I am pleased to present you with this prospectus for S*ProCom², a distinctive research center at a unique educational institution, The Cooper Union for the Advancement of Science and Art.

The Cooper Union was founded in the 19th century by philanthropist, visionary, and inventor Peter Cooper, who believed that education should be “as free as the air and water.” Consequently, the Cooper Union awards tuition scholarships to all its students, with admission based solely on academic merit. The School of Engineering offers Bachelor’s and Master’s Degrees in chemical, civil, electrical and mechanical engineering.

S*ProCom², the Center for Signal Processing, Communications and Computer Engineering Research, involves Cooper Union students and faculty with cutting edge research and development projects. When institutions, including corporations, government agencies, medical centers and other academic entities have challenging problems that require creative solutions, they come to S*ProCom². We engage alumni, entrepreneurs, research scientists, and professionals in a variety of disciplines to support these collaborative research efforts and enhance our students' educational experience. Our students are among the best in the nation, and have the benefit of accelerated exposure to subjects in engineering and science coupled with early “hands-on” experience. With strengths in both theory and practice they can actually “get things done.” The unique nature of The Cooper Union enables S*ProCom² to provide unique services to our collaborators, and conversely the S*ProCom² supports the unique mission of The Cooper Union.

Here we describe some of the exciting projects we have undertaken, we present the principles of S*ProCom², and we point to the future of the Center as we envision it.

Sincerely,

Prof. Fred L. Fontaine, Ph.D.
Director of S*ProCom²
The focus areas of S*ProCom2- signal processing, communications and computer engineering- intersect a wide variety of applications and disciplines, and are essential to creative, interdisciplinary solutions to real-world problems.

Modern technology is based on information, and that information takes on a variety of forms that must be analyzed, converted, compressed, transmitted, enhanced and integrated. These tasks are achieved with systems of ever-increasing sophistication. Not only is computational power ever expanding, but also there is a growing prevalence of cognitive systems that are self-learning and adaptive.

S*ProCom2 is positioned to undertake research and development projects in areas such as:

- streaming of HDTV and other high bandwidth signals through the internet
- acoustic noise cancellation
- automated diagnostics from bio-electrical signals such as EKG and EEG
- parallel, distributed and cooperative communication and computing systems
- sensor array data fusion and integration
- autonomous navigation for mobile robots

- cognitive wireless communication systems
- adaptive resampling in space-time for magnetic resonance imaging
- identification of chemicals from NMR spectra
- real-time network intrusion detection
- artificial vision, hearing and other sensory systems
- secure data transmission and authentication
- algorithmic and architectural designs for low power ad-hoc sensor networks
Signals—the representation of information—are ubiquitous. Communication—the transfer of information—is ubiquitous. Computation—the processing of information—is ubiquitous.

S*ProCom2 unites these fields to address the challenges of the modern world!

S*ProCom2 matches the faculty and students of The Cooper Union with exciting research projects. The strategic vision of S*ProCom2 as a symbiotic partner to the academic programs at The Cooper Union is based on the following paradigms:

- Students are exposed to advanced concepts early on. Students take on challenging theoretical and project based courses from their first year. Graduate level electives are available to undergraduates.
2007: Two juniors derive expressions for parallelizing a cryptographic algorithm.

2008: Freshmen debug their digital logic design project.
In 2003, an informal exchange between Prof. Fred Fontaine (Electrical Engineering, Cooper Union) and Prof. Yi Wang (Radiology, Weill-Cornell Medical Hospital) led to a formal project that created S*ProCom2. Prof. Wang and his chief researcher, Prof. Pascal Spincemaille had a problem: their MRI (magnetic resonance imaging) machines weren’t fast enough.

MRI works by using strong magnetic fields to excite hydrogen atoms (H+) in the body. As shown in the figure below, these atoms then release radio-frequency (RF) energy that is picked up by a receiver coils (antenna). The coil signal is digitized by a data acquisition system (DAQ) and then processed with a powerful computer to create the image that your radiologist can analyze.

The rate at which the magnetic fields can change, and thus the time it takes for the MRI scan, is limited by the laws of physics. One solution to speed up is using multiple receiver coils, which then require multiple DAQs and a computer cluster to work. These are cumbersome and expensive--Cornell wanted us to make MRI faster and cheaper.

S*ProCom2 attacked this research problem from three angles: the coils, the data acquisition and the image reconstruction (signal processing). Over a three-year period (2004-2007), we came up with novel solutions. We invented a prototype microcoil array (right) that packed tens of coils into the space of one existing coil. We developed a data acquisition board that evolved over the years (right) into a system-on-a-chip that is an order of magnitude cheaper than commercial alternatives. A number of master’s students also worked on multi-resolution algorithms that could construct high-quality MR images from a smaller amount of coil data (which means a faster scan).
2008: Information-Theoretic Identification of Chemical Spectra

2008: Quasi Monte-Carlo Simulation
2003–2005

Senior Projects

• Surface Penetrating Radar
• Non-invasive Blood Count Monitor
• Nerve-controlled Tetris
• Doozie Douser Firefighting Robot
• Active Noise Control
• MP3 Stereo System
• Flexible MRI Coil Array
• 64-Channel MRI Data Acquisition System
• Compensation for MRI Field Inhomogenities

• QRD-LSL Interpolators with Dynamic Assignment of Filter Orders
• A 3-Bit Artificial Neural Network Flash A/D Converter
• SNR Computation for Coil Arrays via Moment Methods for Multichannel MRI
• Piecewise Linear Modeling for Nonlinear Control with Optimized Settling Time
• Competitive Learning Algorithms for Low Bit-Rate Image Compression
• Reconstruction of Spiral MRI Using FFT on Hexagonal Lattice Structures
2006

Senior Projects

- Digital Notepad
- Experience Recorder
- Intelligent Automated Saltwater Aquarium
- Low-Cost Scalable Multichannel MRI Receiver
- Microcoil Receiver Array for MRI
- Micromouse
- RF Telemetry
- Speaker Verification for Secure Access

- Optimization of CMOS Analog Amplifiers Using Genetic Algorithms
  
  - A Parallelized Monte Carlo Algorithm for the 1D Steady-State 
    Burger’s Equation
    K. Chatterjee, C. Yu and J. Poggie
  
  - A Two-Dimensional Stochastic Algorithm for the Solution of the 
    Nonlinear Poisson-Boltzmann Equation: Validation with Finite-
    Difference Benchmarks
    K. Chatterjee and J. Poggie
    66, No. 1, pp. 72-84, April 2006
  
  - A Stochastic Algorithm for the Extraction of Partial Inductances in 
    IC Interconnect Structures
    K. Chatterjee
    Applied Computational Electromagnetics Society Journal, Vol. 21, 
    No. 1, pp. 81-89, March 2006.
  
  - A parallelized 3D Floating Random-Walk Algorithm for the 
    Solution of the Nonlinear Poisson-Boltzmann Equation
    K. Chatterjee and J. Poggie
  
  - An Economical Multichannel Integrated Receiver/Reconstruction 
    System for MRI
    I. L. Dalal
    Workshop, Bethesda, MD, July 2006, pp. 134–135
  
  - A Reconfigurable FPGA-based 16-Channel Front-End for MRI
    I. L. Dalal and F. L. Fontaine
    In Proc. of the 40th Annual Asilomar Conference on Signals, 
    Systems and Computers, Pacific Grove, CA, Oct 29–Nov 1, 2006, 
    pp. 1860–1864
  
  - A Low-Cost Scalable Multichannel Digital Receiver for Magnetic
• Multiresolution MR Image Reconstruction from Pseudo-Hex Lattices Using Separable DIS-DID FFT
• A Mathematically Unified Approach to Accelerated MRI Reconstruction
• A Biophysical Study of Cationic Polymers Used for Non-viral Gene Therapy Delivery Using Continuous Configurational Biased Direct Monte Carlo Method

