Improving Design Efficiency for Large-Scale Heterogeneous Circuits

By

Anthony Gregerson

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the
UNIVERSITY OF WISCONSIN-MADISON
2015

Date of final examination: 4/10/2015

The dissertation is approved by the following members of the Final Oral Committee:

Katherine L. Morrow, Associate Professor, Electrical & Computer Engineering

Azadeh Davoodi, Associate Professor, Electrical & Computer Engineering

Nam Sung Kim, Associate Professor, Electrical & Computer Engineering

Parameswaran Ramanathan, Professor, Electrical & Computer Engineering

Eric Bach, Professor, Computer Science

ACKNOWLEDGEMENTS

This work would not have been possible without the continual support, patience, and friendship of my advisor, fellow graduate students, and family. I would like to thank all those who challenged me to be a better researcher, teacher, and student during my time at the University of Wisconsin.

ABSTRACT

Despite increases in logic density, many Big Data applications must still be partitioned across multiple computing devices in order to meet their strict performance requirements. Among the most demanding of these applications is high-energy physics (HEP), which uses complex computing systems consisting of thousands of FPGAs and ASICs to process the sensor data created by experiments at particles accelerators such as the Large Hadron Collider (LHC). Designing such computing systems is challenging due to the scale of the systems, the exceptionally high-throughput and low-latency performance constraints that necessitate application-specific hardware implementations, the requirement that algorithms are efficiently partitioned across many devices, and the possible need to update the implemented algorithms during the lifetime of the system.

In this work, we describe our research to develop flexible architectures for implementing such large-scale circuits on FPGAs. In particular, this work is motivated by (but not limited in scope to) high-energy physics algorithms for the Compact Muon Solenoid (CMS) experiment at the LHC. To make efficient use of logic resources in multi-FPGA systems, we introduce Multi-Personality Partitioning, a novel form of the graph partitioning problem, and present partitioning algorithms that can significantly improve resource utilization on heterogeneous devices while also reducing inter-chip connections. To reduce the high communication costs of Big Data applications, we also introduce Information-Aware Partitioning, a partitioning method that analyzes the data content of application-specific circuits, characterizes their entropy, and selects circuit partitions that enable efficient compression of data between chips. We employ our information-aware partitioning method to improve the performance of the hardware validation platform for evaluating new algorithms for the CMS experiment.

Together, these research efforts help to improve the efficiency and decrease the cost of the developing large-scale, heterogeneous circuits needed to enable large-scale application in high-energy physics and other important areas.

TABLE OF CONTENTS

Abstract ii

1. Introduction 1
   - 1.1. Motivation 1
   - 1.2. Objectives and Contributions 2
   - 1.3. Document Organization 5

2. Background 7
   - 2.1. Field Programmable Gate Arrays (FPGAs) 7
   - 2.2. Processing and Data Acquisition for High Energy Physics 8
   - 2.3. Graph Partitioning 9
     - 2.3.1. Homogeneous Hypergraph Partitioning 9
     - 2.3.2. Heterogeneous Hypergraph Partitioning 10
   - 2.4. Multi-Gigabit Serial Data Transmission 11
   - 2.5. Information Content in Digital Signals 12
     - 2.5.1. Quantifying Information 12
     - 2.5.2. Entropy in Digital Systems 13
     - 2.5.3. Other Information Measurement Metrics 15

3. Mixed-Granularity Reconfigurable Architectures for High Energy Physics Processing 17
   - 3.1. Motivation 17
   - 3.2. Related Work 19
   - 3.3. Characterizing Calorimetry-Based L1 Triggering Algorithms 20
     - 3.3.1. Summary of Common Operations in Trigger Applications 21
     - 3.3.2. Pseudo-code for Algorithms in the Upgraded CMS Level-1 Calorimeter Trigger 22
   - 3.4. Proposed Architectures 35
     - 3.4.1. Direct Arithmetic Logic Implementations (LUT-Based, DSP-Based, and Hybrid LUT-DSP) 36
     - 3.4.2. Compressed Coarse-Grained Look-up Operations (Compressed LUT Tree and Multi-Level Compressed BRAM Table) 37
   - 3.5. Base Implementation Costs 38
   - 3.6. Cost Stability Analysis 40

3.7. Analysis of Implementations 43
3.8. Contributions 44

4. Multi-Personality Graph Partitioning for Heterogeneous Resources 45
4.1. Motivation 45
4.2. Multi-Personality Partitioning Problem Definition 47
4.3. Manipulating Resource Utilization 48
4.4. Related Work 49
4.5. Benchmark Development 52
4.5.1. HDL-Based Benchmark Development 53
4.5.2. CHACO Benchmark Graphs 55
4.6. Multi-Personality Brute-Force Algorithm 56
4.7. Multi-Personality Integer Programming 56
4.7.1. Personality and Partition Selection Constraints 57
4.7.2. Resource Constraints 58
4.7.3. Objective Function 58
4.7.4. Computational and Spatial Complexity 59
4.8. KLFM-Based Partitioning Algorithm Design 60
4.8.1. Base KLFM Algorithm 60
4.8.2. Multi-Personality KLFM 61
4.8.3. Deadlock Prevention for Multi-Personality Graphs 63
4.8.4. Dynamic Gain Buckets 64
4.8.5. Pass-Level Implementation Remapping 67
4.8.6. Multi-Level Partitioning 68
4.9. Implementing the Multi-Personality Partitioning Algorithms 70
4.9.1. Partitioning Algorithms Using Static Personality Mapping 71
4.9.2. Partitioning Algorithms Using Dynamic Personality Mapping 72
4.10. Results 74
4.10.1. Integer-Linear Programming 74
4.10.2. Evaluating KLFM Parameters 76
4.10.3. Performance Validation of Base KLFM Algorithm 77
4.10.4. Cut Size Results for KLFM-Based Algorithms 78
4.10.5. RUR Deviation Results for KLFM-Based Algorithms 79
4.10.6. Run-Time Cost 80

4.10.7. Comparison of Pareto Optimality 81
4.11. Performance Consistency of KLFM Algorithms 82
4.12. Contributions 83
5. Information-Aware Partitioning 84
5.1. Motivation 84
5.2. Related Work 86
5.3. Entropy in Digital Logic Circuits 89
5.3.1. Low-Entropy Input Data 89
5.3.2. Non-Reversible Computations 90
5.3.3. Redundancy / Reduction in Boolean Complexity 90
5.4. Entropy-Based Partitioning in the Regional Calorimeter Trigger 91
5.5. Benchmarks for Information Partitioning 97
5.5.1. Benchmarks and Data Sets 98
5.6. Entropy-Aware Partitioning Results for All Benchmarks 101
5.6.1. Wire Cut Size Results 101
5.6.2. Information Cut Size Results 103
5.6.3. Information Density Results 105
5.6.4. Maximum Theoretical Compression Factors 107
5.6.5. Discussion 109
5.7. Methods for Collecting Entropy Data in Circuits 112
5.7.1. Evaluating Entropy Estimation Methods 112
5.7.2. Structural Estimation 113
5.7.3. Input-Based Estimation 118
5.7.4. Functional Simulation-Based Estimation 121
5.7.5. State-Sampled Functional Simulation-Based Estimation 126
5.7.6. Impact of Entropy Accuracy on Partitioning 130
5.7.7. Summary of Entropy Estimation Methods 132
5.8. Application Case Study: Entropy-Based Partitioning in the CMS Level-1 Trigger Emulator Prototype 133
5.8.1. Prototype Architecture 133
5.8.2. Implementing Compression 134
5.8.3. Temporal Entropy Variation 136
5.8.4. Synchronous L1TE with Bounded Transmission Time 140

5.8.5. Asynchronous L1TE with Bounded Bandwidth 144
5.8.6. Inexact Synchronous Operation 145
5.8.7. Summary of L1TE Case Study 146
5.9. Contributions 147
6. Future Directions 148
7. Conclusion 150
8. Publications 152
9. Appendix A: A Brief History of HEP Triggering Systems 153
9.1. Early Trigger Systems 154
9.2. Trigger Systems of the 1980s 155
9.2.1. Axial Spectrometer Experiment at the Intersecting Storage Rings (1971-1984) 155
9.2.2. The CDF Experiment at the Tevatron (ca. 1987) 156
9.2.3. The DELPHI and L3 Experiments at the Large Electron-Positron Ring (ca. 1989) 158
9.3. Trigger Systems of the 1990s 160
9.3.1. The SLD Experiment at the SLAC Linear Collider (ca. 1991) 160
9.3.2. The NA48, WA98, and NOMAD Experiments at the SPS (ca. 1992-1997) 161
9.3.3. The BaBar Experiment at the PEP-II Collider (ca. 1999) 164
9.4. Trigger Systems of the 2000s 167
9.4.1. The H1 Experiment at the upgraded-luminosity HERA Accelerator (ca. 2001) 171
9.4.2. Upgrades to the CDF Experiment for Tevatron Run-II (ca. 2001) 172
9.4.3. The CDF Upgrade for Tevatron Run-IIb (ca. 2005) 175
9.4.4. The ATLAS, CMS, and LHCb Experiments at the Large Hadron Collider (ca. 2009) 175
9.5. Trends in Future Trigger Design 179
10. Bibliography 181

List of Figures

Figure 1. Entropy (in bits) versus probability of having a value of '1' for a binary variable. 14
Figure 2. The required throughput of trigger systems has been increasing exponentially over time. ..18
Figure 3. One of 126 boards used for processing calorimeter trigger algorithms in the original CMS trigger. The overall system is partitioned across over 700 FPGAs and ASICs and thousands of RAM-based LUTs. 21
Figure 4. A comparison of a simple, full look-up table for the EPIM (a) with a two-level compressed look-up table (b) using calorimeter energy readings (ECAL & HCAL) as inputs. 38
Figure 5. A section of the datapath for performing electromagnetic/hadronic particle identification in the CMS Regional Calorimeter Trigger. 38
Figure 6. Histogram of end-to-end latency requirements for randomly parameterized instances of the LUT-Based Direct design for the Electron/Photon ID (EPIM) algorithm. 41
Figure 7. Total LUT requirements versus number of energy vetoes based on Monte Carlo simulation of EPIM implementation costs. 41
Figure 8. Histogram of the normalized size (in CLBs) of the Monte Carlo generated LUT trees for the EPIM after tree compression. Values are normalized to the base implementation without vetoes. The line indicates the cumulative percentage of Monte Carlo samples that could fit in that number of CLBs. 42
Figure 9. Histogram of the size of the Monte Carlo generated EPIMs converted to memory images for mapping to a look-up table and compressed with the LZW algorithm. Values are normalized to the base implementation without vetoes. 42
Figure 10. Cut sizes for ILP-based partitioning with a run time of 48 hours, normalized to MF statically-mapped partitioning. Missing bars indicate that ILP could not find a solution. 74
Figure 11. ILP cut sizes (normalized to MF) for graphs with less than 30,000 nodes for different amounts of runtime. ILP was unable to find a solution for cti when given runtime equal to MF. 75
Figure 12. Average of the cut size results for all benchmark graphs using DMP and different gain bucket types, normalized to the Multi-Personality (MP) bucket type. Lower values are better.76
Figure 13. Cut size results for KLFM-based partitioners, normalized to map-first, and presented separately for HDL-derived graphs and graphs from partitioning benchmark sets. 79
Figure 14. RUR deviation results for all benchmarks. Lower values are better. 80
Figure 15. Pareto comparison of cut size vs. RUR deviation for all KLFM-based partitioners. 81
Figure 16. Performance consistency of the multi-personality KLFM algorithms. Each point represents a single benchmark and the algorithm's relative performance (in terms of RUR deviation and cut size) relative to the average result of all algorithms on that benchmark. For both performance metrics, lower (more negative) values are better. 82
Figure 17. Histogram of the bits of Shannon entropy per wire in a subsection of the RCT, measured during post-synthesis simulation. The line represents the cumulative percentage of wires (as labeled on the right axis) that have at least the amount of entropy indicated on the y axis. 92
Figure 18. Plot of wire cut size versus information cut size in the RCT for 500 runs of a KLFM algorithm using a traditional wire-minimization objective function. 93
Figure 19. Wire cut size vs. Information cut size in the RCT when using KLFM algorithms with Wire-based and Entropy-based cost functions. 95
Figure 20. The maximum lossless compression factor achievable for compression using entropy coding on inter-partition cuts produced by wire-based partitioning and entropy-based partitioning. 96
Figure 21. Absolute wire cut sizes using wire-based and entropy-based partitioning. 101
Figure 22. Normalized wire cut sizes using wire-based and entropy-based partitioning. 102
Figure 23. Absolute information cut sizes for wire-based and entropy-based partitioning. 103

