting techniques. Bubble sort based technique (99) sorts $2k^2 + 1$ inputs using $k^2(2k^2 + 1)$ two input sorters and has area complexity of $O(k^4)$. Odd-even transposition network (100) needs k^2 compare and swap stages with each stage having $(k^2 - 1)/2$ compare and swap units for a $k \times k$ window size.

Our proposed architecture, *medianPipe*, is a sorting networking based technique that is based on merge sort. Since fetching the whole image from external memory might not be possible for big image sizes, we assume that the image is available in the form of slices. The size of image slice is chosen based on the resources available in device. Once an image slice is fetched from external memory, all the median values corresponding to the slice are computed. In order to compute the median values at the borders of slice, slices are assumed to be overlapping by

couple of pixel width. Since the size of the slice is much bigger than the median window, the number of pixels that had to be re-read are less.

The rest of the paper is organized as follows: The architecture and data flow for *medianPipe* is explained in section 5.3 followed by results and discussion in section 5.4. The paper is concluded in the final section 5.5.

5.3 Proposed Architecture and Data flow

The proposed architecture for *medianPipe* is based on the principle of merge sort. It consists of two stepped sorting process. The first step is to sort the pixels within each row of median window to get sorted rows. The second step is to merge these sorted rows to find the median. The overall block diagram of *medianPipe* is shown in figure 5.3.

5.3.1 Row Sorter

The row sorter receives single pixel per cycle. A single comparator is used to sort the pixels in three clock cycles as shown in figure 5.3.1. Using multiple comparators won't increase the overall performance because the bottleneck is the merger block.

5.3.2 Sorted-Row Buffers

The sorted rows are saved in block memory based FIFOs. These FIFO based buffers are essential for the merge process. Since each sorted row, except the two rows close to boundary, are part of three median windows, a separate row-buffer is used for the same sorted row corresponding to each median window.