• Prostate ultrasound image processing
  D. Stefan
• A New Floating Random Walk Methodology for Neumann and Mixed Boundary Condition Problems Without Reflections at Boundaries: Validation with Laplace’s Equation in One Dimension
  K. Chatterjee, C. Yu, S. Srinivasan and J. Poggie
• A Parallelized Monte Carlo Algorithm for the Maxwell-Helmholtz Equation: Application to a Heterogeneous Transverse Magnetic Benchmark Problem
  K. Chatterjee, H. Yue and Y. L. Le Coz
• A Parallelized Monte Carlo Algorithm for the Solution of the One-Dimensional Wave Equation: Validation with Analytical Benchmark Solutions
  K. Chatterjee, C. Yu, H. Yue and J. Poggie
• A real-time AAC-type audio codec on the 16-bit dsPIC architecture
  I. L. Dalal
  In Proc. of the 4th International Conference on Electrical and Electronic Engineering (IEEE), Mexico City; Sep. 2007 (pp. 205–208)
• A Low-Complexity Navigation Algorithm for a Scalable Autonomous Firefighting Vehicle
  J. Harwayne-Gidansky and Michael Sudano
Senior Projects

- Automatic Transcription of Music
- Cognitive Radio System
- Feedback System for Physical Training
- Gaze Tracking Using Corneal Reflections
- Hardware-Efficient Bloom Filter Implementation
- Information Theoretic Chemical Spectra Identification
- Wireless Electromyography
- Wireless Pen

- A Meta-analysis Approach to Improving the Accuracy and Stability of Linear Discriminant Analysis in DNA Microarray Classification
- A Distributed Particle Swarm Optimization Implementation with Applications in Feed-Forward Neural Networks
- GRIND: Finite Domain Integer Constraint Programming in C++
- A Parallelized Monte Carlo Algorithm for the One-dimensional Wave Equation

- On the Parallelization of the MICKEY-128 2.0 Stream Cipher
  D. Stefan and C. Mitchell In Proc. of the ECRYPT State of the Art of Stream Ciphers

- A hardware framework for fast generation of multiple long-period random number streams
  I. L. Dalal and D. Stefan

- An area-efficient hardware implementation of the CryptMT stream cipher
  D. Stefan, J. Harwayne-Gidansky, D. Nummey, I. Dalal

- A Quasi-Monte Carlo Solver for Partial Inductances in IC Interconnect Structures

- A Parallel Framework for Long-period Random Number Generators in Hardware
  I. L. Dalal and D. Stefan
A Low-Cost Scalable Multichannel Digital Receiver for Magnetic Resonance Imaging

Ishaan Dalal
Ashwin Kirpalani
Fred Fontaine


\textbf{S*ProCom}^2

Center for Signal Processing, Communications and Computer Engineering Research (S*ProCom$^2$)

Department of Electrical Engineering
The Cooper Union
51 Astor Place, New York, NY 10003.

August 2006
Design and Implementation of a Wireless (Bluetooth) Four Channel Bio-Instrumentation Amplifier and Digital Data Acquisition Device with User-Selectable Gain, Frequency, and Driven Reference

Alin Cosmanescu
Ian Kremenic
et al.


S*ProCom²

Center for Signal Processing, Communications and Computer Engineering Research (S*ProCom²)

Department of Electrical Engineering
The Cooper Union
51 Astor Place, New York, NY 10003.

August 2006
Fig. 4. hSPICE(r) generated plots of the channel characteristics showing the three different gain settings and the two different frequency cutoffs, the simulations were done using the model file for the op-amps provided by the manufacturer. Experimental measurements match these simulation results reasonably well, with deviations due to the tolerances of the component values. This deviation was expected and designed for, so for example, the channel was designed to have a gain of 1500 V/V (63 dB) for the highest setting, while it was experimentally measured to have a gain of approximately 1000 V/V, which meets our gain requirements.

Fig. 5. The input to one of the ADC channels of the microcontroller is shown against the output of a pin which toggles for every sample made on that channel. The sampling rate achieved by the ADC is determined from the frequency of the bit toggle, shown on this screen capture as CH2. Various signal statistics for both the signal being sampled and the bit toggle are shown on the left hand of the figure. This oscilloscope screen capture was taken with two channels being sampled by the microcontroller and the data being transmitted using BluetoothR.

measure both EKG and EMG signals. It can digitize and wirelessly transmit the measured signals using BluetoothR, making it ideal for mobile and in-situ data acquisition. It is also simple enough to build in any moderately equipped electrical laboratory using breadboards and other off-the-shelf components.

IV. ACKNOWLEDGMENTS
The authors would like to thank Glenn Gross for his help with obtaining components, lab space, and equipment. We would also like to thank Yifei Huang, Andy Wu, Yuriy Mykula, Alex Scolnik and Michael Dooley for lending a helping hand.

REFERENCES
A Reconfigurable Real-time Image Reconstruction Engine for Parallel MRI

Subject/Keywords: Field-Programmable Gate Arrays, Biomedical Image Processing, Magnetic Resonance Imaging

Ishaan Dalal (IEEE # 41472525)
The Cooper Union for the Advancement of Science and Art
51 Astor Place Rm. 126
New York, NY 10003
Phone: 661-204-0797
ishaan@cooper.edu

Advisor: Fred L. Fontaine
**Abstract**—Parallel MRI acquisitions are generally reconstructed into images off-line, on PCs and computer clusters. Here, we present an innovative single-FPGA engine that performs real-time 2D-FFT image reconstruction from arrays of up to 16 coils. Partial reconfiguration enables rapid switching of FPGA modules for maximal flexibility and lower hardware cost. If the engine is integrated on to an FPGA-based receiver, more complicated parallel MRI algorithms can replace the receiver logic after the acquisition is complete. The engine can also function as a standalone hardware accelerator. As a proof-of-concept for real-time non-cartesian reconstruction, a reconfigurable module for next-neighbor regidding of spiral MRI is also demonstrated.

I. INTRODUCTION

A magnetic resonance imaging (MRI) scanner consists of a back-end and a front-end. The back-end includes static and gradient magnetic-field generators that excite H\(^+\) nuclei in the body, as well as coils (antennae) that pick up RF signals emitted by the nuclei when the external fields relax. The front-end is comprised of receivers that demodulate and digitize these RF-MR signals, followed by a reconstruction engine that converts the frequency-domain RF-MR data into spatial images. The acquisition time of a classical single-coil scanner is limited by electrical and physiological factors. Current MRI research focusses on parallel imaging (pMRI) techniques such as SMASH, SENSE and GRAPPA [1]. These speed up acquisition by employing multiple-coil phased-arrays, followed by data interpolation based on the individual coil sensitivities (radiation pattern). Therefore, pMRI requires multiple receiver channels as well as more sophisticated image reconstruction hardware.

Commercial MRI front-ends use analog RF/IF receivers, baseband sampling and PCs for reconstruction. Other experimental front-ends for pMRI [2], [3] have used specialized digital receivers, bandpass sampling and computer clusters for image reconstruction. Recently, a field-programmable gate array (FPGA)-based 16-channel digital receiver has also been demonstrated [4]. General-purpose computers/CPU's are ill-suited for dedicated pMRI reconstruction algorithms such as the 2D-FFT, requiring clusters of multiple computers for real-time pMRI. This solution is neither economical nor very efficient.

Our reconstruction engine generates real-time images (via the 2D-FFT) from arrays of up to 16 coils (or channels). It is built around a single FPGA, and includes a partial-reconfiguration framework that allows different reconstruction modules to be dynamically loaded/unloaded as needed. Such hardware reuse opens the door for real-time reconstruction with more complicated pMRI algorithms that may be too large to fit on the FPGA. These algorithms can usually be partitioned into sub-modules that operate sequentially upon data; hence, partial reconfiguration allows them to be fully implemented without having to move to larger or multiple FPGAs. For example, if the engine and the digital receiver of [4] are integrated on one FPGA, then as soon as an acquisition has been buffered in on-board memory, the receiver logic is no longer necessary and can be replaced by, say, SENSE reconstruction. To the best of our knowledge, this is the first single-FPGA based pMRI reconstruction engine and has a superior performance-cost ratio than existing solutions.

This paper first discusses the design of the partial-reconfiguration framework. Considering that image quality is paramount in medical MRI, the implementation of an efficient-yet-accurate fixed-point 2D-FFT, including a fast matrix transpose is discussed next. Finally, we demonstrate the utility of the engine beyond conventional (cartesian) MRI with a novel implementation of next-neighbor regidding [5] for spiral MRI. Relevant performance analyses and results are also presented.

