2016

Research Into Computer Hardware Acceleration of Data Reduction and SVMS

Jason Kane
University of Rhode Island, jkane@ele.uri.edu

Follow this and additional works at: https://digitalcommons.uri.edu/oa_diss

Recommended Citation

Kane, Jason, "Research Into Computer Hardware Acceleration of Data Reduction and SVMS" (2016). *Open Access Dissertations*. Paper 428.
https://digitalcommons.uri.edu/oa_diss/428

This Dissertation is brought to you for free and open access by DigitalCommons@URI. It has been accepted for inclusion in Open Access Dissertations by an authorized administrator of DigitalCommons@URI. For more information, please contact digitalcommons-group@uri.edu.

RESEARCH INTO COMPUTER HARDWARE
ACCELERATION OF DATA REDUCTION AND SVMS

BY
JASON KANE

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

UNIVERSITY OF RHODE ISLAND

2016

DOCTOR OF PHILOSOPHY DISSERTATION
OF
JASON KANE

APPROVED:

Dissertation Committee:

Major Professor Qing Yang

Haibo He

Joan Peckham

Nasser H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2016

ABSTRACT

Yearly increases in computer performance have diminished as of late, mostly due to the inability of transistors, the building blocks of computers, to deliver the same rate of performance seen in the 1980's and 90's. Shifting away from traditional CPU design, accelerator architectures have been shown to offer a potentially untapped solution. These architectures implement unique, custom hardware to increase the speed of certain tasking, such as graphics processing. The studies undertaken for this dissertation examine the ability of unique accelerator hardware to provide improved power and speed performance over traditional means, with an emphasis on classification tasking.

In the first study, the compression algorithm Lempel-Ziv-Oberhumer (LZO) 1x-1-15 is analyzed and documented. This algorithm family has seen widespread use and can be found in the NASA mars space rover and the B-tree Linux file system. A thorough analysis of the algorithm is seen to yield x86 vector and other CPU parallelization improvements that can be utilized for acceleration. Real-world datasets are used to concretely benchmark the improved performance.

The second study shifts the focus from CPU instruction acceleration to optimized hardware acceleration. A real-world embedded application of machine learning involving Support Vector Machine (SVM) accelerated hardware is examined. Prior work developed by URI's Biomedical Engineering department investigated the use of a state-of-the-art SVM-based algorithm to control an artificial limb in real-time. Evaluation of the algorithm was performed using general processing means, using a

Core i7 CPU and an Intel ATOM mobile CPU. This study builds on the prior work, investigating the performance advantages imparted by implementing the SVM decision function in hardware and combining it with a hardware-based feature extractor on a Field Programmable Gate Array (FPGA). The design is evaluated for both accuracy and real-time response to determine if the FPGA implementation is a better choice for implementation in a power-limited cyber physical system.

The third study examines the SVM classification portion of the FPGA design that was constructed for use in the artificial limb in further detail. A general purpose hardware architecture for fast, accurate SVM classification, R^2SVM , is proposed. While several similar architectures have been published, our architecture is shown to be superior in a several ways. To prove the performance, accuracy, and power consumption of the architecture, a prototype is constructed and multiple machine learning datasets are run and analyzed.

The final study takes a look at the creation of a smart city architecture. A novel multi-tiered hierarchical architecture, Reflex Tree, is proposed as a solution to automated city management in the future. The four layers of the architecture are able to perform massive parallel sensing, pattern recognition, spatial-temporal association, and system-wide behavioral analysis. Like the human nervous system, each layer in the hierarchy is able to detect specific events and inject feedback without the need for higher level intervention. Simulations of the architecture are performed in two scenarios: a gas pipeline and a city power supply network.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Qing Yang for his guidance, support, and patience throughout the duration of the research performed, which ultimately led to the creation of this dissertation. I have learned a great deal in the last several years and am honored to have had the opportunity to have been mentored by him. In addition to Dr. Yang, I would like to thank my dissertation committee: Professors Haibo He, Joan Peckham, Manbir Sodhi, and J.C. Lo for reviewing the manuscripts within this dissertation and taking the time to serve and participate in my comprehensive examination and dissertation defense.

I am also grateful to other faculty and staff members of the Electrical, Computer, and Biomedical Engineering department. In particular, I am grateful to Dr. Godi Fischer, Dr. Frederick Vetter, Dr. Resit Sendag, and Ms. Meredith Leach Sanders. I would also like to extend my thanks to Dr. Robert Hernandez, Willard Simoneau, Matthew Seaton, Jing Yang, and my other friends and colleagues.

I would like to express my gratitude to my managers at the Naval Undersea Warfare Center (NUWC), including Dr. Brian McKeon, Hector Lopez, John Fastino, Jeffrey Hanson, and Daniel Freitas. Their willingness to allow myself to work on research half-time to allow me to fulfill my educational goals was greatly appreciated. I also would like to thank Dr. Pierre Corriveau, Dr. Anthony Ruffa, and Neil Dubois, for helping to fund the vast majority of my research. Obtaining the NUWC Fellowship to cover the Fall 2014 and Spring 2015 academic sessions was extremely helpful in timely completing my remaining graduate studies requirements. I am grateful to NUWC for offering this program to employees.

Last, but not least by any means, I would like to thank my lovely wife, Anna, who has graciously put up with the degree program she “married into”, and my beautiful, adorable daughter, Molly.

PREFACE

This dissertation is written in the manuscript format. It consists of four manuscripts organized as follows:

Manuscript 1:

Jason Kane and Qing Yang, " Compression Speed Enhancements to LZO for Multi-Core Systems," published in *the proceedings of the IEEE 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2012)*, New York, NY, 2012. pp. 108-115.

Manuscript 2:

Jason Kane, Qing Yang, Robert Hernandez, Willard Simoneau, and Matthew Seaton, "A Neural Machine Interface Architecture for Real-Time Artificial Lower Limb Control," published in *the proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE 2015)*, Grenoble, FR, 2015. pp. 633-636.

Manuscript 3:

Jason Kane, Robert Hernandez, and Qing Yang, "A Reconfigurable Multiclass Support Vector Machine Architecture for Real-Time Embedded Systems Classification," published in *the proceedings of the 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM 2015)*, Vancouver, CA, 2015. pp. 244-251.

Manuscript 4:

Jason Kane, Bo Tang, Zhen Chen, Jun Yan, Tao Wei, Haibo He, and Qing Yang, "Reflex-Tree: A Biologically Inspired Architecture for Future Smart Cities," published

in *the proceedings of the IEEE 44th Annual International Conference on Parallel Processing (ICPP 2015)*, Beijing, China, 2015. pp. 360-369.

TABLE OF CONTENTS

ABSTRACT	ii
ACKNOWLEDGMENTS	iv
PREFACE.....	vi
TABLE OF CONTENTS.....	viii
LIST OF TABLES	xii
LIST OF FIGURES	xiii
1 Compression Speed Enhancements to LZO for Multi-Core Systems	1
Abstract.....	2
1.1 Introduction.....	3
1.2 Analysis Of LZO 1X-1-15	4
1.3 Algorithm Enhancements.....	10
1.3.1 Parallelization of Block Compression.....	10
1.3.2 Optimize Copying of Literal Data.....	12
1.3.3 Search for Matches Every 32-Bits.....	13
1.3.4 Force Cache-Line-Aligned Reads	13
1.3.5 Utilize Hardware CRC-32 Instruction.....	14
1.4 Performance Analysis	16
1.4.1 Test Setup.....	17
1.4.2 Benchmark Test Results	20
1.5 Conclusions.....	26

List of References	27
2 A Neural Machine Interface Architecture for Real-Time Artificial Lower	
Limb Control	29
Abstract.....	30
2.1 Introduction.....	31
2.2 System and Algorithm Design	33
2.3 Prototype System Architecture Implementation	35
2.3.1 Wireless and Inter-Module Communications.....	35
2.3.2 Feature Extraction and Phase Detection.....	36
2.3.3 Multiclass SVM and Majority Vote	38
2.4 Experimental Evaluation Methodology	39
2.5 Performance Evaluation	39
2.6 Conclusions.....	42
List of References	43
3 A Reconfigurable Multiclass Support Vector Machine Architecture for Real-	
Time Embedded Systems Classification	45
Abstract.....	46
3.1 Introduction.....	47
3.2 Background and Related Work	48
3.2.1 Overview of SVM Classification Process	48
3.2.2 Related Work.....	51

3.2.3 The Role of Precision in SVM Classification	51
3.3 Architecture Design	52
3.3.1 Kernel Evaluation.....	55
3.3.2 Coefficient Weighting	58
3.3.3 Voting.....	60
3.3.4 Prototype Implementation Details	60
3.4 Evaluation Methodology	62
3.4.1 Workload Characteristics for Benchmarking	62
3.4.2 Case Study – A Human-Computer Control Interface.....	64
3.4.3 Evaluation Platforms	65
3.5 Results	67
3.5.1 Hardware Synthesis.....	67
3.5.2 Workload and Case Study Performance Results	68
3.6 Conclusion and Future Work	71
List of References	72
4 Reflex-Tree: A Biologically Inspired Architecture for Future Smart Cities..	74
Abstract.....	75
4.1 Introduction.....	76
4.2 Related Work	79
4.2.1 Smart City Infrastructure.....	79
4.2.2 Sensor Networks for Smart City Applications	80

4.3 Overview Of Reflex-Tree Architecture.....	82
4.4 Case Studies in City Management	85
4.4.1 Case Studies: Common Layers.....	85
4.4.2 Case Study 1: Gas Pipeline Distribution System	91
4.4.3 Case Study 2: City Power Supply Network	95
4.5 Results	97
4.6 Conclusions and Future Work.....	101
List of References	101

LIST OF TABLES

TABLE	PAGE
Table 1.1. Dataset Information.....	15
Table 1.2. Cache-Line-Aligned vs. 32-Bit Search Performance.....	24
Table 2.1. Feature Extraction and SVM Prediction Timing	40
Table 3.1. FPGA Performance in SVM Feed Forward Phase.....	68
Table 4.1. Emergency Levels and Response for a Natural Gas Pipeline.....	92
Table 4.2. Emergency Levels and Response for a City Power Supply Network.....	95

LIST OF FIGURES

FIGURE	PAGE
Figure 1.1. LZO 1x-1-15 Algorithm Flow	7
Figure 1.2. LZO 1x-1-15 Input Stream Traversal (assuming no matches found).....	8
Figure 1.3. Multi-Threaded Implementation.....	11
Figure 1.4. Cache-Line Boundary Read Penalty.....	13
Figure 1.5. Compression Ratio at Various Input Block Sizes	18
Figure 1.6. Compression Time at Various Input Block Sizes.....	18
Figure 1.7. Compression Time per Block at 4k, 8k, 16k, 32k Input Block Sizes.....	18
Figure 1.8. Average Compression Time for Various Maximum Thread Settings.....	19
Figure 1.9. Compression Ratio	22
Figure 1.10. Compression Time (small streams)	22
Figure 1.11. Compression Time (large streams).....	22
Figure 1.12. Average Compression Time for Non-Redundant Data	25
Figure 2.1. Proposed Target Control System.....	33
Figure 2.2. System Event Timeline.....	34
Figure 2.3. System Architecture Implementation	35
Figure 2.4. Feature Extraction Unit.....	37
Figure 3.1. R ² SVM FPGA Architecture Overview.....	54
Figure 3.2. Pipelined Kernel Structure.....	56
Figure 3.3. Kernel Output Block.....	57
Figure 3.4. Coefficient Weighting Unit	58
Figure 3.5. Voting Unit.....	60
Figure 4.1. Example Components of a Smart City	76
Figure 4.2. Human “Reflex” Circuits.....	82
Figure 4.3. Novel 4-Layer Reflex-Tree Architecture.....	83
Figure 4.4. Preliminary Results: a) Experimental Setup	
b) Rayleigh Scatter vs. Distance	87
Figure 4.5. Parallel Fiber-Optic Sensing Network.....	88
Figure 4.6. Simulated Natural Gas Pipeline.....	94
Figure 4.7. San Francisco Bay Power Supply Network.....	96
Figure 4.8. Simulation of Fire/Ground Tremor in Gas Pipeline	98
Figure 4.9. Simulation of Earthquake/Ice Storm in Power Supply Network.....	100

MANUSCRIPT 1

Compression Speed Enhancements to

LZO for Multi-Core Systems

by

¹ Jason Kane and Qing Yang

is published in *the proceedings of the IEEE 24th International Symposium on
Computer Architecture and High Performance Computing (SBAC-PAD 2012)*,

New York, NY, 2012. p. 108-115.

¹ Jason Kane and Qing Yang are with Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, 02881, Email {jkane, qyang}@ele.uri.edu.

Abstract

This paper examines several promising throughput enhancements to the Lempel-Ziv-Oberhumer (LZO) 1x-1-15 data compression algorithm. Of many algorithm variants present in the current library version, 2.06, LZO 1x-1-15 is considered to be the fastest, geared toward speed rather than compression ratio. We present several algorithm modifications tailored to modern multi-core architectures in this paper that are intended to increase compression speed while minimizing any loss in compression ratio. On average, the experimental results show that on a modern quad core system, a 3.9x speedup in compression time is achieved over the baseline algorithm with no loss to compression ratio. Allowing for a 25% loss in compression ratio, up to a 5.4x speedup in compression time was observed.

1.1 Introduction

Real-time systems are more and more often being required to process an increasing amount of data. Interface throughput and storage bottlenecks may be reached in such systems because of the amount of data involved. Data compression can be used to help alleviate such problems by reducing the amount of data injected into a pipeline. For the purpose of this paper we will consider targeting a theoretical real-time system, requiring a lossless compression system to pass data between two interfaces. Compression may be required in such a situation due to bandwidth limitations or space constraints on the destination interface. We will assume a constant input stream of data is available to the compression device and that the final compressed output is able to be passed to the secondary interface with zero delay. The target lossless compression algorithm for this system will be Lempel-Ziv-Oberhumer (LZO) variant 1x-1-15.

The LZO compression library is a collection of lossless dictionary based data compression algorithms that favor speed over compression ratio. The LZO library was first released in 1996 and has received periodic updates since. The library has experienced widespread use, being implemented in a variety of technologies, including NASA's Mars Rovers [1] and Oracle Corporation's B-tree Linux file system. A comparison of the LZO 1x-1 algorithm speed performance against common compression formats such as GZIP, can be found in [2]. Of the available LZO algorithms, 1x-1-15 is considered by the author, Oberhumer, to have the fastest compression speeds [1], exceeding that of LZO 1x-1 at the cost of compression ratio.

This paper investigates possible enhancements to the 1x-1-15 algorithm to improve data compression speeds for use in real-time systems.

Utilizing the special architectural features of modern multi-core processors, we will examine the effects of optimizing LZO in the following ways: parallelizing block compression, using Intel SSE (Streaming SIMD Extensions) vector instructions to perform data copy operations, modifying the search algorithm, enforcing cache-aligned reads, and calculating CRC-32 checksums via hardware. All five enhancements have been implemented on the LZO open source code. Performance evaluation and comparison have been carried out using real world data sets. Experimental results have shown significant performance improvements in terms of compression time. For the same compression ratio, over a factor 3 speedup was observed. If trading off a slightly lower compression ratio is allowed and all enhancements are combined, over a factor 5 speedup was observed.

The remainder of this paper is organized as follows. First, an analysis of the existing LZO 1x-1-15 algorithm is conducted, revealing the overall structure and identifying unique characteristics. Next the proposed enhancements to the algorithm are discussed in detail. After this, experimental data is given to compare against baseline performance. Finally, conclusions regarding the obtained results are presented.

1.2 Analysis Of LZO 1X-1-15

LZO 1x-1-15 is a variation of the Lempel-Ziv 1977 (LZ77) compression algorithm, which is described in [3]. LZ77 achieves data compression via a sliding window mechanism: bytes from a look-ahead buffer are shifted one by one into a

search buffer. When matches are found between the look-ahead buffer and locations in the search buffer, tokens are output on the compression stream rather than literals, resulting in compression. Major differences in Oberhumer's LZO variation include: the optimization of operations through the use of integer computer hardware, a quick hash lookup table for match data, and better optimized output tokens.

The LZO 1x-1-15 algorithm is structured to take advantage of the fact that most computers are optimized for performing integer operations. Instruction latency/throughput tables illustrating this on Intel ATOM architecture CPUs can be found in [4]. Throughout the algorithm, no time-consuming floating point operations are used. The most complex instruction performed is an integer-multiply operation, which takes roughly five or less CPU clock cycles on the latest Intel architecture processors [5]. When pipelining and out-of-order execution operations are taken into account, the delay introduced by this multiply instruction is made even less significant.

Fetches to and from cache are also optimized. The algorithm is relatively small when compiled into binary format, likely fitting within a modern level 1 instruction cache. Compiling for i386 target hardware was found to result in roughly 600 instructions, which occupied a total size of roughly 1.84kB. This low instruction count is confirmed by that of the slightly larger, more complex cousin algorithm LZO 1x-1, given in [6]. Since the algorithm operates on data in a block manner, successive iterations do not require the frequent re-fetching of instructions from main memory. Data cache misses are also minimized, as a maximum of 48kB of data at a time is compressed; regardless of the user-defined input block size. On a modern data cache

of size 16kB or larger, this will result in few level 1 cache misses. A 48kB sub-block of input data to be compressed should easily fit within level 2 cache, if not most of level 1 cache. Intel processors in particular predict data fetch patterns and automatically pre-fetch sequential data from a detected input stream, resulting in further performance gain during the block compression algorithm [4][7].