FIGURE 24. NORMALIZED INFORMATION CUT SIZES FOR WIRE-BASED AND ENTROPY-BASED PARTITIONING. 104
FIGURE 25. INFORMATION, IN TERMS OF BITS OF SHANNON ENTROPY PER WIRE FOR WIRE-BASED AND ENTROPY-BASED PARTITIONING. 105
FIGURE 26. NORMALIZED INFORMATION, IN TERMS OF BITS OF SHANNON ENTROPY PER WIRE FOR WIRE-BASED AND ENTROPY-BASED PARTITIONING. 106
FIGURE 27. ABSOLUTE MAXIMUM EFFECTIVE COMPRESSION FACTORS FOR WIRE-BASED AND ENTROPY-BASED PARTITIONING. 108
FIGURE 28. NORMALIZED MAXIMUM EFFECTIVE COMPRESSION FACTOR FOR WIRE-BASED AND ENTROPY-BASED PARTITIONING. 109
FIGURE 29. ABSOLUTE MAXIMUM EFFECTIVE COMPRESSION FACTORS FOR WIRE-BASED AND ENTROPY-BASED PARTITIONING. THE CHART SCALE IS ZOOMED IN TO ILLUSTRATE SCALE OF SMALLER VALUES. 110
FIGURE 30. NORMALIZED MAXIMUM EFFECTIVE COMPRESSION FACTOR FOR WIRE-BASED AND ENTROPY-BASED PARTITIONING. THE CHART SCALE IS ZOOMED IN TO ILLUSTRATE SCALE OF SMALLER VALUES. 111
FIGURE 31. A 2-INPUT OR GATE WITH INPUTS A & B AND OUTPUT Q. 113
FIGURE 32. AN EXAMPLE CIRCUIT THAT USES TWO COMPARATORS TO PRODUCE TWO SINGLE-BIT OUTPUTS (GT100 AND GT50) THAT HAVE CORRELATED VALUES DUE TO THE LOGIC OF THE CIRCUIT. 115
FIGURE 33. HISTOGRAM OF THE ERROR (IN BITS OF ENTROPY) FOR ALL SIGNALS IN THE *RCT* CIRCUIT WHEN USING STRUCTURAL ENTROPY ESTIMATION. 117
FIGURE 34. AVERAGE ABSOLUTE ESTIMATION ERROR FOR ALL BENCHMARKS USING STRUCTURAL ESTIMATION. 118
FIGURE 35. HISTOGRAM OF THE ERROR (IN BITS OF ENTROPY) FOR ALL SIGNALS IN THE *RCT* CIRCUIT WHEN USING INPUT-BASED ENTROPY ESTIMATION. 120
FIGURE 36. AVERAGE ABSOLUTE ESTIMATION ERROR FOR ALL BENCHMARKS USING INPUT-BASED ESTIMATION. 121
FIGURE 37. HISTOGRAM OF THE ERROR (IN BITS OF ENTROPY) FOR ALL SIGNALS IN THE *RCT* CIRCUIT WHEN USING FUNCTIONAL SIMULATION OF THE CIRCUIT AND INCLUDING DATA FROM ALL SIGNALS VISIBLE IN FUNCTIONAL SIMULATION. 123
FIGURE 39. NORMALIZED LUT SYNTHESIS COST FOR THE BENCHMARK CIRCUITS WHEN SYNTHESIZING NORMALLY AND REQUIRING THAT THE SYNTHESIS TOOL KEEP ALL FUNCTIONAL SIGNALS FOR USE WITH FUNCTIONAL-SIMULATION-BASED ENTROPY ESTIMATION. 124
FIGURE 38. NORMALIZED FLIP-FLOP SYNTHESIS COST FOR THE BENCHMARK CIRCUITS WHEN SYNTHESIZING NORMALLY AND REQUIRING THAT THE SYNTHESIS TOOL KEEP ALL FUNCTIONAL SIGNALS FOR USE WITH FUNCTIONAL-SIMULATION-BASED ENTROPY ESTIMATION. 125
FIGURE 40. SPEED-UP OF ALL SIMULATION-BASED ENTROPY GATHERING METHODS, NORMALIZED TO THE CPU TIME REQUIRED FOR POST-SYNTHESIS SIMULATION. 126
FIGURE 41. SPEED-UP OF ALL SIMULATION-BASED ENTROPY GATHERING METHODS, NORMALIZED TO THE CPU TIME REQUIRED FOR POST-SYNTHESIS SIMULATION. THE CHART IS ZOOMED TO ILLUSTRATE THE SCALE OF SMALLER VALUES. 127
FIGURE 42. NORMALIZED LUT SYNTHESIS COST FOR THE BENCHMARK CIRCUITS WHEN SYNTHESIZING NORMALLY, KEEPING ALL FUNCTION SIGNALS, OR KEEPING ONLY STATE SIGNALS. 128
FIGURE 43. NORMALIZED FLIP-FLOP SYNTHESIS COST FOR THE BENCHMARK CIRCUITS WHEN SYNTHESIZING NORMALLY, KEEPING ALL FUNCTION SIGNALS, OR KEEPING ONLY STATE SIGNALS. 129
FIGURE 44. HISTOGRAM OF THE ERROR (IN BITS OF ENTROPY) FOR ALL SIGNALS IN THE *RCT* CIRCUIT WHEN USING FUNCTIONAL SIMULATION OF THE CIRCUIT AND INCLUDING DATA FROM ONLY STATE OUTPUTS IN FUNCTIONAL SIMULATION. 130
FIGURE 45. AVERAGE ABSOLUTE ENTROPY ESTIMATION ERROR FOR ALL BENCHMARKS WHEN USING FUNCTIONAL SIMULATION OF THE CIRCUIT AND INCLUDING DATA FROM ONLY STATE OUTPUTS IN FUNCTIONAL SIMULATION. 130
FIGURE 46. NORMALIZED INFORMATION CUT SIZES ACHIEVED WHEN USING ENTROPY-BASED PARTITIONING WITH DIFFERENT METHODS OF COLLECTING THE WIRE ENTROPY DATA. VALUES ARE NORMALIZED TO PARTITIONING WITH NO ENTROPY INFO (WIRE-BASED PARTITIONING). LOWER VALUES ARE BETTER. 131
FIGURE 47. COMPRESSION FACTORS FOR VARIOUS HUFFMAN SYMBOL SIZES IN THE RCT. 135

FIGURE 48. HISTOGRAM OF THE STANDARD DEVIATION IN WIRE ENTROPY (AS A PERCENTAGE OF EACH WIRE'S AVERAGE ENTROPY) FOR THE *RCT* BENCHMARK WHEN COMPUTING ENTROPY OVER 1 μS TIME WINDOWS. 137
FIGURE 49. STANDARD DEVIATION IN INFORMATION CUT SIZE FOR THE RCT USING DIFFERENT ENTROPY COLLECTION TIME WINDOWS. 138
FIGURE 50. HUFFMAN COMPRESSED FRAME SIZES FOR THE FIRST 4000 INPUT DATA SETS FOR THE RCT CIRCUIT, PARTITIONED FOR THE L1TE. 140
FIGURE 51. THE REDUCTION IN THE REQUIRED INTER-PARTITION BANDWIDTH FOR THE SYNCHRONOUS L1TE WHEN USING FRAME COMBINING TO REDUCE THE VARIATION IN THE DATA THAT MUST BE TRANSMITTED IN EACH TIME WINDOW.....142
FIGURE 52. NORMALIZED REDUCTION IN REQUIRED BANDWIDTH WHEN USING FRAME-COMBINING WITH A TIME WINDOW OF 8 FRAMES FOR NORMAL DISTRIBUTIONS WITH INCREASING DEVIATION IN FRAME SIZES.....143
FIGURE 53. FRAME TRANSMISSION TIMES (IN μS) FOR THE FIRST 4000 FRAMES ON THE ASYNCHRONOUS L1TE WHEN USING THE SAME BANDWIDTH AS THE SYNCHRONOUS L1TE. 144
FIGURE 54. ONE OF THE MANY L1 PROCESSING BOARDS FROM THE L3 EXPERIMENT. 159
FIGURE 55. DELPHI'S TREE-BASED LOOK-UP TABLE. 160
FIGURE 56. TRACKING SENSORS USED IN THE NA48 EXPERIMENT. 162
FIGURE 57. ARCHITECTURE OF THE NA48 TRIGGER. 163
FIGURE 58. HIGH-LEVEL DIAGRAM OF THE BABAR TRIGGER AND DAQ SYSTEM.....166
FIGURE 59. BABAR L1 TRIGGER ALGORITHMS. 167
FIGURE 60. EVENT RATES IN THE PHENIX EXPERIMENT FOR DIFFERENT IONS. 168
FIGURE 61. DATA ACQUISITION IN THE STAR EXPERIMENT. 169
FIGURE 62. SEGMENT RECONSTRUCTION SCHEME IN THE H1 AT HERA. 171
FIGURE 63. LEVEL-1 AND LEVEL-2 TRIGGER ARCHITECTURE FOR THE CDF EXPERIMENT.....174
FIGURE 64. EVENT PROCESSING TIMES FOR DIFFERENT PROCESSOR ARCHITECTURES IN CDF RUN-IIB. 176
FIGURE 65. THE CMS TRIGGER ARCHITECTURE (LEFT) COMPARED TO A TRADITIONAL ARCHITECTURE (RIGHT). 178

LIST OF TABLES

TABLE I. MAX FREQUENCY AND RESOURCE UTILIZATION (IN PERCENT OF THE TX240T'S RESOURCES) FOR IMPLEMENTATIONS OF THE CMS EXPERIMENT'S ELECTRON/PHOTON ID MODULE USING SEVERAL DIFFERENT ARCHITECTURES. UTILIZATION VALUES ARE IN PERCENT, AND REPRESENT THE COST OF A SINGLE INSTANCE OF THE MODULE. 39

TABLE II. RESOURCE UTILIZATION FOR ELECTRON/PHOTON ID ARCHITECTURES WHEN MAXIMALLY MULTIPLEXED. 40

TABLE III. RESOURCE UTILIZATION RANGES FOR ELECTRON/PHOTON ID ARCHITECTURES ACCOUNTING FOR FREQUENCY AND RESOURCE RANGES FROM MONTE CARLO SIMULATION. 44

TABLE IV. MULTI-PERSONALITY BENCHMARK GRAPHS, SORTED BY ASCENDING NUMBER OF NODES. 53

TABLE V. NORMALIZED CUT SIZE AND DEVIATION FROM TARGET RESOURCE UTILIZATION RATIO FOR DIFFERENT INITIAL NODE PERSONALITY SELECTION METHODS (RANDOM VS. PROPORTIONAL). 77

TABLE VI. CUT SIZE COMPARISON OF MF, HMETIS, AND THE BEST OF THE 40 PARTITIONERS IN THE WALSHAW PARTITIONING ARCHIVE FOR THE SINGLE-RESOURCE VERSIONS OF OUR CHACO-BASED BENCHMARK GRAPHS. LOWER VALUES ARE BETTER. 78

TABLE VII. GEOMETRIC MEANS OF THE NORMALIZED CUT SIZE, ABSOLUTE RUR DEVIATION, AND NORMALIZED WALL-CLOCK RUN TIMES FOR THE SIX KLFM PARTITIONING STRATEGIES. 81

TABLE VIII. SUMMARY OF BENCHMARK CIRCUITS USED IN INFORMATION-AWARE PARTITIONING. 98

TABLE IX. TRUTH TABLE WITH PROBABILITIES FOR OR GATE ENTROPY-COMPUTATION EXAMPLE. 114

TABLE X. COMPARISON OF CPU TIME REQUIRED FOR ENTROPY GATHERING USING POST-SYNTHESIS SIMULATION AND STRUCTURAL ESTIMATION. 116

TABLE XI. COMPARISON OF CPU TIME REQUIRED FOR ENTROPY GATHERING USING POST-SYNTHESIS SIMULATION AND INPUT-BASED ESTIMATION. 119

TABLE XII. COMPARISON OF CPU TIME REQUIRED FOR ENTROPY GATHERING USING POST-SYNTHESIS SIMULATION AND FUNCTIONAL SIMULATION-BASED ENTROPY ESTIMATION. 122

TABLE XIII. PUBLICATIONS DERIVED FROM PROJECTS IN DESCRIBED IN THIS DOCUMENT. 152

1. Introduction

1.1. Motivation

In recent years, the rise in prominence of 'Big Data' − a catch-all term for computing problems based on vast datasets generated from domains ranging from scientific experiments to internet search to social networking − has helped to maintain interest in designing large, many-node distributed computing systems in spite of continued increases in per-device logic density. Many systems designed to handle Big Data applications are becoming more and more heterogeneous, as graphical processing units (GPUs) and field programmable gate arrays (FPGAs) are increasingly used to improve these applications' throughput or response time. A prominent example of this type of system is present in the Large Hadron Collider (LHC), a massive high energy physics experiment that processes Petabytes of data per day. To analyze the data produced in its experiments, LHC depends on real-time computing systems made up of thousands of application-specific integrated circuits (ASICs), FPGAs, and general-purpose processors. The original design for these systems was an arduous and expensive process, relying heavily on a manual design process, and resulted in a highly complex system that could not be easily changed.

With the LHC's computing systems due for an overhaul as part of a large-scale upgrade to enable the next generation of high-energy physics experiments [1], there is renewed motivation to rethink the system architecture and automate prototyping techniques to improve the efficiency and reduce the complexity of the system. Previous research in the specific area of high-energy physics computing architectures and on the more general problem of system partitioning does not adequately account for the scale, degree of heterogeneity, or performance requirements of the LHC. This underscores the need for new algorithms capable of improving design automation for massive, heterogeneous computing systems.

The motivation to efficiently partition logic and efficiently utilize communication resources is not limited to the LHC, and applies more broadly to other applications that are distributed across large hardware systems. Multi-chip systems based on heterogeneous FPGAs are frequently used for emulation

and prototyping in ASIC design and for other high throughput, latency-sensitive application domains such as network processing and high-frequency trading. Heterogeneous processor clusters are also increasingly common in supercomputing and cloud computing, where applications must be mapped to heterogeneous compute nodes, and data transmission between partitions is expensive, whether it is at the inter-chip, inter-rack, inter-cluster, or inter-datacenter level. We will examine how the algorithms we developed for automating the design of high-energy physics systems can be more generally applied to the broader problem of partitioning an application across heterogeneous, distributed resources in order to efficiently utilize the available logic and minimize communication costs.

1.2. Objectives and Contributions

The goal of this dissertation is to improve the design process for large-scale, heterogeneous digital systems, with a focus on the special challenges presented by the exceptionally large and high-performance distributed hardware systems used in high-energy physics. Such systems have an essential role in enabling cutting-edge scientific research, but present unique problems that often go unaddressed by the commercial market. This presents an excellent opportunity for meaningful contributions from academic research. The research work we will describe in this document is divided into three major projects.

First, we analyze the design process of a state-of-the-art, real world application in the domain of high energy physics: the Regional Calorimeter Trigger (RCT) system for the Compact Muon Solenoid (CMS) experiment. Because high energy physics systems require their algorithms to co-exist on hundreds of chips, feature large amounts of interconnection, and have tight performance constraints, they can be difficult to modify even when built on reprogrammable hardware. This can be a significant limitation for physics researchers, as they may want to evolve or replace their algorithms over time to adapt to experimental conditions or search for new phenomena [2]. To address this problem, we develop new FPGA-based architectures for implementing important operations in physics processing that reduce variations in performance and resource costs. These design allowing physicists to more easily alter and

upgrade their algorithms without requiring major redesigns of the hardware systems that implement the algorithms. As part of our design exploration process, we also attempt to characterize some of the key challenges presented by the design the upgraded RCT system, which serve as motivational factors for our later work on automated design partitioning.

Second, we formulate a new version of a graph partitioning problem, multi-personality partitioning (MPP), in order capture the new challenges presented when partitioning designs for implementation on large heterogeneous systems with flexible resources, such as FPGAs and supercomputers. Although circuit partitioning has been the subject of much research in the past, prior work does not adequately model heterogeneous devices, resulting in either inefficient use of device hardware or high communication costs between devices. For large-scale applications that require many devices, this can lead to more expensive, complex, and difficult to maintain hardware systems. To address these challenges, we develop general and targeted heuristics for optimizing the use of device hardware and communication resources by creating automated algorithms for solving multi-personality partitioning problem, implement them in partitioning software, and analyze their efficacy on a set of modern-sized benchmarks.

Third, we extend our research into partitioning by attempting to improve the ability of automated algorithms to optimize partitioning decisions based on functional knowledge of application-specific circuits. To this end, we develop and evaluate automated methods for analyzing the data content of circuit signals and exploiting signal redundancy to decrease the cost of inter-partition communication in large, partitioned systems. We create a benchmark set of large application-specific circuits with characteristic input data sets and demonstrate that many of the signals within these circuits have low average information content as measured by their Shannon entropy. We demonstrate that we can exploit knowledge of signal entropy with automated partitioning algorithms to create partitions with significantly lower cross-partition information cut sizes. We will explore different methods to characterize signal entropy, including both simulation and logic analysis-based approaches, and evaluate their relative accuracy, computation times, and impact on circuit quality. Finally, we detail one proof-of-concept

application by introducing inter-partition data compression to the Level-1 Trigger Emulator, a hardware validation platform used in the development of high energy physics applications.

The specific contributions of each project are enumerated below.

Flexible FPGA-based Architectures for High Energy Physics

- We develop firmware and software tools used in the design exploration process of the high-luminosity RCT system for the CMS experiment at the Large Hadron Collider.
- We characterize the key algorithms used in calorimeter-based particle triggering applications and their related implementation costs on heterogeneous FPGAs.
- We design and evaluate logic architectures for creating coarse-grained look-up tables to allow flexible implementation of physics algorithms. We also perform a Monte Carlo analysis of cost variations in these architectures based on changes to the parameters of the algorithms to quantify the impact of altering key algorithm parameters.

Multi-Personality Graph Partitioning

- We provide a formal definition of the new partitioning problem for heterogeneous, flexible-resource systems, describe its complexity, and introduce a new quality metric for balancing resource utilization across heterogeneous processing nodes.
- We create a set of large multi-personality graph benchmarks based on graphs derived from publicly available, real-world circuits and software task graphs
- We design an integer linear programming (ILP) model for multi-personality partitioning, and evaluate its performance and spatial complexity using sophisticated third-party ILP solvers.
- We develop software for implementing MPP using a modified version of the Kernighan-Lin/Fiduccia-Mattheyses (KLFM) heuristic to support multiple personalities, including modification to gain bucket data structures, clustering, personality remapping, and deadlock prevention. In our analysis, we evaluate how various implementation choices affect solution quality and run time.