II. PARTIAL-RECONFIGURATION ON THE FPGA

A. The FPGA Platform

The engine is built around a Xilinx Virtex-II Pro XC2VP30 FPGA that has \( 13,500 \) configurable logic blocks (CLB), \( 136 \times 18 \times 18\)-bit multipliers and two embedded PowerPC 405 microprocessors. The FPGA sits on a XUPV2P development board that provides physical-layer ICs such as a Fast Ethernet transceiver to facilitate I/O via multiple interfaces. One of the PowerPCs acts as a system supervisor and also runs a lightweight TCP/IP stack and FTP server for transferring data/images over Ethernet; the other PowerPC is currently unused.

B. Memory Subsystem and the PLB Bus

The engine has two kinds of memory: primary memory and secondary memory. Primary (or ‘core’) memory is a limited amount (2.5 megabits) of fast, single-cycle-latency static RAM (SRAM) inside the FPGA that is reserved for use by logic modules. Secondary memory is a slower but much larger consumer-grade 512 megabyte DDR dynamic RAM (DRAM) module [6] used mostly for storing for MRI acquisitions. A DRAM memory controller on the FPGA handles low-level interfacing to the DDR RAM module to enable simplified memory access.

The PowerPC communicates with the DRAM and other high-bandwidth logic peripherals such as the fast ethernet MAC over the 64-bit Processor Local Bus (PLB) [7]. Transferring data to/from DRAM over the PLB incurs some transactional overhead in addition to DRAM’s native latencies. Hence, transfers of long bursts of contiguous data are much more efficient than single reads/writes. The DRAM access cost can be linearly modeled as

\[
\tau_{\text{read/write}} + mt
\]

where \( \tau \) is the initial latency, \( m \) is the number of data units and \( t \) is the transfer time/unit. With 64-bit units at a 100 MHz PLB/DRAM frequency, \( \tau_{\text{read}} = 100 \) ns, \( \tau_{\text{write}} = 50 \) ns and \( t = 10 \) ns, leading to burst transfer rates of up to 800 megabytes/s. Read/write latencies are asymmetric since the DRAM controller can only pipeline writes. We limit burst transfers to a maximum of 256 32-bit ‘words’ since maximum burst lengths may vary based on the physical structure of the DRAM module.

C. Dynamic Partial Reconfiguration

The FPGA is programmed in ‘frames’—addressable units that physically stretch from ‘top’ to ‘bottom’. This enables dynamic partial reconfiguration [8], i.e. reprogramming portions (sets of frames) of the FPGA while the remainder of the device is still operational (Fig. 1).
A crucial consideration in PR design is ensuring seamless communication between partially-reconfigurable modules (PRMs) and existing static logic (as well as other PRMs). Signals can only cross PR boundaries via logic bridges called bus macros which must be hard-wired upon initial programming. While bus macros provide physical interconnects, the PR framework on a multiple-module system such as the reconstruction engine must be structured around a logical bus for consistency. Since reconstruction PRMs need direct access to MRI data stored in DRAM (which is connected to the PLB), PLB is used as the framework bus.

Custom FPGA logic modules including PRMs must connect to the PLB via an IP Interface (IPIF); each PRM must have a dedicated IPIF. We have designed a simplified IPIF that uses significantly fewer resources than Xilinx’s IPIF, but relies on the PRM to provide buffers and a finite state machine (FSM) for data transfers. The IPIF is a combined master/slave PLB device supporting techniques important for real-time applications such as bus mastering and variable-length bursts. When the FPGA is initially programmed, a fixed number of IPIFs are placed on PR boundaries and permanently connected to the PLB as shown in Fig. 1. Upon reconfiguration, a PR module is automatically linked to its boundary IPIF. The FPGA’s internal structure places certain constraints upon the placement of PRMs. Therefore, PRMs that include resources in fixed locations inside the FPGA—such as external I/Os, multipliers and SRAM—usually have some overhead compared to a non-PR version.

III. RECONFIGURABLE 2D-FFT

The real-time 2D-FFT consists of a 1D block floating-point FFT, absolute value (CORDIC) and a fast matrix transpose as shown in Fig. 2.

A. Block Floating-point FFT

MRI data-sets belong to the complex spatial-frequency domain, or k-space. Data-sets are generally acquired on cartesian trajectories and reconstructed via the 2D Fast Fourier Transform (FFT) into spatial images. The 2D-FFT is decomposed into 1D ‘row’ and ‘column’ passes. At each stage of a radix-2 FFT, the intermediate output can grow by as much as $1 + \sqrt{2} \approx 2.414$, causing overflows in fixed-point architectures; the outputs of one or more stages must be scaled to prevent overflow. When the nature of the FFT input is unknown, a common scheme is to scale each stage by 2 (1 bit) regardless of whether overflow would have occurred. If a large proportion of the input data are small compared to the FFT’s precision, accumulated roundoff errors from scaling-by-2 severely affect the final output; this problem is compounded with the dual row-column FFTs for 2D data.

Fig. 3 is a magnitude plot of FatWater, a 16-bit 256 x 256-point k-space data-set. Characteristically, the data are very small except for the central region (low frequencies). Reconstructing k-space data-sets with a 16-bit scaling-by-2 fixed-point FFT results in extremely poor image quality. Better quality can be obtained by increasing the FFT precision or through multi-pass normalization at the expense of higher run-times and/or resource usage.

There is a third alternative: at a given precision, the block floating-point FFT (BFP-FFT) [9] is better than fixed- and floating-point, and only slightly more computationally expensive than fixed-point. Each BFP-FFT stage scales intermediate outputs only when they would overflow (Fig. 4) and tracks overall scaling with a block exponent $b$ common to the entire input block. Multiplying the output block by $2^b$ recovers unity-gain values.
Both 16- and 24-bit fixed-point and block floating-point FFTs were tested extensively with a number of MRI data-sets. Table I compares the quality of the final 2D images from the different FFTs to a reference 64-bit floating-point MATLAB reconstruction. The test data-sets are all 256 × 256 complex 16-bit: uniformly distributed random numbers (noise), an artificial phantom (FatWater) and an actual acquisition (Cardiac). The PSNR metric is derived from the mean-squared error (or L2 norm) of the pixel-by-pixel difference between the ‘distorted’ image and the reference. While used extensively in image coding studies, PSNR is not always indicative of an image’s perceived visual quality. Hence, we also employ the Structural Similarity (SSIM) [10] metric that considers both the underlying structure of (i.e. objects in) the image and the human visual system model to calculate an SSIM index between 0–1 (higher is better).

Figs. 5a and 6a show FatWater and Cardiac reconstructed with the 16-bit BFP-FFT that was chosen based on its high quality and negligible resource overhead (<10%) compared to 16-bit fixed-point. Both metrics are excellent, and the difference images Figs. 5b and 6b (with respect to the MATLAB reference) had to be multiplied by extremely large factors so that difference detail would be visible to the naked eye.

Clocked at 200 MHz, Xilinx’s radix-2 ‘minimum resources’ FFT core calculates a 256-point transform in 5.6 μs; operating it in bit-reversed addressing mode allows synchronous load/unload. A fast matrix transpose (section III-C) performs natural reordering and transposition. If all the block-exponents $B = \{b_1, \ldots, b_n\}$ from the first 1D-FFT (rows) are not equal, each row $i$ is divided by $2^{b_i}$ so that the column inputs to the second 1D-FFT have the same relative gain. After the second FFT, each output column is multiplied by $2^{b_i}$ as the next stage (absolute value; section III-B) has a higher precision (24-bit). These block-exponent normalizations are accomplished by using the embedded multipliers as single-cycle barrel shifters.

B. Calculating the Absolute Value

The most common way to form a spatial image is by taking the absolute value (magnitude) of the complex 2D-FFT outputs $X + jY$. This can be computed efficiently in hardware by the recursive CORDIC [11] algorithm using only shifts and adds (no multiplies). CORDIC iteratively rotates the vector $(X, Y) \rightarrow (X', Y')$ until $Y' = 0$, thus calculating magnitude $X' = \sqrt{X^2 + Y^2}$ and phase (not used). Output precision increases by $\approx 1$ bit per iteration, and iterations can be unrolled and pipelined for faster throughput.