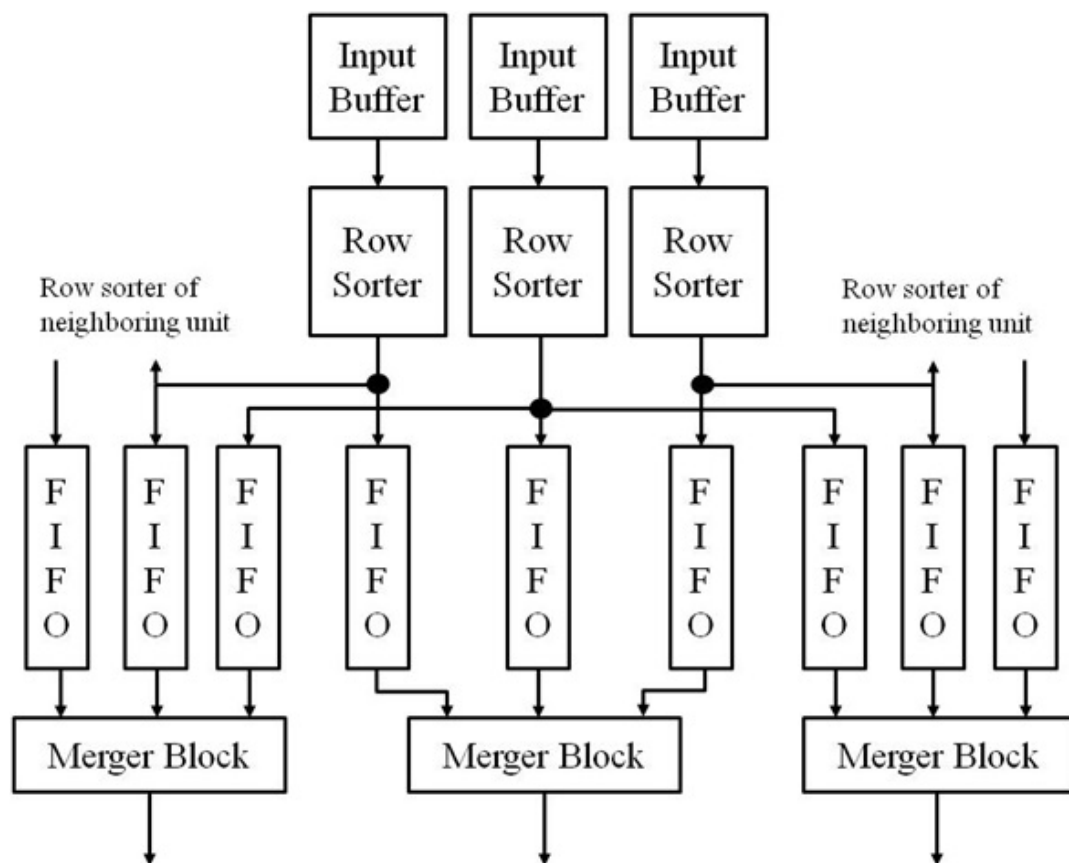


Figure 51. Block Diagram of three *medianPipes*

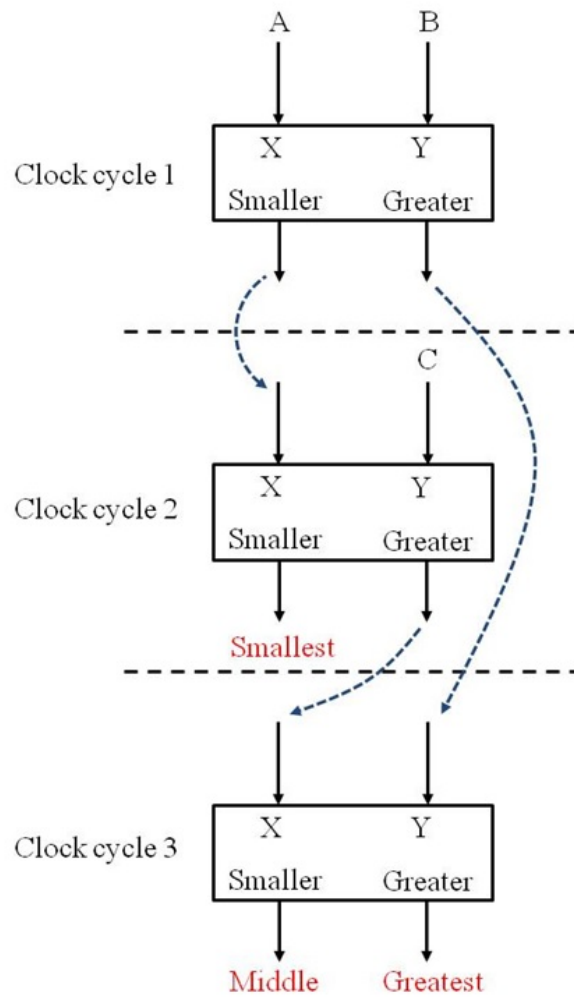


Figure 52. First stage sorting using a single comparator

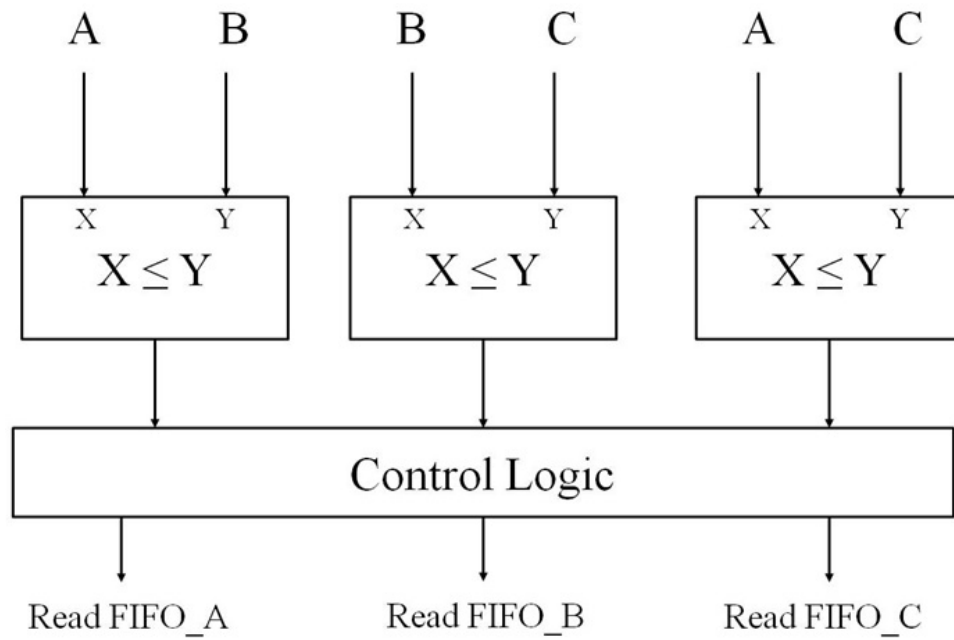


Figure 53. Merger Block using three Comparators

5.3.3 Merger block

The sorted sub rows are merged to find the median value. This is done using the merging block. It has three comparators and control logic to control the read operation of the sorted-row buffers as shown in figure 5.3.3.

Since each row of median window is part of two other neighboring median windows, multiple merger blocks are used to compute the medians corresponding neighboring windows.

5.3.4 Data-flow

The pixels are read from external memory in a burst mode. Without loss of generality, we assume that the pixels are read column wise. Each row sorter receives a single pixel per cycle for three consecutive clock cycles. These pixels are sorted by the row sorter and pushed into the sorted-row buffers for all three median windows the row belongs to. The merger block reads the pixels from the sorted-row buffers and median value is determined during the fifth cycle of merge operation.

The median values corresponding to all pixels in a column of image slice are computed in parallel as shown in figure Figure 54. After sorting all the rows corresponding to the median windows of column C_2 in parallel, a separate merger block per median value is used to compute the median values in parallel.

In order to achieve maximum parallelism, the computation of the median values in column C_3 of image slice is started a cycle after computation for C_2 since the pixels are assumed to be read in a column major form, with one column available every cycle of burst.

Multiple *medianPipes* may be used in parallel to compute median values in a single column. For consecutive columns, the median values are calculated in separate *medianPipes* as well. *MedianPipes* allocated to a column C_i may be reused after 8 clock cycles for computation of medians in $C_{(i+8)}$ since the median corresponding to C_i would have been computed till then.

5.4 Results and Discussion

MedianPipe is described using Verilog Hardware Description Language (HDL). Synthesis and fitting is done using Altera Quartus II 13.1 for Stratix IV EP4SGX230KFC2. The simu-

	C2	C3	C4	C5
(1,1) (1,2) (1,3)	(1,4)	(1,5)	(1,6)	
(2,1) (2,2) (2,3)	(2,4)	(2,5)	(2,6)	
(3,1) (3,2) (3,3)	(3,4)	(3,5)	(3,6)	
(4,1) (4,2) (4,3)	(4,4)	(4,5)	(4,6)	
(5,1) (5,2) (5,3)	(5,4)	(5,5)	(5,6)	
(6,1) (6,2) (6,3)	(6,4)	(6,5)	(6,6)	

Figure 54. Parallel computation of median values in column C_2 using two *medianPipes*. The ordered pairs (R, C) represent a pixel, where R and C represents row and column respectively.

Rows $R_1 - R_6$ are sorted in parallel. Merging sorted rows at R_1 , R_2 and R_3 compute the median value at $(2, 2)$. R_2 , R_3 and R_4 compute median at $(3, 2)$

	C2	C3	C4	C5	
(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)

Figure 55. Computation for median values across the columns. Since a single column of image slice is read from the external memory, if the computation of median values for C_2 starts at i^{th} cycle, the computation for C_3 will start at $(i + 1)^{th}$, C_4 at $(i + 2)^{th}$ and so on. The *medianPipes* dedicated to the columns are re-used once the median values for the columns are computed

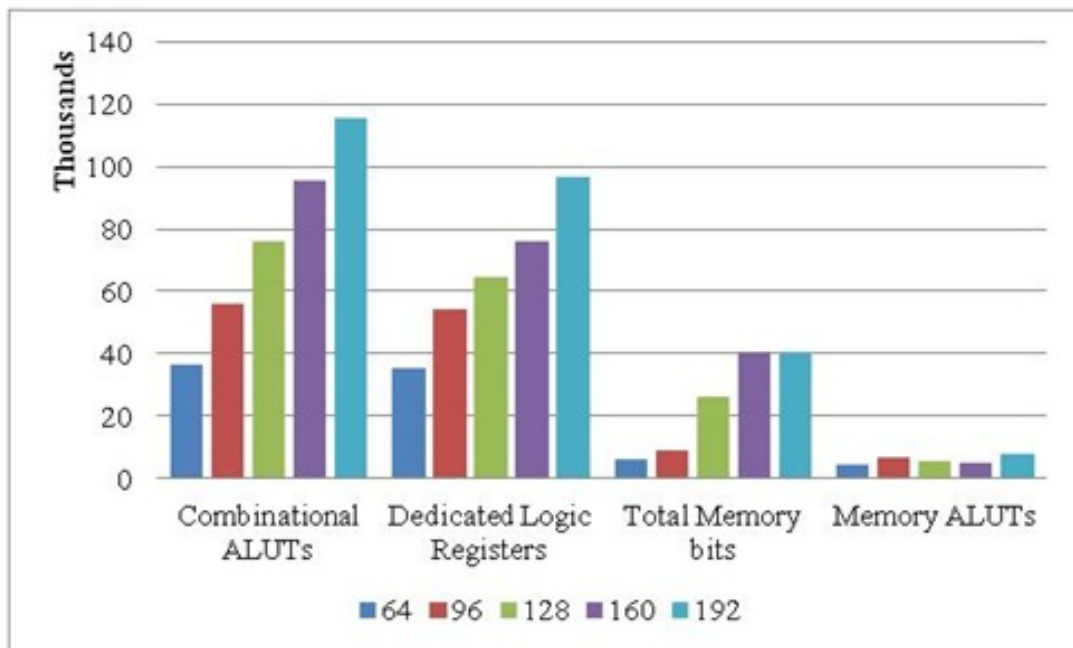


Figure 56. Resource usage for various number of *medianPipes*

lations were run on Modelsim Altera 10.0 d. The design is parameterized for pixel resolution and image slice size. The hardware complexity of the design is specified in terms of number of block memory bits and Adapted Look-Up Tables (ALUT). Throughput is calculated by computing clock frequency and number of clock cycles corresponding to the number of medianPipes employed. Clock frequency is determined using Altera TimeQuest Timing analysis tool and number of clock cycles using Modelsim. Throughput or pixel rate is reported in terms of Pixels per second.