Information-Aware Circuit Partitioning

- We evaluate different metrics for characterizing signal information content in circuits, including simulation-based vs. structure-based metrics.
- We develop methods measuring and estimating the entropy of wires within a circuit and experimentally evaluate them in terms of their computation time, measurement accuracy, and impact on circuit synthesis. We integrate entropy data into the partitioning algorithms developed for MPP, prove that it is possible to achieve information cut sizes that are significantly smaller than conventional wire cut sizes, and observe how partitioning decisions vary when using entropy to select partitioning rather than traditional metrics.
- We quantify the advantages of information-aware partitioning through compression of inter-partition signals using entropy coding.
- We identify other application domains that may benefit from content-aware partitioning by developing a set of diverse application-specific benchmark circuits with representative data sets.
- We demonstrate the practical use of information-aware partitioning on the RCT design as implemented on the Level-1 Trigger Emulator to identify compressible signals and reduce system bandwidth requirements and improve performance.

1.3. Document Organization

The remainder of this document is organized as follows: In Chapter 2, we introduce background information on FPGAs, processing systems for high energy physics, graph partitioning, and high speed serial data transmission, and information measurement in digital systems. In Chapter 3, we describe our work designing flexible hardware designs for the Level-1 Regional Calorimeter Trigger as part of the High-Luminosity Large Hadron Collider project. In Chapter 4, we introduce a new type of graph partitioning problem − multi-personality partitioning (MPP) − designed for partitioning heterogeneous logic devices. We also describe the development and implementation of multiple algorithms for solving

the MPP problem and evaluate their efficacy when partitioning large-scale circuit designs for FPGAs. In Chapter 5, we apply concepts from the field of information theory to the circuit partitioning problem, developing an entropy-based algorithm capable of partitioning circuits to minimize the bandwidth between chips when using compression. We demonstrate that our algorithm can significantly improve the performance of a hardware emulator system built for validating the design of high energy physics computing systems. In the remaining chapters, we summarize our contributions, outline potential future research directions, and enumerate research publications produced from our work. We also include a supplementary survey on triggering systems for high energy physics in the Appendix that provides additional historical context for our research into the development of electronics for physics experiments.

2. Background

In this section we provide a brief introduction to some of the core technology and terminology which will serve as a basis for our later description of the research in high energy physics architecture, heterogeneous netlist partitioning, and information-aware circuit partitioning. Additional related work for each project is included within their respective chapters and in the appendix.

2.1. Field Programmable Gate Arrays (FPGAs)

Field Programmable Gate Arrays are devices commonly used for implementing digital logic circuits. A key feature of FPGAs is that they are reconfigurable: that is, they can be programmed multiple times, allowing designers to change the logic functions implemented on the same FPGA. In this way, FPGAs offer some compromises between the performance and efficiency of application-specific integrated circuits (ASICs) and the flexibility of software running on general-purpose processors. Unlike the ASIC design model, where a custom circuit is built to implement a specific logic design, FPGAs are mass produced in different sizes, and designers must fit their logic into the available resources of the FPGA. FPGAs are frequently used in the following types of applications:

- Applications that need higher throughput, lower latency, or better efficiency than is offered by software, but must also be easily upgraded or must support multiple different designs.
- Specialized, small market applications where the number of devices needed does not justify the high fixed costs of ASIC development.
- Time-critical circuit development or design exploration, where the time to produce an ASIC is prohibitive.
- Circuit emulation for the purpose of functional testing and debugging in the process of developing ASICs.

Modern FPGAs contain heterogeneous logic resources. In our work, we will use FPGAs produced by Xilinx [3] in our design processes, but for the purposes of our work they are not substantially different

from FPGAs from other vendors. There are three primary resource types on these FPGAs: small look-up table (LUT)-based blocks that can implement generic functions with a small number of inputs (we refer to these as configurable logic blocks or CLBSs), high density configurable memories (block RAMs or BRAMs), and units for performing fast arithmetic operations, such as multiplication and addition (digital signal processing blocks or DSPs). Circuits are implemented by programming the LUTs and connecting them to one another and the other resources using a configurable routing network. Internally, the resources are organized into blocks and regions, using a hierarchal routing scheme that allows high intra-block connectivity and restricts the amount of global connectivity. For off-chip communication, FPGAs have traditionally contained a large number of parallel I/O pins, but in recent years some FPGAs have added multiple high-speed serial transceivers. We will discuss these further in Section 2.4.

2.2. Processing and Data Acquisition for High Energy Physics

High-energy physics (HEP) is a scientific discipline that studies particle interactions at energy levels that exceed those that occur naturally on earth. HEP relies on the use of particle accelerators/colliders, enormous machines that accelerate sub-atomic particles to near-luminal velocities and then collide them together. An accelerator houses one or more *experiments*: collections of complex sensors, signal processing, and computing systems that collect and analyze data from the collections. At various points in this document, we will refer to systems belonging to the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC), the largest and most powerful particle accelerator in the world (soon to be upgraded to the High Luminosity LHC (HL-LHC) [4]; this upgrade serves as part of the motivation for our work).

One of the characteristic processing challenges of HEP is the massive amount of data that is produced by experiments. The amount of data produced by an experiment is the product of the rate at which the accelerator can produce collision events and the amount of data that is collected for each event. For instance, the LHC can produce new data sets every 25 ns, and CMS's sensors collect data at a rate of 1 Petabyte (PB) per second [5]. It is infeasible to store such large amounts of data, so HEP experiments

employ *trigger systems* − specialized processing units that perform real-time analysis of the sensor data and rapidly decide which data should be acquired for further study and which data should be immediately discarded. Because sensor data must be buffered during this decision process, trigger response time must be very fast in order to keep buffer sizes reasonable. We will describe the specific properties of trigger algorithms in greater depth in Chapter 3.

2.3. Graph Partitioning

Graph partitioning is one of the classic NP-Complete problems in computing [6]. Partitioning has many practical applications, but within the context of this document, we focus on its application to the design of digital systems. Circuits or computational tasks can be expressed as hypergraphs, where nodes are used to model circuit element, logic function, or tasks (depending on the level of granularity) and edges are used to model wires, buses, or communication channels. Partitioning such graphs is a key step in processes such as circuit floorplanning, rapid prototyping of multi-chip systems, and scheduling of large software tasks in distributed computing.

2.3.1. Homogeneous Hypergraph Partitioning

The hypergraph bipartitioning problem is defined based on two components: a constraint related to balancing the nodes in each partition and an optimization objected related to the edges that span the partitions. Formally, the node constraint is expressed as follows: Given a hypergraph $G = (V, E)$, composed of nodes $v \in V$ and hyperedges $e \in E$, with node and edge weights $w(v)$ and $w(e)$ respectively, divide all nodes of G into two disjoint sets, V_1 and V_2 (i.e., $V_1 \cup V_2 = V,\ V_1 \cap V_2 = \emptyset$), subject to the constraint that the sum of node weights in the two subsets are equal:

$$\sum_{v_i \in V_1} w(v_i) = \sum_{v_j \in V_2} w(v_j) \tag{1}$$

For a circuit netlist, the node weight might represent the area or number of resources consumed by a given logic block, and for a software task it might represent the number of cores or memory footprint

needed by the task. For most practical problems it is sufficient to simply require that the partitions are of similar size rather than being equal. For example, if the constraint were modified to ensure the difference in size was less than 1%, the previous equation becomes:

$$\left| \sum_{v_i \in V_1} w(v_i) - \sum_{v_j \in V_2} w(v_j) \right| \leq 0.01 \sum_{v \in V} w(v) \qquad (2)$$

The actual imbalance threshold can be chosen to ensure that each partition can be implemented within a given resource budget. On its own, the node constraints given in Equations (1) and (2) are formulations of the well-studied set partitioning problem [7]. The graph partitioning problem uses set partitioning as a constraint for the graph's vertices and adds an optimization goal based on the graph topology. The other component of the graph partitioning problem is the optimization objective. The most common objective is to minimize the sum of the weights of all edges (or hyperedges in the case of hypergraphs) that span between the partitions − a metric referred to as the *span* or *cut size*. Cut size reduction is a useful optimization goal because inter-partition connections may model expensive communication paths, such as global wires on a circuit or inter-node communication in a supercomputer. Formally, if we define $e_{v1,v2,\ldots vk}$ as the hyperedge connected to nodes $v_1, v_2, \ldots, v_k$, then:

$$cut\ size(V_1, V_2) = \sum_{e \in E} w(e_{vi,\ldots vj}) \quad \text{s.t. for some } i, j,\ v_i \in V_1 \text{ and } v_j \in V_2 \qquad (3)$$

Homogeneous graphs were long used as a model for partitioning computing systems, and the majority of partitioning algorithms were developed for homogenous graphs. However, the transition to heterogeneous computing resources motivated the development of a newer graph model, as we will discuss in the next section.

2.3.2. Heterogeneous Hypergraph Partitioning

The homogenous node model used in previously-described graph partitioning problem is insufficient to model many modern digital systems, such as applications to be implemented in Field-Programmable Gate Arrays (FPGAs), Systems-on-Chip (SoCs), and supercomputers, as these systems often include

multiple heterogeneous resource types. This has led to a new problem, referred to alternately in academic literature as *heterogeneous*, *multi-constraint*, or *complex-resource graph partitioning*: Given a graph with nodes that have weights in *R* resource types, the partitioning solution must balance all *R* resources within their specified maximum margins of imbalance. The weight of each node is now described by an *R*-entry weight vector *w*. If we define $w_r(v)$ as node *v*'s weight in resource *r*, and I_r as the maximum weight imbalance between partitions for resource *r*, then the imbalance constraint equation becomes:

$$\frac{\left| \sum_{v_i \in V_1} w_r(v_i) - \sum_{v_j \in V_2} w_r(v_j) \right|}{\sum_{v_k \in V} w_r(v_k)} \leq I_r \quad \forall \quad r \in 1..R \tag{4}$$

Although multi-constraint graphs allow a more detailed model of systems containing heterogeneous logic, further refinement is needed to model the ability to flexibly map computations to different resource types. Specifically, existing work on using multi-constraint partitioning for heterogeneous circuits is based on the assumption that all nodes can only be implemented in one way − a significantly limiting assumption for FPGAs and heterogeneous processors. We will introduce our new model that eliminates this limitation, multi-personality graphs, in Section 4.2. This new model will serve as the basis for one of the major research projects we will describe in this document, Multi-Personality Partitioning.

2.4. Multi-Gigabit Serial Data Transmission

Multi-gigabit serial transceivers (MGTs) are becoming a widely used communication method in systems requiring high data throughput. MGTs offer several advantages over traditional parallel buses, such as reduced pin counts, decreased power consumption, and avoidance of clock-skew problems [8]. MGTs have long been used at the network level in technologies such as Ethernet, and they are becoming increasingly prevalent in the design of high performance digital circuits. At the chip and circuit level, FPGA manufacturers are increasingly replacing parallel input/output (I/O) pins with MGTs [9], serial interfaces have been proposed for next-generation memory architectures [10], and serial links for on-chip

communication have been proposed in research [11]. The transition to the use of MGTs rather than parallel I/Os introduces important changes to the computational constraints associated with communication. Devices that use MGTs for communication typically have more available bandwidth, but using MGTs comes with the cost of higher latency than parallel I/Os due to the time it takes to serialize and de-serialize data. These changes will contribute to the motivation for our development of an entropy-based and bandwidth-focused circuit partitioning method in Chapter 5.

2.5. Information Content in Digital Signals

The study of the information content of signals and stored data − information theory − is a broad topic with applications in many fields. Within the context of this document, we will focus on providing a basic introduction to the way information is quantified and measured in digital systems.

2.5.1. Quantifying Information

The field of information theory was established with Claude Shannon's foundational work to develop a mathematical framework for describing and analyzing communication [12]. Shannon introduced the idea that the amount of information contained in a signal or data set could be uniformly quantified using a concept he referred to as 'information entropy.' Information entropy should not be confused with thermodynamic entropy. Although they are linked [13], and the laws of thermodynamic entropy have been used to define constraints on information entropy [14] [15] and vice versa [16] [17], they are not synonymous. Throughout this document, when we refer to 'entropy', it will refer to Shannon's concept of information entropy unless otherwise stated.

If we consider a signal that is capable of producing different values (also referred to as 'symbols') at a given time, Shannon's information entropy is based on the statistical likelihood of specific data values occurring. If we define *S* as the set of all possible symbols and *p(s)* as the probability that a given symbol, *s*, is produced by the signal, then the Shannon entropy can be defined as:

$$E = -\sum_{s \in S} p(s) \log_b(p(s)) \quad \text{where} \quad p(s)\log_b(p(s)) = 0 \quad \text{for} \quad p(s) = 0 \tag{5}$$

Shannon's definition of information entropy mirrors Boltzmann's equation for thermodynamic entropy in statistical mechanics. The base of the logarithm, *b*, may be chosen differently depending on the application. In our discussion of entropy in digital systems, we will use $b = 2$, which results in the entropy having units of 'bits'.

From its inception, there has been debate over whether entropy is truly a measure of the information content of a signal or whether it is better described as a measure of the uncertainty of predicting a signal's value [18]. From a practical standpoint in the design of communications systems, entropy is a useful metric for quantifying information. Entropy-based coding has been used to compress data in many application domains [19] [20] [21], demonstrating that signals with high entropy require more bits to transmit (and conversely that signals with low entropy can be more effectively compressed). Since our interest in entropy is related to communication efficiency, we treat entropy as a measurement of the quantity of information and defer the semantic argument to others.

2.5.2. Entropy in Digital Systems

In Equation (5) we gave the formula for computing the entropy of a signal with a set of possible symbols. Within the context of digital logic circuits, the signal is a wire or group of wires and the symbols are the possible values the wire(s) can transmit. For a single wire in a binary digital system, there are two

possible values: '0' and '1'. If $p(s_0)$ is the probability of the symbol '0' occurring, the entropy equation can be rewritten as:

$$E = -p(s_0)\log_2(p(s_0)) - p(s_1)\log_2(p(s_1)) \quad (6)$$

Since $p(s_0) + p(s_1) = 1$, Equation (6) can be further rewritten as:

$$E = -(1 - p(s_1))\log_2(1 - p(s_1)) - p(s_1)\log_2(p(s_1)) \quad (7)$$

Thus, we can consider the entropy of any given wire in the circuit to be a function of the probability that its value will be '1' at a given time (we can also derive an identical equation based on the probability

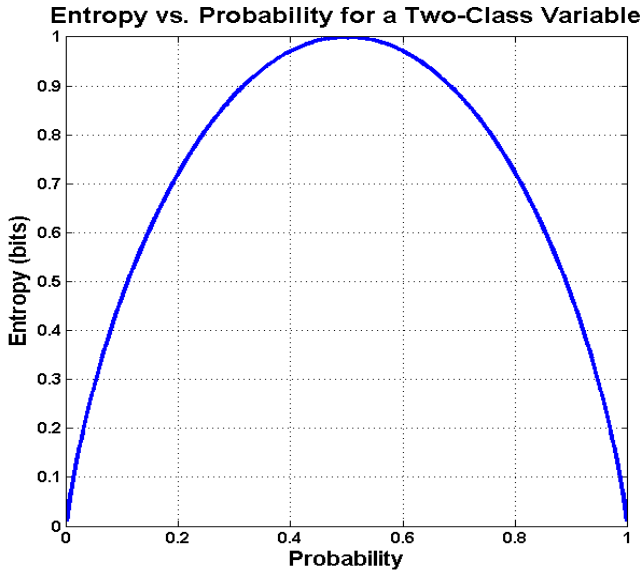

Figure 1. Entropy (in bits) versus probability of having a value of '1' for a binary variable [221].

the wire will be '0'). Equation (7) is represented graphically in Figure 1. From this diagram, we can observe two important properties of the entropy of a binary signal. First, the maximum entropy of the wire is 1 bit. We consider an individual wire to physically represent 1 bit in a digital system, but the actual amount of information that wire caries is at most 1 bit and may be less than that. Second, the maximum entropy occurs when there are equal probabilities that the wire will have a value of '0' or '1'. This agrees with our earlier description of entropy as quantifying the predictability of a signal's value. These observations also hold true when we consider logical groupings of multiple wires that may have

more than two possible values. The entropy (in bits) of a multi-wire signal is always less than or equal to the number of wires in the signal, and the maximum entropy (which is equal to the number of wires) occurs when there is an equal probability of all symbols occurring.