Xilinx’s vector rotation core could not provide 24-bit outputs at 200 MHz on our device. Hence, we use two multipliers and an adder (Fig. 2) to calculate $X^2 + Y^2$, followed by a simplified hyperbolic mode CORDIC which calculates $\sqrt{X^2 + Y^2}$. The circuit’s performance can be summed up with the observation that running various MRI data-sets (including those in Table I) through it resulted in negligible PSNR decreases of < 0.02 dB over MATLAB’s absolute value function.

C. Fast Matrix Transpose

As Fig. 2 illustrates, the 2D-FFT requires two transpositions: one after the ‘rows’ and another after the ‘columns’. Common $N \times N$ matrices for MRI applications are $N = 128$ and $N = 256$, but more complicated algorithms (such as the regridder of section IV) may use up to $N = 4096$. Transposing large matrices stored ‘out-of-core’, i.e. in secondary memory (DRAM), requires an intermediate SRAM buffer. Data are read/write in bursts of $M$ 32-bit words, and the maximum burst transfer length is $M_{\text{max}}$ (in our case, 256 words). For simpler notation, read/write latency $\tau_{\text{RW}} = \tau_{\text{read}} + \tau_{\text{write}}$. Derivations for the transposition run-times that follow are detailed in the appendix.

From (A.1), the minimum transpose time for a $N \times N$ matrix ($N \geq M_{\text{max}}$) on our system is

$$T_{\text{min}} = \frac{N^2}{M_{\text{max}}} (\tau_{\text{RW}} + 2M_{\text{max}})$$

(the factor of 2 is for reading and writing). What sets different transposition algorithms apart is the size $B$ of the intermediate buffer necessary to achieve this bound, e.g. a direct transpose requires $B = M_{\text{max}}^2$, i.e. $256^2 \times 32$-bits, or 2 megabits. Considering the extremely limited SRAM (2.5 megabits) available, our
goal is minimizing transpose time across different $N$ with as small a buffer as possible.

For this purpose, we have adapted Kaushik, et al’s single-
radix transposition (SRT) algorithm [12]. SRT is out-of-place and
requires temporary storage equal to the original matrix’s
in secondary memory; this is not a problem since consumer
DRAM modules are available in large capacities at reasonable
cost. If $N$ is factorized as

\[ N = 2^n = \prod_{i=1}^{p} s_i \]  

the SRT algorithm makes $p$ passes over the entire matrix with $N/s_1$ steps-per-pass. In each step, $s_1$ consecutive rows are read from DRAM into a $B = s_1 \times N$ buffer, linearly permuted, and written back as non-consecutive rows. Fig. 7 illustrates the SRT of a $4 \times 4$ matrix. From (3), the number of passes $p$ depends on the choice of factors $s_i$; the largest $s_i$ determines the size $B$ of the buffer. As the ‘buffer/permute’ parts of Fig. 7 show, permutation can be performed by filling the $s_1 \times N$ buffer column-wise and then emptying it row-wise into DRAM. In hardware this simply requires rewiring the $\log_2(s_1)$ LSBs of the buffer’s address input as MSBs before it is filled, e.g. rewrite $[b_2b_1b_0]$ as $[b_0b_2b_1]$ for the two-row buffer of Fig. 7.

IV. NEXT-NEIGHBOR REGRidding

Non-cartesian MRI trajectories such as spirals are increasingly
time for fast imaging. Regridding refers to the process of mapping the non-cartesian trajectories to cartesian coordinates so that the 2D-FFT can be used for reconstruction. Conventional regidding methods convolve the data with an interpolation kernel before resampling onto a rectangular grid; they are computationally intensive and are run off-line. The next-neighbor (NN) regridding algorithm [5] is less complex and can provide comparable quality. It maps k-space data in a spiral data-set $S$ to a large $2^q \times 2^q$ sparse substitute matrix $\hat{R}$ with minimal interpolation. $\hat{R}$ starts out as all zeros, and k-space positions $(u_s, v_s) \in \mathbb{R}^2$ are quantized to q-bit integers to obtain ‘next-neighbor’ indices $$(\hat{u}_s, \hat{v}_s) \in \mathbb{N}^2 \text{ in } \hat{R}.$$ Data $S(u_s, v_s)$ are regridded to $(\hat{u}_s, \hat{v}_s)$, although redundant data points can occur when quantization causes multiple k-space positions $(u_s, v_s)$ to regrid to the same location $(\hat{u}_s, \hat{v}_s)$ in $\hat{R}$. The number of redundant points depends on the size of $\hat{R}$, i.e. $q$ and for spiral scans, on the number of overlapping interleaves. $\hat{R}(\hat{u}_s, \hat{v}_s)$ containing redundant points must be averaged before FFT reconstruction. Also, if the positions $(u_s, v_s)$ do not have a uniform density in k-space, $S(u_s, v_s)$ must be multiplied with weighting factors $W(u_s, v_s)$ before reconstruction.

After quantizing each $(u_s, v_s) \rightarrow (\hat{u}_s, \hat{v}_s)$, the regriddr must check for redundancy by verifying that $\hat{R}(\hat{u}_s, \hat{v}_s)$ is zero and does not contain any previously regridded data points. A primary concern of our design was efficiently detecting redundancy when the data are stored in secondary memory (DRAM). Even for a contiguous block of $(u_s, v_s)$, the corresponding quantized $(\hat{u}_s, \hat{v}_s)$ are effectively non-contiguous, making naïve read-back of each $(\hat{u}_s, \hat{v}_s)$ for verification too slow. One-to-one occupancy maps such as a $2^q$ bit-array are also inefficient since $\hat{R}$ is sparse. Instead, we use Bloom filters [13], [14]. The Bloom filter is a randomized data structure that compactly represents a set for querying membership. Its space-efficiency can be traded off against a small probability of false positives.

A Bloom filter consists of $k$ hash functions $h_i$, $i = 1, 2 \ldots k$ and an $m$-bit array ($M$) initially set to all zeros. Given an input $x$, the hash functions generate outputs $h_i(x)$ with range $\{1 \ldots m\}$ that address individual bits of the array $M$. For a set $A = \{x_1, \ldots x_n\}$ with $n$ elements, the filter is programmed by setting the $k$ bits in $M$ that correspond to $h_i(x)$ to one for each
Fig. 8. Flow-diagram for next-neighbor regridding

\[ x \in A. \text{ To check if an item } y \text{ is in } A, \text{ we check the bits in } \mathcal{M} \text{ corresponding to } h_1(y). \text{ If any bit is zero, } y \notin A. \text{ If all the bits are one, we assume } y \in A, \text{ with a false positive probability } P_{FP} \approx (1 - e^{-k \cdot x/m})^k. \]

Hash functions from the \( H_3 \) universal class [15] are computed with only exclusive-OR (XOR, \( \oplus \)) operations, and are thus very efficient for hardware implementation. It is easiest to illustrate \( H_3 \) hashes with an example. If the hash inputs are 8 bits and the outputs (addresses) must be 4 bits, we choose a matrix \( D \) of 4-bit random numbers \( \{d_1, d_2, \ldots, d_8\} \). Then, for an input \( x = 10001010 \),

\[
D = \begin{bmatrix}
1100 \\
0001 \\
1111 \\
1100 \\
0100 \\
1110 \\
1011 \\
1110 
\end{bmatrix}
\]

\[ h(x) = h(10001010) = 1 \cdot d_1 \oplus 0 \cdot d_2 \oplus 1 \cdot d_3 \oplus 1 \cdot d_4 \oplus 0 \cdot d_5 \oplus 0 \cdot d_6 \oplus 1 \cdot d_7 \oplus 0 \cdot d_8 = 1100 \oplus 1001 \oplus 1101 = 1000 \]

Fig. 8 is a flow-diagram of the regriddor. It is optimized for spiral scans with up to \( n = 65536 \) total points and regrrids them to substitute matrices up to \( 4096 \times 4096 \), i.e. \( q \leq 12 \). First, each quantized \((\hat{u}_r, \hat{v}_r)\) is concatenated to a 24-bit number \((\hat{u}_r \cdot 2^{12} + \hat{v}_r)\) that is input to 8 \( (k) \) hashes. The 16-bit hash outputs individually address 64-kbit chunks of the 512-kbit array \( \mathcal{M} \). Since the Bloom filter is always used in a simultaneous query/program mode, the average false positive probability \( \bar{P}_{FP} \) is

\[
\bar{P}_{FP} \approx \frac{1}{n} \sum_{j=1}^{n} (1 - e^{-k \cdot j/m})^k \ll P_{FP} \quad (5)
\]

Offsets of \( \hat{R}(\hat{u}_r, \hat{v}_r) \) containing redundant points are stored and these are averaged after all \( n \) points have been regridded.