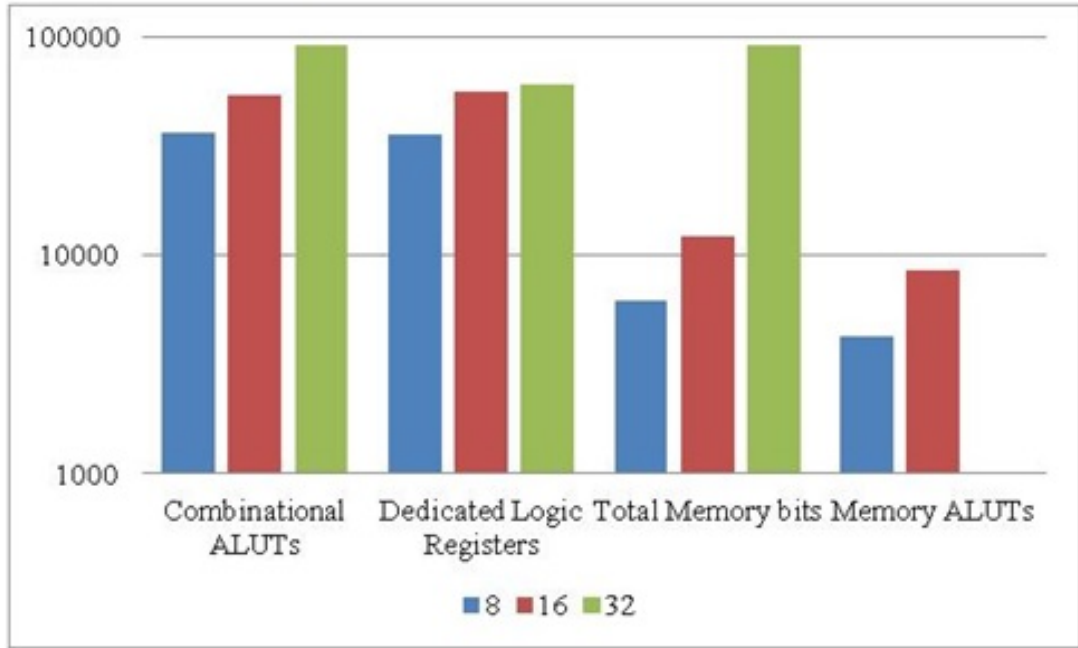


Figure 57. Resource usage for pixel sizes of 8, 16 and 32 bits

We study the scaling of hardware resources with the number of *medianPipes*. It can be seen from Figure 56 that all hardware resources scale linearly with the number of *medianPipes* used. Hence depending on the resources available on device, more *medianPipes* and hence bigger image slice can be targeted. Hardware resources for various pixel sizes were also investigated and again the resources scaled linearly with pixel size. Going from 16 to 32 bit pixel, the number of block memory bits did not scale as for 32-bit pixel, the fitting operation did not use any memory ALUTs.

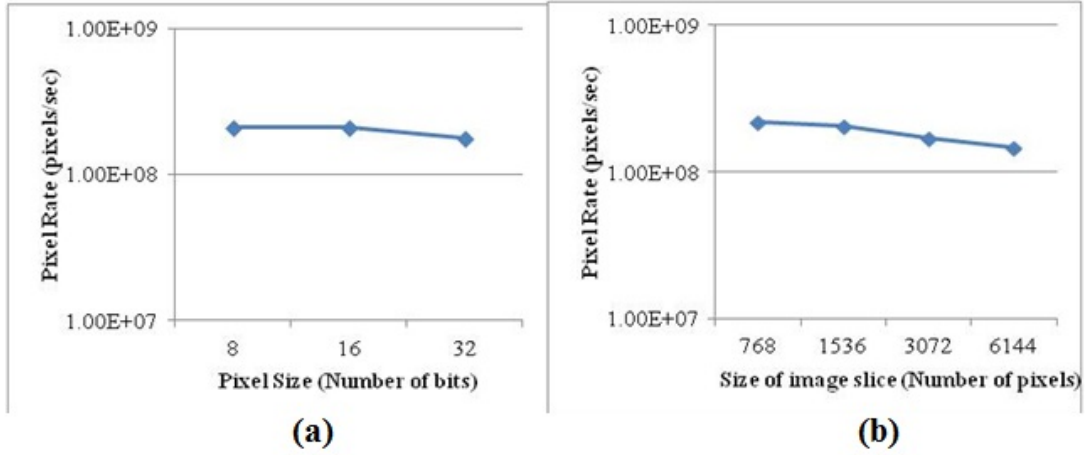


Figure 58. (a) Trend in pixel rate with respect to pixel size using 128 median pipes for 768 pixels (b) Trend in pixel rate with respect to image slice size (using 8-bit pixel)

Pixel rate is studied by varying various parameters in the design space. Figure 58 shows the pixel rates for various image slice sizes. The pixel rate drops for bigger image slice size if the number of *medianPipes* is kept constant. The pixel rate was also studied with respect to the number of *medianPipes*.

5.5 Conclusion and Future Work

A highly pipelined FPGA based architecture for median filtering is proposed. The proposed technique scales linearly with hardware resources and pixel size. Pixel rates for various sizes of image slices and pixel sizes give pixel rate higher than 124 MHz that is the standard for 1080p High-Definition. As part of the future work, the pixel rate as well as the hardware complexity could be improved. Applying different configuration of *medianPipes*, bigger image slices could

TABLE IX

THROUGHPUT IN TERMS OF IMAGE SLICE RATE FOR VARIOUS NUMBER OF

MEDIANPIPES

Number of <i>MedianPipes</i>	Size of image slice (pixels)	Slice Rate Kilo Slices per sec
64	1536	113
96	2304	75.2
128	3072	55.8
160	3840	43.4
192	4608	35.3

be targeted using the same number of *medianPipes* by trading the pixel rate. The architecture could be applied to video streaming applications.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

High-performance solutions for all four applications discussed in the previous chapters establish the importance of heterogeneous solutions in improving the performance and energy-efficiency of memory, data and compute intense applications. Considering the performance and power limitations of high-end processors, heterogeneous solutions based on computing platforms like GPUs and FPGAs are being used to keep up with the increasing performance and power-efficiency demands. Microsoft's Catapult project (16) is an example of FPGA-based heterogeneous system that achieved two times speed up for image and text searches in Bing search. Looking at the ever-growing market and a variety of applications for the embedded systems, like cellular devices, the number of heterogeneous SoCs-based solutions is continuously going up. Qualcomm's Snapdragon (10) is one such example that has dedicated processing units for applications such as sensor interfacing, video processing and computational photography.