The idea of combining the entropies of multiple signals into a single value will become important in our later discussion of entropy-based partitioning. Formally, Shannon defined the 'joint entropy' to describe the combined entropy of multiple signals. To compute the joint entropy of wires (or wire groups) W_1, ..., W_n with symbol sets S_1, ..., S_n respectively:

$$E(W_1,...,W_n) = -\sum_{s_1 \in S_1} ... \sum_{s_n \in S_n} p(s_1,...,s_n) \log_2(p(s_1,...,s_n)) \quad \text{where} \quad p(s_1,...,s_n)\log_2(p(s_1,...,s_n)) = 0 \quad \text{for} \quad p(s_1,...,s_n) = 0 \tag{8}$$

In Equation (8), $p(s_1,...,s_n)$ is the joint probability that the symbols s_1, ..., s_n will occur together. If the wires' values are independent, the joint probability is the product of the individual symbol probabilities and the joint entropy is equal to the sum of the entropies of the individual wires. This represents the upper bound on the joint entropy since we cannot use any wire's value to predict another wire's value. Thus:

$$E(W_1,...,W_n) \le \sum_{i=1}^{n} E(W_i) \tag{9}$$

We will utilize this upper bound on the joint entropy of groups of wires when we explore the idea of computing and minimizing the maximum joint entropy of the wires that cross between circuit partition boundaries in Chapter 5.

2.5.3. Other Information Measurement Metrics

Thus far, we have used Shannon's definition for entropy. A key aspect of Shannon's definition of entropy is that it treats signals as random variables. In real digital systems, do not necessarily have uniform probability distributions over time [22]. If we assume that the signals are not independent

random variables, the Shannon entropy may overestimate their information content [23]. As a result, other entropy formulations have been proposed for quantifying the amount of information in binary sequences. Several alternate entropy metrics are based on computing a binary derivative of a signal's value changes over time [24] [25] and have led to new methods of computing entropies [26].

If we approach metrics for quantifying information based on their applications to compression, we can define the amount of information in a signal based on its Kolmogorov-Chaitlin complexity [27] [28], a measure of the minimal amount of information that would be required to reproduce a signal. Kolmogorov complexity has been proven to be incomputable [29]; this implies that there is no feasible metric we can compute that will ensure minimal compression. As such, we will continue to use the Shannon entropy as our information metric since it is easily computed, well understood, and commonly used as a metric for optimizing communications.

3. Mixed-Granularity Reconfigurable Architectures for High Energy Physics Processing

In this chapter, we describe our work to develop and evaluate an FPGA-based processing system for the high-luminosity calorimeter trigger system of the Compact Muon Solenoid experiment at the Large Hadron Collider. This project also helped us to gain insight into areas of the circuit design process that could derive significant benefit from the development of new algorithms for design automation and rapid prototyping. These insights will serve as the motivational foundation of the research described in Chapters 4 and 5.

3.1. Motivation

On-going trends in the fields of high energy particle accelerators and electronics have created new design challenges for their real-time processing systems. For example, increases in collision rates and event resolution have driven sensor data rates to 1 Petabyte per second at the state-of-the-art Large Hadron Collider (LHC) particle accelerator. To handle this large amount of data, scientists employ triggering systems consisting of hundreds of specialized processing units capable of handling huge amounts of data with µs-scale processing latencies. High throughput and low latency is particularly important in the Level-1 (L1) trigger system responsible for making the first triggering decisions. As shown in Figure 2, the output throughput of L1 systems has been scaling exponentially, increasing by 1-2 orders of magnitude with each new generation of accelerator. This has driven trigger design away from the use of general-purpose processors, and toward application-specific custom electronics. Meanwhile, the rising cost and complexity of constructing particle accelerators has had a major impact on their design process. The LHC has been designed to be upgraded to support higher luminosities and newer experimental objectives, and the supporting electronics must be capable of adapting to perform new algorithms. As the process of simply shutting down and restarting the experiment for maintenance

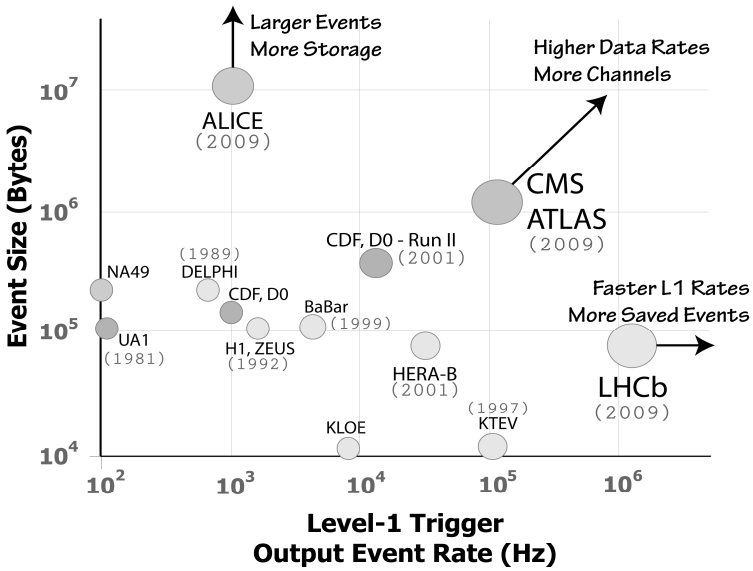

Figure 2. The required throughput of trigger systems has been increasing exponentially over time [23]. Note: Both axes are log-scale.

requires several months and generates enormous costs, it is essential that the processing architecture be flexible and robust to change over the lifetime of the accelerator.

Designing systems for a large-scale, collaborative project like the LHC presents its own challenges. Components of the hardware and firmware design are handled by many disparate scientific and academic groups. The design specifications and hardware platforms may change frequently during development. This has motivated a need for automated tools to aid in rapid prototyping and retargeting based on changes in system specifications.

Initially, the extremely high performance requirements of the LHC necessitated the use of high-speed ASICs [30]. With advancements in process technologies and the associated boost in logic density and performance, FPGAs have become an attractive platform to replace ASICs for real-time processing, due to their reprogrammable nature. However, reprogrammability alone does not guarantee that an FPGA-based system architecture will be able to easily support changes to in the experiments, as new circuits for new algorithms may have vastly different timing and area requirements.

3.2. Related Work

The inception of modern computer-based trigger systems can arguably be traced back to the 1970s at ADONE electron-positron collider at Frascati National Laboratory [31] and its ‘spiral reader’ system [32] for bubble physics. The spiral reader was perhaps the first on-line processing system based on software running on mainframe computers. A few years later, CERN’s Intersecting Storage Rings (ISR) was the first particle accelerator to achieve proton-proton collisions by colliding separate beams [33]. The development of colliding beam accelerators vastly increased collision rates; since then, data rates have been a dominating constraint in trigger design.

In the 1980s and 1990s, the increasing cost of constructing accelerators began motivating thoughts of how to increase their architectures’ flexibility and lifespan. The Axial Field Spectrometer Trigger introduced a heterogeneous, multi-level design, using both simple processing based on ‘coincidence counting’ and pre-computed RAM look-up tables (LUTs) [34] and a custom, 16-bit programmable DSP processor [35]. Both the contents of the RAM LUT and the software running on the DSP could be modified to change the physics algorithms after the system was installed. The CDF experiment at Fermilab extended heterogeneity with a 3-level trigger system that used ASIC-based L1 & L2 triggers and a software-based L3 running on a computer farm [36]. CDF was one of the first triggers built for modularity and flexibility; its L2 'Mercury' ASICs were designed to be physically swappable. This helped CDF stay in operation for 20 years. The Large Electron Positron (LEP) ring was built to study events caused by beam-gas interactions that were not well understood prior to the start of real experiments. LEP therefore needed to be able to alter trigger algorithms after the system was deployed [37]. It was also one of the earliest systems to incorporate limited use of FPGAs (alongside RAM LUTs) for this purpose [38].

By the 2000s, exponential increases in detector luminosity and data throughput had massively increased the scale of computing systems necessary to process the data being collected. The ATLAS and CMS experiments are characterized by event rates that dwarf other experiments, giving them each a data rate of over 1 Petabyte per second [39]. Both ATLAS and CMS feature a hardware-based L1 Trigger and

software-based High-Level Trigger. The L1 hardware is primarily based on Xilinx Virtex-II Pro and Virtex-4 FPGAs [40] [41], but high-speed ASICs were needed for operations like addition, sorting, and table look-up to meet throughput constraints. Despite the use of FPGAs, the large number of devices and complex system partitioning has made it difficult to modify trigger algorithms. This complexity can be seen in the design of the CMS calorimeter trigger board (one of 126), pictured in Figure 3.For further discussion of historical trends in the development of triggering systems, we provide broader survey of trigger architectures in Appendix A.

Our implementation work is based on algorithms originally developed at the University of Wisconsin for calorimeter-based triggering in the CMS experiment under high luminosity, high pile-up conditions [42] [43]. Compared to the previous algorithms, the high-luminosity algorithms offer greater positional resolution and better filtering of background noise. In part, the research we will describe in this chapter helped to inform changes to the initial algorithm proposals for more efficient hardware implementation. The Wisconsin trigger design represents one of two alternatives being evaluated for implementation in the HL-LHC, and is based on the concept of a spatially-partitioned, pipelined hardware design. The other design, based on work done by Imperial College [44] [45], avoids spatial multiplexing and instead time-multiplexes bunch crossings across multiple chips in order to meet throughput requirements. Both designs are expected to be used in parallel during the prototyping phase to study their relative performance and cost [46]. Although the results we present on the development of the Wisconsin prototype, both systems share the same basic physics goals, so insights derived the work presented in this chapter should be broadly applicable to both architectures.

3.3. Characterizing Calorimetry-Based L1 Triggering Algorithms

In order to evaluate the possible system architectures, it was first necessary to develop a more general characterization of the triggering algorithms to determine how they might be efficiently implemented in an FPGA. Based on consultation with the physicists developing algorithms for the Regional Calorimeter Trigger (RCT) of the LHC's Compact Muon Solenoid (CMS) experiment, as well as analysis of historical

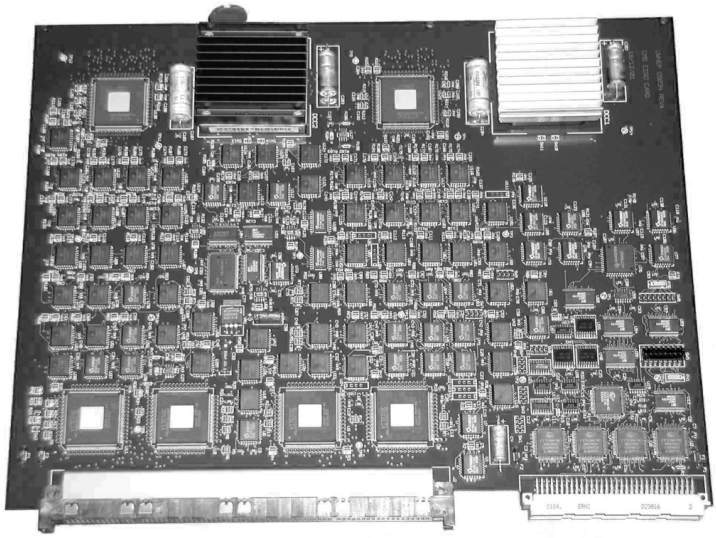

Figure 3. One of 126 boards used for processing calorimeter trigger algorithms in the original CMS trigger. The overall system is partitioned across over 700 FPGAs and ASICs and thousands of RAM-based LUTs.

trigger architectures for other particle accelerators, we have categorized common operations present in triggering systems. Later in this section, we will also provide pseudo-code descriptions of some of the major functions of the RCT.

3.3.1. Summary of Common Operations in Trigger Applications

Based on our survey of triggering systems (see Appendix A for a brief description of some historically significant trigger architectures included in our study) we identified the following key operations for triggering algorithms:

- *Energy Magnitude* − Compute the total energy over a given sensor region, sort regions by their energy, and check whether the energy exceeds a threshold. This typically involves simple arithmetic such as addition and comparison operations.
- *Energy Composition* − Analyze the different energy components in a region (e.g. electromagnetic energy, hadronic energy, transverse energy) and make a decision based on the distribution of each component. This may involve intermediate arithmetic operations, such as multiplication and division.

- *Background Energy Removal* − Compute the background energy (noise) in each region by summing energy over large areas, removing energy belonging to significant collision events, averaging, and adjusting based on sensor geometry and the distance of a given sensor region from the particle collision point in 3D space. Because sensors are cylindrically distributed, this may involve complex arithmetic operations, such as trigonometric functions.
- *Regional Maxima Formation* − Cluster collision data into small or large regions, arbitrating between adjacent regions in order to form regional energy maxima that are as large as possible. This typically only requires simple arithmetic, but requires complex routing of data.
- *Fine-grained Veto* − Provide the ability to reject a triggering decision based on the value of a single input tower if that input was used in making a positive triggering decision. Veto power is a valuable feature for practical implementation of triggering algorithms. Not only can it act as a notch filter to remove undesirable background noise at known energy levels, it is also used to mask out data produced by inputs suffering from permanent faults that cause them to provide faulty results. This is simple to implement, but may require design modification after the system has already been deployed and in service.

3.3.2. Pseudo-code for Algorithms in the Upgraded CMS Level-1 Calorimeter Trigger

In this section, we provide pseudo-code for selected algorithms under development as part of the high-luminosity upgrade for the CMS Level-1 Calorimeter Trigger system [47], based on the current versions of the algorithms being evaluated in the CMS L1 Trigger Emulator [48] (see Section 5.8 for further discussion of the CMS L1TE). These algorithms were initially designed by physicists at the University of Wisconsin − Madison [42] [49] and collaboratively refined for efficient implementation on hardware as part of the research described in this dissertation [50] [51]. Note that these algorithms are subject to change prior to the final design. Additionally, although algorithms are described procedurally in the emulator, they must be highly parallelized in the hardware implementation in order to meet the high

throughput requirements and partitioned for a distributed hardware system (See Chapters 4 and 5 for more details on our research into efficiently partitioning large-scale circuits).

Primary Inputs and Outputs

The input to the RCT system is a group of approximately 10,000 calorimeters (devices for measuring particle energy) deployed within the CMS detector, a large group of sensors that absorb sub-particles produced during particle annihilation when hadrons are collided in the LHC. The calorimeters are distributed in 5,000 discrete locations on the detector, with each location containing a calorimeter for measuring electromagnetic energy (ECAL) and one for measuring hadronic energy (HCAL). These locations are referred to as 'towers'.