The NN regriddor was tested with a GE resolution phantom \((n = 2048 \times 16 \text{ interleaves})\) and a coronary scan \((n = 4096 \times 16)\); both were regridded to different sizes of \( \hat{R} \) as shown in Table III. The eight regridding tests were repeated 1000 times with different hashes, i.e. random numbers obtained from an on-board 32-bit maximal-length linear-feedback shift register. As expected, the mean number of false positive points \( (FP \text{ Mean}) \) is very close to what theory, i.e. (5), predicts: \( n\bar{P}_{FP} \). The regridding time \( t \) is nearly constant, and asymptotically depends only on the number of points \( n \) to be regridded.

Fig. 9 shows reconstructions of the GE resolution phantom with conventional regridding (Kaiser-Bessel, window width 2.5) and next-neighbor regridding \((2048 \times 2048 \text{ substitute matrix } \hat{R})\). The NN-regridded data was transformed with the on-board 2D-FFT in \( \approx 320 \) ms.

V. DISCUSSION

The partial-reconfiguration framework was designed in VHDL and synthesized with Xilinx XST. It was then tested on the FPGA by downloading reconfiguration bitstreams from a PC as well as a CompactFlash memory card (standalone mode). Bus-macro functionality and memory throughput were verified by connecting the IPIF to a ‘stress-test’ PR module that interacted with PLB control signals and performed a wide variety of read/write transfers. Preliminary design and functional simulation of the 2D-FFT and NN-regriddor were done with Xilinx’s high-level System Generator tool; these modules were then synthesized and simulated with ModelSim using behavioral models of the PowerPC and DDR RAM for bit- and cycle-accurate results.

Table IV shows the resource usage of each module; the 2D-FFT is ‘light’ enough to be integrated with the digital receiver of...
[4] on an XC2VP30 FPGA as a complete front-end. At ≈ 4 ms per 256×256 FFT, a 250 images/sec throughput can be sustained with sum-of-squares interpolation of multiple coil images. Thus, real-time reconstruction at near-movie (16 channels @ 15 fps) to movie (8 channels @ 30 fps) frame-rates is possible for applications such as cardiac MRI. Reconstructed image quality is excellent as demonstrated earlier.

**CONCLUSION**

An innovative multi-channel reconfigurable engine for MR imaging has been presented. The block-floating point 2D-FFT module reconstructs up to 16 channels in real-time with very high quality. Partially reconfiguring the FPGA allows idle acquisition/imaging modules to be dynamically replaced with other modules, increasing hardware reuse and thus flexibility as well as economy. The utility of partial reconfiguration beyond conventional cartesian MRI has also been demonstrated with a fast next-neighbor regrider for spiral MRI. Possible future directions include developing real-time modules for accelerated pMRI techniques such as SENSE and GRAPPA.

**APPENDIX: DERIVING TRANPOSITION RUN-TIMES**

If length-M burst transfers are used, larger reads writes are broken up into multiple transfers. Hence, direct transposition requires reading M row-wise blocks (of M units) from DRAM into a B = M^2 buffer so that M column-wise blocks (of M units) can be written out to DRAM. Run-time is:

$$T_{\text{direct}} = \frac{N^2}{M} (\tau_{\text{RW}} + 2Mt) \quad \text{(A.1)}$$

The absolute minimum time $T_{\text{min}}$ (2) is obtained when $M = M_{\text{max}}$, the maximum possible burst transfer length.

The time for the single-radii algorithm is:

$$T_{\text{SRT}} = \frac{\text{passes \times \ steps \times \ rows \times M-blocks \times \ time}}{\text{pass \ step \ row \ M-block}} = \frac{N}{s_1} \times \frac{N}{k_1} \times \frac{N}{M} \times (\tau_{\text{RW}} + 2Mt) \quad \text{(A.2)}$$

and can be minimized to $2T_{\text{SRT}} = 2T_{\text{min}}$ for $p = 2$ and $M = M_{\text{max}}$. This requires a $s_1$-row buffer, i.e. $B = s_1 \times N$.

For $p = 2$ and length-M transfers, we can reduce the buffer to $s_1 \times M$ if the SRT permutation logic is modified for the appropriate offsets. Each step $s$ is now divided into $s/k_1$ sub-steps, and in each sub-step, only $k = s_1 \cdot M/N$ rows are filled row-wise into the $s_1 \times M$ buffer so that column-wise blocks of length-M can be written to main memory. As noted in section III-C, transposing the FFT outputs on their way back to DRAM gives us a ‘free’ first pass. These strategies form the basis for the compact-SRT algorithm we have implemented, with run-time:

$$T_{\text{cSRT}} = \frac{\text{passes \times \ steps \times \ rows \times \ time}}{\text{step \ substep \ row \ row}} = \frac{1}{s_1} \times \frac{1}{k_1} \times \frac{N}{M} (\tau_{\text{RW}} + 2Mt \text{)}$$

Thus, if $M = M_{\text{max}}$ with buffers $[B_{\text{SRT}} = s_1 \times M_{\text{max}}] < [B_{\text{direct}} = M_{\text{max}}]$, $T_{\text{SRT}} = T_{\text{direct}} = T_{\text{min}}$. If $M < M_{\text{max}}$ and $[B_{\text{SRT}} = s_1 \times M] < [B_{\text{direct}} = M^2]$, $T_{\text{SRT}} < T_{\text{direct}}$ since the direct transposte must read and write in length-M bursts while compact-SRT can always read in length-M bursts (because SRT reads are contiguous rows).

**REFERENCES**


MR Image Reconstruction from Pseudo-Hex Lattice Sampling Patterns Using Separable FFT

Jae-Ho Kim
Fred Fontaine


S*ProCom²

Center for Signal Processing, Communications and Computer Engineering Research (S*ProCom²)

Department of Electrical Engineering
The Cooper Union
51 Astor Place, New York, NY 10003.

July 2007
A Hardware Framework for Fast Generation of Multiple Long-period Random Number Streams

Ishaan Dalal
Deian Stefan


S*ProCom2

Center for Signal Processing, Communications and Computer Engineering Research (S*ProCom2)

Department of Electrical Engineering
The Cooper Union
51 Astor Place, New York, NY 10003.

February 2008
System-on-a-Chip for Real-Time Network Intrusion Detection using Counting Bloom Filters

Jared Harwayne Gidansky
Deian Stefan

Technical Report CU/EE/S–TR08F01
Won Second Prize at the IEEE Region 1 Student Conference/Paper Contest,

S*ProCom²
Center for Signal Processing, Communications and Computer Engineering Research (S*ProCom²)

Department of Electrical Engineering
The Cooper Union
51 Astor Place, New York, NY 10003.

April 2008
System-on-a-Chip for Real-Time Network Intrusion Detection using Counting Bloom Filters

Jared Harwayne-Gidansky, Student Member (#80513874)
Deian Stefan, Student Member, IEEE (#80294970)

Department of Electrical Engineering
The Cooper Union for the Advancement of Science and Art
51 Astor Place, Room 406B, New York, NY 10003 USA
Branch Counselor: Dr. Fred L. Fontaine
I Introduction

II Overview of Bloom Filters and Hash Functions
   II-A Definition ................................................................. 2
   II-B Choosing Hash Functions for Hardware Bloom Filters .............. 2
   II-C $H_3$ Hash Functions .................................................. 2
   II-D Formal definition of Bloom Filters .................................. 3
       II-D1 Programming the Bloom Filter (Fig. 2a) ..................... 3
       II-D2 Querying the Bloom Filter (Fig. 2b) .......................... 3
   II-E False Positive Probability: Derivation ............................. 3

III Counting Bloom Filters and FPGA Overview ....................... 4
   III-A Designing Counting Bloom Filters ............................... 4
   III-B Overview of FPGA Architecture .................................. 4

IV Hardware Implementations and Analysis ............................... 4
   IV-A Standard Bloom Filter Implementation ............................ 5
   IV-B Counting Bloom Filter Implementation ........................... 5

V Application: Network Intrusion Detection ............................. 5

VI Conclusion .......................................................................... 6

References .............................................................................. 6
Abstract—Computers face an ever increasing number of threats from hackers, viruses, and other malware; effective Network Intrusion Detection (NID) before a threat affects end-user machines is critical for both financial and national security. As the number of threats and network speeds increase (over 1 gigabit/sec), users of conventional software based NID methods must choose between protection or higher data rates.