As seen in Fig. 1.1, the 1x-1-15 algorithm itself can be divided up into four major sections of code:

1. Search for a match
2. Write unmatched literal data
3. Determine match length
4. Write match tokens

When searching for match data, the algorithm examines 32-bits of data from the input stream and computes a hash value into a small 8192 (8k) entry dictionary of 16-bit pointers to recently found data. Initially all entries in this dictionary are initialized to point to the same first 32-bit data element in the input stream, which can result in misses for the first several iterations of the search loop, until the dictionary becomes sufficiently populated. The hash value into the dictionary is completed quickly by first using an integer operation to multiply the current 32-bit unsigned data with the fixed value 0x1824429D. Pseudo-randomness, to reduce the frequency of hash table conflicts, is guaranteed based on the fact that the number being multiplied is a fairly large 32-bit prime number. Next, integer shift operations perform a quick division, resulting in the final 13-bit hash index to be used for simple indexing into the dictionary. If a match is found, or if the end of the input stream is reached, the

Figure 1.1. LZO 1x-1-15 Algorithm Flow

algorithm jumps to copying any previously unmatched literal data to the output stream. Otherwise, the pointer to the next potential data match is obtained from the input stream according to (1.1); where ip is the pointer to the current search location in the input stream and ii is a pointer to the location immediately following the last detected match in the input stream, or the beginning of the current 48kB sub-block of data in the instance that a match has yet to be found.

$$ip += 1 + ((ip - ii) >> 5); \quad (1.1)$$

In (1.1), first the difference between ip and ii is taken and the intermediate result is then divided by 32. Finally ip is incremented by this value plus one. A difference between ip and ii indicates the number of missed matches that have occurred. Larger differences have a greater effect on the equation. Fig. 1.2 illustrates the effect of misses on the input data stream. As consecutive misses become more frequent, ip , jumps exponentially until either a match is found or the end of the current 48kB sub-

Figure 1.2. LZO 1x-1-15 Input Stream Traversal (assuming no matches found)

block is reached. The search algorithm appears to make the assumption that data matches are related by spatial locality. When a match is found, the search algorithm assumes nearby matches will also occur, only incrementing the search pointer, ip , by one byte. As match detections repeatedly fail, ip is increasingly incremented in an exponential fashion.

Consider the following simplified example: a 100-byte data set with 4-byte long matches located at byte offsets 30, 34, 38, and 84. Misses will occur at offsets 0 through 29 and the search pointer ip will be incremented by one byte offset each time. Keep in mind that every time a miss occurs, the dictionary is still populated. Then three consecutive matches will occur, with ip being updated to the next unmatched location for successive search iterations. From offsets 42 through 73 ip is incremented by one byte. From offsets 74 through 82, the difference between the last match, ii , and the search pointer has grown beyond a distance of 32, and as such a byte offset of 2 is

added to *ip* upon each detected miss. At byte offset 84 a match is found once again. As the match is only 4 bytes in length, the search will resume at byte offset 88 where a miss will occur. Since a match was detected, *ii* was updated and the difference between *ip* and *ii* is once again less than 32. This results in *ip* being incremented by one byte once again rather than two. Miss detections and single offset increments will continue to the end of the dataset.

An interesting aspect of the code used by the algorithm to perform the searches is that it involves only integer operations and no comparison instructions. The effect of this is that the compiled code will produce no conditional branch instructions to calculate the exponential behavior. Consequently, no part of the CPU pipeline will need to be reserved for branch prediction to determine how far to advance into the input data stream on the next iteration. This results in overall faster code.

The second major portion of the LZO 1x-1-15 algorithm performs the copying of unmatched literal data to the output stream. First, the number of bytes of uncompressed literal data is written in an encoded manner to the output stream. This is followed by the literal data. The algorithm assumes a 32-bit data bus, and, when possible, copies data to the output stream in groups of 32-bits in an attempt to optimize any writes performed. After copying the literal data, if the end of the data stream was reached, a special block marker is written to the output stream. Otherwise, the match length is calculated.

The algorithm calculates match length by performing 32-bit XOR operations on sequential sets of data pointed to by the input stream and the dictionary. If a 32-bit match comparison is successful, the result of the XOR operation will be zero. When

the comparison fails for the first time, 8-bit byte comparisons are performed to determine if any leftover partial-word singular bytes in the stream matched. Thus, similar to the literal copy operation, the algorithm assumes that the computer's hardware will be optimally used if matches are primarily performed on a 32-bit integer basis.

Following the match length calculation, the algorithm encodes the following tokens on the output stream: a marker denoting the type of match that occurred, the offset to the location of the data that matched the current set, and the length of the match. To minimize the number of bytes used to store this information and improve the compression ratio, five different encodings exist. Each of the five encodings offers a unique variation of offset/length encoding. After outputting the match information, the algorithm returns to searching for the next 32-bit set of matching data.

1.3 Algorithm Enhancements

The five enhancements that follow in this section have been applied to the existing LZO 1x-1-15 algorithm to determine the impact on compression performance.

1.3.1 Parallelization of Block Compression

The original LZO algorithm is serial in nature. Data is compressed on a block by block basis, but there is no explicit support for multiple CPU hardware cores. By implementing a divide and conquer approach to individually compress and reassemble blocks of data in the input stream, a performance gain should be attained. This performance gain should be directly proportional to the amount of CPU cores utilized.

To investigate this improvement, a thread-based variation of the 1x-1-15 algorithm has been constructed as follows. As seen in Fig. 1.3, three main types of threads are created: control, compression, and reconstruction. Communication between the three types of threads is accomplished through the use of semaphore-protected shared memory. The main thread is created to control the flow of input data, manage memory utilization, and initiate compression threads. N user-defined threads are created to perform the actual compression on input data blocks when commanded. The compression threads operate independently from one another, each with their own set of temporary resources. The software and operating system ensure the compression threads are load-distributed to all available processor cores in the system.

Figure 1.3. Multi-Threaded Implementation

The main control thread will initiate a compression thread when the following conditions are determined: available input data exists, temporary output compression buffers exist, and an idle compression thread exists. Finally, a reconstruction thread is created to accept the output LZO-compressed block data from the individual compression threads. It is the job of the reconstruction thread to ensure that the resultant output stream is created in-order. Since data is being compressed in parallel on separate CPUs, there is no guarantee as to when a particular block of data will finish compression, i.e., Input Block 1 could be compressed first, followed by Block 3, followed by Block 2. Thus, without the reconstruction thread, there would be no mechanism to correctly reconstruct the compressed output data in-order. Unless special headers were added to identify the out-of-order data, existing LZO decompression routines would be incompatible with the produced output format.

1.3.2 Optimize Copying of Literal Data

In the current LZO algorithm, uncompressed literal data is copied to the output stream on a 32-bit long word basis. The copying of this data is one of the more time intensive operations. Instead of copying on a 32-bit integer basis, data can be copied faster using available vector instructions. On current generation Intel processors, this allows for up to 128 bits of data to be copied at a time. This should potentially quadruple performance when this portion of the code is executed. To investigate this optimization, Intel SSE (Streaming SIMD (Single Instruction Multiple Data) Extensions) vectorized memory copy related source code from Agner Fog's freely available `asmlib` library [8] was utilized.

1.3.3 Search for Matches Every 32-Bits

LZO 1x-1-15 shifts data from a look-ahead buffer into the search buffer by one byte, similar to LZ77. This algorithm variant has been created to advance the input stream by 32-bits (one integer) each time a match detection fails. In this manner, the input stream will be traversed faster, although with the extra side effect that the dictionary will be less frequently populated. A compression ratio loss is expected, the magnitude of which will need to be determined experimentally.

1.3.4 Force Cache-Line-Aligned Reads

Most current generation Intel processors suffer a penalty when reading data that exists between cache line boundaries [7]. Figure 1.4 illustrates this issue. The search portion of the 1x-1-15 algorithm has the potential of suffering the most from this observation. In the worst case, successive new matches could continually be found on cache line boundaries, incurring a performance penalty each time. By modifying the search equation and match length detection to operate entirely on a 32-bit basis, all accesses to memory can be ensured to be cache-line-aligned.

Assuming the cache line boundaries in a system lie on an address location that is divisible by 4 bytes, and knowing that up to 4 bytes of data can be read from memory

Figure 1.4. Cache-Line Boundary Read Penalty

at a time, the following replacement search equation was constructed to ensure that a boundary would never be crossed:

$$ip += 4 + (((ip - it) >> 5) \& 0xFFFFFFFFC); \quad (1.2)$$

Four bytes are added to the current input pointer instead of one byte to ensure that the input data is always advanced by 32 bits. The exponential portion of the equation is bit-wise ANDed to ensure that the result of that mathematical operation is 32-bit aligned as well.

Match length calculation was also required to be modified to ensure that reads from the input stream remain on cache line boundaries. Since the existing algorithm allowed for match lengths to be determined on a byte basis, this variation was altered to only perform matches on a 32-bit basis, guaranteeing inter-cache line boundary locations will not be read the next time the search equation is executed.

This modification is in essence an extension of the previously described 32-bit search variation. In addition to searching every 32-bits for a match, the search pointer is verified to lie on a 4-byte boundary and the lengths of matching datasets are terminated early to ensure they result in multiples of 32-bits. Similar to the 32-bit search algorithm, a compression ratio loss is expected. The compression ratio loss will likely be greater due to the fact that matches will be shorter. A speed enhancement may result due to the cache line read penalty being avoided.

1.3.5 Utilize Hardware CRC-32 Instruction

The existing LZO library incorporates a CRC-32 calculation routine derived from the freely available zlib library [9]. This variation implements a tabular method using the 0x4C11DB7 polynomial. This method requires a considerable amount of

Table 1.1. Dataset Information

Compression Item	File Type / Description	File Size (GB)
images_en.nq	Text (Dump of Wikipedia Thumbnail Links)	1.2
e-coli	HDF RAW Binary (E-Coli C227 Strain Information)	30.0
gb-1gram-(n).csv	Text, where n= 0 to 9. (From Google Books 1-gram corpus)	0.95
ERR007772_(n).fastq	FASTQ Text-Based, where n= 1,2. (Mouse Genome Sequence Data)	5.6
xtf_Files.tar	XTIFF Images (NY Hudson River Side-scan SONAR images)	1.5
enwiki.xml	Text (Wikipedia dump of articles, templates, media/file descriptions, and primary meta pages)	37.9
ERR007772_1.fastq.gz	Compressed GZIP (Compressed ERR007772_1.fastq)	2.4
enwiki.xml.bz2	Compressed BZIP2 (Compressed enwiki.xml)	8.6

processor utilization to generate checksums. The particular approach taken for this paper was to observe the performance of the relatively new Intel SSE 4.2 assembly instruction “CRC32”.

It should be noted that the improved CRC-32 library function is not explicitly utilized by the LZO 1x-1-15 algorithm, or any other of the provided LZO library algorithms for that matter. The function is provided by Oberhumer to the end-user so that CRC’s may be calculated on a need basis as determined by the writer of the final produced compression executable. In this manner the end-user is able to adjust the balance between speed and error-checking capability. The author of the LZO library has written and maintained the executable "lzop". The current version of lzop, 1.03, incorporates a small subset of the LZO library for its compression, including the 1x-1, 1x-1-15, and 1x-999 algorithms. This executable calls the CRC-32 library function twice for every block of input data compressed; once to determine the CRC for the

original uncompressed data block and once to determine the CRC for the compressed output block. By default, the application sets the user-defined input block size to 256kB.

1.4 Performance Analysis

To determine compression performance, first a dataset of sample files to be compressed was constructed. The particular set was chosen to demonstrate performance over a variety of file types that could represent potential data streams in a real-time system. A brief description of the files used can be seen in Table 1.1.

Text files such as the Wikipedia backups and the Google Books 1-gram corpus were chosen to demonstrate performance on highly redundant data. Since LZO is a dictionary-based method, it was expected that the compression time and ratio should yield fairly good results for these files. The files already containing various degrees of compressed data (GZIP and BZIP2) were chosen to show compression performance when file expansion has a high potential of occurrence. An uncompressed tarball compilation of extended TIFF image data was picked to demonstrate image compression characteristics of the algorithm. A rather large uncompressed tarball compilation of HDF biological E-coli data was chosen to show performance against binary data. Finally, genome data in FASTQ format was chosen, as it is considered the standard for storing the output of high throughput sequencing instruments. Thus, the FASTQ data should provide a somewhat realistic real-time data stream.

As stated in Section 1.1, performance analysis was evaluated in regards to a theoretical real-time compression system consisting of two interfaces: a constant input stream and an output stream that can be written to with zero delay. A software

command-line executable was created to simulate such a system and facilitate batch compression of the files identified in Table I. The executable loads input data and compresses it in memory to reduce the impact of slow file I/O subsystems. Time spent reading data from the hard disk is not counted against compression time. This serves two purposes; first, the algorithms can be more fairly evaluated independent of hardware overhead limitations, and second, the input data stream is simulated as a continuous stream, as desired by the real-time system simulation. Parameters were passed to the executable dictating which algorithm to utilize and how many iterations of compression to perform on a per file basis. After testing a particular algorithm's performance on a file from the dataset, the program outputs the average throughput, the average compression time, the average compression time per input data block, the compression ratio, and the final compressed file size.

1.4.1 Test Setup

A Dell Optiplex 790 computer running Microsoft Windows 7 Professional 64-bit was utilized to determine the impact of the algorithm modifications. The system was configured with an Intel 3.4GHz i7-2600 Sandy Bridge Quad Core Processor (4 physical cores, 8 logical cores), 8 GBytes of Dual Channel 1333MHz DDR3 RAM, and a 500GB 7200RPM hard disk drive.

Figure 1.5. Compression Ratio at Various Input Block Sizes

Figure 1.6. Compression Time at Various Input Block Sizes

Figure 1.7. Compression Time per Block at 4k, 8k, 16k, 32k Input Block Sizes

Prior to the start of testing, an optimal parameter was determined for the input block size. The first Google Book 1-gram dataset was run with several input block sizes, as seen in Fig. 1.5 through Fig 1.7. Compression ratio and compression timing performance noticeably increased from about 1kB to 16kB. From 48kB onward, performance seemed to level off and remain relatively constant. Compression time per block was found to be on the order of milliseconds. It should be noted that the time to compress one block of data is virtually the same for the baseline and multi-threaded implementations, as the multi-threaded compression routine is identical. The multi-threaded implementation gains its speed due to the fact that it is processing multiple compression blocks simultaneously. For testing purposes, an input block size of 256kB was chosen, as it is the default used by Oberhumer's lzop executable and demonstrated no obvious disadvantage in the preliminary tests.

For those algorithms utilizing multiple CPU cores, in addition to the input block size parameter, the optimum number of compression threads was determined empirically prior to testing. Some results of this testing can be seen in Fig. 1.8.

Figure 1.8. Average Compression Time for Various Maximum Thread Settings
(Control and Reconstruction Threads Not Counted in Totals)

Performance with one thread in the multi-core version of LZO was found to be slightly worse than the baseline. This is most likely due to the overhead involved in coordinating the multi-core algorithm. Performance for both multi-threaded implementations increased up to a maximum value of four compression threads. It was originally expected that performance would increase up to eight maximum threads as the system under test has eight logical cores. Further testing showed that due to the specific software implementation, the CPU cores never reached maximum utilization of all eight cores simultaneously. In order to accommodate large file sizes, only 64 MBytes at a time were read to physical memory from disk and compressed in a loop. The stall time imparted from performing the disk reads was accounted for and subtracted from the total compression time. However, the operating system, due to the frequency of the stalls, appeared to determine that delegating work to the logical cores was not necessary, as the physical cores remained slightly underutilized. A modified version of the compression benchmark program was created to iterate on the first 512 MB of file data in memory. This software was able to exercise all eight cores, showing the expected performance boost from one to eight threads.

For the purposes of this testing it was decided to continue utilizing the original version of software with the maximum thread parameter set to 4 or greater. The memory-only software variation would result in inaccurate compression ratio measurements, as larger file streams would not be able to fit within available memory space.

1.4.2 Benchmark Test Results

Testing was conducted in the following manner. First the performance of the existing LZO 1x-1-15 algorithm was determined. Then each of the four new performance enhancements to 1x-1-15 was tested separately. A version of the compression algorithm that combined all of the enhancements was tested and the results recorded. Finally, the CRC-32 library modification was tested against the baseline library function using the unmodified LZO 1x-1-15 algorithm with a 256kB input block size. The CRC-32 function calls were constructed similar to those used in lzop, calling the function twice per each processed input data block.

Results of testing the LZO 1x-1-15 variants on the dataset can be found in Fig. 1.9 through Fig. 1.11. In terms of compression ratio, in all cases, the baseline, optimal memory copy, and multi-core implementations yielded the best results. This was expected, as the three of these algorithms did not deviate from the original algorithm in a manner that would affect file traversal behavior. The remaining three algorithms employed modifications to the search portion of the algorithm to enhance compression speed. Compared to the baseline algorithm, the 32-bit search, cache aligned read, and combination implementations experienced losses in compression ratio. The 32-bit search algorithm resulted in files up to 1.3 times larger than the baseline, whereas the cache aligned read and combination variants created files up to 2.3 times in size.

Figure 1.9. Compression Ratio

Figure 1.10. Compression Time (small streams)

Figure 1.11. Compression Time (large streams)

In terms of compression time, the baseline algorithm proved the slowest for most files. In all cases, the optimized vector memory copy algorithm outperformed the baseline speed-wise by roughly 5 percent. The multi-core and combination algorithms consistently completed file compression within 25 and 20 percent respectively of the time taken by the baseline algorithm. On average, for the files tested, the speedup gained was 3.9x for multi-core and 5.4x for the combination algorithm. The 32-bit search algorithm resulted in a 10 percent speed increase for text files and a 25 to 50 percent increase for binary/image files. The cache-line-aligned read algorithm, with one exception, proved to be slightly faster than the 32-bit search modification.