The detector is capsule-shaped, and can be divided into a cylindrical "barrel" region, and two "end caps". We will focus on algorithms used in the barrel region, which contains 80% of the towers. Geometric locality is important in physics algorithms. In pseudo-code, the barrel cylinder can be mapped into a 2-dimensional array of towers, where one dimension represents transverse movement along the barrel (eta) and the other dimension represents rotation around the axis (phi). Thus, we can define the tower inputs as:

```
towers[1..eta_max][1..phi_max]
```

Each tower's data consists of 17 bits, composed of ECAL energy (8 bits), HCAL energy (8 bits), and an additional finegrain veto bit that experiment operators can set to indicate a calorimeter hardware fault in the tower. Additional error correction/detection bits are also provided, but are processed at the link level and do not factor into the functional descriptions of the algorithms. For some algorithms, towers are grouped into 2x2 'clusters', as energy from an individual particle may be distributed across cluster-sized regions of towers [42]. Since the detector is cylindrical, it wraps around in the phi dimension and connects to the end cap in the eta direction, and thus an individual cluster can be defined as a 2x2 array of towers:

```
cluster[i][j] = {towers[i][j], towers[i + 1][j],
                towers[i][j + 1 % eta_max], towers[i + 1][j + 1 % eta_max]}
```

Code Listing I. Pseudocode for the Tower Clustering operation in the RCT.

Input data sets (also known as a 'bunch crossing' or 'bx') arrive at a rate of up to 40 million bx/s resulting in an input throughput of approximately 160 billion towers per second in the barrel region.

The output consists of an array of 4,000 particle objects and 4,000 jet objects. Each particle object is composed of a particle type (2 bits, representing isolated electron/photon, isolated tau, non-isolated electron/photon, or non-isolated tau), particle transverse energy (11 bits), center-of-energy position (4 bits), barrel position (12 bits), and valid flag (1 bit). Each jet object is composed of a jet energy (15 bits), barrel position (12 bits), an isolation flag (1 bit) and a valid flag (1 bit). Objects are sorted based on energy and particle type in a separate system [52].

Tower Filtering

Although the barrel has enough sensors to detect thousands of particles, each bunch crossing produces around 20-25 particles on average [5]. This means that the majority of sensor input data consists only of energy noise produced by background radiation in the detector (we will revisit the optimization opportunities available by this property in Section 5.4). Towers are filtered based on comparing them to a programmable, position-dependent energy threshold value. This can be expressed as:

```
for each tower[i][j] in towers[1..eta_max][1..phi_max] {
    if (tower[i][j] > threshold(i, j)) {
          filtered_tower[i][j] = tower[i][j];
    } else {
          filtered_tower[i][j] = 0;
    }
}
```

Code Listing II. Pseudocode for the Tower Filtering operation in the RCT.

Channel Transverse Energy Sum

The channel transverse energy (E_T) is sum of the electromagnetic and hadronic energies in each tower, and represents to total energy localized to a barrel position.

```
for each filtered_tower[i][j] in filtered_towers[1..eta_max][1..phi_max] {
    channel_et[i][j] = filtered_tower[i][j].ecal +
                       filtered_tower[i][j].hcal;
}
```

Code Listing III. Pseudocode for the Channel E_T computation in the RCT.

Cluster Energy Sums

For each tower cluster, we compute a transverse energy sum, a cluster ECAL sum and a cluster HCAL sum. Recalling that each cluster is a 2x2 array of towers:

```
for each cluster[i][j] in clusters[1..eta_max - 1][1..phi_max] {
    cluster.ecal[i][j] = cluster[i][j][0][0].ecal +
                         cluster[i][j][0][1].ecal +
                         cluster[i][j][1][0].ecal +
                         cluster[i][j][1][1].ecal;
    cluster.hcal[i][j] = cluster[i][j][0][0].hcal +
                         cluster[i][j][0][1].hcal +
                         cluster[i][j][1][0].hcal +
                         cluster[i][j][1][1].hcal;
}
for each filtered_tower[i][j] in filtered_towers[1..eta_max][1..phi_max] {
    cluster.et[i][j] = filtered_tower[i][j] +
                       filtered_tower[i][j + 1 % phi_max] +
                       filtered_tower[i + 1][j] +
                       filtered_tower[i + 1][j + 1 % phi_max];
}
```

Code Listing IV. Pseudocode for the Cluster Energy Sum operations in the RCT.

Cluster Overlap Filtering

Each cluster overlaps some of its tower data with eight other clusters. Conversely, each individual tower is covered by four different clusters. To avoid having this energy data be counted more than once,

each tower must contribute its energy to only one cluster and contribute a zero energy value to the other clusters. The process of selecting which cluster to assign a tower's energy is known as 'overlap filtering'. The cluster overlap filtering algorithm is designed to create the largest global cluster energy maxima, then the next largest local maxima (after removing the global maxima), and so on. This can be expressed as:

```
// Compute Cluster ET values prior to Overlap Filtering
for (i = 2 to eta_max - 1) {
    for (j = 1 to phi_max) {
          jp1 = j + 1 % phi_max;
          prefilter_et[i][j] = channel_et[i][j] +
                               channel_et[i][jp1] +
                               channel_et[i+1][j] +
                               channel_et[i+1][jp1];
    }
}

// Generate 'Keep Masks' which defines which of a clusters 4 towers it gets
// to keep.
for (i = 2 to eta_max - 1) {
    for (j = 1 to phi_max) {
          ip1 = i + 1;
          jp1 = j + 1 % phi_max;
          im1 = i - 1;
          jm1 = j - 1 % phi_max;

          // Define the energies of the 8 neighboring clusters, treating
          // eta and phi as cardinal directions:
          // northwest, north, northeast, west, east, southwest, south, and
          // southeast.
          nw_et = prefilter_et[im1][jm1];
          n_et = prefilter_et[i][jm1];
          ne_et = prefilter_et[ip1][jm1];
          w_et = prefilter_et[im1][j];
          e_et = prefilter_et[ip1][j];
          sw_et = prefilter_et[im1][jp1];
          s_et = prefilter_et[i][jp1];
          se_et = prefilter_et[ip1][jp1];
```

```
            // Transverse energy for the current cluster
            my_et = prefilter_et[i][j];

            keep_tower_00[i][j] =
                  (w_et < my_et) && (nw_et < my_et) && (n_et > my_et);
            keep_tower_01[i][j] =
                  (n_et < my_et) && (ne_et < my_et) && (e_et > my_et);
            keep_tower_10[i][j] =
                  (w_et < my_et) && (sw_et < my_et) && (s_et > my_et);
            keep_tower_11[i][j] =
                  (e_et < my_et) && (se_et < my_et) && (s_et > my_et);
      }
}

// Apply filtering based on Keep Masks for each cluster. If the remaining
// energy is below the threshold value, filter out the cluster.
for (i = 2 to eta_max - 1) {
      for (j = 1 to phi_max) {
            whole[i][j] = keep_tower_00[i][j] && keep_tower_01[i][j] &&
                        keep_tower_10[i][j] && keep_tower_11[i][j];
            filtered_sum = 0;
            if (keep_tower_00[i][j]) {
                  filtered_sum += channel_et[i][j];
            }
            if (keep_tower_01[i][j]) {
                  filtered_sum += channel_et[i][j + 1 % phi_max];
            }
            if (keep_tower_10[i][j]) {
                  filtered_sum += channel_et[i + 1][j];
            }
            if (keep_tower_11[i][j] {
                  filtered_sum += channel_et[i + 1][j + 1 % phi_max];
            }
            threshold_pass = filtered_sum > filtered_sum_threshold(i, j);
            if (keep_tower_00[i][j] &&  threshold_pass) {
                  filtered_cluster[i][j][0][0] = channel_et[i][j];
            } else {
```

```
            filtered_cluster[i][j][0][0] = 0;
        }
        if (keep_tower_01[i][j] &&  threshold_pass) {
            filtered_cluster[i][j][0][1] =
                                channel_et[i][j + 1 % phi_max];
        } else {
            filtered_cluster[i][j][0][1] = 0;
        }
        if (keep_tower_10[i][j] &&  threshold_pass) {
            filtered_cluster[i][j][1][0] = channel_et[i + 1][j];
        } else {
            filtered_cluster[i][j][1][0] = 0;
        }
        if (keep_tower_11[i][j] &&  threshold_pass) {
            filtered_cluster[i][j][1][1] =
                                channel_et[i + 1][j + 1 % phi_max];
        } else {
            filtered_cluster[i][j][1][1] = 0;
        }
    }
}
```

Code Listing V. Pseudocode for the Cluster Overlap Filter computation in the RCT.

Despite being a global algorithm, it we are able to efficiently parallelized it in hardware, as each cluster only needs local information to make its filtering decisions [53] [46].

Electron / Photon Identification

Based on the amount of electromagnetic and hadronic energy in a cluster, it is possible to characterize whether the particle that produced the deposit was electromagnetic (such as an electron or photon) or non-electromagnetic (such a tau particle). The original Electron / Photon ID algorithm computes a ratio between the electromagnetic and hadronic energy in a cluster and compares it to a threshold value.

```
for each cluster[i][j] in clusters[1..eta_max][1..phi_max] {
    if (cluster[i][j].ecal / cluster[i][j].et > egamma_threshold(i, j)) {
          cluster[i][j].is_egamma = true;
    } else {
          cluster[i][j].is_egamma = false;
    }
}
```

Code Listing VI. Pseudocode for the Electron/Photon Identification computation in the RCT.

When designing the implementation of the Electron / Photon ID function, there was desire from physicists to keep hardware flexibility to alter or replace the algorithm after the hardware had been designed and deployed. For similar reasons, the Electron ID cards in the original low luminosity RCT design used energy-based look-up tables to implement the algorithm [5]. We studied several different ways of implementing the algorithm in hardware with the assumption that it would be subject to change. We discuss this work further throughout Section 3.4.

Particle Isolation

The particle isolation algorithm determines whether particle energy deposits are isolated on the detector. We classify an energy deposit in a given cluster as isolated if the amount of energy located in its 'annulus region' − a 6x6 region of towers with the cluster at its center − is relatively small compared to the energy in the central cluster. This determination is made by first comparing the energy of the clusters in the annulus to a programmable threshold value and counting the number of clusters that pass the threshold, then using the energy in the central cluster as the address to a look-up table containing a count threshold for that energy level, and finally comparing the annulus count to this count threshold. This operation is computed separately for ECAL and HCAL energy, using different thresholds. The look-up tables are programmable to allow them to be experimentally tuned. To reduce the look-up table size, we also perform a quantization operation prior to look-up.

```
for each cluster[i][j] in clusters[3..eta_max - 3][1..phi_max] {
    annulus_ecal_count = 0;
    annulus_hcal_count = 0;
    for (annulus_i = i -2 to i + 3) {
          for (annulus_j = j - 2 % phi_max to j + 3 % phi_max) {
                if (cluster[annulus_i][annulus_j].ecal >
                    annulus_ecal_threshold(i, j)) {
                      annulus_ecal_count++;
                }
                if (cluster[annulus_i][annulus_j].hcal >
                    annulus_hcal_threshold(i, j)) {
                      annulus_hcal_count++;
                }
          }
    }
    quantized_ecal = cluster[i][j].ecal / ecal_quantization_factor(i, j);
    quantized_hcal = cluster[i][j].hcal / hcal_quantization_factor(i, j);
    ecal_count_threshold = lookup_ecal_count_threshold(quantized_ecal);
    hcal_count_threshold = lookup_hcal_count_threshold(quantized_hcal);
    cluster[i][j].is_ecal_isolated =
                      annulus_ecal_cout > ecal_count_threshold;
    cluster[i][j].is_hcal_isolated =
                      annulus_ecal_cout > ecal_count_threshold;
}
```

Code Listing VII. Pseudocode for Core-Relative Particle Isolation.

Particle Identification

Particle Identification sorts clusters into groups of isolated electromagnetic particles, non-isolated electromagnetic particles, isolated taus, or non-isolated taus. Electromagnetic/tau classification and isolation have already been computed by the Electron/Photon ID and Isolation algorithms respectively, but the Particle ID algorithm also considers the existence of tower veto bits in the cluster, which can disqualify it from being sorted into any particle group.

```
for each cluster[i][j] in clusters[1..eta_max][1..phi_max] {
    cluster[i][j].cluster_veto = cluster[i][j][0][0].tower_veto ||
                                 cluster[i][j][0][1].tower_veto ||
                                 cluster[i][j][1][0].tower_veto ||
                                 cluster[i][j][1][1].tower_veto;
    if (cluster[i][j].is_egamma) {
          if (cluster[i][j].is_ecal_isolated) {
                isolated_egamma_objects.add(cluster[i][j]);
          } else {
                non_isolated_egamma_objects.add(cluster[i][j]);
          }
    } else {
          if (cluster[i][j].is_hcal_isolated) {
                isolated_tau_objects.add(cluster[i][j]);
          } else {
                non_isolated_tau_objects.add(cluster[i][j]);
          }
    }
}
```

Code Listing VIII. Pseudocode for Particle Identification in the RCT.

Cluster Energy Weighting

An individual particle may have its energy absorbed by multiple adjacent towers − hence why tower clusters are used in algorithms that construct particle objects. However from a physics standpoint it is worthwhile to compute a higher positional resolution for the particle than a 2x2 tower cluster. To obtain sub-tower positional resolution, we compute a 'center-of-energy' value that describes how the energy is distributed within the cluster. This center-of-energy is based on the ratio of the amount of transverse energy distributed in the eta direction and phi direction within the cluster's sub-towers. Here a zero value for the position indicates that the energy is equally distributed in a given direction and a value of one (or negative one) indicates the energy is entirely distributed in one half of the cluster.

```
for each cluster[i][j] in clusters[1..eta_max][1..phi_max] {
    eta_m_towers_et = cluster[i][j][0][0].channel_et +
                      cluster[i][j][0][1].channel_et;
    eta_p_towers_et = cluster[i][j][1][0].channel_et +
                      cluster[i][j][1][1].channel_et;
    phi_m_towers_et = cluster[i][j][0][0].channel_et +
                      cluster[i][j][1][0].channel_et;
    phi_p_towers_et = cluster[i][j][0][1].channel_et +
                      cluster[i][j][1][1].channel_et;
    cluster[i][j].eta_pos = (eta_p_towers_et - eta_m_towers_et) /
                            (eta_p_towers_et + eta_m_towers_et);
    cluster[i][j].phi_pos = (phi_p_towers_et - phi_m_towers_et) /
                            (phi_p_towers_et + phi_m_towers_et);
}
```

Code Listing IX. Pseudocode for the Cluster Energy Weighting computation in the RCT.

Jet Reconstruction

Sometimes the individual particles we construct in the previous algorithms are parts of a 'jet' − a group of multiple particles, distributed over adjacent clusters, representing particles that may have been generated from a single particle decay event.

Jet reconstruction is an algorithm to assemble jets, which are 12x12 towers in size, by assigning tower energies to individual jets in order to create a global jet energy maximum followed by local jet energy maxima. This is accomplished by summing the filtered cluster energies to create energy regions. Jets are then created by grouping them as 3x3 region clusters and performing overlap filtering on the borders of the region to determine which regions are assigned to each jet when jets overlap. The region-forming portion of the algorithm can be described as:

```
for each filtered_cluster[i][j] in
                    filtered_clusters[2..eta_max-2][1..phi_max] {
   region[i][j].et = 0;
   for (c_i = i -1 to i + 2) {
         for (c_j = j - 1 % phi_max to j + 2 % phi_max) {
               region[i][j].et += filtered_cluster[c_i][c_j].et;
         }
   }
}
```

Code Listing X. Pseudocode for Region-Building phase of Region-Based Jet Reconstruction algorithm. The regions are then subject to Overlap Filtering (See Code Listing V).