To address this shortcoming, we have designed a hardware-based NID system-on-a-chip using novel data structures called Counting Bloom Filters (CBFs). Our design has extremely high throughput (up to 3.3 gigabits/sec) and can successfully detect and mitigate known threats.

Index Terms—Data structures, Computer network security, Field programmable gate arrays

I. INTRODUCTION

Viruses, worms, and hacker attacks on networks cost billions of dollars (Klez virus: $9 billion[1]) and affect hundreds of thousands of users (MSBlast worm: over 350,000 hosts) [2]. Additionally, recent high profile attacks on our national security infrastructure—such as the infiltration of DoD networks by Chinese hackers [3]—reveal that defense against network intrusion is now a matter of not just financial, but also national security.

It is clearly incorrect to assume that each user or individual machine on a network is secure [4]. Therefore, to maintain national security and prevent such massive financial losses, internet service providers and other organizations are moving toward Network Intrusion Detection (NID): preemptive detection of hacking attacks, worms, and other threats present in data when it enters the network, i.e. before it can reach the user’s machine.

Network Intrusion Detection involves looking at patterns in network data that match known threats stored in an existing threat-database. Currently, NID is performed using dedicated devices from manufacturers such as Cisco. These devices are essentially full-blown computers using software solutions such as pattern matching with hash tables to perform NID [5]. However, as both network data rates and the number of potential threats increase, the ability to effectively scan network data in real time using such devices becomes impractical. For example, while Gigabit (1000 megabits/sec) networks are increasing in popularity, a typical Cisco NID device [5] has a maximum scanning rate of only 150 megabits/sec.

As a result, there has been a move toward custom hardware implementations of network intrusion detection which can have significantly higher scan rates (throughput) [2], [6]. Rather than fabricate custom ICs, Field-Programmable Gate Arrays (FPGAs) are used for this purpose. FPGAs are reconfigurable chips that contain programmable logic and can perform operations in parallel, and have become the dominant platform for hardware–based NID. Such NID devices can offer real-time protection against a wide array of threats while achieving throughputs of over 1 gigabit/sec [7].

Given that FPGAs are very different from software programming, what is the most efficient way to implement the pattern matching necessary for NID? Using traditional hash tables is not a good idea since they have a large amount of overhead and require enough memory to store the full threat-database. Instead of hash tables, a much more efficient data structure for hardware-based pattern matching is a Bloom filter, which only stores a compact randomized representation of the threat-database. Consequently, Bloom Filters are faster, have much lower memory requirements, and can be computed with elementary logical operations (XOR, AND, etc) that are the specialty of FPGAs. The basic Bloom Filter can be improved into a Counting Bloom Filter that allows for even more flexibility.

In this project, we design a Network Intrusion Detection “system-on-chip” (i.e., on a single FPGA chip). We first discuss the probabilistic mathematics that make Bloom Filters work, and the evolution into Counting Bloom Filters. Then, we discuss the design process and computer architectural challenges in hardware implementations of Bloom Filters. Finally, we use hardware Counting Bloom Filters to set up a Network Intrusion Detection system, test it with real-world threats and present performance benchmarks.

Outline

The outline of our paper is as follows:

• Overview of Bloom Filters and Hash Functions: The structure and mathematical background of Bloom Filters, followed by performance analysis. •Choosing a hardware-efficient hash function for Bloom Filters.

• Counting Bloom Filters (CBFs) and overview of FPGAs: The limitations of Bloom Filters, and their evolution into CBFs. •Discussion of trade-offs in CBF design. •A brief introduction to FPGAs and their suitability for Bloom Filter implementation.

• Hardware Implementations: Three Bloom Filters and three Counting Bloom Filters running at up to 3.3 gigabits/sec. •Design results and conformance to theoretical predictions.

• Network Intrusion Detection using Counting Bloom Filters: Discussion, implementation and analysis of an innovative one-chip system for real time, full-speed (gigabit ethernet) detection of network threats using hardware Counting Bloom Filters.

• Conclusion: Review of presented material, summary of results, and discussion of future work.
II. OVERVIEW OF BLOOM FILTERS AND HASH FUNCTIONS

A Bloom Filter is a data structure used to determine if a given piece of data belongs to a predefined set (e.g., threat-database). Bloom Filters consist of two key parts: an array of bits (known as the Bloom array) and initialized to ZEROS and one or more hash functions. The Bloom array has a size of m-bits and a total of k hash functions. The Bloom Filter must be programmed with a predefined set which consists of n elements. This is accomplished by hashing each element of the set through the k hash functions; the output of the hash functions are used as indices (addresses) for the m-bit array and the bits at the corresponding addresses are ‘set’ (i.e. changed to ONE).

After programming, to check if a piece of data is a member of the set (known as querying), that data is hashed. If all the corresponding bits of the Bloom array are set (ONE), then that piece of data is almost certainly part of the predefined set; if any of the corresponding bits are not set (ZERO), the data is definitely not a part of the set (i.e., for NID, there is no threat). The check thus consists of performing an AND logical operation on the corresponding bits and checking if the output is ZERO or ONE. Thus, a Bloom Filter can definitely tell if a piece of data does not belong to the set (i.e., is not a threat), but there is a very small chance of a false positive: data is identified as belonging to the set (a threat) when it actually is not. This false positive probability can be controlled and made arbitrarily small, so is not usually an issue.

B. Choosing Hash Functions for Hardware Bloom Filters

The Bloom Filter’s performance depends almost completely on the choice of hash function [9]. A hash function is essentially a mapping between two sets. For two sets U and V, an ideal (or perfectly random) hash function uniformly maps each element in U to a unique element in V (Fig. 1a). Real-world hash functions are rarely ideal and can have collisions, i.e. the mapping is no longer unique and more than one element in U may map to the same element in V (Fig. 1b). Expressed mathematically, for elements x, y ∈ U, x ≠ y, an ideal hash function H(·) will have no collisions, i.e. H(x) ≠ H(y); on the other hand, for a non-ideal hash function G(·) collisions can occur (i.e., G(x) = G(y)).

Collisions in a Bloom Filter lead to false positives and decrease its effectiveness. Hence, hash functions for Bloom Filters in hardware must not only be as close to ideal as possible (i.e., have a low collision rate), but also have low hardware complexity (high throughput). Additionally, since the size of the Bloom array varies depending on the application, the hash function should have variable-length outputs. To choose the hash function for our Bloom Filter, we began by evaluating a number of common hash functions: MD5 (Message Digest 5), SHA-1 (Secure Hash Algorithm-1), and CRC (Cyclic Redundancy Check).

MD5 and SHA-1 are cryptographic hash functions that have large fixed-length outputs, minimal collisions and other security features (e.g. they are one-way and cannot be inverted). Bloom Filters do not require such security features which, unfortunately, make MD5 and SHA-1 extremely computationally intensive [12], [13]. The Cyclic Redundancy Check hash function is used for verifying data integrity (e.g., in Ethernet transmission) and is fairly simple with limited complexity. It is based on polynomial division and can have variable-length output depending on the degree of the polynomial chosen. The biggest problem with CRC is its relatively large collision rate (Table I).

After further research, we considered the H_3 hash functions, a member of the Universal Class of hash functions. These hash functions are low-complexity randomized algorithms that attempt to distribute the data inputs along the set of hash outputs as evenly as possible; with the appropriate choice of random numbers they have low collision rates and can be configured for arbitrary length outputs [14]. Therefore, H_3 hash functions are an excellent choice for Bloom Filters, and have recently been successfully used for this purpose [7], [8]. Table I presents an informative comparison of the output length, collision rate and hardware complexity of the four hash functions we discussed above.