The one exception to this trend occurred with the file “image_en.nq”. This is a text file composed of Wikipedia thumbnail hyperlinks. A brief examination of the file shows that the contents appears to be highly redundant, as it consists of many very similar hypertext Uniform Resource Locator (URL) ASCII text strings, all beginning with “<http://”. The baseline algorithm confirms that this is the most redundant of the files tested, as it achieved the greatest compression on this file in Fig. 1.9. It was expected that the compression ratio would be the worst for the cache-line-aligned version, as this version not only searches every 32-bits for a match and ignores matches less than 32-bits in size, but also moves the search pointer forward if it is not currently on a 32-bit boundary. The extent of the speed degradation, however, was not anticipated.

To determine why the cache-line-aligned modification may experience issues with highly redundant data, a special version of the compression software was created to gather more in-depth timing information. This version of software recorded

execution time spent in the four main sections of the compression code, described in Section 1.2. Compression of the file “image_en.nq” was re-run using the 32-bit search and cache-line-aligned algorithms. The results can be found in Table 1.2.

It was found that the cache-line-aligned algorithm overall executed roughly 1.28 times the number of loop iterations, resulting in a total compression time of 1.40 times longer than that of the 32-bit search variation. Supplemental testing on an Intel Core2 Duo system (not shown), which according to [7] should benefit more from the aligned reads than the i7 processor used for benchmarking, showed that the algorithm still performed 1.30 times slower than the 32-bit search variation. Examining Table 1.2, fewer match length determination loops were executed, implying that fewer dictionary matches were found. This makes sense, as during dictionary searches the 32-bit alignment of the input pointer may skip up to three consecutive bytes, leading to a more sparsely populated dictionary. Those matches found would also have been shorter in length due to the imposed 32-bit boundary restriction. This led to more

Table 1.2. Cache-Line-Aligned vs. 32-Bit Search Performance
(Higher Ratios Indicate Worse Performance for Cache-Line-Aligned Alg.)

Code Portion	% Total Compression Time	%Total Compression Time	Ratio of Loop Iterations	Ratio of Compression Time
	32-Bit Search Alg.	Cache – Line Alg.		
	(Cache-Line Alg. : 32-Bit Search Alg.)			
Search	27.0%	31.9%	1.35	1.65
Literal Copy	14.8%	16.3%	2.15	1.55
Match Length	41.7%	32.3%	0.95	1.08
Token	16.5%	19.5%	1.70	1.66
Overall	100%	100%	1.28	1.40

Figure 1.12. Average Compression Time for Non-Redundant Data

execution time in the rather expensive search and uncompressed literal copy code segments. The ability of the search function to exponentially skip through a 48kB sub-block is also hindered by the high level of data redundancy: the algorithm periodically finds short matches, resulting in an inability to skip forward to the end of a sub-block, as it would with data of lower redundancy. These combined factors appear to have attributed to the poor speed performance for highly redundant data.

Compression timing performance for highly non-redundant data can be seen in Fig. 1.12. Once again, the multi-threaded algorithms experienced considerable speed boosts compared with the others. Results indicated that file expansion only achieved an excess growth of 0.4 percent of the original file size for all tested algorithm variants.

CRC-32 testing was conducted on one of the Google Book 1-gram data files. Testing showed that on average an extra 1.47 msec per block was added to compression time when using the tabular method, and an extra 0.23 msec per block was added when using the hardware method. As expected, the hardware

implementation demonstrates a considerable performance increase of about six times that of the baseline CRC-32 function.

1.5 Conclusions

This paper examined several methods in which the compression speed of LZO 1x-1-15 can be improved. Slight adjustments to the code were made to increase parallelization, improve the copying of literal data, speed up input stream traversal, optimize cache accesses, and perform hardware CRC-32 calculations. The multi-core and optimized memory copy algorithms both demonstrated their ability to speed compression without affecting compression ratio. Combining these two mutually exclusive operations should likely achieve a more robust implementation of LZO 1x-1-15. Hardware CRC-32 calculations were seen to add verification to the output stream with minimal impact on total compression time. Of the algorithm variants tested, the multi-core algorithm displayed the greatest speed enhancement without degradation to compression ratio - an improvement of 3.9x over that of the baseline. In instances where compression ratio is not of primary importance, the combined optimization algorithm demonstrated the ability to increase compression speed performance by 5.4x.

In general, the results suggest that LZO and other token-based block compression algorithms similar to it can benefit from recent CPU hardware optimizations. Increased processor bus widths allow for larger block memory copies when storing uncompressed literal data. The introduction of multiple cores allow for multiple blocks to be simultaneously and independently compressed. These optimizations

should port to embedded multi-core architectures containing data buses greater than 32-bits.

Future work may be explored by investigating potential optimizations from the Intel AVX (Advanced Vector Extensions) and AVX2 vector instruction sets [10]. These instruction sets extend upon SSE, allowing for SIMD instructions to run on 256-bit data types. Further vector optimizations may be possible in hash calculation, literal copying, and match length determination. Preliminary prototypes of the software under test showed that utilizing existing x86 SSE 128-bit vector operations for these purposes often resulted in performance loss due to operand alignment related setup costs. Operands typically become unaligned due to the byte-oriented searching nature of the LZO algorithm. Performing enough vector calculations in parallel may outweigh the setup disadvantage. It should be noted that these costs are entirely architecture dependent. Migrating to another processor family may prove to have better or worse performance when executing similar instructions.

Another area of future work may be to implement an adaptive version of the algorithm similar to that described in [11]. Such a system would attempt to maintain a minimum compression ratio, switching back and forth between the different algorithms for speed gains as need allows.

List of References

- [1] Oberhumer, M.F.X.J., 2011. LZO real-time data compression library. [Online]. Available: <http://www.oberhumer.com/opensource/lzo/>. [20 June 2012].
- [2] L. Yang, R.P. Dick, Haris Lekatsas, and Srimat Chakradhar, "Online memory compression for embedded systems", ACM transactions on Embedded Computer Systems. vol 9, issue 3, no. 27, February 2010.

- [3] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, vol. 23, no. 3. pp.337-343, 1977.
- [4] Intel Corporation. (2012, April). “Intel 64 and IA-32 Architectures Optimization Reference Manual” [online]. Available: <http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf> [20 June 2012]
- [5] Agner Fog. (8 June 2011). “Instruction tables. Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA Cpus” [online]. Available: http://www.agner.org/optimize/instruction_tables.pdf [20 June 2012].
- [6] C.M. Sadler and M. Maronosi, “Data Compression Algorithms for energy-constrained devices in delay tolerant networks”, Proc. Of IEEE SenSys ‘06, pp. 265-278, Nov 2006.
- [7] Agner Fog. (8 June 2011). “The microarchitecture of Intel, AMD and VIA CPUs” [online]. Available: <http://www.agner.org/optimize/microarchitecture.pdf> [20 June 2012]
- [8] Agner Fog. (21 August 2011). “Instructions for asmlib. A multi-platform library of highly optimized functions for C and C++” [online]. Available: <http://www.agner.org/optimize/asmlib-instructions.pdf> [20 June 2012]
- [9] Mark Adler, 2012. zlib Home Site. [Online]. Available: <http://www.zlib.net>. [20 June 2012]
- [10] Intel Corporation. (2011, April). “Intel® 64 and IA-32 Software Developer’s Manuals” [online]. Available: <http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html> [20 June 2012]
- [11] Chandra Krintz and Sezgin Sucu, “Adaptive On-the-Fly Compression”, Transactions on Parallel and Distributed Systems, January 2006, Vol. 17 No. 1

MANUSCRIPT 2

**A Neural Machine Interface Architecture for
Real-Time Artificial Lower Limb Control**

by

Jason Kane¹, Qing Yang¹, Robert Hernandez¹, Willard Simoneau²,
and Matthew Seaton²

is published in *the proceedings of the Design, Automation & Test in Europe
Conference & Exhibition (DATE 2015)*,
Grenoble, FR, 2015. p. 633-636.

¹ Jason Kane, Qing Yang, and Robert Hernandez are with Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, 02881, Email {jkane, qyang, rhernandez}@ele.uri.edu.

² Willard Simoneau and Matthew Seaton are with Torpedo Systems Department, Naval Undersea Warfare Center, Newport, RI, 02842, Email {willard.simoneau, matt.seaton}@navy.mil.

Abstract

This paper presents a novel architecture of a lower limb neural machine interface (NMI) for determination of user intent. Our new design and implementation paves the way for future bionic legs that require high speed real-time deterministic response, high accuracy, easy portability, and low power consumption. A working FPGA-based prototype has been built, and experiments have shown that it achieves average performance gains of around 8x that of the equivalent software algorithm running on an Intel Core i7 2670QM, or 24x that of an Intel Atom Z530 with no perceivable loss in accuracy. Furthermore, our fully pipelined and parallel non-linear support vector machine-based FPGA implementation led to a 6.4x speedup over an equivalent GPU-based design. In this paper, we also characterize our achieved timing margin to show that our design is capable of supporting real-time wireless communications. With additional refinement, such a wireless personal area network (PAN) system will provide improved flexibility on an individual basis for electromyography (EMG) sensor placement.

2.1 Introduction

Until recently, commercially available lower limb prosthetics have mainly focused on the use of simple passive devices. Surface electromyography (EMG) is one sensory interface that has been used in the past to successfully predict active user intent to control upper limb devices [1]. The main issue with applying this pattern recognition (PR) technology to patients with transfemoral (TF) amputations is that the detected lower limb EMG signals are typically non-stationary during locomotion. These EMG signals are however quasi-cyclic with respect to locomotion gait and somewhat stationary within short windows. In [2], it was shown that a phase-dependent EMG pattern classifier could be constructed to accurately predict intended user movements. When the EMG data is fused with measured moments and ground reaction force from the artificial limb, a more accurate algorithm was developed, achieving over 95% accuracy for locomotion detection [3] [4] [5]. The algorithm in [3] and [4] used linear discriminate analysis (LDA) for classification, and was later realized in a field programmable gate array (FPGA) to obtain real-time performance [6]. To achieve higher accuracy, a support vector machine [7] (SVM) based classifier has been studied [5] and shown to provide greater accuracy than the LDA approach. Because the ultimate goal is to control an artificial limb in a real-time and wearable environment, a portable embedded system is desirable with low form factor and low power consumption.

In this paper, we present a new lightweight design of the more accurate SVM-based algorithm in an FPGA. Our FPGA implementation differs greatly from that presented by the LDA approach in [6], as it utilizes a non-linear SVM-based detection

algorithm. While this increases the design complexity, it has been shown to provide higher accuracy in determining user intent over the LDA approach [5]. Furthermore, in an effort to increase performance via pipelining and parallelism, our FPGA implementation was written and optimized in VHDL. In contrast, the previously created LDA-based FPGA was auto-generated using Impulse C C-to-HDL CoDeveloper [8].

This paper makes the following contributions:

(C1) Development of an architecture providing a well-defined, deterministic response using a previously published highly accurate SVM-based algorithm. An FPGA implementation should execute faster than a general purpose CPU due to the exploitable parallelism of the algorithm. Dedicated pipelined hardware driven by state machines will guarantee lower and more consistent response times than a traditional CPU or GPU, both of which are reliant on operating systems. This becomes important when one's stability and safety are dependent on a functional system.

(C2) Introduction of a power and area efficient architecture for artificial lower limb control. Most importantly, this work should serve as the first step in creating a physically smaller and lower power ASIC implementation of the SVM-based algorithm. This is of the utmost importance, since our computational engine needs to fit within the volume constraints of the artificial leg and provide a minimum of 6 to 8 hours of continuous use.

(C3) Demonstration of a system capable of real-time wireless communications. Making comparisons with an existing real-time wireless protocol, we will show that sufficient prediction time slack exists such that our design can communicate with

Figure 2.1. Proposed Target Control System

sensory devices in a time critical manner. Wireless sensors are desired for EMG readings because, depending on a TF amputee subject's residual nerves, some muscle locations may be preferred over others for placement. Our eventual wireless system should provide flexibility, allowing us to target those regions on an individual basis.

The remainder of this paper is organized as follows. In the next section we briefly introduce our proposed real-time system and describe our existing NMI algorithm. Section 2.3 details our prototype hardware implementation. In Sections 2.4 and 2.5 we introduce our testing methodology and discuss our results. We then present our conclusions and discuss future work in this area.

2.2 System and Algorithm Design

The proposed final state of our prosthetic system can be seen in Fig. 2.1. A detailed description of data flow in the system is summarized in [9]. Capabilities and limitations of the implemented prototype design are discussed in Section 2.3.

To obtain force and moment measurements, the artificial lower limb contains an integrated six degrees of freedom (6-DOF) load cell manufactured by Bertec

Figure 2.2. System Event Timeline

Corporation, model AM6514. Intended user movements are captured through the use of seven EMG sensors. These sensors are placed at several strategic muscle locations that are dependent on the subset of classifications being performed. Analog readings from all thirteen channels are amplified and digitized as 16-bit samples at a 1kHz sampling rate and then transmitted over a wireless interface. The amplification, A/D conversion, and wireless transmission occur at each sensor individually. In the final design a real-time protocol, such as RT-WiFi [10], will be used to transfer the data within a guaranteed time window. This creates a personal area network (PAN) and allows for the sensors to be placed with much greater flexibility.

The wireless data is received by an input/output (I/O) core which forwards the raw ADC data to the FPGA's feature extractor. Once the feature extractor has obtained enough sensor data to occupy one window, a set of features are computed and forwarded to the multiclass SVM module. The SVM module takes these features and performs the current classification of user intent, which is then forwarded to a ten-point majority vote algorithm. The result of the majority vote is the final predicted user intention, which is sent through the wireless link back to a device on the artificial limb that will use the information to adjust limb impedance and position accordingly via its own separate state machine. Fig. 2.2 depicts a timeline of the major events, as seen from the FPGA, that take place during operation. Details of notable algorithm events are described in [9].

Figure 2.3. System Architecture Implementation

2.3 Prototype System Architecture Implementation

For our concept prototype, we implement the FPGA portion of the design in Fig 2.1. We later use offline EMG/Load Cell data gathered from our existing lower limb prosthesis hardware for validation testing in Sections 2.4 and 2.5. A high level block diagram of the hardware implementation can be seen in Fig. 2.3. In the subsections that follow, the major three components of the prototype design are discussed.

2.3.1 Wireless and Inter-Module Communications

A dual core implementation of a superscalar MIPS-like CPU was synthesized to act as a transmitter/receiver between incoming raw EMG and load cell data and outgoing predictions. Due to its small footprint, USB device compatibility, and built-in support for several standard wireless communication protocols, a real-time embedded build of Linux was chosen for an operating system. One core on the system is dedicated to handling kernel tasking. The other core is responsible for receiving wireless data and transmitting it to the Feature Extractor/SVM module over a fast

5Mbit serial link as well as receiving predictions from the serial link and relaying them wirelessly back to the artificial limb.

2.3.2 Feature Extraction and Phase Detection

The feature extraction unit is depicted in detail in Fig. 2.4. At the front-end of the unit, incoming digital samples are tagged by channel and sample number. To minimize computational noise, our hardware implementation performs as many lossless operations as possible in the integer domain before converting to floating point for use with the existing SVM classifier models. The use of less complex integer operations also saves clock cycles and reduces hardware complexity. Fig. 2.4 uses a dashed line to delineate between the use of integer and floating point operations in the hardware.

Data initially enters the feature extraction unit in unsigned 16-bit format, but is converted to signed 32-bit format for performing feature calculations. Transferring the data in its original 16-bit format saves valuable wireless transmission time. Information regarding incoming frames is kept, and each functional unit automatically begins calculating its features upon receiving the last sample it needs.

Six load cell units operate in parallel, one for each channel. Integer operations are performed to determine the minimum, maximum, and mean of the last eight data windows. A conversion to floating point then takes place and the resulting 18 features are normalized. Like the load cell units, one EMG unit exists for each EMG channel, resulting in a total of 7 parallel units. Three features (the mean of the absolute value of all samples, number of zero crossings, and number of turns) are calculated using only integer operations, then converted to floating point and normalized. On the other

Figure 2.4. Feature Extraction Unit

hand, the length feature is mostly calculated using floating point arithmetic. This is necessary due to the presence of a square root operation, which would result in loss of precision if equivalent integer operations were performed. To reduce latency in the length calculations, a seven-stage pipelined floating point addition unit at the final calculation stage is used to compute eight partial sums that are continually re-injected into the pipeline, rather than stall the pipeline for each addition operation. When all partial sums are computed, they are combined together, and the result is normalized and presented as a feature to the SVM unit.

The phase detection unit performs a 3-tap IIR filter operation on the fifth oldest window of vertical ground reaction force data. The integer data is converted to fixed point Q18.14 format to reduce the amount of time taken for calculations. The output

of the filter is used as input to a simple state machine to predict the current gait phase. The prediction is stored in a ten entry history buffer to aid in future predictions.

2.3.3 Multiclass SVM and Majority Vote

The Multiclass SVM unit is designed to handle three classes. The SVM unit's block RAM is primed with model data prior to run-time, including support vectors, combined coefficient and class labels, and bias values. The RAM is designed to accept and store up to four different models, corresponding to the four gait phases. When the SVM unit is signaled to begin a classification, the stored SVM model corresponding to the current gait phase at the time of feature latching is used for all subsequent operations. A multi-stage pipelined floating point Gaussian kernel computation [11] then begins. Once the first valid sample exits the pipeline, subsequent valid samples are output on each clock cycle until all remaining calculations are complete. The scalar outputs from the kernel operation are stored in RAM and later retrieved as needed by the SVM weighting operation that follows.