The remaining filtering process is very similar to that used in the Cluster Overlap Filtering algorithm, with the primary difference being that it uses 4x4 energy regions rather than clusters. For brevity, we refer the reader to that section and will not replicate that pseudo-code here.

Pileup Level Computation

This algorithm computes the level of pileup in calorimeter tower readings. 'Pileup' refers to the high background noise created in the detector when performing collisions under high luminosity, and is a crucial aspect of the new high-luminosity experiments in the proposed upgrades to the LHC [54]. It is important to be able to filter out this background noise in order for the other algorithms to perform efficiently, and is done by computing a pileup level over a tower region and then subtracting the pileup energy from all towers in that region [55].

Our pileup computation algorithm re-uses the 4x4 energy regions from the jet expansion. First, the total energy of each region is compared to a programmable threshold value. Regions that are above the threshold are assumed to contain energy deposits from particles. The algorithm uses only those regions which do not pass the threshold, as they are assumed to contain only background noise. The total energy of the background noise regions is accumulated, and the total is divided by the number of regions to obtain the average background energy.

```
background_region_count = 0;
background_region_total_energy = 0;
for each region[i][j] in regions[2..eta_max-2][1..phi_max] {
    if (region[i][j].et < region_threshold(i, j)) {
          background_region_count++;
          background_region_energy += region[i][j].et;
    }
}
if (background_region_count == 0) {
    pileup_level = 0;
} else {
    pileup_level = background_region_energy / background_region_count;
}
```

Code Listing XI. Pseudocode for the computing the background pileup energy for high-luminosity CMS trigger algorithms.

Jet Isolation

Jet isolation is an algorithm that determines whether a jet's core (the 4x4 tower region at its center) is considered isolated within its 12x12 tower annulus. This is determined by removing the pileup energy from the jet, computing the ratio between the energy in the annulus and the core, and then comparing that ratio to a programmable threshold value.

```
for each region[i][j] in regions[2..eta_max - 1][1..phi_max] {
    annulus_energy = 0;
    for (r_i = i  - 1 to i + 1) {
          for (r_j = j - 1 % phi_max to j + 1 % phi_max) {
                if (r_i != i && r_j != j) {
                      annulus_energy += (region[r_i][r_j] - pileup_level);
                }
          }
    }
    core_ratio = (region[i][j] - pileup_level) / annulus_energy;
    if (core_ratio > jet_isolation_threshold(i, j)) {
          region[i][j].jet_isolated = true;
    }
}
```

Code Listing XII. Pseudocode for the computing the Core-Relative Jet Isolation.

3.4. Proposed Architectures

The operations and algorithms we developed and described in Section 3.3 will inevitably need to be modified and adjusted over the lifetime of the system. As experimental goals change, different physical phenomena are studied, and the capabilities of the accelerator and sensors change, it may be necessary to adjust thresholds, change region sizes, search for different energy compositions, or add new veto points. However, significant hardware redesigns can be extremely costly in HEP, as accelerators require very long and expensive warm-up and cool-down periods. To this end, trigger designs typically seek to incorporate as much programmability as possible to allow changes to experimental parameters without replacing hardware. Throughout the mid-20th century, this motivated the use of general-purpose processors and later off-the-shelf digital signal processor in L1 triggers. As the throughput and latency demands on L1 systems increased, they transitioned to the using a mixture of ASICs for fixed operations and using large external memories as look-up tables (LUTs) to handle programmable operations. The inputs for a LUT-based calculation are used as the address for the memory, and the data at that address is the pre-computed "result" based on those input values. Although conceptually similar to the small LUTs used as a basic logic element in an FPGA, in this context they refer to much larger structures holding and indexed by multi-bit values.

We examined several architectural approaches for implementing the HEP algorithms while maintaining flexibility to alter the algorithm parameters. These approaches can be classified into three categories.

Coarse-Grained Look-Up Memories (BRAM Table and LUT Tree)

Some older trigger architectures have relied on performing computations using LUTs in discrete RAM chips, using input data as an address to access pre-computed results stored in the memory. This allows the memories to be programmed with arbitrary logic functions, limited only by the number of inputs. However, discrete RAM chips are expensive, complicate logic board designs, and significantly limit the amount of data that can be processed on each FPGA due to pin limitations. The latency imposed by

accessing a memory controller and external chip are also costly when considering the required response time of trigger processors. FPGAs provide some options for on-chip data storage in both block RAM and configurable logic blocks. A **BRAM Table** implements the coarse LUT using block RAMs (BRAMs), high density programmable memory blocks available on FPGAs. Because the contents of these memories can be (re)loaded at run-time, they allow alterations to the algorithm with no need to re-synthesize or reprogram the FPGA. FPGAs also contain fine-grained SRAM-based LUTs and multiplexers in their configurable logic blocks (CLBs). These CLBs can be assembled into a large, coarse LUT. We form a **LUT Tree** by organizing CLBs into a balanced binary tree, where decision data is stored in leaf CLBs and interior CLBs are used to implement addressing logic. Changing the data stored in a CLB's SRAM requires the FPGA to be reconfigured, which may necessitate a service interruption. It will not, however, change the number of LUTs required or how they are connected, which avoids changes to the rest of the circuitry implemented within the FPGA.

3.4.1. Direct Arithmetic Logic Implementations (LUT-Based, DSP-Based, and Hybrid LUT-DSP)

The coarse-grained LUT architectures common in trigger electronics may be very area inefficient when implementing logic functions with large numbers of inputs. Many common HEP operations, such as simple arithmetic, comparison, and sorting [52] can be efficiently mapped to general-purpose CLBs and arithmetic-optimized DSP blocks by expressing them as arithmetic functions instead of a lookup. Because some operations can be mapped to multiple resource types, many implementation choices are possible. It may be necessary to conserve certain resources for other logic on the same chip. For instance, the FIFOs used by high-speed serial communication links require BRAM resources, reducing the BRAM resources available for computation [53]. Unlike the coarse-grained LUT solutions, changing arithmetic functions will require re-synthesis of the circuit, and the sizes of the pre- and post-modification circuits may not match if the function differs sufficiently. We implemented the Electron ID module (EPIM) [43] of the

HL-LHC's regional calorimeter trigger using **LUT-based**, **DSP-based**, and **Hybrid LUT-DSP** designs to examine their relative performance and costs in each resource.

3.4.2. Compressed Coarse-Grained Look-up Operations (Compressed LUT Tree and Multi-Level Compressed BRAM Table)

Alternatively, we can exploit the fine-grained nature of the CLBs and BRAM blocks to reduce the cost of coarse-grained look-up tables by applying offline compression of the table's data. For the **Compressed LUT Tree**, we compress the original tree by recursively collapsing nodes that have leaves with the same electron decision value. For the BRAM table, we designed a **Multi-Level Compressed BRAM Table**, which uses a two-stage look-up process. In the first stage, a subset of the inputs is used to perform look-up operations in smaller BRAM-based tables. These tables implement a compressive hash function. The output of the compressing tables is multiplexed and used as the input to the second-stage table. To avoid complex questions about the suitability of lossy compression and its implications on the performance of the physics algorithms, our later experiments will focus only on lossless compression. However, a well-designed lossy function might provide the opportunity to further reduce implementation costs. An example of this concept is highlighted in Figure 4, which shows single-level and multi-level BRAM tables for implementing the Electron/Photon ID module (EPIM). The actual size and arrangement of table inputs may be subject to change; for instance the original HCAL sensors were found to be less precise than the ECAL sensors, so fewer HCAL bits may be needed to achieve the same physics performance. Although the multi-level design as shown requires three memory structures rather than one, the total storage required is $2^{10} + 2^{10} + 2^{14} = 18$ kilobits, as compared to $2^{20} = 1$ megabit, representing a reduction of 98%. Because BRAMs are used to implement these tables, the tables can be modified at run-time without reprogramming the FPGA. However the size and organization of the smaller 2-level tables must be selected to ensure that they are capable of implementing the desired algorithms. We will examine issues of resource and performance variability based on changes to the algorithm further in Section 3.6.

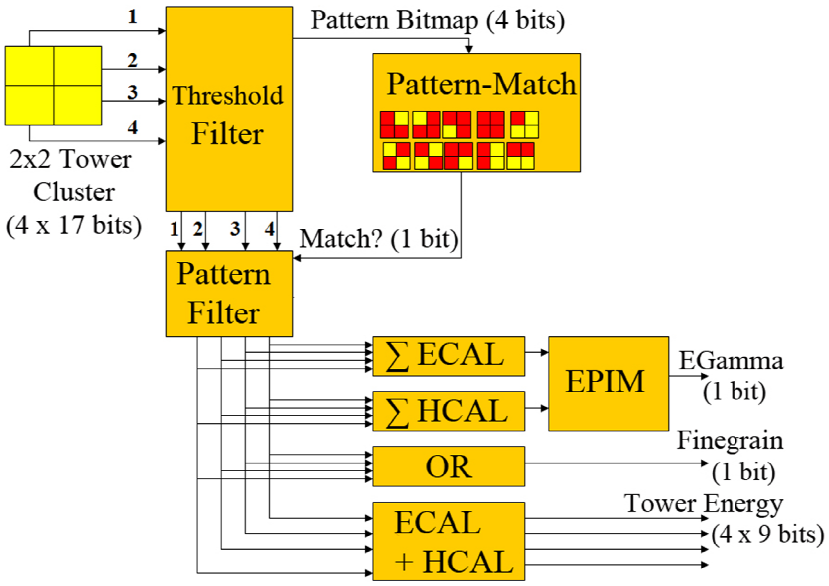

Figure 5. A section of the datapath for performing electromagnetic/hadronic particle identification in the CMS Regional Calorimeter Trigger.

3.5. Base Implementation Costs

We evaluated the performance and resource cost of the designs described in Section 3.4 by implementing the datapath for the CMS Regional Calorimeter Trigger's Electron/Photon ID module (EPIM) − an algorithm for classifying energy deposits as electromagnetic or hadronic based on readings from the electromagnetic (ECAL) and hadronic (HCAL) calorimeters. A block diagram of the datapath is shown in Figure 5. For brevity, we will omit detailed description of the individual blocks, but more can be found in the related publications [50] [43] [53]. The datapath contains many tasks representative of HEP algorithms, including energy-magnitude filtering (threshold filter, ECAL, HCAL), energy composition analysis, and fine-grained decision vetoing (OR).

Table I shows the performance and resource costs for implementing a single instance of the datapath, based on synthesis, placement, and routing on a Xilinx Virtex-5 TX240T FPGA using the Xilinx ISE tool. The full system must duplicate this data path for thousands of different regions in the experiment, thus the resource costs may play a role in determining how many FPGAs are necessary to fit the triggering

algorithms. However the Electron/Photon algorithm must coexist with other algorithms on the same FPGA, so these are not the sole determinants.

Table I. Max frequency and resource utilization (in percent of the TX240T's resources) for implementations of the CMS experiment's Electron/Photon ID module using several different architectures. Utilization values are in percent, and represent the cost of a single instance of the module.

Category	Architecture	Freq. (MHz)	CLB Utilization	BRAM Utilization	DSP Utilization
Direct	LUT-Based	440	**0.08**	---	---
	DSP-Based	370	0.02	---	**4.20**
	Hybrid LUT-DSP	320	0.01	---	**1.00**
Full Look-Up	BRAM Table	390	0.01	**9.80**	---
	LUT Tree	90	**14.60**	---	---
Compressed Look-Up	Compressed LUT Tree	270	**0.12**	---	---
	Multi-Level BRAM	450	0.02	**0.62**	---

It may be surprising that compression is so effective on the LUT Tree implementation, given that tree compression is relatively unsophisticated. The reason such compression is able to reduce the logic cost so drastically is that a large amount of the LUT Tree's logic cost is actually due to routing; FPGAs use programmable routing that also consumes CLBs. Therefore reducing the size of the tree not only reduces the number of storage elements, but also the routing overhead. However it should be noted that part of its success is due to the relative simplicity of the base physics algorithms used when computing these resource costs, as we will show in Section 3.6.

When interpreting the resource utilization numbers, the timing characteristics of the logic must also be considered. The LHC can supply new data sets at a rate of up to 40 MHz. Thus, it is possible for logic that operates at a higher frequency to be time multiplexed to process multiple data points while still meeting the throughput requirements. Time multiplexing trades latency for area. Since the trigger system has tight end-to-end latency constraints, time multiplexing is only feasible if the latency constraints can still be met. We demonstrate the area-reduction effect of time multiplexing in Table II.

Table II. Resource utilization for Electron/Photon ID architectures when maximally multiplexed.

Category	Architecture	Multiplexing Factor	CLB Utilization per Particle	BRAM Utilization per Particle	DSP Utilization per Particle
Direct	LUT-Based	11	**0.007**	---	---
	DSP-Based	9	0.002	---	**0.467**
	Hybrid LUT-DSP	8	0.001	---	**0.125**
Full Look-Up	BRAM Table	9	0.001	**1.089**	---
	LUT Tree	2	**7.300**	---	---
Compressed Look-Up	Compressed LUT Tree	6	**0.020**	---	---
	Multi-Level BRAM	11	0.002	**0.056**	---

3.6. Cost Stability Analysis

Because the ability to alter the HEP algorithms without redesigning the system is critical, it is important to compare implementation costs for multiple algorithm variations. For the full, uncompressed look-up table implementations (BRAM Table, LUT Tree) the resource requirements and timing characteristics do not vary with changes to the algorithm, provided the number of inputs and outputs do not change. For the other designs, these properties may change based on both the complexity of implementing the logic and the routing costs. Additional delay and resource utilization margins can be incorporated into the design to allow for future changes in the algorithms. The amount to reserve is important, as underestimating future needs may limit future algorithms, but overestimating will result in a larger, more complex, and less efficient system than necessary.

To evaluate the variation in timing and resource costs, we performed a Monte Carlo analysis of the LUT-Based Direct, Compressed LUT Tree, and Multi-Level BRAM designs for the EPIM, allowing up to 20 energy composition regions and up to 500 energy veto values. For LUT-Based Direct, we generated HDL files and for Compressed LUT Tree and Multi-Level BRAM we generated memory images for each algorithm variant. For the LUT Tree, compression was handled by the Xilinx Vivado ROM compiler, and for Multi-Level BRAM we compressed the image using Lempel-Ziv-Welch (LZW) compression with an 8-bit block size.

Figure 6 shows the latency histogram for the direct implementations. The ratio of worst-to-best latency is 1.6. The distribution is multi-modal, reflecting an important aspect of FPGA-based architecture. Modern FPGAs are composed of clustered logic cells connected with hierarchal routing, so as logic costs grow, they incur non-linear routing penalties when logic can no longer fit within a cluster. Figure 7 shows that for this algorithm, the area variation depends almost entirely on the number of vetoes. Designs that included up to 500 veto values required ~30X the logic resources as designs with no vetoes. Figure 8 shows a histogram of the normalized number of CLBs required to implement the Static LUT Tree design after tree compression. Within our Monte Carlo parameters, area requirements were more stable than for

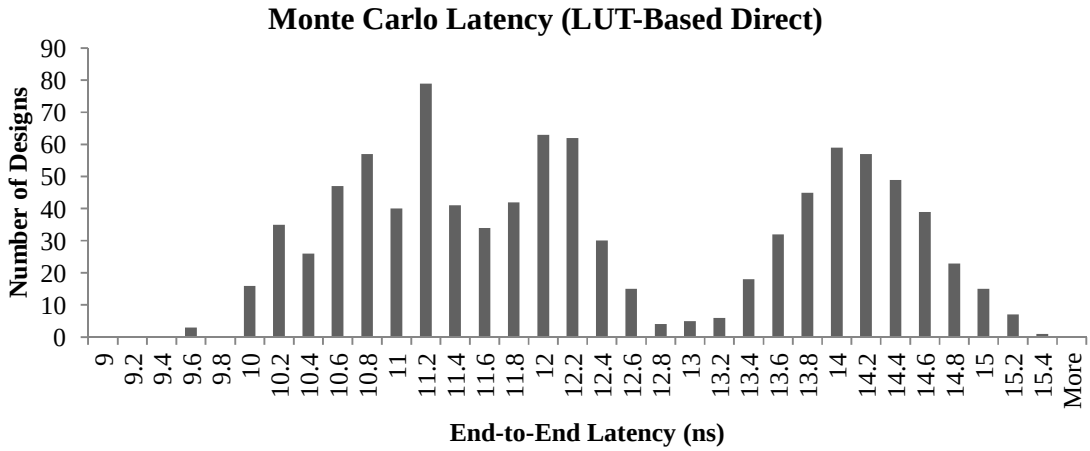

Figure 6. Histogram of end-to-end latency requirements for randomly parameterized instances of the LUT-Based Direct design for the Electron/Photon ID (EPIM) algorithm.