C. H_3 Hash Functions

Given its desirable properties (low-complexity and low collision rate), we decided to use the H_3 hash function. H_3 hash functions can be computed with only exclusive-OR (XOR) operations, making them very efficient for hardware implementation. Let us illustrate H_3 hashes with an example. If the data inputs to the hash function are, say, 8 bits and the hash outputs must be 4 bits, we begin by choosing a matrix D

![Table I: Comparing CRC, MD5, SHA-1 & H_3 hash functions [11]](image)

* CRC and H_2 use 24-bit outputs for this comparison.
of eight 4-bit random numbers \( \{d_1, \ldots, d_8\} \). Then, an element \( x = 00011010 \) is hashed as:

\[
D = \begin{bmatrix}
  d_1 & 1100 \\
  d_2 & 0001 \\
  d_3 & 0100 \\
  d_4 & 1110 \\
  d_5 & 1001 \\
  d_6 & 0110 \\
  d_7 & 1101 \\
  d_8 & 1111 \\
\end{bmatrix}
\]

\[
h(x) = h(00011010) = x \cdot D
\]

where \( \cdot \) (AND) and \( \oplus \) (XOR) are logical operations. The result \( h(x) \) is used to address the bit array of the Bloom Filter; this is demonstrated in Fig. 2. By using different random matrices, as many different hash functions as necessary can be generated. While the idea of \( H_3 \) hashing may seem simplistic when compared to complex cryptographic hash functions such as MD5 and SHA-1, it has been demonstrated that \( H_3 \) hash functions approximate an ideal hash function very well when used with real-world data [15].

D. Formal definition of Bloom Filters

Mathematically, a Bloom Filter is defined as a compact representation of a set \( S = \{x_1, x_2, \ldots, x_n\} \) of \( n \) elements stored in an array of \( m \) bits (all of which are initially 0); the \( k \) hash functions \( \{h_1, \ldots, h_k\} \) are independent and have a range of \( \{1, \ldots, m\} \) (i.e. each hash function maps uniformly to a random number from 1 to \( m \)).

1) Programming the Bloom Filter (Fig. 2a): The set \( S \) must first be programmed in the Bloom Filter (e.g., \( S \) can be a certain set of known threats). For all \( x \in S \), each \( x \) is hashed by the \( k \) hash functions \( h_i \) \( (1 \leq i \leq k) \) and the bits in the Bloom array corresponding to the outputs \( h_i(x) \) are set to ONE as shown in Fig. 2a.

2) Querying the Bloom Filter (Fig. 2b): After the Bloom Filter is programmed, it is put into operation. To check if an item \( y \) is a member of \( S \) (e.g., for intrusion detection, does a packet contain a virus \( y \)’s?), you must hash it with all \( k \) hashes and then check to see if all the bits in the Bloom array corresponding to \( h_i(y) \) are set to ONE (Fig. 2b). If they are not ONE, then \( y \) is definitely not a member of \( S \); but, if they are all ONEs, then we assume \( y \) is a member of \( S \), although it may not be with a very small finite probability (i.e. a false positive) [9].

E. False Positive Probability: Derivation

A false positive occurs when an element that is not in the set is hashed and all the corresponding \( k \) bits in the Bloom array turn out to be ONE; let us derive the probability of false positives for a Bloom Filter.

Consider the \( m \)-bit array; since the output of each hash function is uniformly and independently distributed over the range \( \{1, \ldots, m\} \), the probability that a bit is set (i.e., one) after a single hash of one element is \( \frac{1}{m} \). The probability that a bit is not set (i.e., ZERO) is, therefore, \( \left(1 - \frac{1}{m}\right) \).

To program a Bloom Filter using \( k \) hash functions on a set with \( n \) entries, a total of \( n \cdot k \) hashing operations are performed. The probability \( p \) that a bit is not set after the filter is completely programmed is therefore

\[
p = \text{Prob(bit is not set)} = \left(1 - \frac{1}{m}\right)^{n \cdot k}
\]

Assuming \( m \) is large (which is true in practice), \( p \) can be approximated as

\[
p \approx \lim_{m \to \infty} \left(1 - \frac{1}{m}\right)^{n \cdot k} = \left[ \lim_{m \to \infty} \left(1 + \frac{1}{-m}\right)^{-m \cdot \frac{n \cdot k}{m}} \right] = e^{-\frac{n \cdot k}{m}},
\]

where the expression inside the square brackets is the definition of the exponential \((e)\) function. Thus, the probability \( p \) that a bit is not set is approximately

\[
p = \text{Prob(bit is not set)} \approx e^{-\frac{n \cdot k}{m}}
\]

The complementary probability \((p')\) that a bit is set to ONE is simply \( p' = 1 - p \). A false positive occurs when all \( k \) bits for the entry being hashed are set to ONE, thus the false positive probability, \( p(\text{FP}) \), is

\[
p(\text{FP}) = \text{Prob(k bits are set)} = (p')^k = (1 - p)^k \\
\approx (1 - e^{-\frac{n \cdot k}{m}})^k
\]

Given eq. (4), the false positive probability \( p(\text{FP}) \) is minimized when

\[
\frac{n \cdot k}{m} = \ln(2)
\]

which also gives the optimal value for one of the three variables \( \{m, n, k\} \) if the other two are kept constant; e.g., the optimal number of hash functions \( k \) for a given \( m \)-bit Bloom array that will hold \( n \) entries is \( k = \frac{m}{n} \cdot \ln(2) \).
III. COUNTING BLOOM FILTERS AND FPGA OVERVIEW

A major limitation of Bloom Filters is that once an element has been programmed into the array, it cannot be deleted without erasing and reprogramming the filter from scratch. This is because there is no way to reset the bits corresponding to one element to ZERO and ensure that none of those bits were needed by another element still in the Bloom Filter. To address this deficiency, Bloom Filters have been adapted into Counting Bloom Filters (CBFs).

A CBF is identical to a standard Bloom Filter in concept, but differs in implementation: the Bloom array for a CBF consists of counters and not individual bits (Fig. 3). By using counters for a CBF, you can add and delete items from the CBF. Instead of setting a bit when programming it, you increment (+1) the corresponding counters; when deleting an item you decrement (−1) the corresponding counters. To query the CBF for the existence of some piece of data, you check if all the corresponding counters in the Bloom array are non-zero (like checking if all corresponding bits are set to ONES in a standard Bloom Filter).

A. Designing Counting Bloom Filters

Since the each counter in the CBF array has a limited size, some new challenges are introduced, such as what to do if a counter overflows because too many elements are added. Empirically, a reasonable counter size that minimizes overflows for most applications is four bits [9].

Regardless of the counter size, there is always the possibility that you reach the maximum value of a counter and an overflow occurs. In this situation, researchers suggest [9] it is best to leave the counter at its maximum value. This complication can potentially cause a false negative later (which never occurs with standard Bloom Filters). However, if deletions from the CBF are more-or-less random (as is the case with real-world data), the average time until a false negative occurs is extremely large [9].

Thus, CBFs have both advantages and disadvantages over standard Bloom Filters. They allow dynamic re-programming (add/delete) of the filter, but require more memory for the Bloom array and may result in a rare false negative. The choice of which to use depends on the application; for network intrusion detection, it is desirable to be able to reprogram the Bloom Filter, e.g. if a new worm outbreak occurs (add), or if an attack is no longer a threat because the affected software was patched (delete), etc. Therefore, we use Counting Bloom Filters for our NID implementation.

B. Overview of FPGA Architecture

The basic element of a Field Programmable Gate Array is the 4-bit Look-Up Table (LUT)—a LUT can be programmed to compute any Boolean function of 4 inputs. An FPGA contains thousands of these LUTs connected by reprogrammable interconnects, allowing immense flexibility for implementing any digital function and then optimizing it by performing operations in parallel. Fig. 4a shows the basic structure of a contemporary FPGA; apart from LUTs, the FPGA also contains blocks of fast static random-access memories (block RAMs) and an embedded microprocessor, such as the PowerPC, to perform control and data transfer functions. The FPGA chip itself is usually integrated on a board that contains additional I/O and storage peripherals such as Ethernet controllers, DDR RAM modules, and serial ports.

Now, Bloom Filters require fast RAM for the bit array along with large amounts of logic to perform multiple H3 hashes (XOR operations) in parallel. The LUTs and block RAMs thus make an FPGA ideal for implementing Bloom Filters; a number of such implementations exist [7, 8, 10, 16] for applications such as spell checking, medical imaging and network intrusion detection.