After kernel function execution, the SVM decision function [7] is evaluated. The hardware achieves high throughput by attempting to maximize the total number of parallel weighting operations. The final value for each SVM comparison is delivered in a pipelined fashion to a vote counting unit, which determines the winner for each binary classification and tallies votes. When it has detected that a target number of votes have been placed, the final classification is sent to the majority vote unit.

The majority vote hardware maintains a sliding window of the last 10 predictions. Prior to receiving a new prediction, it adjusts its window and tallies the votes for each

stored classification. When the new classification is available only a few clock cycles are required to update the vote tally and output the final result.

2.4 Experimental Evaluation Methodology

This study was conducted with Institutional Review Board (IRB) approval at the University of Rhode Island and informed consent of subjects. Testing occurred in two phases: feature extraction and SVM performance evaluation, and real-time wireless system performance evaluation. For all testing, 3-Class offline data from [9] was utilized to benchmark the performance of the new system.

Two sets of 3-Class data were obtained from a male able-bodied subject. The first set of data was used to predict stair ascent and consisted of 6905 individual classification tests. The second set of data was used to predict walking motion and consisted of 7391 classification tests.

A software program was created to transmit required input data wirelessly to the FPGA module under test and receive responses. Cycle-accurate counters embedded within the FPGA hardware determined the amount of time to independently perform both feature extraction and multiclass SVM. This timing information was reported back to the software program along with the resulting prediction. Accuracy and timing benchmarks were achieved by comparing against a current generation Intel i7 CPU running a previously published C software-based implementation [9]. The Intel i7 system under test consisted of the following major components: an Intel Core i7 2670QM CPU, 6 GB of RAM, and Windows 7 64-bit operating system.

2.5 Performance Evaluation

Table 2.1. Feature Extraction and SVM Prediction Timing

Trial	Average Time Taken (ms)		FPGA Speed Increase		
	Intel i7	FPGA	Min	Max	Average
3 Class Stair Up	0.25	0.03	5.51x	171.46x	8.40x
3 Class Walking	0.24	0.03	5.54x	25.14x	8.03x

After design compilation, the Altera Quartus II tool predicted that our total thermal design power (TDP) is 2.3 Watts, roughly matching that of the Intel Atom Z530 mobile CPU that ran the comparable NMI algorithm in [9]. Our complete system should require less power than this mobile device, as all memory and most communications resources are contained within the Stratix V.

During the first phase of testing, the offline A/D sample data from the 20ms 3-Class trial was sent to the feature extractor in real-time. The three classes predicted by this model include stair ascent, standing, and level-ground walking. The system reported back the class prediction and timing taken to perform feature extraction and multiclass SVM prediction. The mean timing and overall speed-up achieved over the Intel i7 by the FPGA to complete both feature extraction and multiclass SVM predictions can be seen in Table 2.1.

In [9], the Intel Z530 Atom achieved a mean prediction time of 0.721ms over the same set of trials and in [12], a highly parallelized GPU-based version of our algorithm was run on a 35W GeForce GT 540m with 96 CUDA Cores, yielding an average prediction time of 0.192ms. Comparing with our results in the table, we have achieved a speedup of 24x and 6.4x respectively over these platforms. In the case of the GPU, a 15.2x power advantage is also achieved. Another interesting comparison can be seen between the SVM and LDA-based FPGAs. In an equivalent 3-Class trial,

[6] reports the average prediction time of its FPGA as being 0.23ms. As seen in Table 2.1, the SVM-based FPGA only takes 0.03ms on average – a 7.6x performance increase. While LDA generally requires fewer computations than SVM, the hardware optimizations present in our design, combined with the fact that the LDA FPGA was created with a software to HDL converter both contribute to this unexpected difference in performance.

Unlike the CPU implementation, our feature extraction consistently occurs within a fixed number of clock cycles, and since the amount of time taken for a classification decision is proportional to the selected gait phase, predictions can be expected to occur within a precise, guaranteed timeframe. Further analysis showed that with the exception of one prediction, results were identical to those produced by the Intel i7, and met the high level of accuracy reported in [5] and [9]. The one mismatch was determined to be a near-boundary case.

In the second phase of testing, the software program used previously for data injection was modified to record the round trip timing of the system. Timing was recorded for both wired Ethernet and 802.11g WiFi using TCP/IP with the Nagle algorithm disabled for speed. The wired Ethernet implementation was chosen as a control to ensure that the soft-core CPU and SVM-based limb algorithm hardware were capable of handling the latency and throughput required for real-time communications. The desired processing slack time was determined using data from [10]. In their paper, the developers of RT-WiFi found that in a noisy environment, a payload of 460 bytes will incur a transmission delay of at most 4.2ms. Extrapolating

linearly to the 520 bytes required for each 20ms window prediction, it can be expected that our system should achieve a guaranteed slack time of at least 4.75ms per window.

The same trial subsets of 3-Class data used during the first phase of testing were again used to determine performance. In the wired implementation, all prediction responses were received within the desired 20ms prediction timeframe. Most arrived within 4ms, with a few taking as long as 14ms. This yields a minimum slack time of 6ms, which satisfies our derived timing requirement. As expected, during 802.11g WiFi testing, the system was incapable of meeting real-time demand, with responses of up to 60 ms. Clearly, a solution similar to RT-WiFi is needed to achieve the desired response.

2.6 Conclusions

This paper explored using FPGAs to improve the real-time performance of an existing NMI algorithm for lower limb control. It was found that by implementing the real-time limb control algorithm in an FPGA we are able to meet required accuracy, while completing all computations within a much smaller and bounded timeframe when compared to the previous software-based implementations. In addition, the FPGA provides a wearable footprint and power rating required by limb control. Analyzing the published data from the authors of [10] along with our wired Ethernet results, we found that the prototype system should be capable of real-time communications if a proper custom data link layer is substituted. Future work may be performed on the topic of further power reduction. ASIC implementations have been known to yield considerably better power ratings than FPGAs and require significantly

less volume. A study will need to be undertaken to quantify the advantages an ASIC design would have over an FPGA.

List of References

- [1] K. Englehart and B. Hudgins, "A robust, real-time control scheme for multifunction myoelectric control," IEEE Transactions on Biomedical Engineering, vol. 50, pp. 848-854, Jul 2003.
- [2] H. Huang, T. A. Kuiken, and R. D. Lipschutz, "A strategy for identifying locomotion mode using surface electromyography," IEEE Trans Biomed Eng, vol 56, pp. 67-73, 2009.
- [3] H. Huang, Y. Sun, Q. Yang, F. Zhang, X. Zhang, Y. Liu, J. Ren and F. Sierra, "Integrating Neuromuscular and Cyber Systems for Neural Control of Artificial Legs" in The ACM/IEEE First International Conference on Cyber-Physical Systems, Sweden, 2010.
- [4] Fan Zhang; DiSanto, W.; Jin Ren; Zhi Dou; Qing Yang; He Huang, "A Novel CPS System for Evaluating a Neural-Machine Interface for Artificial Legs," Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on , pp.67-76, 12-14 April 2011.
- [5] H. Huang, F. Zhang, L. J. Hargrove, Z. Dou, D. R. Rogers, and K. B. Englehart, "Continuous locomotion-mode identification for prosthetic legs based on neuromuscular-mechanical fusion," IEEE Trans Biomed Eng, vol 58, pp. 2867-2875, 2011.
- [6] Xiaorong Zhang; He Huang; Qing Yang, "Implementing an FPGA system for real-time intent recognition for prosthetic legs," Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE, pp.169-175, 3-7 June 2012.
- [7] C. Cortes and V. Vapnik, "Support-Vector Networks," Machine Learning, vol 20, no 3, pp. 273-297, Sept. 1995.
- [8] Impluse Accelerated Technologies. "Impulse CoDeveloper C-to-FPGA Tools." Internet: www.impluseaccelerated.com/products_universal.htm, 2013 [October 10, 2013].
- [9] Hernandez, R.; Qing Yang; He Huang; Fan Zhang; Xiaorong Zhang, "Design and implementation of a low power mobile CPU based embedded system for artificial leg control," Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE , pp. 5769-5772, 3-7 July 2013.
- [10] Yi-Hung Wei; Quan Leng; Song Han; Mok, A.K.; Wenlong Zhang; Tomizuka, M., "RT-WiFi: Real-Time High-Speed Communication Protocol for Wireless

Cyber-Physical Control Applications,” Real-Time Systems Symposium (RTSS), 2013 IEEE 34th, pp.140-149, 3-6 Dec. 2013.

- [11] J. H. Friedman, “Another Approach to Polychotomous Classification,” Stanford University, Stanford, CA, 1996.
- [12] Hernandez, R.; Faella, J., “Towards policy and guidelines for the selection of computational engines,” Systems Conference (SysCon), 2013 IEEE International , pp.88-95, 15-18 April 2013.

MANUSCRIPT 3

**A Reconfigurable Multiclass Support Vector Machine Architecture for
Real-Time Embedded Systems Classification**

by

¹Jason Kane, Robert Hernandez, and Qing Yang

is published in *the proceedings of the 23rd IEEE International Symposium on Field-
Programmable Custom Computing Machines (FCCM 2015)*,
Vancouver, CA, 2015. p. 244-251.

¹ Jason Kane, Qing Yang, and Robert Hernandez are with Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, 02881, Email {jkane, qyang, rhernandez}@ele.uri.edu.

Abstract

A great architectural challenge facing many of today's embedded systems is in combining physical sensory inputs with often power constrained computational elements to timely and accurately achieve an understanding of the current operating environment. Classification processing is one manner in which some designs accomplish this task. Support Vector Machines (SVMs) encompass one field of classification that has proven to yield high accuracy and has recently seen increased widespread use, however, the required computational intensity places a challenge on computer architects to design a hardware structure capable of performing real-time classifications while maintaining low power consumption. This paper proposes the first ever fully pipelined, floating point based, multi-use reconfigurable hardware architecture designed to act in conjunction with embedded processing as an accelerator for multiclass SVM classification. Several tasks, involving an extensive sample set of over 100,000 classifications, are evaluated for speed and accuracy against the same suite running on both an Intel Core i7-2600 system with libSVM and an Nvidia GT 750M graphics card GPU. Our promising results show that our architecture is able to achieve a measureable speed increase in performance of up to 53x while matching 99.95% of all libSVM predictions.

3.1 Introduction

Recently, classification processing has become a growing field of interest for many embedded systems. In the past, pattern recognition techniques have been employed to achieve good classification accuracy in various areas, including data mining [1], artificial limb control [2], and security systems [3]. One popular subset of machine learning classifiers, support vector machines [4] (SVMs), have been shown to achieve high accuracy without the need of complex parameter tuning that is found in some neural networks [5]. While they yield strong results, the computation process of classification suffers greatly from a large number of iterative mathematical operations and an overall complex algorithmic structure. Traditional processors with limited parallelization in their data pipelines have exhibited poor real-time performance, giving rise to the challenge for computer architects to devise a classification engine suitable for embedded systems that is both performance and energy optimized.

In this paper, we present an accelerator-based *reconfigurable real-time* architecture for the feed forward phase of SVM classification, R^2SVM . Accelerator-based architectures [6] are an area of potential growth for embedded systems, yielding solutions to the issues of power, space, and timing constraints. They offer a solution to improving energy efficiency and performance by optimizing logic to perform specific tasks. To date, only limited such architectures have been studied, which either target a specific SVM task or trade off important factors, including precision, accuracy, speed, and general purpose use. Our paper makes the following contributions to the community:

(C1) A real-time energy efficient, accurate, general purpose run-time reconfigurable accelerator for the purpose of multiclass SVM classification. Unlike all existing works, our architecture is designed to work with four of the most commonly used SVM kernels, and any number of features and classes, up to a maximum specified at synthesis. This allows for multiple diverse SVM workloads to be targeted.

(C2) We have developed a fully functional prototype of the R²SVM architecture in an FPGA, which will soon be made publically available. Our unique design uses model input data identical to that provided by libSVM [7], allowing for direct performance comparisons.

(C3) We present data from several well-known machine intelligence benchmarks to demonstrate the benefits of the R²SVM architecture over a variety of scenarios. To the best of the authors' knowledge, this is the first work to provide comparison results for multiclass classification with SVM among FPGA, CPU, and GPU hardware.

The remainder of this paper is organized as follows. In the next section we briefly discuss the background and related work in the area of SVM classification and SVM hardware optimization. Section 3.3 describes the system architecture of R²SVM in detail. In Sections 3.4 and 3.5 we introduce our testing methodology and discuss our performance results. Last, in Section 3.6, we present our conclusions and discuss future work in this area.

3.2 Background and Related Work

3.2.1 Overview of SVM Classification Process

Support vector machines are a form of supervised machine learning, whose current soft margin incarnation was pioneered by Vapnik and Cortes [4].

Classification is divided into two phases, a training phase and a test phase, also known as the feed forward phase. During the training phase, a set of data is supplied to a learning algorithm in the form of real vector-class pairs to create a model, which is later required for use by the feed forward phase. Given an accurate, ideal set of labeled input data, model creation in traditional SVMs is generally a one-time process that can be performed prior to run-time [2] [3]. The feed forward phase, however, is often run periodically or as needed to perform classifications requiring high performance and real-time processing. We therefore concentrate on optimizations of this phase of the classification process in this paper.

The feed forward phase can be viewed as a two-step process. First, data of interest must be collected by some sensory means and relevant attributes/features extracted. Feature extraction itself is a task that is often highly tailored to the type of data being collected [8]. Features presented to classification may change over time as better indicators are discovered or redundant/misleading elements are removed, and thus extraction is best left to general purpose processing. In contrast, SVM classification is a redundant operation that, if accelerated in hardware, an embedded system could serve to benefit. Following feature extraction, the SVM classification algorithm, known as the decision function (3.1), is run in which stored model data is used to characterize the set of gathered features into classes.

$$f(x) = \text{sign}(\sum_{i=1}^l \alpha_i y_i k(x, x_i) + b) \quad (3.1)$$

A vector consisting of multiple feature elements, x , is supplied to (3.1) and a binary classification is performed. For each iteration of the summation in (3.1), a

stored model support vector $\mathbf{x}_i$, is presented along with feature vector $\mathbf{x}$ to the chosen kernel function $\mathbf{k}(\mathbf{x}, \mathbf{x}_i)$.

Kernel functions [9] are special mapping functions that efficiently map non-linear datasets to a highly dimensional linear feature space. They allow for support vector machines, which were originally developed for linear classification, to also perform non-linear classifications. The particular kernel to employ is entirely dependent on the characteristics of the data in use and is beyond the scope of this paper. Four common kernels implemented by our FPGA design include linear (3.2), polynomial (3.3), Gaussian radial basis function (RBF) (3.4), and sigmoid (3.5).

$$\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j) \quad (3.2)$$

$$\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{a}\mathbf{x}_i^T \mathbf{x}_j + r)^d \quad (3.3)$$

$$\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = e^{(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)}, \text{ where } \gamma > 0 \quad (3.4)$$

$$\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{a}\mathbf{x}_i^T \mathbf{x}_j + r) \quad (3.5)$$

The result of the kernel function in (3.1) is multiplied by the coefficient weight associated with the support vector, $\mathbf{a}_i$, and class label, $\mathbf{y}_i$, where $\mathbf{y}_i = +1$ to denote one class and $\mathbf{y}_i = -1$ is used to denote the other class. After $\mathbf{l}$ iterations, where $\mathbf{l}$ refers to the total number of support vectors in the given model, the bias value associated with the class comparison is combined with the summation. The resulting classification is then determined by comparing the final value against zero.

SVMs have been further extended to solve problems requiring classification among more than two classes of data. Similar to libSVM, our design implements the one against one approach [10]. One against one SVM performs multiple back-to-back

binary classifications. The winner of each classification receives a vote and the class with the greatest number of votes is presented as the resultant classification.

3.2.2 Related Work

Researchers have repeatedly employed computer systems to accelerate the processing of the SVM decision function. Implementations have targeted CPUs [7], GPUs [11] [12] [13], as well as custom accelerator hardware. Many existing custom hardware implementations focus on accelerating the decision function for a specific task [14]. These designs cannot easily be reused for other purposes. Often-times they support a small, fixed number of classes [15] or make use of low precision calculations [16] for speed improvements. Most existing designs attempt to limit or refrain from using more precise floating point hardware for several reasons, including space and design complexity [17]. Unlike existing works, the reconfigurable architecture provides the ability to accurately compute multiple diverse workloads in succession. This is desirable in any real-time system that requires numerous machine learning decisions to be made within a given timeframe.

3.2.3 The Role of Precision in SVM Classification

While researchers have exploited fixed point and other reduced precision numerical systems to implement efficient, task specific SVM implementations, the exact nature of the relationship between SVM classification accuracy and precision has only been vaguely addressed by existing literature. Rather than submit to trial-and-error tactics in determining the required arithmetic precision, [18] delves into the theory behind this topic. Through running several experiments, the authors conclude that SVM parameter quantization and rounding error are the two main factors that

contribute to determining the minimal amount of precision required for SVM computations, both of which are largely data set driven. For the limited data sets studied in the paper, the authors found that a minimum of 15 bits of floating-point precision were required to ensure no loss in accuracy. In devising an architecture meant to tackle a wide variety of classification problems, great flexibility in precision will be required to ensure high accuracy under all conditions. For this reason, while not the most resource or power-efficient approach, we have elected to maintain the IEEE-754 floating point number system when performing computations for classification in our architecture.