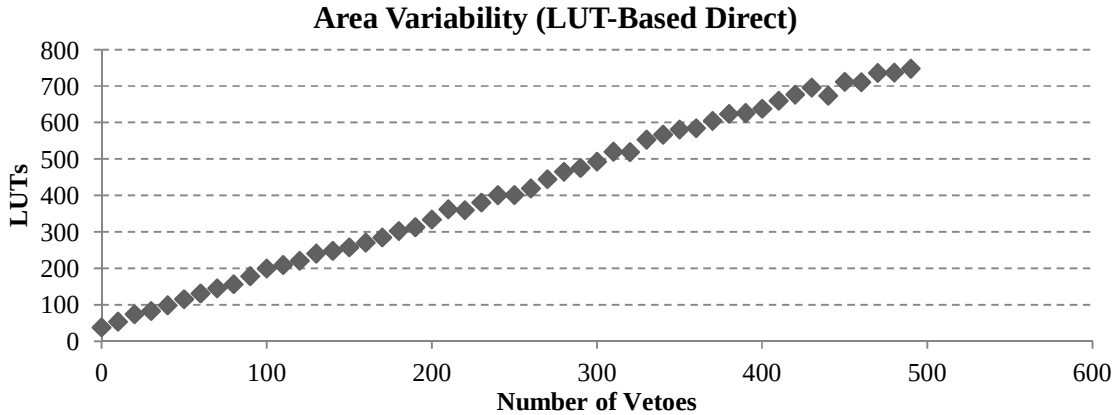

Figure 7. Total LUT requirements versus number of energy vetoes based on Monte Carlo simulation of EPIM implementation costs.

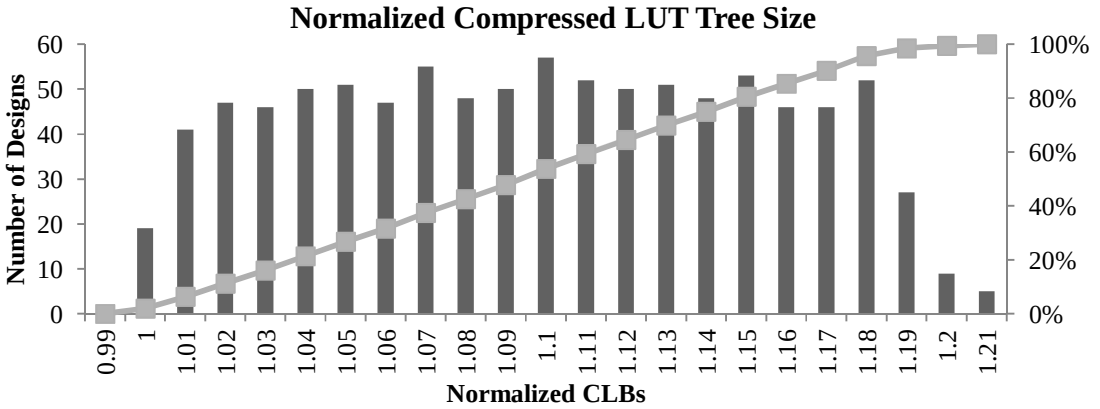

Figure 8. Histogram of the normalized size (in CLBs) of the Monte Carlo generated LUT trees for the EPIM after tree compression. Values are normalized to the base implementation without vetoes. The line indicates the cumulative percentage of Monte Carlo samples that could fit in that number of CLBs.

direct implementations, with a maximum increase of 21% over the base implementation

For the multi-level BRAM design, area and latency do not vary, provided the algorithm can be mapped into the pre-chosen memory sizes after compression. Figure 9 shows a histogram of the memory size needed to fit the Monte Carlo-generated EPIM algorithms an LZW-based compression is implemented in the BRAMs. The key result demonstrated here is that the variation in memory size is only 4% from the best to worst case. To examine a more extreme case, we increased the number of vetoes to

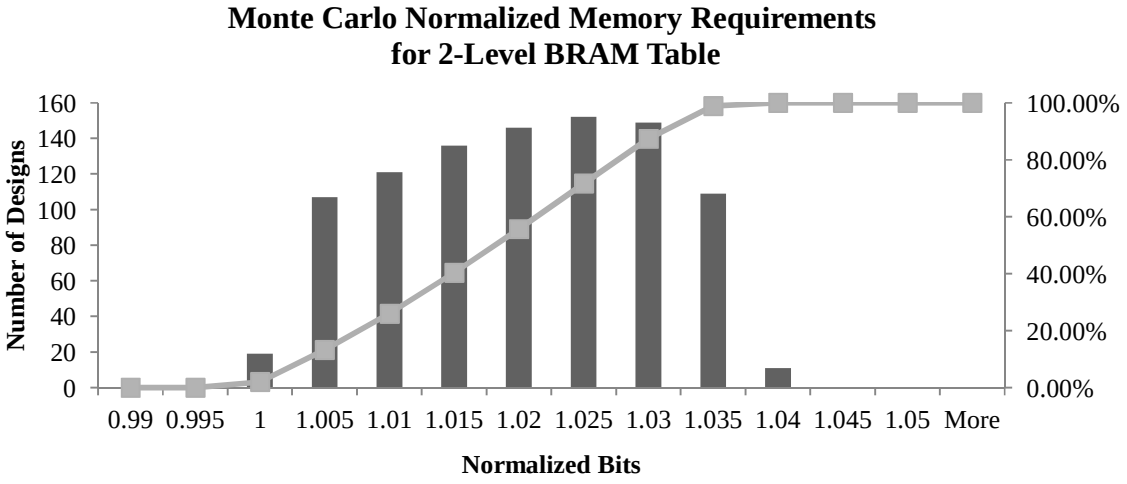

Figure 9. Histogram of the size of the Monte Carlo generated EPIMs converted to memory images for mapping to a look-up table and compressed with the LZW algorithm. Values are normalized to the base implementation without vetoes.

50,000 − a 100X increase over the Monte Carlo limits − and found that this required a 4.5X increase in memory size. It should be noted that other compression algorithms could be used; 8-bit LZW was chosen for ease of implementation in hardware, but we found that Huffman coding could reduce the memory size by up to 40% for designs with small numbers of vetoes.

3.7. Analysis of Implementations

By incorporating the area and frequency variability results from the Monte Carlo experiment into our model for the per-particle resource costs of the multiplexed EPIMs, we challenge some of the initial conclusions drawn from the costs of the base implementations. The resource ranges for the smallest to largest instances are shown in Table III. The gulf between direct implementations and full look-up tables is significantly narrowed, and the compressed look-up tables have the lowest costs when implementing the more complex variations of the algorithms. The Compressed LUT Tree and Multi-Level BRAM are competitive in size; if resource cost is the deciding factor, selection of one over the other will depend on whether CLBs or BRAMs are in greater supply after considering all of the other logic that must fit on the chip. All things being equal, we believe the multi-level BRAM design to be superior, as it offers other advantages − like run-time reprogrammability and consistent latency independent of routing − that make it more attractive for physics applications. If resource costs are not an issue − for instance if the number of particles that can be processed per FPGA is bottlenecked by some other aspect of the system, the full BRAM table may be preferred as its single level allows for lower end-to-end latency.

Table III. Resource utilization ranges for Electron/Photon ID architectures accounting for frequency and resource ranges from Monte Carlo simulation.

Category	Architecture	Multiplexing Range	CLB Utilization Range per Particle	BRAM Utilization Range per Particle	DSP Utilization Range per Particle
Direct	LUT-Based	7 - 11	**0.007 - 0.317**	---	---
	DSP-Based	7 - 9	0.002 - 0.252	---	**0.467**
	Hybrid LUT-DSP	7 - 8	**0.001 - 0.223**	---	**0.125**
Full Look-Up	BRAM Table	9	0.001	**1.089**	---
	LUT Tree	2	**7.300**	---	---
Compressed Look-Up	Compressed LUT Tree	2 - 6	**0.020 - 0.072**	---	---
	Multi-Level BRAM	11	0.002	**0.056**	---

3.8. Contributions

In this work, we collaborated with scientists at the European Center for Nuclear Research (CERN) to develop, analyze, and characterize the properties of next-generation high-energy physics triggering algorithms and identify the key challenges they present for the design of computing systems that govern data acquisition. We proposed several architectures for implementing physics algorithms on FPGAs designed to leverage different aspects of an FPGA's heterogeneous resource pools. We used Monte Carlo simulation to perform a cost-variability analysis to evaluate the resource cost and delay stability of different implementations as algorithm parameters change. The key insight of our analysis is that look-up based approaches, even when compressed, exhibited much greater stability than direct arithmetic implementations, and that this greater stability may lead to lower overall implementation costs when designers need to plan for the worst-case requirements. We showed that compression mechanisms can be effectively applied to algorithms like Electron/Photon ID that traditionally rely on memory-based look-up-tables. Compression can reduce resource costs by over 90% with low cost variation. This work served as a component of the design exploration and planning process for the Phase I and Phase II high-luminosity upgrades [2] of the Compact Muon Solenoid experiment at CERN's Large Hadron Collider.

4. Multi-Personality Graph Partitioning for Heterogeneous Resources

In this chapter, we will discuss the second major project, our research into algorithms for efficient automated partitioning of devices with flexible heterogeneous logic resources.

4.1. Motivation

In our discussion of the implementation options for the high energy physics architecture in Chapter 3, we discussed the fact that when using a heterogeneous FPGA-based fabric, we sometimes have to choose between different implementations that require different mixtures and quantities of the FPGA's heterogeneous resources. In that discussion, we mentioned that the designer might choose to use a certain implementation based on the amount of resources available on the device. When dealing with a large application that is partitioned over many devices this problem becomes more difficult, as it may be necessary to select different implementations on a per-device basis. This is further complicated if we later need to change one of these modules (for example to alter its function or meet performance constraints). This may change the available resources and make our previous implementation and partitioning choices inefficient. In order to avoid such problems, it is desirable to have an automated way to efficiently select implementations while partitioning an algorithm across heterogeneous devices. However, prior research on partitioning algorithms does not have a way of accommodating the concept of making implementation choices based on the available resources. To achieve the goal of simultaneously making intelligent partitioning and implementation choices, a new partitioning model and problem formulation are required.

We will refer to any logic that can be implemented in multiple ways as "multi-personality" logic. In Chapter 3, we demonstrated that multiple personalities may exist based on different designs created to implement the same task. Those implementation choices were based on coarse-grained designs in which the resources to use in the implementation were explicitly specified by the designer. On FPGAs, the

concept of multiple personalities is even more fundamental and pervasive; because CLBs and BRAMs can implement arbitrary logic functions of a given input size (including the functions of DSP blocks), it is possible to map functions to different resources at a relatively primitive level. When the designer does not explicitly specify resource types for the implementation, they are chosen automatically during the synthesis and technology mapping phases of the CAD process. Selecting the best implementation for each instance of multi-personality logic is an important decision with many high-level consequences, as it will impact whether the design can fit within a given FPGA and how much slack is available in each of the FPGA's resources for additional necessary features such as communication logic or for future modification to the algorithms. A given personality may also allow the design to be implemented using fewer FPGAs and reduce the amount of inter-FPGA communication, reducing the complexity of the system. When creating a design for a single device, it may be sufficient to allow the designer to manually specify the implementations of certain modules and rely on the automated technology mapping algorithms to choose the implementations of the remaining modules. However, for designs that must be partitioned across tens or hundreds of FPGAs we are faced with a dilemma: partitioning the design between chips requires knowledge about the resource costs of the logic; otherwise the partition may not fit on the FPGA. At the same time, resource costs for multi-personality logic cannot be determined until an implementation is selected, but selecting the best implementation requires knowledge of the partition in which the logic will be placed.

The multi-personality partitioning problem is not unique to specialized scientific applications. Large FPGA-based arrays remain a valuable platform for rapid prototyping and ASIC emulation, and vendors of commercial CAD tools have cited multi-resource balancing as one of the key open problems of performing design partitioning on FPGAs [56] [57]. Moreover, with the adoption of heterogeneous design frameworks like OpenCL that can allow a single design to be targeted at heterogeneous devices including CPUs, GPUs, DSPs, and FPGAs [58] [59], multi-personality partitioning is becoming a more general problem that applies to partitioning of both hardware and software systems. Achieving high quality partitioning solutions for heterogeneous systems requires new partitioning algorithms that are capable to

integrating knowledge about possible implementations into the partitioning process. In this research, we will establish the foundation for research into such algorithms by formally defining the problem, analyzing its key differences from traditional circuit partitioning problems, developing new problem-specific quality metrics, demonstrating modifications to well-understood algorithms for traditional partitioning problems, and experimentally evaluating the benefits of algorithms that can make use of the flexible resources on FPGAs.

4.2. Multi-Personality Partitioning Problem Definition

To update the complex-constraint partitioning problem to these modern devices capable of implementing designs using different heterogeneous resources, we defined a new *multi-personality partitioning problem (MPP)*. Given a graph with one or more multi-personality nodes, divide the nodes into two disjoint subsets and select personalities for every multi-personality node, such that for every resource, *r*, the difference in the sum of the node weights in each subset is less than some pre-defined maximum margin of imbalance, I_r. Extending the notation from our discussion of heterogeneous graph partitioning in Section 2.3 − where V is the set of all nodes in the graph, V_1 and V_2 are the partitions of V, and v_i is an individual node − if we further define all possible implementations of node v as P_v, the node's selected personality as $p(v)$, and the weight in resource r for that personality as $w_r(p(v_i))$, we can reformulate the heterogeneous partitioning constraints from Equation (4) as:

$$\frac{\left| \sum_{v_i \in V_1} w_r(p(v_i)) - \sum_{v_j \in V_2} w_r(p(v_j)) \right|}{\sum_{v_k \in V} w_r(p(v_k))} \leq I_r \quad \forall \quad r \in 1..R \tag{10}$$

It is important to note that every node has only one 'selected personality', $p(v)$, even when there are multiple possible personalities. The selected personality is the personality that has been chosen to implement the node, and the choice of which personality to select becomes an optimization decision for the partitioning problem. The addition of the personality selection component to the multi-resource

personality problem adds greater degrees of freedom and more potential solution candidates. Whereas the previous complex-constraint partitioning problem (hereafter referred to as *single-personality partitioning*) has $O(2^{|V|})$ potential solutions, where $|V|$ is the number of nodes in the graph, multi-personality partitioning increases the number of potential solutions to $O(C(P)\times 2^{|V|})$ potential solutions, where

$$C(P) = \prod_{v \in V} |P_v| \tag{11}$$

The number of combinations of personalities scales non-polynomially with the number of multi-personality nodes, and potentially faster than the number of partitions. The larger number of potential solutions may mean that it is possible to find superior solutions compared to single-personality partitioning, and the ability to change personalities provides an additional axis of freedom that can be exploited in partitioning algorithms.