IV. HARDWARE IMPLEMENTATIONS AND ANALYSIS

Before moving on to Network Intrusion Detection, we first designed, optimized and implemented the necessary Bloom Filters on the FPGA. We have implemented three Bloom Filters in both standard and Counting variants on the Xilinx Virtex-4 FXL2 FPGA hosted on the ML403 development board (Fig 4b). The FPGA consists of 5,472 slices (a slice contains two LUTs), 36 block RAMs (of 18-kbits each), and an embedded PowerPC processor. The embedded processor eliminates the need for a separate computer to perform control functions such as TCP/IP data transfer over a network, making intelligent decisions in the NID threat detection process, etc.

For our proof-of-concept NID implementation, we chose a realistically-sized n = 2048-element data set containing signatures for a variety of intrusions: viruses, worms, hacking/unauthorized access attempts and even spam. Given n = 2048, we use eqns. (4) and (5) with the additional constraint...
of a small false positive probability (< 2.5%) to arrive at an optimal Bloom array size of \( m = 16384\) bits and \( k = 8\) hash functions. Since the patterns identifying possible intrusions can be of different sizes, each Bloom Filter accepts a differently-sized input: 2-, 4-, or 8-bytes respectively.

Fig. 5 compares our Bloom Filter’s false positive rate with the false positive rate of a theoretical Bloom Filter as the number of elements in the filter, \( n\), goes from 128 to 2048. As is apparent, our experimental false positive rate closely tracks the theoretical value, with a mean error of 5.4% and standard deviation of 6.2%.

A. Standard Bloom Filter Implementation

The Bloom array (\( m = 16384\)-bits) is stored in Block RAM on the FPGA. To maximize throughput and minimize the control and routing logic, we give each hash function a dedicated I/O port and exclusive access to the block RAM. We achieve this as follows: since each block RAM has two I/O ports (i.e., you can perform two independent reads/writes to different addresses per clock cycle), and we have \( k = 8\) hashes, we split the bit array across \( \frac{k}{4} = 4\) block RAMs that contain \( m = 16384\) bits each. While each hash function has to be modified so its output falls within the range of its “exclusive” block RAM, this does not affect the false positive rate or the effectiveness of the Bloom Filter [7]. An AND gate is used to perform the final membership check.

B. Counting Bloom Filter Implementation

As described in section III, going from a standard Bloom Filter to a counting Bloom Filter design simply involves modifying the Bloom array so that each “bit” is a 4-bit counter instead. We thought that since the 4-bit LUTs are the basic unit of the FPGA and can be used as 4-bit counters, they were a perfect fit for our CBF implementation (which requires 4-bit counters); unfortunately, we found that implementing CBFs this way was to be a poor decision and did not work out for two reasons. First, connecting the LUTs together to have them behave as a single Bloom array of counters required a substantial amount of routing overhead. The second reason was that the Virtex-4 FPGA we were using has only \( \sim 11,000\) LUTs, meaning that even a single 8,000-entry CBF (our NID device requires three 16,384-entry CBFs) could not be implemented (more than the remaining \( \sim 3,000\) LUTs were needed for routing and other operations).

After our initial failure of implementing a CBF with LUTs we successfully implemented a CBF by storing the counters in block RAM; each block RAM (there are 36 on the FPGA) has a capacity of 18-kbits. The RAM was used at a 4-bit depth, and 4-bit adders were included to increment/decrement the RAM counters while comparators were used to check if the counter was ‘set’ (i.e., its value was greater than zero).

CBFs are somewhat slower at programming than a standard Bloom Filter; while the standard Bloom Filter takes one clock cycle to change a bit, the CBF takes two clock cycles to modify a counter; the original value must be read, incremented (for an add) or decremented (for a delete), and then written back to update counter. The extra programming time is immaterial since after the initial programming, the CBF array only needs occasional reprogramming (for updates). We note that checking (reading) the CBF array, which is by far the major operation in an NID device, only requires a single clock cycle; thus, using CBFs has negligible effect on throughput compared to standard Bloom Filters.

Our maximum throughput was approximately 3.3 Gbps, more than an order of magnitude faster than current system from Cisco [5]; a complete summary of our results for the CBF prototype can be found in Table II.

V. APPLICATION: NETWORK INTRUSION DETECTION

As mentioned earlier, networks face a large variety of threats from hackers and malware that can cause billions of dollars of damage and breach national security. Assuming that end-users can protect their systems is both inaccurate and ineffective. Detecting malicious data or events at the network boundaries (Network Intrusion Detection) allows for preemptive and broader protection from threats and creates a single point to maintain network security (as opposed to at every machine in the network). The NID device is installed at the gateway to the network it is protecting.

We have designed and implemented a Network Intrusion Detection (NID) device using Counting Bloom Filters to detect and mitigate threats before they enter a network.

Fig. 6 shows the structure of our NID device. The input data—Ethernet packets (frames)—is fed to the preprocessor which removes the headers, extracts the data and sends it to three Counting Bloom Filters. The filters take 2-, 4- and 8-byte data inputs respectively; this is because the threat patterns (such as ILOVEYOU for the Love Bug worm) can be of variable size; the preprocessor must be used to keep track of patterns.
longer than 8-bytes—we have not implemented this capability yet.

A flow chart showing how our NID device operates is shown in Fig. 7. After the CBFs, an intelligent processor is needed to analyze the output of the CBFs and decide if a threat is detected. Instead of offloading this to a separate computer or microprocessor (e.g., as done by Cisco [5]) we decided to design the NID device as a complete system-on-chip by using the embedded PowerPC processor. Once the PowerPC detects a threat from the CBF outputs, it eliminates the possibility of a false positive by verifying if that piece of data in question is present in the secondary hash table (stored in DDR RAM on the FPGA board). Note that this hybrid “bloom filter + small secondary hash table” approach is much faster and uses less memory than a full hash-table approach.

If the threat is confirmed, the NID device drops the data packet and sends a DESTINATION UNREACHABLE message to the sender (to prevent it from resending the malicious data), and a warning message to the system administrator. For known senders of malicious data, the NID device can also take more drastic steps such as IP blocking. Additionally, since we use CBFs—as opposed to standard Bloom Filters—an administrator is able to modify the database (add or remove signatures) in real-time using the FPGA’s serial port to rapidly respond to emerging threats.

**NID System-on-a-Chip: Testing and Performance**

After designing the NID system-on-a-chip, we tested it with 2048 2-, 4- and 8-byte threat-signatures from the SNORT database (an industry standard database used for network intrusion prevention and detection). We generated 60 megabytes of simulated network data, consisting of 10% malicious packets and 90% benign packets, which were processed by the NID at a peak theoretical rate of ~3.3 gigabits/sec (because all three Counting Bloom Filters must operate at the rate of the slowest filter). This experiment resulted in an average false positive rate of 2.32%, which, surprisingly, was slightly lower than the theoretical false positive rate of 2.55%.

**VI. CONCLUSION**

We presented a single-chip, FPGA-based hardware Network Intrusion Detection (NID) system using Counting Bloom Filters. Our design identifies and can neutralize threats such as hackers and viruses at the network boundary before they can attack end-user computers. As network data rates increase, such real-time preemptive action may prove impractical for software NID solutions.

Counting Bloom Filters are efficient, randomized data structures that are much faster and use less memory than the hash tables usually used in software applications. Consequently, our NID device has a throughput of ~3.3 Gbps—over an order of magnitude higher than typical software-based NID systems from manufacturers such as Cisco.

Future work includes increasing the number and variety of threat-signatures that the system can detect as well as full scale testing on a live network. We hope that by increasing the flexibility and speed of effective Network Intrusion Detection we can help secure computers against malicious attacks, reduce associated financial losses and prevent the compromise of national security.

**REFERENCES**

On Parallelizing the CryptMT Stream Cipher

Deian Stefan
Dave Nummey
Jared Harwayne-Gidansky
Ishaan Dalal


S*ProCom²

Center for Signal Processing, Communications and Computer Engineering Research (S*ProCom²)

Department of Electrical Engineering
The Cooper Union
51 Astor Place, New York, NY 10003.

May 2008
Low Discrepancy Sequences for Reconfigurable Platforms

Ishaan Dalal
Deian Stefan
Jared Harwayne-Gidansky


S*ProCom$^2$

Center for Signal Processing, Communications and Computer Engineering Research (S*ProCom$^2$)

Department of Electrical Engineering
The Cooper Union
51 Astor Place, New York, NY 10003.

July 2008