3.3 Architecture Design

In this section, we describe the design decisions made in the construction of our unique architecture. We envision the R^2SVM architecture as being implemented as a critical processing section in a variety of real-time embedded devices. Classification is often a very resource and time-intensive process for general purpose computing. By relieving this burden in a power/area constrained environment, our architecture provides an attractive alternative. Our main intention in designing R^2SVM was to develop an accurate, fast, highly efficient SVM classifier, capable of meeting the demands of a diverse set of workloads that may be encountered by real-time systems.

To achieve a level of accuracy on-par with existing software packages, all mathematical operations related to the evaluation of the SVM decision function are carried out using IEEE-754 floating point arithmetic. While fixed point may offer speed and resource benefits, popular software-based packages such as libSVM use floating point for their calculations. Precision, dynamic range, and error accumulation

in evaluating the SVM decision function will differ between the two methods, leading to inconsistent results when making comparisons. A secondary benefit to utilizing floating point arithmetic is that most software packages store their models and kernel parameters in this format. Thus, through maintaining consistent IEEE-754 operations throughout our design, R^2SVM 's internal calculations should identically mirror those performed by the software packages. This is a particularly desirable trait for developers wishing to port their software-developed SVM models to embedded devices and achieve similar results.

Meeting hard real-time speeds is achieved through extensive pipelining and parallelism. We designed R^2SVM such that a given model will always take a known number of clock cycles to compute a result. We will show in the subsections that follow that access delays to both parameter and model data are kept to a minimum to minimize pipeline stalls. In Section 3.5, our results clearly indicate the ability of the architecture to maintain real-time response.

The final goal of our architecture is to provide seamless support for a diverse set of classification workloads through offering reconfiguration at run-time. Versatility is available in this area in that header information from the model is used during run-time to reconfigure the number of classes to be used in the current classification. We will demonstrate in the subsections that follow that while our hardware may be synthesized for a maximum of K classes, models ranging anywhere from 2 to K classes may be used without any performance degradation. This versatility is also extended to the number of features in the system. So long as an SVM model applies zero weights to unused feature inputs during kernel calculations, any number of

Figure 3.1. R^2 SVM FPGA Architecture Overview

features up to the maximum allowed by the synthesized hardware implementation may be used. We feel that the creation of a general purpose hardware SVM classifier requires flexibility and locking the design to specific characteristics, as has been done in the prior work of Section 3.2, would defeat this purpose. In addition to allowing for modifications to the number of classes/features, kernel selection and parameters are also reconfigured at run-time whenever classification processing is requested.

R^2 SVM, as seen in Fig. 3.1, consists of three major design portions based on the algorithmic structure of (3.1): A kernel value calculation block, a coefficient weighting engine, and a voting unit. The operation of the proposed architecture behaves as follows: Sets of model data derived from training events and computed on a separate system is sent to the hardware and stored prior to use. At run-time, kernel parameters along with features are periodically provided to the hardware. Additional control signals inform R^2 SVM of the model to be used along with the number of features and classes for the current task at hand. This allows for different classification tasks to share the same hardware in the event that a system requires multiple classifications. Upon receipt of a “Begin” pulse, R^2 SVM will initiate computing of (3.1) in a pipelined fashion, starting with kernel values. When the first SVM kernel value is available, coefficient weighting starts. As the scalar values associated with the output of the decision function become available, votes are taken among classes. When the last vote has been cast, R^2 SVM determines the winner,

informs the higher level system of its completion, and waits to receive the next set of parameters.

3.3.1 Kernel Evaluation

The first step in the evaluation of (3.1) is to use an appropriate kernel function to map an input vector set of features to a scalar value, which can later be used by the coefficient weighting process. Because of the latency involved with floating point calculations, our kernel is fully pipelined such that, once populated, the hardware will be able to output a new kernel result on each clock cycle. To ensure that the coefficient weighting unit is not overrun by incoming kernel values, a feedback mechanism is incorporated to stall the pipeline if necessary. The design periodically reconfigures itself according to run-time classification parameters to support either the linear, polynomial, Gaussian radial basis function, or sigmoid kernel, given by (3.2) through (3.5).

We attempt to improve speed and routing in the design by reusing common portions of the data path where possible. Commutative properties are exploited to reduce the number of floating point units required. Examining the kernel equations, all involve some specialized input operation, followed by a summation, which is in turn followed by a specialized output operation. This is reflected in our design by Fig. 3.2. Here, the design uses an adder tree to perform the task of summation. This removes the need for any iterative stalling when producing a kernel output.

Figure 3.2. Pipelined Kernel Structure

Our input block to the kernel pipeline operates based on the kernel selected. One of two operations will occur: the multiplication of two vectors, or the subtraction and squaring of those vectors. The first is required by the linear, polynomial, and sigmoid kernels, and the latter is used by the Gaussian radial basis function. Fig. 3.3 shows our output block architecture. Again, based on kernel selection, different operations will occur. For the linear kernel, the output from the adder chain is directly written to the kernel result register. The polynomial kernel requires that the output of the adder tree be scaled by some value α and an offset r be added, prior to raising the resulting number to a power, d . To determine a number raised to a power, the relationship given by (3.6) is used. One issue that arises is that the natural log of a negative number is undefined. Because this equation only holds true for a positive number x , if x is negative, the number must be multiplied by -1. The negation is reapplied to the

Figure 3.3. Kernel Output Block

result at the end of calculations if and only if the power being raised to, y , is an odd power.

$$x^y = e^{(y \cdot \ln(x))} \text{ , where } x > 0 \quad (3.6)$$

The Gaussian radial basis function is able to leverage a large portion of the polynomial data path. Parameters α and r are fixed at 1.0 and 0.0 respectively, and the natural log unit is bypassed. The resulting data path produces the expected RBF result.

With the sigmoid kernel, alternative mathematical operations are once again performed. To calculate sigmoid, we use the trigonometry identity of (3.7).

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x}-1}{e^{2x}+1} \quad (3.7)$$

The data path chosen for sigmoid initially follows that of the Gaussian radial basis function, with the exception that parameters α and r are specified. The output of the natural exponent unit is fed into additional logic to complete the determination of the sigmoid kernel value.

Figure 3.4. Coefficient Weighting Unit

3.3.2 Coefficient Weighting

Once scalar kernel values are available, the coefficient weighting phase may commence (Fig. 3.4). Given a model with k classes, each support vector will have $(k - 1)$ associated coefficients. These coefficients must be multiplied by their associated kernel value. By instantiating $(k - 1)$ multiply-accumulate (MAC) blocks, each kernel value is only required to be presented to the hardware once. This reduces circuit complexity as well as iterative behavior and redundant memory access. While a particular model may have k classes, our hardware is designed to handle a maximum K classes, where $2 \leq k \leq K$. MAC blocks are reconfigured to be enabled/disabled such that only those required for the current classification task are used.

When the first kernel value associated with a particular class comparison is made available to the coefficient weighting unit, a header word is read from RAM to indicate the number of coefficients associated with it. In addition, dependent on the current comparison, from zero to $(k - 1)$ negated bias values are loaded to initialize the MAC accumulators. This is done to eliminate the need for an additional floating point subtraction unit at the output of each MAC. The class coefficients and their associated kernel values are then presented over consecutive cycles to the input of the pipelined MACs. When all multiplications required for the current class comparison have completed, all $(k - 1)$ MAC blocks store their current accumulator values to dedicated temporary registers simultaneously. A flag is set signifying that these registers are full and cannot yet be re-written to, and the MAC units begin to repeat the same process for the next class comparison.

The control logic in Fig. 3.4 is used to iteratively retrieve the temporary data and either store it in the intermediate sums RAM or forward it to create a final summation for comparison. The control logic maintains knowledge of the two classes being compared and uses a hash function, given by (3.8), based on the triangular number sequence to quickly index into the intermediate sums RAM. The entries in this RAM consists of a floating point data field along with a single valid bit. If the valid bit is not set, then incoming data from a MAC temporary register is stored to the intermediate sums RAM and the valid bit is set. Otherwise, the data in RAM is retrieved and presented to a floating point addition unit along with the incoming temporary register data to compute a final sum for class comparison. Also forwarded with this sum are the two corresponding classes, a valid flag, and a flag to indicate

Figure 3.5. Voting Unit

whether or not this is the last comparison. These values are all sent to the voting unit for class determination.

3.3.3 Voting

The voting unit, as seen in Fig. 3.5, maintains separate counters for each class. When a comparison value arrives from the coefficient weighting engine and the valid flag is set, it is compared with a floating point value of zero. As mentioned earlier, two classes are forwarded to the voting unit to inform it of the counters involved in the current comparison. If the value is greater than zero, the lower numbered class receives a vote. Otherwise the higher numbered class is incremented. On each iteration, the elected class is updated by checking for the lowest labeled class holding the maximum number of total votes. When the unit receives the “last comparison” flag, the final vote is cast, and a signal is pulsed to inform external hardware that the classification has completed and the elected class is valid.

3.3.4 Prototype Implementation Details

Our prototype was implemented in an Altera 5SGSMD5N Stratix V FPGA using a combination of Altera Megafunction IP blocks [19] and VHDL. The 5SGSMD5N was selected due to its abundance of DSP resources, required to efficiently implement floating point hardware logic. Altera IP blocks were used to provide floating point calculations in most instances due to their tested and proven accuracy, level of optimization, and relatively low latency.

Since Altera IP blocks only support single and double precision IEEE-754 calculations, unique reduced precision floating point operations like those in [18] are not possible in our implementation. Of the two available numerical formats, we decided to use single precision due to its lower latency, lower use of resources, and the fact that the authors in [18] determined that enough bits of precision should be available to accurately perform most classification tasking.

In designing our VHDL codebase, we created five top-level generics to control synthesis of the design. This allowed for us to characterize the optimal performance when run with a variety of input models. The five generics include: the maximum number of features, the maximum number of classes, the maximum number of models supported by the design, the total number of support vectors required for storage, and the number of support vectors present in the largest model. The features and class parameters are used to construct an optimal set of hardware that will function up to the defined limitations. In the case of the kernel unit, our VHDL code uses the maximum number of features to construct an optimal-sized adder tree. The coefficient weighting and vote units use the maximum number of classes to determine the required amount

of redundant MAC blocks and voting counters. The remaining three parameters are used to characterize the memory subsystem.

While not realistic for all SVM datasets, to simplify data access for the initial prototype, all storage was implemented using internal FPGA block RAM. The total number of support vectors parameter, in combination with the maximum number of classes determines the amount of memory required for model data storage. To access a particular model's data, an index to the start of that data is sent to the hardware along with kernel parameters. The maximum number of models parameter is used to determine the maximum number of supported models, and therefore the total number of unique model indices that may exist in the system. The final parameter, the maximum number of support vectors in the largest model, is used to determine the amount of the RAM required by the kernel unit. While the kernel does not require any RAM according to our design methodology, one was implemented to accommodate post-run debugging of kernel results. While included in our performance results, the RAM could be easily removed in a mature version of the hardware implementation. This should further reduce the area and power required by the design.

3.4 Evaluation Methodology

In evaluating the performance of our design, we wanted to demonstrate scalability as well as accuracy and efficiency in solving complex classification tasks. We have selected six workloads and one embedded system case study to prove the ability of our architecture to meet the demands of real-time systems.

3.4.1 Workload Characteristics for Benchmarking

We first selected a diverse set of existing real-world datasets obtained from both the UCI Machine Learning Repository [20] and the Statlog collection [21]. Details regarding the number of classes and features in the datasets can be seen in Table 3.1 in the Results section.

Adult. This is a modified version of the UCI Adult dataset, obtained from the libSVM authors [7]. We have selected this two-class dataset as a means to baseline performance against the publically available KMLib GPU library.

DNA. This workload was chosen to demonstrate performance with a large number of features (180). The classification problem is related to molecular biology and could represent a portion of a medical system: given a DNA sequence, the task is to determine the boundaries between introns and exons.

Letter. The purpose of this classification task is to determine the English alphabet letter corresponding to a given set of 16 attributes. 20 different fonts were used in the creation of the dataset to add to the complexity. This dataset could be viewed as an example of a potential embedded robotics application: automating the process of text recognition as part of a control loop in understanding the operating environment.

Shuttle. This dataset was supplied by NASA and consists of shuttle control data with regard to the position of radiators, essentially an embedded real-time defense system. It was chosen to aid in characterizing the error rate of our system, as a large number of test cases were available.

Satimage. This workload is used to identify different types of terrain present in a Landsat satellite image. Classes include red soil, cotton crop, vegetation stubble, grey soil, mixture, damp grey soil, and very damp grey soil. If the data were gathered and

analyzed in real-time, the dataset could be representative of an embedded communications system.

Vowel. This dataset consists of features gathered from spoken British English vowels from several speakers. Speech recognition could be a critical task for a real-time embedded system functioning as an assisted living device. The Vowel dataset was selected in particular to examine performance with a small number of support vectors.

3.4.2 Case Study – A Human-Computer Control Interface

To fully evaluate the utility of R^2SVM , we also examine its use in a real-world embedded system. The selected system is a human-computer wireless interface that takes classification of lower arm electromyography (EMG) signals to perform user-intended cursory actions on a personal computer. The intent of this medical device is to allow amputees to seamlessly control a computer much like an able-bodied subject.

The overall setup of our system is as follows: EMG sensors are employed to gather data from the following muscles: *Extensor Carpi Ulnaris*, *Extensor Carpi Radialis*, *Flexor Digitorum Superficialis*, *Flexor Carpi Radialis*, *Flexor Carpi Ulnaris*. Using an Intel Atom embedded processor, four features are extracted from each muscle: mean absolute value, zero crossings, sign/slope changes, and length. Classification is then performed among seven classes to determine the intended cursor action to be performed. The seven classes include: Cursor Up, Cursor Down, Cursor Left, Cursor Right, Left Click, Right Click, and Rest. To guarantee a timely response for the governing top-level control algorithm, classifications must be made available every 20ms. The results of classification are transmitted periodically over a wireless interface to a nearby computer where the intended operation is executed.

Our prototype system under development conservatively allocates 60% of the 20ms window to classification. In our testing, we run three trials from collected data traces to show that R^2SVM is capable of meeting the real-time demands of the classification portion of such an embedded system, while operating at a very low power level.

3.4.3 Evaluation Platforms

Models were constructed from the datasets using libSVM. In all tested platforms, models were loaded prior to run-time such that only classification performance would be examined. Dedicated timers were used to determine and record the amount of time taken for each classification. For our testing purposes, we define accuracy as being able to compute the same resulting class as a CPU running the well-known libSVM software package. The extent to which the SVM decision function correctly predicts with a given model is dependent on several factors that are beyond this paper's scope. Because of this, we omit details of the classification accuracy of the models themselves, as this data would not add much value to our SVM performance discussion.

A Dell Optiplex 790 computer running Microsoft Windows 7 Professional 64-bit was utilized to determine the baseline CPU performance with libSVM during the SVM feed forward phase. The system was configured with an Intel 3.4GHz i7-2600 Sandy Bridge Quad Core Processor with 8 Gbytes of Dual Channel 1333MHz DDR3 RAM. While libSVM is a single-threaded library for SVM classification, to the best of the authors' knowledge, there currently exists no equivalent publically available multi-threaded CPU implementation.

The GPU selected for benchmarking was an Nvidia Geforce GT 750M graphics card (Kepler architecture) coupled with an Intel i7-4850HQ as the supporting CPU. The mobile card was chosen because of its abundance of resources (384 cores) coupled with fairly low power consumption (35 Watts). While there are many publically available software packages that use GPUs for optimizing the SVM training phase, few exist that perform the feed forward phase of computation. To the best of the author's knowledge, KMLib [11] and GPUSVM [12] are the only two publically available packages to support current GPU drivers, and both are only capable of performing nonlinear feed forward SVM classification with at most two classes. In addition, only KMLib supports the libSVM algorithm for the SVM training phase, allowing for comparable models to be used. Unfortunately KMLib currently only supports the Gaussian kernel in this mode of operation. A private libSVM-compatible GPU optimized feed forward library from the authors of [13] was used for our evaluation, as it supports all four kernels implemented by our FPGA design. In the Results section, we compare this GPU library's relative performance to KMLib using the Adult dataset.

Our Altera 5SGSMD5N Stratix V FPGA was recompiled separately for each of the datasets to determine the maximum performance in each case and demonstrate the scalability of our prototype implementation. While we could have synthesized a single design with the maximum number of features/classes specified to support all testing, we instead compiled multiple times to examine the impact on resource utilization as the number of features/classes increased. In all test cases we elected to

fix the main clock speed at 60MHz. The same models and kernel parameters developed for the Core i7 were sent to our FPGA prior to run-time.

3.5 Results

3.5.1 Hardware Synthesis

The full results of optimal hardware synthesis for our test cases are omitted for space, but are briefly described here. As anticipated, the number of DSP blocks required was seen to grow with an increasing number of both features and classes. This was most evident when comparing the DNA design (202 DSP), which had the highest number of features, against the one created for Vowel (41 DSP). Because of our design's scaling nature, supporting more input features requires that more floating point addition units are implemented in the kernel hardware. Likewise, supporting a greater number of classes requires additional floating point multiplication and addition modules to be present in the coefficient weighting portion of the design. A direct correlation was also seen between the amount of block memory bits in use and the number of support vectors present in the model. This is because our present implementation relies on internal block RAM for storage and auto-sizes the resources required. Referring ahead to Table 3.1, Letter has the largest model, and, as expected, Letter required more block RAM than any other test case. Looking at system power usage, our estimated TDP (Thermal Design Power) as provided by Altera's PowerPlay Analysis tool ranged from 1.18 Watts to 2.78 Watts. In comparison, the GPU and Intel Core i7 system being used for benchmarking have dramatically greater TDPs of 35 and 95 Watts respectively: a difference of 12x/34x of our highest TDP.