The demand for heterogeneous computing systems is going to increase with the advent of large-scale IoT deployment. The idea of IoT is to have enormous number of nodes connected to internet that are communicating with each other as well as with the cloud data-center. These nodes could be any device, sensor or actuator that is connected to the internet. Having this unprecedented number of devices on the network, real-time computation and security requirements will demand high performance as well as low-energy computing devices at the network level. At the cloud data-center level, there will be enormous increase in the amount

of data-processing for sensing, analysis and visualizing of this Big-data. At the node level, exploring the design space for CPU-FPGA-GPU SoC based hybrid platforms is required. As many of the devices connected to internet are battery operated, providing high-performance at affordable energy budget would require innovative heterogeneous computing architecture solutions.

This thesis explored the design space for FPGA-based architectures targeting a variety of memory and compute intense applications. In a broader perspective, this design space exploration could be extended to include other computing platforms in the design space and finding the best solution for a particular application on heterogeneous computing system. The proposed solutions have applications in multiple domains. Future directions for individual proposed solutions are given below.

BWT architecture described in chapter 4 could be extended to target compressed string matching that has applications in large scale string processing applications like Big-data analytics in data-centers, Deep Packet Inspection in cyber network security and genome alignments problems in Bio-informatics.

Matrix inversion architecture described in chapter 3 could be extended to target problems like real-time cryptography, Multiple-Input-Multiple-Output Orthogonal Frequency Division Multiplexing (MIMO-OFDM) systems and compressive sensing based signal reconstruction.

Re-gridding architecture described in chapter 2 could be extended to target image reconstruction in Radar systems. High-performance Image reconstruction applications, like MRI,

are also a straight forward extension of the proposed architecture. The proposed architecture targets 2-dimensional trajectories. The solution could also be extended for 3D MRI trajectories.

Median Filtering architecture proposed in chapter 5 could be extended for numerous variants of Median filter. The technique for finding median could be applied to other applications that require median filtering, for example, Web-mining.

The thesis establishes that high-performance embedded solutions are the way forward to meeting the ever increasing demands of energy-efficient computing. Considering the future of big-data processing, Network-security, and IoTs, the importance of these specialized solutions in heterogeneous computing systems is going to increase tremendously and hence more research efforts should be channeled in developing efficient embedded solutions as part of heterogeneous computing architectures.

CITED LITERATURE

1. Increased Complexity in Embedded Software Systems, 2014. Available at: <http://www.ni.com/white-paper/52165/en/>.
2. EPFL, Switzerland: MRI Simulation and Reconstruction - Matlab Framework for MRI Simulation and Reconstruction, 2015. Available at: <http://bigwww.epfl.ch/algorithms/mri-reconstruction/>.
3. Carrara, W. G.: Spotlight Synthetic Aperture Radar: Signal Processing Algorithms. Artech, 1995.
4. Kestur, S., Park, S., Irick, K., and Narayanan, V.: Accelerating the nonuniform fast fourier transform using fpgas. In Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on, pages 19–26, 2010.
5. Kestur, S., Irick, K., Park, S., Al Maashri, A., Narayanan, V., and Chakrabarti, C.: An algorithm-architecture co-design framework for gridding reconstruction using fpgas. In Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE, pages 585–590, 2011.
6. Sorensen, T., Schaeffter, T., Noe, K., and Hansen, M.: Accelerating the nonequispaced fast fourier transform on commodity graphics hardware. Medical Imaging, IEEE Transactions on, 27(4):538–547, 2008.
7. Duarte, R., Neto, H., and Véstias, M.: Double-precision gauss-jordan algorithm with partial pivoting on fpgas. In Digital System Design, Architectures, Methods and Tools, 2009. DSD'09. 12th Euromicro Conference on, 2009.
8. de Matos, G. and Neto, H.: Memory optimized architecture for efficient gauss-jordan matrix inversion. In Programmable Logic, 2007. SPL'07. 2007 3rd Southern Conference on, pages 33–38. IEEE, 2007.
9. Amor, M., Doallo, R., Fraguera, B. B., Herrero, J. R., Quintana-Ortí, E. S., and Strzodka, R.: Graphics processing unit computing and exploitation of hardware accelerators. Concurrency and Computation: Practice and Experience, 25(8):1104–1106, 2013.

10. Qualcomm Snapdragon, 2015. Available at: <https://www.qualcomm.com/products/snapdragon>.
11. Hennessy, J. L. and Patterson, D. A.: Computer architecture: a quantitative approach. Elsevier, 2011.
12. Fernandez, E. B. C.: Hardware Implementation of a String Matching Algorithm Based on the FM-Index, 2013.
13. Buell, D., El-Ghazawi, T., Gaj, K., and Kindratenko, V.: High-performance reconfigurable computing. COMPUTER-IEEE COMPUTER SOCIETY-, 40(3):23, 2007.
14. OpenFPGA. Available at: <http://www.openfpga.org/>.
15. Center for High-Performance Reconfigurable Computing. Available at: <http://chrec.ufl.edu/>.
16. Putnam, A., Caulfield, A., Chung, E., Chiou, D., Constantinides, K., Demme, J., Esmailzadeh, H., Fowers, J., Gopal, G. P., Gray, J., Haselman, M., Hauck, S., Heil, S., Hormati, A., Kim, J.-Y., Lanka, S., Larus, J., Peterson, E., Pope, S., Smith, A., Thong, J., Xiao, P. Y., and Burger, D.: A reconfigurable fabric for accelerating large-scale datacenter services. In 41st Annual International Symposium on Computer Architecture (ISCA), June 2014.
17. Why hyperscalers and clouds are pushing intel into FPGAs?, 2015. Available at: <http://www.nextplatform.com/2015/07/29/why-hyperscalers-and-clouds-are-pushing-intel-into-fpgas/>.
18. Thinh, T. N., Hieu, T. T., Dung, V. Q., and Kittitornkun, S.: A fpga-based deep packet inspection engine for network intrusion detection system. In Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2012 9th International Conference on, pages 1–4. IEEE, 2012.
19. Tessier, R. and Burleson, W.: Reconfigurable computing for digital signal processing: A survey. Journal of VLSI signal processing systems for signal, image and video technology, 28(1-2):7–27, 2001.
20. Altera: Floating-Point Mega Functions, 2013. Available at: <http://www.altera.com/literature/ug/>.