Table 3.1. FPGA Performance in SVM Feed Forward Phase

Trial (#Class, #Feat) [Source]	Sub-trial	#SV	FPGA Time (μ S)	Average FPGA Speedup		# Test Trials	Acc. FPGA (%)
				Over CPU	Over GPU		
Adult (2,123) libSVM, UCI	Lin	600	13.4	37.3x	14.7x	1000	100
	Poly	705	14.0	39.5x	23.3x	1000	100
	RBF	785	17.2	39.6x	14.0x	1000	100
	Sig	687	13.5	42.8x	17.2x	1000	100
DNA (3,180) Statlog	Lin	402	10.0	39.7x	17.7x	1187	100
	Poly	1053	21.8	39.1x	18.3x	1187	100
	RBF	696	13.5	53.7x	21.1x	1187	100
	Sig	444	9.4	48.7x	19.7x	1187	100
Letter (26,16) Statlog	Lin	6060	125.0	9.0x	6.6x	5000	99.97
	Poly	4408	100.0	10.0x	13.5x	5000	100
	RBF	5701	120.0	9.6x	7.2x	5000	100
	Sig	14602	266.0	14.4x	13.7x	5000	100
Shuttle (7,9) Statlog	Lin	3994	72.7	5.96x	6.9x	14500	99.97
	Poly	878	21.7	7.4x	14.9x	14500	100
	RBF	2208	43.6	7.1x	8.1x	14500	100
	Sig	4290	78.4	10.2x	8.6x	14500	100
Satimage (6,36) Statlog	Lin	1216	25.6	11.5x	9.5x	2000	99.95
	Poly	1051	21.7	13.8x	18.1x	2000	99.95
	RBF	1165	23.3	13.5x	11.0x	2000	100
	Sig	2468	47.3	15.2x	10.4x	2000	100
Vowel (11,10) UCI	Lin	268	13.7	4.0x	12.1x	463	100
	Poly	299	13.0	5.1x	6.72x	463	100
	RBF	290	56.8	4.0x	11.8x	463	100
	Sig	427	17.1	6.9x	11.4x	463	100
Arm1 (7,20) C. Study	RBF	443	14.1	9.1x	11.3x	1818	100
Arm2 (7,20) C. Study	RBF	572	14.1	8.9x	12.0x	1818	100
Arm3 (7,20) C. Study	RBF	435	14.0	8.6x	11.4x	1818	100

3.5.2 Workload and Case Study Performance Results

The results of testing for all kernels and scenarios can be viewed in Table 3.1. We display both the average time taken for the FPGA to compute the classification as well as the speedup offered by R^2 SVM. In all cases, R^2 SVM was able to predict with 99.95% or higher accuracy the same final class as libSVM on the CPU. The few encountered failures were examined individually to determine the reason for which the

incorrect class was reported. In all instances, failure was determined to be due to the result of floating point rounding error, which accumulates faster on the single precision FPGA. This indicates a near-boundary case for classification, which in a real setting could be remedied through either improvements to model features or kernel parameters.

Additionally, it was found that for our case study we were able to perform each prediction with 100% accuracy and within 14.1 μ S. The time required for classification occupies only 0.07% of the budgeted 20ms mentioned earlier, leaving ample time for accompanying hardware to perform data collection, feature extraction, and control. Leftover time could even be used to send alternative models/kernels to the classification module to assist in strengthening the results.

Average FPGA speed-ups of up to 53.74x over the CPU and 23.33x over the GPU can be observed in Table 3.1. It was found that the GPU library used for testing was a more optimal implementation than the publically available KMLib, which it outperformed by 418.53x on the Adult dataset. Examining both the CPU and GPU data results in detail, a few local maximums exceeding several dozen milliseconds were discovered, reaffirming the need for a dedicated well-bounded accelerator to support real-time tasking. Since the R²SVM design is entirely hardware driven with no facility for preemption, the FPGA classification times given in the table are guaranteed to be the actual times required to evaluate the decision function for a given model.

The amount of time taken for R²SVM to perform a classification should be roughly proportional to the number of support vectors in the system. Features increase

the latency of the kernel function somewhat, as they add additional stages to the addition tree hardware. Once the kernel pipeline is completely filled, the limiting factor in calculations quickly becomes the coefficient weighting unit. In our current design, there are two factors that can impose considerable impacts on latency when calculating weights: the number of classes and the number of support vectors in the system. In our test cases, and in any well-constructed model, the total number of support vectors should be far greater than the number of classes in the system. Thus R^2SVM should perform classifications in about the same amount of time for models with similar numbers of support vectors. We further confirm this fact by comparing two such models: DNA linear (402 SV) and Vowel sigmoid (427 SV). In this instance DNA requires $10\mu s$ and the vowel sigmoid requires a slightly longer $17\mu s$ to complete classification. The slight difference can be attributed to both the extra depth in the DNA kernel's adder tree as well as the extra delay imparted through use of the sigmoid kernel.

To determine why R^2SVM is able to outperform the CPU and GPU, we performed an analysis of their codebase. On the CPU, when libSVM begins a classification, it first computes all kernel values before beginning the weighting process. This results in a huge performance penalty when compared with the FPGA, which is allowed to begin coefficient weighting while kernel evaluation is ongoing. The CPU is limited in parallelization in this regard and must iterate, relying on cache and a higher maximum clock rate to attempt to gain performance. A similar issue occurs in the coefficient weighting section of the libSVM software where $(k - 1)$ weighting coefficients must be applied to all support vectors in the system and

summed. The CPU evaluates each class comparison separately in a linear fashion. This can be a hugely expensive task for any processor. Meanwhile, R^2SVM is constructed to perform $(k - 1)$ weightings and subsequent accumulations in parallel. R^2SVM outperforms the GPU for similar reasons. Unlike the FPGA, the GPU must also perform all kernel operations first followed by weighting. During kernel calculations, a large GPU performance hit is taken when a low number of features are presented. Indeed in many low-feature instances, the CPU outperforms the GPU. This is because the GPU incurs a time penalty with each kernel invocation, and with a small number of features the amount of time to copy the results back to the accompanying CPU cancels any advantage. As evident in the DNA dataset, the FPGA design is still able to outperform the GPU when a large number of features are presented. Thus, it would appear that hardware optimizations to both the kernel and coefficient weighting process have allowed our design to achieve performance levels that rival today's desktop processors and GPUs.

3.6 Conclusion and Future Work

This paper presented a novel design and implementation of R^2SVM , a reconfigurable real-time high performance SVM classification architecture. While providing accuracy comparable to that of available software packages, we have shown that our hardware design is scalable in terms of both features and classes. Using real-word datasets, we implemented our hardware and concretely determined the performance with regards to two of today's popular general purpose processing methods. In all cases, R^2SVM exhibited an increase in speed in comparison to both the CPU and GPU being used for benchmarking, with an average speed-up as high as

53x, and energy savings reaching an estimated minimum of 12x. In future work we will explore alternatives to the kernel hardware. We envision that derivatives of our design will be deployed for use in many diverse settings. We intend to release our design publically under an open source license to promote further innovation in this exciting area.

List of References

- [1] M. Kezih, M. Taibi, "Evaluation effectiveness of intrusion detection system with reduced dimension using data mining classification tools," in *Int. Conf. on Systems and Computer Science (ICSCS)*, 2013, pp.205-209.
- [2] H. Huang, Y. Sun, Q. Yang et al., "Integrating neuromuscular and Cyber Systems for neural control of artificial legs," in *ACM/IEEE Proc. of ICCPS*, Apr 2010, pp. 129-138.
- [3] S. Tang, X. Li, H. Zhang, et al. "TelosCAM: Identifying Burglar through Networked Sensor-Camera Mates with Privacy Protection," in *IEEE Real-Time Systems Symposium (RTSS)*, 2011, pp. 327-336.
- [4] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [5] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning*, 1998, pp. 137–142.
- [6] S. Patel and W.-m. Hwu, "Accelerator architectures," *Micro, IEEE*, vol. 28, no. 4, pp. 4–12, 2008.
- [7] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, 2011.
- [8] H. Liu and H. Motoda, *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.
- [9] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge, UK: Cambridge University Press, March 2000.
- [10] J. H. Friedman, "Another approach to polychotomous classification," Department of Statistics, Stanford University, Tech. Rep., 1996.

- [11] K. Sopyła, P. Drozda, P. Górecki, “SVM with CUDA Accelerated Kernels for Big Sparse Problems,” in *Proc. of the 11th Int. Conf. on Artificial Intelligence and Soft Computing*, 2012, pp. 439-437.
- [12] B. Catanzaro, N. Sundaram and K. Keutzer, “Fast Support Vector Machine Training and Classification on Graphics Processors,” in *Int. Conf. on Machine Learning*, 2008, pages 104-111.
- [13] J. Faella, “On Performance of GPU and DSP Architectures for Computationally Intensive Applications,” M.S. thesis, Dept. Elec. Eng., Univ. of RI, Kingston, RI, 2013.
- [14] T. Kryjak, M. Komorkiewicz, and M. Gorgon, “FPGA implementation of real-time head-shoulder detection using local binary patterns, SVM and foreground object detection,” in *Conf. On Design and Architectures for Signal and Image Processing (DASIP)*, 2012, pp. 1–8.
- [15] C. Kyrkou and T. Theocharides, “A parallel hardware architecture for real-time object detection with support vector machines,” *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 831–842, 2012.
- [16] S. Cadambi, I. Durdanovic, V. Jakkula, et al. “A Massively Parallel FPGA-Based Coprocessor for Support Vector Machines,” in *17th IEEE Symp. On Field Programmable Custom Computing Machines*, 2009, pp. 115–122.
- [17] D. Anguita, L. Carlino, A. Ghio, and S. Ridella, “A fpga core generator for embedded classification systems,” *Journal of Circuits, Systems and Computers*, vol. 20, no. 2, pp. 263–282, 2011.
- [18] B. Lesser, M. Mücke, W. Gansterer, “Effects of Reduced Precision on Floating-Point SVM Classification Accuracy,” in *Proc. of the Int. Conf. on Computational Science (ICCS)*, 2011, pp. 508-517.
- [19] Altera Corporation, “Altera megafunctions,” May 2014. Available: <http://www.altera.com/products/ip/altera/mega.html>
- [20] A. Asuncion and D. Newman, “UCI machine learning repository,” 2007. Available: <http://archive.ics.uci.edu/ml/>
- [21] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, Eds., *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, New York, NY, 1994.

MANUSCRIPT 4

Reflex-Tree: A Biologically Inspired Architecture for Future Smart Cities

by

¹Jason Kane, Bo Tang, Zhen Chen, Jun Yan, Tao Wei, Haibo He, and Qing Yang

is published in *the proceedings of the IEEE 44th Annual International Conference on*

Parallel Processing (ICPP'15),

Beijing, China, 2015. p. 360-369.

¹ Jason Kane, Bo Tang, Zhen Chen, Jun Yan, Tao Wei, Haibo He, and Qing Yang are with Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, 02881, Email { jkane, btang, chen, jyan, wei, he, qyang }@ele.uri.edu.

Abstract

We introduce a new parallel computing and communication architecture, Reflex-Tree, with massive sensing, data processing, and control functions suitable for future smart cities. The central feature of the proposed Reflex-Tree architecture is inspired by a fundamental element of the human nervous system: reflex arcs, the neuromuscular reactions and instinctive motions of a part of the body in response to urgent situations. At the bottom level of the Reflex-Tree (layer 4), novel sensing devices are proposed that are controlled by low power processing elements. These “leaf” nodes are then connected to new classification engines based on machine learning techniques, including support vector machines (SVM), to form the third layer. The next layer up consists of servers that provide accurate control decisions via multi-layer adaptive learning and spatial-temporal association, before they are connected to the top level cloud where complex system behavior analysis is performed. Our multi-layered architecture mimics human neural circuits to achieve the high levels of parallelization and scalability required for efficient city-wide monitoring and feedback. To demonstrate the utility of our architecture, we present the design, implementation, and experimental evaluation of a prototype Reflex-Tree. City power supply network and gas pipeline management scenarios are used to drive our prototype as case studies. We show the effectiveness for several levels of the architecture and discuss the feasibility of implementation.

4.1 Introduction

Urbanization—the demographic transition from small, rural communities to large urbanized cities—is associated with shifts from an agriculture-based economy to one grounded in mass industry, technology, and service delivery. The combined population of the world’s urban areas is predicted to surpass six billion by 2050 [1-3]. With this accelerated growth will come unprecedented increases in the consumption of resources and services, leading to material and energy shortages, which will in turn ultimately drive climate change [4-7]. The “Smart City” is an emerging concept aimed at dramatically enhancing the efficiency, sustainability, and safety of these urban communities. Integrating infrastructure and services into a cohesive whole allows them to be both monitored and managed using intelligent devices and systems [8]. Smart cities encompass enhancements in energy, building, mobility, healthcare, infrastructure, technology, governance, and citizenry (Figure 4.1). The technologies driving these enhancements have a predicted collective market worth of \$3.3 trillion

Figure 4.1. Example Components of a Smart City

by 2025 [9].

Accurate, distributed, and real-time sensing platforms with high performance computing and communication structures are the core components of the smart city concept, enabling municipalities to monitor and respond to changing conditions within the community in real-time. A prime example is the use of a large amount of distributed sensors to monitor critical elements of public infrastructure (bridges, tunnels, gas and oil pipelines, etc.) and massive parallel and distributed computing infrastructures to intelligently process vast amounts of data collected [10]. Such infrastructure enables both appropriate and efficient resource allocation during normal operation and quick response in real-time to storms, earthquakes, or other natural disasters [11-13].

Realizing the “intelligent” infrastructure at the foundation of future smart cities presents significant parallel processing challenges:

Firstly, widely distributed, real-time, precise, and massively parallel sensor networks are essential to the success of smart city systems. Most of the existing proposed sensing network architectures [14][15] follow a centralized approach for data gathering and processing, where information is gathered through various sensors and routed directly to the cloud. They lack the sensing capacity, high spatial and temporal resolution, heterogeneous sensing capability, and reliability necessary to meet the critical requirements of future smart cities. Alternatively, a decentralized, layered approach could offer much in terms of fault tolerance for data gathering. A failure in one area of the network will not impact other adjacent sites, resulting in improved up-time. In addition, filtering the data being sent throughout a layered

hierarchy should minimize the amount of data needed to be sent to the highest layer, reducing the overall bandwidth required for operation, as well as the associated costs.

Secondly, parallel computing nodes must be deployed at various geographical locations in order to process the massive volume of data generated by distributed sensors in real-time. Often the dataset is not only immense in volume, but also heterogeneous in nature, representing the status information of public infrastructure, healthcare systems, transportation networks, energy distribution, and other critical systems. Currently, no computing platform based on massively parallel and geographically distributed nodes exists that is capable of delivering the processing performance consistent with the low power envelope demanded by smart cities to reduce their operational costs.

Thirdly, in order to provide distributed, real-time control and decision making in complicated urban environments, advanced machine intelligence with spatial-temporal association and complex system behavior analysis is essential. Computational complexity of such machine intelligence increases from broadly distributed local nodes (neighborhood, community, and districts) to a central node that requires citywide control and decision-makings with data volumes representing the entire urban area.

Finally, and more importantly, management and control functions must be implemented at each distinct level of an urban environment, from individual elements of infrastructure to blocks, neighborhoods, districts, and the entire city. As an example, if sufficient computing intelligence is partitioned to lower levels, a local gas leakage could be quickly and efficiently handled by an edge-computing node,

eliminating the time required for centralized citywide control decisions to take place and be communicated.

To tackle these challenges, we propose a transformative parallel computing and communication architecture specifically suitable for smart cities, the Reflex-Tree, which will be described in detail in Section 4.3.

This paper makes the following contributions:

- Introduction of a novel deployment of a distributed fiber-optic sensing network (FOSN), able to provide timely measurements of both temperature and strain measurements at millimeter level spatial resolution.
- The creation of a unique hierarchical four layer, decentralized, large scale, and application specific parallel computing and communication structure capable of carrying out sensor-based decision-making processes.
- Detailed simulations of Reflex-Tree in two real-world problems: city power supply network and natural gas pipeline management. We describe how our architecture efficiently detects and handles problems that arise.

4.2 Related Work

4.2.1 Smart City Infrastructure

A number of existing works examine potential applications and control infrastructure for the smart cities of the future. For example, the authors of [16] propose a storage system application for city-wide video surveillance. Data from high definition real-time security cameras is sent to a cloud storage system and split into a database based on metadata information. Dividing the streams was shown to allow for fast retrieval when on-demand classification analysis is required by the cloud.

Another suitable application for smart cities is smart lighting [17]. Here the authors discuss an IP based approach to lighting control. The intention is to implement an efficient method of automated lighting control that will ultimately result in cost savings. Unlike these two simple applications, we intend to present a high level, broad architecture, capable of handling a multitude of city tasking.

More related to the work we are pursuing, the authors of [14] propose a framework for effective disaster management. They suggest using crowd sourcing as a method for reporting on environmental and other conditions. Mobile or other wireless devices are used to relay information to the cloud, where it is assessed, filtered, and correlated with data from known reliable infrastructure sensors installed on city buildings or public transportation vehicles. Similarly, the work in [15] proposes another unique implementation of a cloud based management system used to gather sensor data from both citizens and places of infrastructure. The information undergoes sorting and classification at the cloud, where the appropriate public agency is contacted in the event a response is required.

To the best of our knowledge, our unique multi-tiered approach for city management differs from all existing works. In section 4.3 we describe the framework of our parallel architecture, which to an extent allows lower level nodes the ability to be responsible for decision making, instead of sole reliance on the cloud. This difference should work to increase critical response speed as well as scalability and fault-tolerance. Not only can our architecture be relied on for disaster management, but it can also be extended to day-to-day management of city infrastructure.