21. Ramdas, T. and Egan, G.: A survey of fpgas for acceleration of high performance computing and their application to computational molecular biology. In TENCON 2005 2005 IEEE Region 10, pages 1–6. IEEE, 2005.
22. García, G. J., Jara, C. A., Pomares, J., Alabdo, A., Poggi, L. M., and Torres, F.: A survey on fpga-based sensor systems: Towards intelligent and reconfigurable low-power sensors for computer vision, control and signal processing. Sensors, 14(4):6247–6278, 2014.
23. Cheema, U. I., Nash, G., Ansari, R., and Khokhar, A. A.: Memory optimized re-gridding for non-uniform fast fourier transform on fpgas. In Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22th Annual International Symposium on, 2014.
24. Cheema, U., Nash, G., Ansari, R., Khokhar, A., et al.: Power-efficient re-gridding architecture for accelerating non-uniform fast fourier transform. In Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, pages 1–6. IEEE, 2014.
25. Cheema, U. I., Nash, G., Ansari, R., and Khokhar, A. A.: Invarch: A hardware efficient architecture for matrix inversion. In Computer Design (ICCD), 2015 33rd IEEE International Conference on, pages 180–187, Oct 2015.
26. Cheema, U. I. and Khokhar, A. A.: Department of electrical and computer engineering university of illinois at chicago chicago, usa. In Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on, pages 1–6. IEEE, 2013.
27. Cheema, U. I., Nash, G., Ansari, R., and Khokhar, A. A.: Medianpipes: An fpga based highly pipelined and scalable technique for median filtering. In 23rd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2015.
28. Cheema, U. I., Nash, G., Ansari, R., and Khokhar, A. A.: Power efficient re-gridding architecture for accelerating non-uniform fast fourier transform. In submitted to Field-Programmable Logic (FPL), 2014 IEEE 24th Annual International Symposium on, 2014.
29. Kajbaf, H., Case, J., Zheng, Y., Kharkovsky, S., and Zoughi, R.: Quantitative and qualitative comparison of sar images from incomplete measurements using compressed sensing and nonuniform fft. In Radar Conference (RADAR), 2011 IEEE, pages 592–596, 2011.

30. O'Connor, Y. and Fessler, J.: Fourier-based forward and back-projectors in iterative fan-beam tomographic image reconstruction. Medical Imaging, IEEE Transactions on, 25(5):582–589, 2006.
31. Kalamkar, D., Trzasko, J., Sridharan, S., Smelyanskiy, M., Kim, D., Manduca, A., Shu, Y., Bernstein, M., Kaul, B., and Dubey, P.: High performance non-uniform fft on modern x86-based multi-core systems. In Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, pages 449–460, 2012.
32. Potts, D., Steidl, G., and Tasche, M.: Fast fourier transforms for nonequispaced data: A tutorial. In Modern sampling theory, pages 247–270. Springer, 2001.
33. Zhang, Y., Kandemir, M., Pitsianis, N. P., and Sun, X.: Exploring parallelization strategies for nufft data translation. In Proceedings of the Seventh ACM International Conference on Embedded Software, EMSOFT '09, pages 187–196, New York, NY, USA, 2009. ACM.
34. Feng, L., Grimm, R., Block, K. T., Chandarana, H., Kim, S., Xu, J., Axel, L., Sodickson, D. K., and Otazo, R.: Golden-angle radial sparse parallel mri: Combination of compressed sensing, parallel imaging, and golden-angle radial sampling for fast and flexible dynamic volumetric mri. Magnetic resonance in medicine, 72(3):707–717, 2014.
35. Candès, E. J. and Donoho, D. L.: Ridgelets: A key to higher-dimensional intermittency? Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 357(1760):2495–2509, 1999.
36. Delaney, A. H. and Bresler, Y.: A fast and accurate fourier algorithm for iterative parallel-beam tomography. Image Processing, IEEE Transactions on, 5(5):740–753, 1996.
37. Buonincontri, G., Methner, C., Krieg, T., Carpenter, T. A., and Sawiak, S. J.: Trajectory correction for free-breathing radial cine mri. Magnetic resonance imaging, 32(7):961–964, 2014.
38. Xie, J., Lai, P., Huang, F., Li, Y., and Li, D.: Cardiac magnetic resonance imaging using radial k-space sampling and self-calibrated partial parallel reconstruction. Magnetic resonance imaging, 28(4):495–506, 2010.
39. Kim, D.-h., Adalsteinsson, E., and Spielman, D. M.: Simple analytic variable density spiral design. Magnetic resonance in medicine, 50(1):214–219, 2003.

40. Meyer, C. H., Hu, B. S., Nishimura, D. G., and Macovski, A.: Fast spiral coronary artery imaging. Magnetic Resonance in Medicine, 28(2):202–213, 1992.
41. Liang, Z.-P. and Lauterbur, P. C.: Principles of magnetic resonance imaging. SPIE Optical Engineering Press, 2000.
42. Sadi, F., Akin, B., Popovici, D. T., Hoe, J. C., Pileggi, L., and Franchetti, F.: Algorithm/hardware co-optimized sar image reconstruction with 3d-stacked logic in memory. In High Performance Extreme Computing Conference (HPEC), 2014 IEEE, pages 1–6. IEEE, 2014.
43. Fessler, J. and Sutton, B.: Nonuniform fast fourier transforms using min-max interpolation. Signal Processing, IEEE Transactions on, 51(2):560–574, 2003.
44. Schomberg, H. and Timmer, J.: The gridding method for image reconstruction by fourier transformation. Medical Imaging, IEEE Transactions on, 14(3):596–607, 1995.
45. Dutt, A. and Rokhlin, V.: Fast fourier transforms for nonequispaced data. SIAM J. Sci. Comput., 14(6):1368–1393, November 1993.
46. Zhang, Y., Liu, J., Kultursay, E., Kandemir, M., Pitsianis, N., and Sun, X.: Scalable parallelization strategies to accelerate nufft data translation on multicores. In Euro-Par 2010 - Parallel Processing, eds. P. DAmbr, M. Guarracino, and D. Talia, volume 6272 of Lecture Notes in Computer Science, pages 125–136. Springer Berlin Heidelberg, 2010.
47. Akin, B., Milder, P., Franchetti, F., Hoe, J. C., et al.: Memory bandwidth efficient two-dimensional fast fourier transform algorithm and implementation for large problem sizes. In Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on, pages 188–191. IEEE, 2012.
48. Akin, B., Franchetti, F., and Hoe, J.: Understanding the design space of dram-optimized hardware fft accelerators. In Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on, pages 248–255, June 2014.
49. Nash, G. T., Cheema, U. I., Ansari, R., and Khokhar, A. A.: Power-efficient rma sar imaging using pipelined non-uniform fast fourier transform. In Radar Conference (RadarCon), 2015 IEEE, pages 1600–1604. IEEE, 2015.

50. Huang, T.-Y., Tang, Y.-W., and Ju, S.-Y.: Accelerating image registration of mri by gpu-based parallel computation. Magnetic resonance imaging, 29(5):712–716, 2011.
51. Ieee standard for floating-point arithmetic. IEEE Std 754-2008, pages 1–70, 2008.
52. Altera: Avalon Interface Specifications, May 2013. Available at: www.altera.com/literature/manual/.
53. Altera: PowerPlay Power Analysis, 2013. Available at: <http://www.altera.com/literature/hb/qts/>.
54. Altera: Simulating Memory IP, 2013. Available at: <http://www.altera.com/literature/hb/external-memory/>.
55. Khronos Group: The open standard for parallel programming of heterogeneous systems. Available at: www.khronos.org/opencv/.
56. Altera: Altera SDK for OpenCL, 2015. Available at: <https://www.altera.com/products/design-software/embedded-software-developers/opencv/overview.tablet.html>.
57. Terasic: DE5-Net FPGA Development Kit. Available at: de5-net.terasic.com/.
58. Altera: PowerPlay Power Analyzer Support Resources, 2015. Available at: <https://www.altera.com/support/support-resources/operation-and-testing/power/sof-qts-power.tablet.html>.
59. Wang, D. and Ali, M.: Synthetic aperture radar on low power multi-core digital signal processor. In High Performance Extreme Computing (HPEC), 2012 IEEE Conference on, pages 1–6. IEEE, 2012.
60. Rabah, H., Amira, A., Mohanty, B. K., Almaadeed, S., and Meher, P. K.: Fpga implementation of orthogonal matching pursuit for compressive sensing reconstruction. 2014.
61. Moussa, S., Abdel Razik, A. M., Dahmane, A. O., and Hamam, H.: Fpga implementation of floating-point complex matrix inversion based on gauss-jordan elimination. In Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on, pages 1–4. IEEE, 2013.

- 62. Zirra, P. and Wajiga, G.: Cryptographic algorithm using matrix inversion as data protection. Journal of Information & Communication Technology, 10, 2011.
- 63. Altera: Accelerating Design Development with Hard Floating-Point DSP Blocks in FPGAs, December 2014. Available at: <http://www.altera.com/>.
- 64. de Matos, G. M. and Neto, H. C.: On reconfigurable architectures for efficient matrix inversion. In Field Programmable Logic and Applications, 2006. FPL'06. International Conference on, pages 1–6. IEEE, 2006.
- 65. LogiCORE IP Floating-Point Operator,.
- 66. Arias-Garcia, J., Jacobi, R. P., Llanos, C. H., and Ayala-Rincon, M.: A suitable fpga implementation of floating-point matrix inversion based on gauss-jordan elimination. In Programmable Logic (SPL), 2011 VII Southern Conference on, pages 263–268. IEEE, 2011.
- 67. Rosado, A., Iakymchuk, T., Bataller, M., and Wegrzyn, M.: Hardware-efficient matrix inversion algorithm for complex adaptive systems. In Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on, pages 41–44. IEEE, 2012.
- 68. Auras, D., Leupers, R., and Ascheid, G.: Efficient vlsi architectures for matrix inversion in soft-input soft-output mmse mimo detectors. In Circuits and Systems (ISCAS), 2014 IEEE International Symposium on, pages 1018–1021. IEEE, 2014.
- 69. Burg, A., Haene, S., Perels, D., Luethi, P., Felber, N., and Fichtner, W.: Algorithm and vlsi architecture for linear mmse detection in mimo-ofdm systems. In ISCAS, 2006.
- 70. Burrows, M. and Wheeler, D. J.: A block-sorting lossless data compression algorithm. 1994.
- 71. Manber, U. and Myers, G.: Suffix arrays: a new method for on-line string searches. siam Journal on Computing, 22(5):935–948, 1993.
- 72. Ferragina, P. and Manzini, G.: Opportunistic data structures with applications. In Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on, pages 390–398. IEEE, 2000.

73. Adjero, D., Zhang, Y., Mukherjee, A., Powell, M., and Bell, T.: Dna sequence compression using the burrows-wheeler transform. In Bioinformatics Conference, 2002. Proceedings. IEEE Computer Society, pages 303–313. IEEE, 2002.
74. Lucito, R., Healy, J., Alexander, J., Reiner, A., Esposito, D., Chi, M., Rodgers, L., Brady, A., Sebat, J., Troge, J., et al.: Representational oligonucleotide microarray analysis: a high-resolution method to detect genome copy number variation. Genome research, 13(10):2291–2305, 2003.
75. Baik, H., Ha, D. S., Yook, H.-G., Shin, S.-C., and Park, M.-S.: Selective application of burrows-wheeler transformation for enhancement of jpeg entropy coding. In International Conference on Information, Communications & Signal Processing, 1999.
76. Adjero, D., Kandaswamy, U., Zhang, N., Mukherjee, A., Brown, M., and Bell, T.: Bwt-based efficient shape matching. In Proceedings of the 2007 ACM symposium on Applied computing, pages 1079–1085. ACM, 2007.
77. Yamaguchi, T. J., Ha, D. S., Ishida, M., and Ohmi, T.: A method for compressing test data based on burrows-wheeler transformation. Computers, IEEE Transactions on, 51(5):486–497, 2002.
78. Adjero, D., Bell, T., and Mukherjee, A.: The Burrows-Wheeler Transform:: Data Compression, Suffix Arrays, and Pattern Matching. Springer, 2008.
79. Mukherjee, A., Motgi, N., Becker, J., Friebe, A., Habermann, C., and Glesner, M.: Prototyping of efficient hardware algorithms for data compression in future communication systems. In Rapid System Prototyping, 12th International Workshop on, 2001., pages 58–63. IEEE, 2001.
80. Martinez, J., Cumplido, R., and Feregrino, C.: An fpga-based parallel sorting architecture for the burrows wheeler transform. In Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference on, pages 7–pp. IEEE, 2005.
81. Schindler, M.: A fast block-sorting algorithm for lossless data compression. In Proceedings of the Conference on Data Compression, volume 469. Citeseer, 1997.
82. Koch, D. and Torresen, J.: Fpgasort: a high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sort-

- ing. In Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays, pages 45–54. ACM, 2011.
83. Mueller, R., Teubner, J., and Alonso, G.: Sorting networks on fpgas. The VLDB JournalThe International Journal on Very Large Data Bases, 21(1):1–23, 2012.
 84. Marcelino, R., Neto, H. C., and Cardoso, J. M.: Unbalanced fifo sorting for fpga-based systems. In Electronics, Circuits, and Systems, 2009. ICECS 2009. 16th IEEE International Conference on, pages 431–434. IEEE, 2009.
 85. Marcelino, R., Neto, H., and Cardoso, J. M.: Sorting units for fpga-based embedded systems. In Distributed Embedded Systems: Design, Middleware and Resources, pages 11–22. Springer, 2008.
 86. Random. Org., 2013. Available at: <http://www.random.org/strings/>.
 87. J.W., T.: Nonlinear methods for smoothing data. EASCON, 1974.
 88. Meena, S. and Linganagouda, K.: Rank based merge sorting network architecture for 2d median and morphological filters. In Advance Computing Conference, 2009. IACC 2009. IEEE International, pages 473–479, March 2009.
 89. Vasicek, Z. and Sekanina, L.: Novel hardware implementation of adaptive median filters. In Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop on, pages 1–6. IEEE, 2008.
 90. Sanny, A. and Prasanna, V.: Energy-efficient median filter on fpga. In Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on, pages 1–8, Dec 2013.
 91. Hwang, H. and Haddad, R.: Adaptive median filters: new algorithms and results. Image Processing, IEEE Transactions on, 4(4):499–502, 1995.
 92. Brownrigg, D.: The weighted median filter. Communications of the ACM, 27(8):807–818, 1984.
 93. Marshall, S.: New direct design method for weighted order statistic filters. IEE Proceedings-Vision, Image and Signal Processing, 151(1):1–8, 2004.