4.2.2 Sensor Networks for Smart City Applications

Two general types of sensor networks hold significant promise to be widely adopted by future smart cities: active wireless sensor networks (WSN) and passive FOSNs. Currently, WSN, in which each sensor node is both a transducer and radio frequency (RF) transceiver, has been the most widely utilized network type [13, 18-20]. Examples include Adhoc networks and the more recent radio-frequency identification (RFID) systems [21-29]. The most distinct advantage of a WSN is that the formation of a network can be accomplished in-situ, making a WSN favorable for use with mobile devices. However, as an active network, each sensor node must be powered locally, requiring frequent maintenance (e.g., battery recharging or replacement). Additionally, wireless communication is limited by the ambient environment, as RF waves have very short penetration depths in water or wet soil [30, 31]. In a municipal gas distribution system, a large portion of the pipeline network is embedded underground, making WSN impractical or impossible.

In contrast, a FOSN is directly connected through optical fibers, allowing the passive network to be completely embedded within an infrastructure element and to be maintenance-free during its entire lifetime [32-36]. These features, along with FOSN's unique advantages of compactness, high spatial and temporal resolution, resistance to chemical corrosion, immunity to electromagnetic interference (EMI), large multiplexing capacity, and remote operation with ultralow loss (0.2 dB/km), make FOSN the most promising technology to serve as a sensing layer for city-wide management [33, 37-40]. Passive FOSNs have been demonstrated to reliably measure temperature, strain, and pressure [41]. The optical nature of the sensors allows them to operate without electricity or mechanical parts. While this offers a cost advantage

Figure 4.2. Human “Reflex” Circuits

over WSN, it also renders such networks immune to electromagnetic interference. This makes FOSNs ideal for monitoring extreme environments including high voltage lines and areas containing volatile materials. A downside to the city-wide deployment of fiber could be the initial installation costs. This would need to be weighed against the cost of WSN maintenance and the need to monitor underground systems. Existing fiber optics in a city could be integrated to help offset initial deployment costs. We elaborate further on our proposed FOSN implementation in section 4.4.

4.3 Overview Of Reflex-Tree Architecture

The Reflex-Tree architectural approach is inspired by the human nervous system, which uses several distinct hierarchical layers to process and react to millions of data streams of biological sensory information in real-time [42-44]. The key element of the reflex-tree concept is the inclusion of automated “reflex” circuits in the sensing and distributed computing architecture.

Physiologically, the myotatic (or stretch) reflex, acts as a direct neural circuit to maintain muscle position without the need for centralized control input from the brain. We present Figure 4.2 as an example to illustrate this phenomenon: while the brain is the central controller of body activities, a direct neural circuit causes an individual to

pull their hand away from a very hot object without direction from the brain. Similarly, drawing a parallel to one of our target city management applications, the detection of a burst pipeline governed by an edge device or intermediate computing node could trigger an automatic valve closure, providing immediate action that minimizes potential damage without necessarily waiting for direction from a centralized command center. Through coupling both afferent (sensor input) and efferent (control output) elements to individual edge devices and intermediate computing nodes in a multi-level hierarchy, we should be able to develop feedback loops to effectively and efficiently stabilize important parameters at the appropriate levels of the system in question.

The envisioned Reflex-Tree architecture, as seen in Figure 4.3, is comprised of a 4-layer hierarchy. At the leaves of the tree, layer 4 contains a distributed sensor network with numerous sensory nodes, which monitor public infrastructure by acquiring and processing critical data in parallel and in real-time, allowing for

Figure 4.3. Novel 4-Layer Reflex-Tree Architecture

unprecedented temporal and spatial resolution. Ideally, the sensors should be non-invasive, highly reliable, and low-cost. Some examples of useful city-wide measurements obtained by the sensors could include temperature, strain, light, vibration, and motion.

Layer 3 consists of a vast array of low-power, high-performance computing nodes, or edge devices. Each edge device is directly connected to a local sensor network (covering a neighborhood or a small community) and is responsible for raw data processing tasks, such as data classification and pattern recognition. These edge devices are intended to implement hardware architectures that operate with maximal parallelism and pipelining with minimal power consumption. This embedded, customized hardware should guarantee minimal operating cost for the city while still meeting the required levels of data processing. The edge nodes possess some limited simple feedback control over local infrastructure to respond to small, isolated events in a timely manner.

A cluster of such edge devices is then connected to an intermediate computing node that forms the next level of the hierarchy, layer 2. The key to this layer is our new spatial-temporal association based on the inputs from lower layers to support decision-making. Layer 2 is responsible for performing the automated, localized “reflex” decisions mentioned earlier. For example, if a localized section of a power grid is experiencing problems, layer 2 will detect this, disable the non-functioning section of the grid and re-route power accordingly until a dispatched electric crew has fixed the issue. The spatial-temporal patterns detected at this layer are still output to the top layer for more complex behavior/spatial analysis.

Layer 1, the top level at the root of the Reflex-Tree, is the cloud with the high computing power necessary to provide city-wide monitoring and control functions. This layer will use the inputs from the second layer to perform complex system behavior analysis and execute any required dynamic decision-making algorithms. This allows for a city-wide response in the event of a natural disaster or other potential cause of service outage. The end result of our architecture is a new computing platform with massive parallelism across all four layers, providing the necessary computing power and intelligence demanded by smart cities of the future.

4.4 Case Studies in City Management

In this section we present two simplified case studies to demonstrate the potential utility of our architecture. First we describe the lower two layers (layers 4 and 3), which are common to the two case studies. In a real-world setting a combination of shared and unique sensing devices will be deployed for each city management task. After describing these two common layers, we examine the unique specific use of the gathered information in layers 2 and 1 for both a natural gas distribution system and a city power supply network.

4.4.1 Case Studies: Common Layers

4.4.1.1 Common Layer 4: Fiber-Optic Sensor Network

To gather detailed information regarding the environment, we employ a new fiber-optic sensor network. At its foundation, molecular-level “finger-print” Rayleigh backscatter extracted by optical frequency domain reflectometry (OFDR) is used as a sensing mechanism. A small-scale proof-of-concept fiber sensing system has been constructed and tested. These tests used an optical fiber section 1 meter in length

mounted on a Swagelok tube. Figure 4.4(a) depicts the experimental setup. Here light from a tunable laser source (TLS) is split into two devices – “clock” and “signal.” The “clock” is an interferometer used to calibrate the non-linear sweep effect of the TLS. The light in the “signal” section is split between the reference and measurement arms of an interferometer via a 50/50 coupler (CPL). In the measurement path, an optical circulator (CIR) further splits the light to interrogate a length of fiber under test (FUT) and return the reflected light. A polarization controller (PC) is used to tune the state of polarization in the system. Another 50/50 CPL then recombines the measurement and reference fields. The Rayleigh scatter as a function of length is obtained via Fourier transform. A distributed sensor is formed by measuring and storing the Rayleigh scatter of the FUT at an ambient temperature. The data sets are broken into interval lengths ($\Delta z = 2$ cm) along the FUT and are converted using a Fourier transform into frequency domain measurements. To determine the spectral shift between the reference and perturbed scans, a cross-correlation is performed. A real-time temperature distribution profile is obtained by compiling the spectral shifts for each Δz along the FUT (equivalent to 50 sensing elements over 1 meter) with an update rate of ~ 50 Hz, shown in Figure 4.4(b). This system successfully demonstrates that the proposed FOSN is competent to serve as layer 4 of the proposed reflex-tree, owing to its high spatial resolution, high sensitivity, high multiplexing capacity, real-time operation, and minimal physical invasion.

Figure 4.4. Preliminary Results: a) Experimental Setup
b) Rayleigh Scatter vs. Distance

One issue in integrating such sensors into infrastructure is that the branch-like configurations of fielded gas pipeline and power delivery infrastructure systems can be highly complex. As a result, a configuration of fiber-optic sensors with massive numbers of sensing nodes must be organized in an efficient manner. In order to meet this challenge, we propose a novel parallel sensing network based on hybridized wavelength-time-switch domain multiplexing (WTSDM) to organize millions of

under severely cold temperature. “Normal” is simply a recording performed under room temperature conditions. The frequency shifts of Rayleigh scatter along the fiber coil under different conditions were measured and plotted in Figure 4.4(b). It is worth noting that, under laboratory conditions, the Rayleigh scatter profile is a clear indicator of the three temperature states. Unfortunately, in the field, the sensors are subject to various noise signals, and an advanced classifier is designed and used to identify the three situations with much improved accuracy, which will be elaborated in the following session. The collected data from our experiments, consisting of the injected “hot”, “cold”, and “normal” conditions, was used to construct an extensive set of raw frequency shift model data to be used as the layer 4 output sensory information in our gas pipeline and electric utility simulations.

4.4.1.2 Common Layer 3: SVM Pattern Recognition

Recall that layer 3 of the architecture consists of edge devices connected to multiple sensing nodes. The main purpose of these edge devices is to perform one or multiple pattern classifications on the incoming raw data from layer four and provide the results to the next higher level. The edge devices are intended as a low cost, low power embedded solution for pre-filtering an enormous amount of raw measurement data. The results of pattern classification are transmitted to the next layer up (layer 2) for further analysis and decision-making. Transmission would again occur through an optical fiber to reduce the amount of power required to maintain the system and allow for data to propagate uncorrupted through potentially adverse environments.

For our case study, support vector machine (SVM) classification [45] will be used to determine one of the three main temperature conditions (hot, cold, or normal) from

the vast gathered amount of raw sensor network data. We have selected SVM for supervised machine learning in part due to its ability to achieve high accuracy without the need of complex parameter tuning, often found in neural networks [46]. While SVM may not be the most efficient classification method for our simplified case study, it should prove beneficial in a deployed environment where the edge sensor is relied on for processing a multitude of diverse workloads, many of which may require dozens or hundreds of feature dimensions.

Using the raw data gathered from our layer 4 sensor network experiments, we have determined that three features should be sufficient in yielding a high accuracy for temperature classifications. Statistical software was used to verify that the three features in question achieved at least a 98% correlation between model data and the expected temperature classification. The features we use include: the mean absolute value, the simple square integral, and the minimum value. Thus, the resulting SVM should be multi-class in nature, supporting three features and three classes.

For this purpose, we designed and implemented a highly pipelined/parallelized SVM on an FPGA that required an equivalent level of feature computations. The benchmarking performed demonstrated that roughly $30\mu\text{S}$ should be required for each individual feature extraction and classification. For our case studies in power distribution and gas management, pattern recognition updates will need to be performed with a timing resolution of roughly several seconds. Other works [47][48] have shown that similar low power, embedded hardware systems are capable of performing SVM classification updates well within the rate of one per second, making

this an ideal platform for pattern classification. The fast rate of classification allows for multiple sensing nodes to be efficiently processed by the same edge device.

While we had mentioned earlier that an advantage to our architecture over a WSN was the fact that the passive FOSN will not require power, the level 3 processing nodes will still require some power. We still believe our design will offer substantial power savings over a WSN approach because an order of magnitude less level three nodes should exist, since each node is responsible for processing a multitude of individual optical sensors. To further minimize power consumption in the edge nodes, we implemented several power modes allowing the edge device to be placed in a sleep mode when activity is low, which should decrease the required power, and therefore reduce the cost of long-term operation.

4.4.2 Case Study 1: Gas Pipeline Distribution System

We now look to describe our first case study in future city management: the control/monitoring of a natural gas pipeline. Gas pipeline systems play an essential role in supplying energy within cities. Several threats can affect pipeline integrity, including ground movements (landslides, seismic activity), harsh environments (sudden temperature changes), third party intrusion (construction work), corrosion, and aging. These hazards significantly hinder/endanger pipeline function, leading to damage, leakage, and pipeline failure, each of which entails serious economic and ecologic consequences [49-51]. A smart energy infrastructure providing accurate, widely distributed, real-time, in-situ monitoring and control should significantly improve pipeline management and safety [52-55]. The autonomous, scalable reflex feedback control loops at the core of the Reflex-Tree architecture are ideally suited to

monitor and control these dynamic and critical components of municipal infrastructure.

Three possible levels of emergency detection for gas pipeline events and their responses, as well as corresponding hierarchical layers, and the intelligence algorithms employed at each layer, are listed in Table 4.1. For a third level emergency (disturbances), an edge device can both locate the damaged spot from the massively parallel fiber-optic sensing network and identify the corresponding emergency level via advanced FPGA embedded pattern recognition algorithms in real-time, allowing a maintenance team to quickly and safely repair the damaged pipe section while maintaining service to surrounding areas. A second level emergency, such as a ruptured pipeline, autonomously triggers a layer 2 “reflex arc,” reporting critical information to the intermediate computing node, which diverts natural gas transport to circumvent the newly-detected rupture while maintaining optimal gas service to the community. In the case of the most catastrophic damage (multiple emergency events)

Table 4.1. Emergency Levels and Response for a Natural Gas Pipeline

Emergency Level	Description	Emergency Response	Corresponding Reflex-Tree Layer	Intelligence Algorithm Category
3rd	Disturbances (small leakage, bent pipeline, etc.)	Locate the damaged point; dispatch trained personnel	Layer 3: Edge Device	Pattern Recognition
2nd	Significant perturbation (ruptured pipeline, only one damage point)	Isolation of section of damaged pipeline within a local area; prevent overloading in local network	Layer 2: Intermediate Computing Nodes	Spatial-Temporal Association
1st	Multiple emergency events (earthquake, extremely cold weather, etc.)	City-wide isolation of damaged pipelines; global flow rate control to mitigate cascading effects.	Layer 1: Data Center on the Cloud	Complex System-Wide Behavior Analysis

a layer 1 emergency response is elicited in which the Reflex-Tree uses global control to isolate damaged pipelines and prevent further damage to the grid. The cloud will use complex behavior analysis to cause a system-wide cascaded overloading effect across the entire gas distribution network, with the potential to continue to propagate and evolve within the system.

For the purposes of this case study, we seek to show that the sensor data gathered at layer 4 can be used to detect several potentially hazardous conditions. It is known that in a gas pipeline, local temperature drop is an indication of leakage owing to the Joule-Thomson effect: local pressure release induces a cold spot in a compressed gas line, allowing for the quick determination of the precise location of a leak. Also, detection of heat can be indicative that an explosion has occurred. Thus, although we are limited to sensing “hot”, “cold”, and “normal” at layer 3 of the Reflex-Tree, we should be able to correlate these detections with an event occurring in the pipeline. To this extent, we have constructed a small scale simulation of such a pipeline.

Figure 4.6 depicts a diagram of the gas pipeline system structure used in our simulation. We briefly describe the prominent features. The pipeline itself is represented by the blue lines in the figure. X and Y axis markings are distance measurements given in kilometers. Layer 3 edge devices are spread alongside the pipeline with a distance of one kilometer between each. All edge devices in the given grid are attached to the same layer 2 intermediate computing node. Edge devices located in adjacent grids would route to additional layer 2 nodes. At layer 2 we have employed the density-based spatial clustering of applications with noise (DBSCAN) clustering algorithm [56] to detect significant sized sections of damaged pipes (several

Figure 4.6. Simulated Natural Gas Pipeline

adjacent kilometers). In an actual system, the layer 2 nodes would run multiple detection algorithms, each tailored to the particular portion of infrastructure being monitored. The local grid covered by this computing node is, along with multiple other layer 2 nodes, interfaced directly to the cloud via optical fiber.

The cloud layer is not simulated in our case study due to its complexity. The cloud, which would consist of a large, powerful server cluster, would piece together the detections from the layer 2 nodes and look for data patterns and trends that may reveal a wide-scale or systematic issue using highly parallel techniques such as MapReduce [57].

For the case study we run a simulation of both a fire and ground tremor that increase in area over time. The fire results in a pipeline explosion, and the tremor is intended to produce pipeline leakage over time. “Reflex-arc” feedback is disabled in the simulation such that detections at higher layers can be clearly observed. We discuss our findings in detail in Section 4.5.

4.4.3 Case Study 2: City Power Supply Network

In the second case study, we focus our investigation on the feasibility of applying the Reflex-Tree architecture to a city power supply network modeled after a portion of a real-world power transmission network located in the San Francisco Bay area. Power lines are another common manner in which cities supply and distribute energy. Natural and other hazards can damage power grids, requiring re-routing or other manual intervention. We believe that the feedback loops present in Reflex-Tree are again an efficient and viable solution for management. Table 4.2 presents several potential emergency scenarios for the power supply network along with the Reflex-Tree layers/algorithms responsible for their detection and remediation.

For this simplified case study, we simulate a deployed FOSN along power lines together with multiple layer 3 edge devices running SVM to determine one of three temperature detections: “normal”, “hot”, or “cold”. We consider a “hot” detection to correlate with an overcurrent condition, as the lines should heat up in such a condition.

Table 4.2. Emergency Levels and Response for a City Power Supply Network

Emergency Level	Description	Emergency Response	Corresponding Reflex-Tree Layer	Intelligence Algorithm Category
3rd	Disturbances (small abnormal fluctuations)	Locate the damaged point in the line; dispatch trained personnel	Layer 3: Edge Device	Pattern Recognition
2nd	Significant perturbation (downed power line, only one major damage point)	Local isolation of damaged power line; prevent overloading in local network	Layer 2: Intermediate Computing Nodes	Spatial-Temporal Association
1st	Multiple emergency events (earthquake, extremely cold weather, etc.)	City-wide isolation of damaged power lines; global control to mitigate cascading effects.	Layer 1: Data Center on the Cloud	Complex System-Wide Behavior Analysis

Figure 4.7. San Francisco Bay Power Supply Network

A “cold” condition will be used to identify ice forming on the power lines – a potentially hazardous condition which can result in downed lines. A more sophisticated model would likely include strain measurements to determine the extent of which the power lines are in danger of failing.

Figure 4.7 depicts a diagram of the structure of the city power line distribution system used in our simulation. The dataset used to develop this simulated network has been extracted from the latest real-world commercial POWERmap database that covers the electric transmission grid of North America, with voltages ranging from 10 kV to 765 kV, provided and updated by Platts [58]. We briefly describe the features of interest in the figure. The power lines themselves are represented by the red lines. In our simulation, we have deployed a multitude of edge devices spaced one kilometer apart alongside the power lines. As with the gas pipeline, all edge devices are attached to a single layer 2 intermediate computing node running DBSCAN to detect clusters of issues, and the local grid covered by this computing node is, along with multiple other layer 2 nodes, interfaced directly to the cloud with optical fiber. Again, we have yet to

simulate the server cluster representing the cloud layer and leave it as future work due to its vast complexity.

For this case study, we run a simulation of both an earthquake and an ice storm event over a period of time. The earthquake causes a ground fault in the power line network, resulting in an overcurrent condition. The ice storm, as expected, should result in an accumulation of ice on the power lines, resulting in a detectable, but potentially catastrophic condition that should be attended to immediately. Our findings for this case study are described in the next Section.

4.5 Results

A visualization of our full simulations can be found in the video files located at [59, 60]. For both simulations, SVM classification in the layer 3 edge devices was found to be extremely accurate, predicting over 98% of all simulated pipeline states correctly over the duration of the simulation. In an actual deployment, some measurement noise will likely be present in the system, however, the great multitude of FOSN sensors as well as their close proximity should aid in filtering this out. We leave this investigation to future work, as the objective of this initial paper is to present our proposed parallel architecture. In both case studies, the clustering algorithm present at layer 2 was able to accurately detect portions of the pipeline/power line that were at risk once at least four individual, adjacent layer 3 classifications became available (this resolution is the result of a DBSCAN input parameter). To illustrate this, we present Figure 4.8, which is a snapshot of the gas pipeline near the end of simulation.

Figure 4.8. Simulation of Fire/Ground Tremor in Gas Pipeline

Figure 4.8 is composed of four diagrams for the gas pipeline case study. From the top left in clockwise order, they are: the current status of the simulation, the current status of layer 3 classification, the “cold” clusters detected, and the “hot” clusters detected. For both the simulation and classification diagrams, a pipeline color of blue indicates “normal” status, a pipeline color of red indicates a “hot” status, and a pipeline color of white indicates a “cold” status. External heat events are displayed by a gradient pattern of red (high intensity) to yellow (low intensity), and external cold events are displayed by a gradient pattern of dark blue (high intensity) to teal (low intensity).

In the displayed frame of the simulation, a fire has broken out on the left side of the pipeline, causing an explosion to occur in a portion of the surrounding pipeline. The top right portion of the pipeline meanwhile is experiencing a ground tremor, resulting in major leakage to two separate sections of the pipeline that are located within the vicinity of the disturbance. Moving to the diagram of detected classifications, it can be visually determined that the layer 3 edge devices have performed classifications that roughly matched the exact state of the simulated pipeline. In an actual deployed Reflex-Tree system, warnings would be relayed to city workers as the defects in the pipeline slowly continued to mount, hopefully preventing further damage by quick intervention. The layer 3 data is collected by a layer 2 node, and all individual hot and cold detections are separated to perform DBSCAN clustering. This results in the bottom two hot/cold cluster diagrams in Figure 4.8. Note that the colors chosen in these diagrams have no real significance: in each of the cluster figures, a different color represents membership with a different cluster. One hot cluster is detected, which corresponds to the one large section of pipeline experiencing an explosion, and two cold clusters are detected, which correspond to the two separate sections of the pipeline that are affected by the ground tremor. Although the feedback loop of “reflex arc” in this preliminary simulation was not completed, layer 2 nodes are expected to instantaneously attempt to intelligently shut down selected sections of the pipeline to prevent further damage. The cloud at layer 1 would then collate all layer 2 gathered data to see if additional action should be taken. The higher level behavior of the Reflex-Tree is currently under both theoretical and experimental investigation, and will be reported in publications to follow.

A very similar situation for the power line case study can be seen in Figure 4.9. Here a simulated earthquake occurs in the upper left portion of the power supply network, while an ice storm is gradually taking place in the lower right half. The earthquake causes multiple ground faults due to downed wires, resulting in an overcurrent condition where the nearby lines heat up. The ice accumulation causes the temperature of the power lines themselves to drop considerably. Like the previous case study, SVM classifications yielded high accuracy (greater than 98%), and hot/cold DBSCAN clustering is seen to successfully distinguish sections of the power network that are in need of immediate attention.

Figure 4.9. Simulation of Earthquake/Ice Storm in Power Supply Network

4.6 Conclusions and Future Work

To the best of the authors' knowledge, we have presented the first hierarchical, parallel approach to city infrastructure management modeled after the human nervous system. Through simulation of realistic case studies, we have proven that the concept appears to be both feasible and highly reliable in detecting potential issues at multiple stages in the hierarchy for the chosen scenarios. In an actual deployed system, the "reflex-arc" feedback should aid considerably in performing timely adjustments to city infrastructure, before the cloud-level intervention.

While we consider this initial simulation successful, an extensive amount of further work exists before such a system could be viably deployed. First, more complex simulations should be undertaken to simulate not only detectable problematic situations, but also to apply feedback and model the resulting response of the system. In addition to energy deployment, a number of other situations to be simultaneously monitored should be included in the simulation, including traffic, weather, and lighting. Also, an extensive simulation of the complex cloud layer will need to be fully defined and implemented. In improving the simulation quality, numerous efficient and parallel algorithms will need to be explored and tailored to specific tasks. Certain tasking may require that unique algorithms be used, while others may share some in common. A final area of future research could include merging FOSN, WSN, and other versatile sensing technologies for situations in which FOSN may not be the ultimate solution in acquiring and gathering data.

List of References

- [1] R. Münz, Overcrowded world? : global population and international migration. London: Haus, 2009.

- [2] M. R. Montgomery, "The urban transformation of the developing world," *Science*, vol. 319, pp. 761-764, 2008.
- [3] K. Davis, "The urbanization of the human population," *City Reader*, p. 20, 2011.
- [4] R. E. Grumbine, "China's Emergence and the Prospects for Global Sustainability," *American Institute of Biological Sciences BioScience*, vol. 57, pp. 249-255, 2007.
- [5] N. I. Suslov, "UN Project Future of the World Economy," *Problems of Economic Transition*, vol. 56, pp. 53-68, 2013.
- [6] D. Sherbinin, "The vulnerability of global cities to climate hazards," *Environment and Urbanization*, vol. 19, pp. 39-64, 2007.
- [7] K. C. Seto, R. Sánchez-Rodríguez, and M. Fragkias, "The new geography of contemporary urbanization and the environment," *Annual Review of Environment and Resources*, vol. 35, pp. 167-194, 2010.
- [8] B. Zagami, "There is no definition of a smart city," *Planning News*, vol. 40, p. 19, 2014.
- [9] N. Bombourg, "Smart Cities - Infrastructure, Information, and Communications Technologies for Energy, Transportation, Buildings, and Government: City and Supplier Profiles, Market Analysis, and Forecasts," *PRNewswire* 2013.
- [10] G. Hancke, B. Silva, and J. Hancke, Gerhard, "The Role of Advanced Sensing in Smart Cities," *Sensors*, vol. 13, pp. 393-425, 2012.
- [11] S. Jang, H. Jo, S. Cho, K. Mechitov, J. A. Rice, S.-H. Sim, H.-J. Jung, C.-B. Yun, B. F. Spencer, Jr., and G. Agha, "Structural health monitoring of a cable-stayed bridge using smart sensor technology: deployment and evaluation," *Smart Structures and Systems*, vol. 6, pp. 439-459, Jul-Aug 2010.
- [12] J. A. Rice, K. Mechitov, S.-H. Sim, T. Nagayama, S. Jang, R. Kim, B. F. Spencer, Jr., G. Agha, and Y. Fujino, "Flexible smart sensor framework for autonomous structural health monitoring," *Smart Structures and Systems*, vol. 6, pp. 423-438, Jul-Aug 2010.
- [13] F. Stajano, N. Hoult, I. Wassell, P. Bennett, C. Middleton, and K. Soga, "Smart bridges, smart tunnels: Transforming wireless sensor networks from research prototypes into robust engineering infrastructure," *Ad Hoc Networks*, vol. 8, pp. 872-888, Nov 2010.
- [14] Asimakopoulou, E.; Bessis, N., "Buildings and Crowds: Forming Smart Cities for More Effective Disaster Management," *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2011 Fifth International Conference on, pp. 229, 234, June 30 2011-July 2 2011

- [15] Khan, Z.; Kiani, S.L., "A Cloud-Based Architecture for Citizen Services in Smart Cities," Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, pp.315,320, 5-8 Nov. 2012
- [16] Dey, S.; Chakraborty, A.; Naskar, S.; Misra, P., "Smart city surveillance: Leveraging benefits of cloud data stores," Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on, pp.868,876, 22-25 Oct. 2012
- [17] Castro, M.; Jara, A.J.; Skarmeta, A.F.G., "Smart Lighting Solutions for Smart Cities," Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on , pp.1374,1379, 25-28 March 2013
- [18] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," Computer Networks, vol. 38, pp. 393-422, Mar 15 2002.
- [19] J. N. Laneman, D. N. C. Tse, and G. W. Wornell, "Cooperative diversity in wireless networks: Efficient protocols and outage behavior," Ieee Transactions on Information Theory, vol. 50, pp. 3062-3080, Dec 2004.
- [20] H. Jo, S.-H. Sim, T. Nagayama, and B. F. Spencer, Jr., "Development and Application of High-Sensitivity Wireless Smart Sensors for Decentralized Stochastic Modal Identification," Journal of Engineering Mechanics-Asce, vol. 138, pp. 683-694, Jun 2012.
- [21] L. M. Ni, Y. H. Liu, Y. C. Lau, and A. P. Patil, "LANDMARC: Indoor location sensing using active RFID," Wireless Networks, vol. 10, pp. 701-710, Nov 2004.
- [22] G. De Vita and G. Iannaccone, "Design criteria for the RF section of UHF and I microwave passive RFID transponders," Ieee Transactions on Microwave Theory and Techniques, vol. 53, pp. 2978-2990, Sep 2005.
- [23] P. V. Nikitin and K. V. S. Rao, "Theory and measurement of backscattering from RFID tags," Ieee Antennas and Propagation Magazine, vol. 48, pp. 212-218, Dec 2006.
- [24] L. Yang, A. Rida, R. Vyas, and M. M. Tentzeris, "RFID tag and RF structures on a paper substrate using inkjet-printing technology," Ieee Transactions on Microwave Theory and Techniques, vol. 55, pp. 2894-2901, Dec 2007.
- [25] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Computer Networks, vol. 54, pp. 2787-2805, Oct 28 2010.
- [26] M. G. Di Benedetto and B. R. Vojcic, "Ultra wide band wireless communications : A tutorial," Journal of Communications and Networks, vol. 5, pp. 290-302, Dec 2003.

- [27] F. Borgonovo, A. Capone, M. Cesana, and L. Fratta, "ADHOC MAC: New MAC architecture for ad hoc networks providing efficient and reliable point-to-point and broadcast services," *Wireless Networks*, vol. 10, pp. 359-366, Jul 2004.
- [28] H. Bolcskei, R. U. Nabar, O. Oyman, and A. J. Paulraj, "Capacity scaling law's in MIMO relay networks," *Ieee Transactions on Wireless Communications*, vol. 5, pp. 1433-1444, Jun 2006.
- [29] M. D. Dikaiakos, A. Florides, T. Nadeem, and L. Iftode, "Location-aware services over vehicular ad-hoc networks using car-to-car communication," *Ieee Journal on Selected Areas in Communications*, vol. 25, pp. 1590-1602, Oct 2007.
- [30] C. A. Balanis, *Advanced engineering electromagnetics*. New York: Wiley, 1989.
- [31] D. B. Davidson, *Computational electromagnetics for RF and microwave engineering*. Cambridge; New York: Cambridge University Press, 2005.
- [32] T. A. Dickinson, J. White, J. S. Kauer, and D. R. Walt, "A chemical-detecting system based on a cross-reactive optical sensor array," *Nature*, vol. 382, pp. 697-700, Aug 22 1996.
- [33] K. T. V. Grattan and T. Sun, "Fiber optic sensor technology: an overview," *Sensors and Actuators a-Physical*, vol. 82, pp. 40-61, May 15 2000.
- [34] B. Lee, "Review of the present status of optical fiber sensors," *Optical Fiber Technology*, vol. 9, pp. 57-79, Apr 2003.
- [35] R. A. Perez-Herrera and M. Lopez-Amo, "Fiber optic sensor networks," *Optical Fiber Technology*, vol. 19, pp. 689-699, Dec 2013.
- [36] D. Sanchez Montero and C. Vazquez, "Remote Interrogation of WDM Fiber-Optic Intensity Sensors Deploying Delay Lines in the Virtual Domain," *Sensors*, vol. 13, pp. 5870-5880, May 2013.
- [37] H. N. Li, D. S. Li, and G. B. Song, "Recent applications of fiber optic sensors to health monitoring in civil engineering," *Engineering Structures*, vol. 26, pp. 1647-1657, Sep 2004.
- [38] S. Liehr, P. Lenke, K. Krebber, M. Seeger, E. Thiele, H. Metschies, B. Gebreselassie, J. C. Mönich, and L. Stempniewski, "Distributed strain measurement with polymer optical fibers integrated into multifunctional geotextiles," pp. 700302-700302, 2008.
- [39] A. D. Kersey, M. A. Davis, H. J. Patrick, M. LeBlanc, K. P. Koo, C. G. Askins, M. A. Putnam, and E. J. Friebele, "Fiber grating sensors," *Journal of Lightwave Technology*, vol. 15, pp. 1442-1463, Aug 1997.

- [40] A. J. Rogers, Understanding optical fiber communications. Boston: Artech House, 2001.
- [41] Rogers, A.J., "Distributed optical-fibre sensors for the measurement of pressure, strain and temperature," *Electronic and Radio Engineers, Journal of the Institution of* , vol.58, no.5, pp.S113,S122, July-August 1988
- [42] D. Lloyd, "Integrative pattern of excitation and inhibition in two-neuron reflex arcs," *Jour Neurophysiol*, vol. 9, pp. 439-444, 1946 1946.
- [43] G. L. Gottlieb and G. C. Agarwal, "Response to sudden toques about ankle in man - myotatic reflex," *Journal of Neurophysiology*, vol. 42, pp. 91-106, 1979 1979.
- [44] P. Crenna and C. Frigo, "Excitability of the soleus H-reflex arc during walking and stepping in man," *Experimental Brain Research*, vol. 66, pp. 49-60, 1987 1987.
- [45] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [46] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning*, 1998, pp. 137–142.
- [47] C. Kyrkou and T. Theocharides, "A parallel hardware architecture for real-time object detection with support vector machines," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 831–842, 2012.
- [48] T. Kryjak, M. Komorkiewicz, and M. Gorgon, "Fpga implementation of real-time head-shoulder detection using local binary patterns, svm and foreground object detection," in *Design and Architectures for Signal and Image Processing (DASIP)*, 2012 Conference on, 2012, pp. 1–8.
- [49] B. Anifowose, D. Lawler, D. Horst, and L. Chapman, "Evaluating interdiction of oil pipelines at river crossings using Environmental Impact Assessments," *Area*, vol. 46, pp. 4-17, 2014.
- [50] R. Alzbutas, T. Iešmantas, M. Povilaitis, and J. Vitkutė, "Risk and uncertainty analysis of gas pipeline failure and gas combustion consequence," *Stochastic Environmental Research and Risk Assessment*, pp. 1-16.
- [51] M. Carnicero and P. Hryciuk, "Landslide Followed by a Leak at the Patagonian Andes, Argentina," in *ASME 2013 International Pipeline Geotechnical Conference*, 2013, pp. V001T03A002-V001T03A002.

- [52] G. Bernasconi, S. Del Giudice, G. Giunta, F. Dionigi, R. Schiavon, and F. Zanon, "Advanced Pipeline Monitoring Using Multipoint Acoustic Data," in Offshore Mediterranean Conference and Exhibition, 2013.
- [53] F. A. V. Martinez and C. R. Acero, "Monitoring and Control System in an Area of Geotechnical Risk Unstable in the Camisea Gas Transport System," in ASME 2013 International Pipeline Geotechnical Conference, 2013, pp. V001T03A004-V001T03A004.
- [54] F. Ravet, E. Rochat, and M. Niklès, "Application of distributed sensing technology to the energy industry."
- [55] G. Bernasconi, S. Del Giudice, G. Giunta, and F. Dionigi, "Advanced Pipeline Vibroacoustic Monitoring," in ASME 2013 Pressure Vessels and Piping Conference, 2013, pp. V005T10A008-V005T10A008.
- [56] Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A 'Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In E. Simoudis, J. Han, & U. M. Fayyad (Eds.), Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (pp. 226-231).
- [57] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM vol. 51, no. 1, pp. 107-113, January 2008.
- [58] Platts, "Electric Power Prices & Energy Market Data by Platts.com," October 2014. Available: <http://www.platts.com/commodity/electric-power>
- [59] J. Kane (2015, June 6). Reflex Tree: Gas Pipeline Simulation - Youtube. [Online]. Available: <https://youtu.be/U-Q9UotWcmk>
- [60] J. Kane (2015, June 6). Reflex Tree: Power Grid Simulation - YouTube. [Online]. Available: <http://youtu.be/LRHsWZ3tLgM>