94. Bates, G. L. and Nooshabadi, S.: Fpga implementation of a median filter. In TENCON'97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications., Proceedings of IEEE, volume 2, pages 437–440. IEEE, 1997.
95. Chang, L.-W. and Yu, S.-S.: A new implementation of generalized order statistic filter by threshold decomposition. Signal Processing, IEEE Transactions on, 40(12):3062–3066, 1992.
96. Kar, B. K. and Pradhan, D. K.: A new algorithm for order statistic and sorting. IEEE transactions on signal processing, 41(8):2688–2694, 1993.
97. Fahmy, S., Cheung, P. Y. K., and Luk, W.: Novel fpga-based implementation of median and weighted median filters for image processing. In Field Programmable Logic and Applications, 2005. International Conference on, pages 142–147, Aug 2005.
98. Huang, T., Yang, G., and Tang, G.: A fast two-dimensional median filtering algorithm. Acoustics, Speech and Signal Processing, IEEE Transactions on, 27(1):13–18, 1979.
99. Benkrid, K., Crookes, D., and Benkrid, A.: Design and implementation of a novel algorithm for general purpose median filtering on fpgas. In Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on, volume 4, pages IV–425. IEEE, 2002.
100. Knuth, D. E.: "Sorting and Searching", The art of Computer Programming., Edison-Wesley Publishing Company.
101. Arming, S., Fenkhuber, R., and Handl, T.: Data compression in hardware the burrows-wheeler approach. In Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on, pages 60–65. IEEE, 2010.
102. Akin, B., Milder, P., Franchetti, F., and Hoe, J.: Memory bandwidth efficient two-dimensional fast fourier transform algorithm and implementation for large problem sizes. In Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on, pages 188–191, 2012.

103. Zhao, G., Li, S., Zhang, Z., and Liu, B.: Nufft-based near-field imaging method for target scattering diagnostics. In Microwave Technology Computational Electromagnetics (ICMTCE), 2011 IEEE International Conference on, pages 104–106, 2011.
104. Jacob, M.: Optimized least-square nonuniform fast fourier transform. Signal Processing, IEEE Transactions on, 57(6):2165–2177, 2009.
105. Altera: Introduction to UniPHY IP, 2013. Available at: <http://www.altera.com/literature/hb/external-memory/>.
106. Perreault, S. and Hebert, P.: Median filtering in constant time. Image Processing, IEEE Transactions on, 16(9):2389–2394, Sept 2007.
107. Alekseychuk, A.: Hierarchical recursive running median. In Image Processing (ICIP), 2012 19th IEEE International Conference on, pages 109–112, Sept 2012.
108. Perreault, S. and Hébert, P.: Median filtering in constant time. Image Processing, IEEE Transactions on, 16(9):2389–2394, 2007.
109. Weiss, B.: Fast median and bilateral filtering. In ACM Transactions on Graphics (TOG), volume 25, pages 519–526. ACM, 2006.
110. Candès, E. J. et al.: Compressive sampling. In Proceedings of the international congress of mathematicians, volume 3, pages 1433–1452. Madrid, Spain, 2006.
111. Baraniuk, R.: Compressive sensing. IEEE signal processing magazine, 24(4), 2007.
112. Maechler, P., Studer, C., Bellasi, D. E., Maleki, A., Burg, A., Felber, N., Kaeslin, H., and Baraniuk, R. G.: Vlsi design of approximate message passing for signal restoration and compressive sensing. Emerging and Selected Topics in Circuits and Systems, IEEE Journal on, 2(3):579–590, 2012.
113. Septimus, A. and Steinberg, R.: Compressive sampling hardware reconstruction. In Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, pages 3316–3319. IEEE, 2010.
114. Mallat, S. G. and Zhang, Z.: Matching pursuits with time-frequency dictionaries. Signal Processing, IEEE Transactions on, 41(12):3397–3415, 1993.

115. Donoho, D. L., Maleki, A., and Montanari, A.: Message-passing algorithms for compressed sensing. Proceedings of the National Academy of Sciences, 106(45):18914–18919, 2009.
116. Stanislaus, J. L. and Mohsenin, T.: High performance compressive sensing reconstruction hardware with qrd process. In Circuits and Systems (ISCAS), 2012 IEEE International Symposium on, pages 29–32. IEEE, 2012.
117. Ren, F., Dorrace, R., Xu, W., and Markovic, D.: A single-precision compressive sensing signal reconstruction engine on fpgas. In Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on, pages 1–4. IEEE, 2013.
118. Bai, L., Maechler, P., Muehlberghuber, M., and Kaeslin, H.: High-speed compressed sensing reconstruction on fpga using omp and amp. Rn, 1:0, 2012.
119. Matam, K. K., Le, H., and Prasanna, V. K.: Energy efficient architecture for matrix multiplication on fpgas. In Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on, pages 1–4. IEEE, 2013.
120. Chen, W. and Wassell, I. J.: Energy efficient signal acquisition via compressive sensing in wireless sensor networks. In Wireless and Pervasive Computing (ISWPC), 2011 6th International Symposium on, pages 1–6. IEEE, 2011.
121. Xu, X., Ansari, R., and Khokhar, A.: Power-efficient algorithms for fourier analysis over random wireless sensor network. In Distributed Computing in Sensor Systems (DCOSS), 2012 IEEE 8th International Conference on, pages 109–115. IEEE, 2012.

APPENDIX

PUBLISHING PERMISSIONS

Permission for (23):



[Home](#)[Create Account](#)[Help](#)



Requesting permission to reuse content from an IEEE publication

Title: Memory Optimized Re-gridding for Non-uniform Fast Fourier Transform on FPGAs

Conference Proceedings: Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on

Author: Cheema, U.I.; Nash, G.; Ansari, R.; Khokhar, A.A.

Publisher: IEEE

Date: 11-13 May 2014

Copyright © 2014, IEEE

[LOGIN](#)

If you're a [copyright.com](#) user, you can login to RightsLink using your [copyright.com](#) credentials. Already a [RightsLink](#) user or want to [learn more?](#)

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

APPENDIX (Continued)

Permission for (28):



Copyright
Clearance
Center



RightsLink®

[Home](#)
[Create Account](#)
[Help](#)



IEEE
Requesting
permission
to reuse
content from
an IEEE
publication

Title: Power-efficient re-gridding architecture for accelerating Non-uniform Fast Fourier Transform

Conference Proceedings: Field Programmable Logic and Applications (FPL), 2014 24th International Conference on

Author: Cheema, U.I.; Nash, G.; Ansari, R.; Khokhar, A.A.

Publisher: IEEE

Date: 2-4 Sept. 2014

Copyright © 2014, IEEE

LOGIN

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials. Already a RightsLink user or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.


If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

APPENDIX (Continued)

Permission for (25):



Copyright
Clearance
Center



RightsLink®

[Home](#)
[Create Account](#)
[Help](#)



IEEE
Requesting
permission
to reuse
content from
an IEEE
publication

Title: InvArch: A hardware efficient architecture for Matrix Inversion

Conference Proceedings: Computer Design (ICCD), 2015 33rd IEEE International Conference on

Author: Cheema, Umer I.; Nash, Gregory; Ansari, Rashid; Khokhar, Ashfaq A.

Publisher: IEEE

Date: 18-21 Oct. 2015

Copyright © 2015, IEEE

LOGIN

If you're a [copyright.com](#) user, you can login to RightsLink using your [copyright.com](#) credentials. Already a RightsLink user or want to [learn more?](#)

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

APPENDIX (Continued)

Permission for (26):



The screenshot shows the IEEE RightsLink interface. At the top left is the Copyright Clearance Center logo. To its right is the RightsLink logo. On the far right are three buttons: Home, Create Account, and Help. Below the Copyright Clearance Center logo is a blue box with the IEEE logo and the text: "Requesting permission to reuse content from an IEEE publication". To the right of this box is a table with the following information:

Title:	A high performance architecture for computing burrows-wheeler transform on FPGAs
Conference Proceedings:	Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on
Author:	Cheema, U.I.; Khokhar, A.A.
Publisher:	IEEE
Date:	9-11 Dec. 2013

Below the table is the text: "Copyright © 2013, IEEE". To the right of the table is a box with a "LOGIN" button and the text: "If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials. Already a RightsLink user or want to learn more?"

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

VITA

NAME	Umer Iftikhar Cheema
EDUCATION	Ph.D. , Electrical & Computer Engineering, University of Illinois at Chicago, 2016 M.S. , Electrical & Computer Engineering, University of Illinois at Chicago, 2014 B.Sc. , Electrical Engineering, University of Engineering & Technology, Lahore, Pakistan, 2008
RESEARCH EXPERIENCE	PhD Research , ECE Department, University of Illinois at Chicago, <i>Spring 2011 - Fall 2015</i> Visiting Researcher , Center for Research on Embedded Systems, Halmstad University, Sweden <i>Jun. 2014 - Aug. 2014</i> Research Assistant , ECE/College of Nursing, University of Illinois at Chicago, <i>Jan. 2012- Aug. 2013</i> Professional Researcher/Developer , National University of Science & Technology, Pakistan, <i>2009 - 2010</i>
TEACHING EXPERIENCE	Instructor , ECE Department, University of Illinois at Chicago, <i>Spring 2014 - Fall 2015</i> Teaching Assistant , ECE Department, University of Illinois at Chicago, <i>Spring 2011 - Fall 2013</i>
INDUSTRY EXPERIENCE	ASIC/FPGA Design & Verification Engineer , Whizz Silicon Inc. (Off-shore) <i>Sept 2008 - Dec. 2010</i> Intern Engineer , Wateen Telecom Private Limited, Pakistan <i>Summer 2007</i>

VITA (Continued)

- JOURNAL ARTICLES**
- Cheema, Umer I.;** Nash, Gregory; Ansari, Rashid; Khokhar, Ashfaq A., Memory-Optimized and Power-Efficient Re-gridding Architecture for Non-uniform Fast Fourier Transform based applications” [In-review - 2015] IEEE Transaction on Computers
- Lodhi M., **Cheema U.**, Stifter J., Wilkie D., Keenan G., Yao Y., Ansari R., Khokhar A. (2014). Death Anxiety in Hospitalized End-of-Life Patients as Captured from a Structured Electronic Health Record: Differences by Patient and Nurse Characteristics. *Research in Gerontological Nursing*. 7(5) 224-234. doi: 10.3928/19404921-20140818-01
- Johnson, J., Lodhi, M., K., **Cheema, U.**, Stifter, J., Dunn-Lopez, K., Yao, Y., Johnson, A., Keenan, G., M., Ansari, R., Khokhar, A., Wilkie, D. (In review, Dec. 2014). Outcomes for End-of-Life Patients with Anticipatory Grieving: Insights from Practice with Standardized Nursing Terminologies within an Inter-operable Internet-based Electronic Health Record. *Journal of Hospice & Palliative Care Nursing*
- CONFERENCE PUBLICATIONS**
- Cheema, Umer I.;** Nash, Gregory; Ansari, Rashid; Khokhar, Ashfaq A, ”InvArch: A Hardware-Efficient Architecture for Matrix Inversion,” 2015 IEEE 33rd International Conference on Computer Design (ICCD 2015)
- Nash, Gregory T.; **Cheema, Umer I.;** Ansari, Rashid; Khokhar, Ashfaq A., ”Power-efficient RMA SAR imaging using pipelined Non-uniform Fast Fourier Transform,” 2015 IEEE Radar Conference (RadarCon 2015)
- Cheema, Umer I.**, et al. ”MedianPipes: An FPGA based Highly Pipelined and Scalable Technique for Median Filtering” abstract at Proceedings of 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2015)
- Cheema, Umer I.;** Nash, Gregory; Ansari, Rashid; Khokhar, Ashfaq A., ”Power-efficient re-gridding architecture for accelerating Non-uniform Fast Fourier Transform,” 2014 24th International Conference on Field Programmable Logic and Applications (FPL 2014)

VITA (Continued)

Cheema, Umer I.; Nash, Gregory; Ansari, Rashid; Khokhar, Ashfaq A, "Memory Optimized Re-gridding for Non-uniform Fast Fourier Transform on FPGAs," (FCCM 2014)

Cheema, Umer I.; et al. "A high performance architecture for computing burrows-wheeler transform on FPGAs," Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on, Dec. 2013

Lodhi M., **Cheema U.**, Stifter J., Keenan G.M., Wilkie D.J., Yao Y., Ansari R., Khokhar A.(In Review, 2014). Death Anxiety - How Nurses Can Help Patients Die Comfortably. Anxiety.org <https://www.anxiety.org/>.

Cheema, U. I., et al.Nursing Care of End-of-Life Patients Facing Death Anxiety, abstract at Midwest Nursing Research Society's (MNRS) Annual Research Conference 2013

Lodhi, M. K., **Cheema, U. I.**, et al. State of Nursing Care for Patients with Anticipatory Grieving: Lessons Learned from Standardized Nursing Data, abstract at MNRS Annual Research Conference 2013

AWARDS

UIC Graduate Student Council Travel Award Sept 2014, Feb 2015, Oct 2015

Recipient of Full tuition Waiver and Assistantship by ECE Department, UIC for the duration of doctoral studies

Awarded Merit Scholarship (2004 - 2008) by UET, Lahore on scoring top grades in higher secondary education (Top 5% students get this scholarship).

Awarded Dr. M.I.D Chughtai Medal by Forman Christian College for scoring overall first position in Higher Secondary Exams (In a batch of over 1,200 students)

Awarded Talent Award 2005 by The Punjab School for extra-ordinary academic performance