4.3. Manipulating Resource Utilization

The incorporation of personality mapping into partitioning also necessitates new quality metrics for the partitioning problem. As mentioned in Section 4.1, balancing the utilization of multiple resources is important in practical applications and has been identified as one of the key open problems for partitioning algorithms targeting heterogeneous devices. In contrast to existing research on complex-constraint partitioning where the total resource utilization across all partitions is fixed, in MPP the resource utilization is dynamic, because changing a node's personality changes its resource requirements. Achieving a specific ratio of resource utilization is often desirable, as the design being partitioned may need to be mapped to a supercomputer, array of FPGAs, or other set of devices with a predefined ratio of available resources. To this end, we define a *resource utilization ratio* (RUR), which allows the partitioning problem to also specify a desired ratio for the use of the heterogeneous resources within each partition. To evaluate how well a solution achieves the RUR, we also introduce the concept of a new optimization metric for MPP: *RUR deviation*. This metric is computed as the root-mean-square (RMS) of the percent deviation of each resource, from its target value. That is,

$$\text{RUR Deviation} \equiv \sqrt{\frac{1}{R} \cdot \sum_{1}^{R} \frac{\left(U_{T,r} - U_{A,r}\right)^2}{U_{T,r}{}^2}} \qquad (12)$$

where $U_{T,r}$ is the target utilization in resource r, and $U_{A,r}$ is the actual utilization in resource r.

As we analyze the efficacy of MPP algorithms later in this chapter, we will evaluate them both in terms of the traditional cut size optimization problem and as a multivariate optimization problem that attempts to optimize for both cut size and RUR deviation.

4.4. Related Work

The basic graph partitioning problem is one of the classic optimization problems of Computer Science [6]. Kernighan and Lin's eponymous "KL" algorithm [60] − a local search optimization algorithm developed for the Traveling Salesman problem − was widely applied to the problem of graph partitioning. The basic idea of the KL algorithm was to perform iterative improvement on an existing, non-optimal bi-partitioning solution by selecting one node from each partition and swapping their places. The algorithm chose the two nodes that would most reduce the cut size when swapped. This process was then repeated until there is no further improvement in the solution. Among the key limitations of KL was the fact that every move required the algorithm to examine every possible pair of nodes that could be swapped was an $O(N^2)$ operation in terms of the number of nodes in the graph. This was acceptable for small graphs, but did not scale to the size of the graphs needed to represent many important partitioning problems, including circuit partitioning. Fiduccia and Mattheyses extended KL to "KLFM" (sometimes referred to as just "FM") by transitioning from graphs to hypergraphs (allowing them to model key features of networks and circuits, such fanout) and adding the gain bucket data structure and a new move heuristic to guarantee linear-time complexity [61]. Crucially, by relaxing the requirement that graph partitions be perfectly balanced, KLFM allowed asymmetric moves − that is, moving a single node to a different partition without needing to simultaneously swap it with another node. This, along with the use of a special data structure for finding the best node to move, allowed KLFM to improve the KL algorithm's

move operation to linear time complexity. Since KLFM will be one of the partitioning algorithms that we modify for MPP, we will discuss the details of the algorithm more thoroughly in Section 4.8.1. KLFM has remained a very popular algorithm for the circuit partitioning domain, in part because the low degree of fanout in hypergraphs based on real circuit designs leads to very fast performance for KLFM. Many incremental improvements to KLFM to improve search efficacy and escape local minima have been proposed; for the sake of brevity in this document, we refer readers to surveys of the topic [62] [63]. In general, KLFM-based local search and stochastic algorithms (sometimes used in concert) have been the most widely used in research related to digital logic partitioning, but other optimization heuristics have also been applied to graph partitioning, including genetic algorithms [64], spectral decomposition of a sparse matrix circuit representation [65], neural networks [66], simulated annealing [67], and Tabu Search [68].

Adaptation of graph partitioning algorithms to the increasing complexity and heterogeneity of computing devices and systems required new innovations. Alpert observed that the performance of graph partitioning algorithms failed to scale with the Moore's Law increase in circuit size, and introduced the concept of multi-level partitioning [69], which partitions a graph in multiple steps with decreasing levels of granularity, to improve performance scaling. Kuznar recognized that the fixed resource sizes of individual FPGAs represented a different problem than ASIC partitioning, and developed a cost minimization algorithm using integer linear programming [70]. Liu introduced the concept of using complex-resource graphs to model partitioning of the heterogeneous logic in FPGAs, proved the problem was NP-Complete, and described a KLFM-based heterogeneous partitioner [71]. Karypis (with Kumar) extended the theoretical framework of multi-constraint partitioning by including multi-resource clustering [72], and later (with Selvakkumaran) implemented practical multi-constraint partitioning for FPGAs in the hMETIS partitioner, including modifications to the gain bucket data structure to support multiple resource types [73]. They also demonstrated that the additional constraints of heterogeneous circuit partitioning significantly increased cut size. The key limitation of this work, when compared to the multi-personality partitioning problem we will describe in this chapter, is that it considered all FPGA resource

mappings as being fixed prior to partitioning. Taura and Chien adapted heterogeneous graph partitioning to the process of mapping a task graph to a computing grid with heterogeneous processors using the idea of balancing processor occupancy between partitions [74]. Wu addressed a similar problem in hardware, using integer programming to partition logic in time-multiplexed FPGAs [75].

As CAD has become more and more essential to managing the complexity of modern systems, there has been greater motivation for intelligent integration of CAD processes. Adya demonstrated that unifying the previously separate CAD processes of partitioning, placement, and floorplanning led to superior results in ASIC design [76]. Taghavi discussed the weaknesses in applying ASIC-driven physical design algorithms to FPGAs, and argued that existing partitioning algorithms needed further development in the problem of heterogeneous resource-balancing [77]. Tessier and Giza noted that an FPGA's configurable logic blocks might be used for either routing or for implementing logic, and that there was an interaction between logic partitioning and placement that impacted the overall area of a design [78].

Partitioning algorithms for hardware/software co-design share some similarities with multi-personality partitioning; Arato provides a formal definition of the problem as a form of graph partitioning [79], and logic may have different implementation costs in hardware or software. Lysecky and Vahid experimented with dynamic hardware/software partitioning on FPGAs using the MicroBlaze soft processor core along with specialized hardware accelerators [80], although both 'hardware' and 'software' in this case resided on the same physical chip. The use of the KLFM algorithm for hardware/software partitioning is also gaining support from some researchers. Mann argued that the KLFM heuristic could provide better results than generic stochastic algorithms [81]. When comparing hardware/software partitioning to the multi-personality graph partitioning problem that we will study in this chapter, it is important to note the major difference in the two problems: In hardware/software co-design, a node's implementation is fixed based on the partition (hardware or software) to which the node is assigned, and each partition is homogeneous. In multi-personality partitioning, partitions are heterogeneous and node personalities are a free variable within a partition. This is necessary to model heterogeneous devices (like FPGAs) or processor clusters, where multiple resource constraints must be managed within each partition. Moreover, the two problems

tend to have different optimization goals. Hardware/software partitioning tends to focus on application performance and meeting real-time deadlines, whereas graph partitioning focuses on logic utilization and minimizing communication costs. From an algorithms standpoint, hardware/software partitioning also tends to use much smaller, coarser-grained graphs where optimization approaches such as integer programming are more traditionally successful [82]. We will develop our own integer program for performing multi-personality partitioning and evaluate its efficacy on large problem sizes later in this chapter.

4.5. Benchmark Development

To evaluate algorithms for multi-personality partitioning, we needed to develop a set of benchmarks appropriate to the problem. We set the following criteria for our benchmarks:

- The benchmarks must cover a range of graph sizes and must include graphs up to sizes of at least hundreds of thousands of nodes. This is necessary to accurately reflect the size of modern graphs that would be partitioned between multiple devices.
- It must be possible to assign multiple personalities and resources to the nodes.
- The benchmarks should cover a variety of application domains and topographies, including both those based on circuit designs and on task graphs.
- The benchmarks must be publicly available.

A detailed description of our analysis of existing circuit and graph partitioning benchmarks can be found in [83]. We selected a benchmark set consisting of 21 graphs derived from hardware description language-based (HDL) designs for circuits and established benchmark graphs from previous partitioning research. Three of the benchmarks (*isolation, jet,* and *rct*) are based off firmware designs for particle accelerator experiments related to the work described in Chapter 3. The graphs in our benchmark suite are summarized in Table IV and described in greater detail in the following sections.

Table IV. Multi-personality benchmark graphs, sorted by ascending number of nodes.

Name	#Nodes	#Resources	%MP Nodes	Type	Description
sha	4K	2	8	HDL	Cryptographic Hashing
diffeq1	4K	2	38	HDL	Differential Equation Solver
raygen	11K	3	25	HDL	Ray-tracing
blob	12K	2	24	HDL	Image Processing
cti	17K	3	14	non-HDL	JOSTLE Benchmark
memplus	18K	3	63	non-HDL	Memory Circuit
boundtop	32K	3	15	HDL	Ray-bounding
wing	62K	3	6	non-HDL	JOSTLE Benchmark
brack2	63K	3	8	non-HDL	JOSTLE Benchmark
fe_tooth	78K	3	7	non-HDL	SuiteSparse Benchmark
fft128	92K	3	8	HDL	Fast-Fourier Transform
fe_rotor	100K	3	4	non-HDL	SuiteSparse Benchmark
598a	111K	3	20	non-HDL	METIS Benchmark
fe_ocean	143K	3	6	non-HDL	SuiteSparse Benchmark
144	145K	3	20	non-HDL	METIS Benchmark
wave	156K	3	11	non-HDL	RIACS Benchmark
isolation	188K	2	2	HDL	Particle Isolation
jet	190K	2	32	HDL	Particle Jet Discovery
m14b	215K	3	4	non-HDL	METIS Benchmark
rct	241K	3	30	HDL	Calorimeter Triggering
mcml	346K	3	15	HDL	Monte Carlo Simulation

4.5.1. HDL-Based Benchmark Development

In order to produce benchmark graphs that would be closely representative of the graph size, topology, and resource distribution that would be encountered in a real FPGA design CAD flow, we based our HDL-derived graphs on circuit designs that were developed to target FPGA platforms. To convert a circuit from HDL to a multi-personality graph, we used the following procedure:

1. We synthesized the HDL design using Synopsys Design Compiler [84] to its GTECH [85] non-technology specific synthesis library. This creates a structural netlist that can be converted to functional graph with simple logic functions bound to nodes.
2. We identified functional nodes within the structural netlist that correspond to functions that can be implemented in multiple resource types in FPGAs. These primarily consist of arithmetic operations (such as additional and multiplication), data storage (such as register files, memories, and FIFOs), and generic logic functions with a moderate number of inputs and outputs, such that they could be

mapped to a look-up table requiring less than the size of a block RAM unit (36 kilobits in our target FPGA architecture).

3. We back-annotated the multi-personality functions in the original HDL design as synthesis "black boxes", which directs the synthesis tool not to map them to hardware resources in its technology library.
4. We re-synthesized the black-box-annotated HDL design using the Xilinx Vivado synthesis tool [86] targeting the Xilinx Virtex-6 FPGA architecture. This optimizes and maps all non-multi-personality logic functions to specific FPGA hardware resources without transforming the multi-personality functions.
5. We separately experimentally synthesized each multi-personality function while directing the Vivado synthesis tool to use slice logic, DSP blocks, or block RAM respectively. We used this data to determine the physical logic costs of each possible personality.
6. We converted the black box netlist to a graph representation, attaching the personality logic costs to each node.

Our HDL-based circuits come from a variety of application domains. These are as follows:

- **sha** − A secure cryptographic stream hashing circuit implementing the SHA1 algorithm [87], included in the ERCBench suite of benchmarks for reconfigurable hardware [88].
- **diffeq1** − A first-order, two-variable differential equation solver using successive approximation included with the VTR academic CAD suite for FPGA research [89].
- **raygen** − A raytracing circuit for producing accurate lighting in computer generated imagery. Included with VTR.
- **blob** − An image processing circuit that identifies similar regions in an image and computes their attributes and bounding boxes. Included with VTR.
- **boundtop** − A collision detection circuit used in ray simulation. Included with VTR.

- **fft128** − A 128-point Fast Fourier Transform circuit that can be used for diverse signal processing applications. Included with ERCBench.
- **isolation** − A large-scale energy-based discrete particle isolation system for high energy physics experiments. Developed for the CMS experiment at the proposed HL-LHC particle accelerator [4].
- **jet** − A multi-particle jet reconstruction and summary circuit, designed for the HL-LHC's global calorimeter trigger.
- **rct** − A region-based electron and tau particle discovery and characterization system, designed for the HL-LHC's regional calorimeter trigger. See Chapter 3 for more details on the design of this system.
- **mcml** − A Monte-Carlo simulation of photon scattering effects in light-based cancer therapy. Included with VTR.

4.5.2. CHACO Benchmark Graphs

In addition to the benchmark graphs we generated from HDL-based circuit descriptions, we also included a variety of partitioning benchmark graphs in the CHACO graph format [90] available from the Walshaw graph partitioning archive [91]. The advantage of including these benchmark graphs is that the Walshaw archive provides extensive partitioning results for these graphs taken from over 40 academic and commercial partitioning algorithms, allowing us to reliably validate the performance of our base partitioning algorithms before adding multi-personality support. Additionally, since the utility of multi-personality partitioning may extend beyond use in the circuit partitioning domain, these benchmarks provide a more diverse set of graph topologies for applications ranging from structural modeling to financial analysis.

Since the CHACO-format graphs from the Walshaw archive do not include HDL code, it is not possible to assign personalities to nodes using the CAD flow described in Section 4.5.1. Instead, we use the personality resource costs obtained from experimental synthesis and randomly assign personalities to

nodes according to the proportions in Table IV. One advantage of including the CHACO graphs in our set of benchmarks is that we can use the single-personality versions of the graphs to compare the performance of our partitioning algorithms to other independently developed partitioners. We will compare our KLFM algorithm to that of hMETIS and the best results from the Walshaw archive in Section 4.10.3.

4.6. Multi-Personality Brute-Force Algorithm

Although multi-personality partitioning is NP-Complete [71], developing an optimal algorithm can yield some insight into the source of the problem's complexity. A simple brute-force algorithm can be formulated in four steps:

(1) Enumerate all possible node partitions − that is, all possible disjoint subsets of nodes.

(2) For each potential node partition from the previous step, enumerate all possible personality assignments to create a set of possible solutions.

(3) For each possible solution, check if the solution meets all resource constraints and compute the solution's cost.

(4) Select the lowest-cost solution that meets all constraints.

Without Step 2, the algorithm is identical to the brute-force algorithm for single-personality partitioning. This added step multiplies the number of possible solutions by a factor equal to the number of personality combinations, as described in Equation (11). Thus, the complexity strongly depends on the number of potential personalities in the graph, and may be significantly higher than in single-personality partitioning.

4.7. Multi-Personality Integer Programming

Although the multi-personality partitioning constraints may initially appear to be non-linear, it is possible to reformulate the problem into a 0-1 integer linear program (ILP) or integer quadratic program (IQP), depending on the chosen cost function. ILP/IQP computation times may be significantly lower

than that of a brute force approach due to the use of sophisticated heuristics exist. However, transforming a problem into an ILP/IQP is often non-trivial; in this section we will describe how the multi-personality partitioning problem may be reformulated using linear constraints.

4.7.1. Personality and Partition Selection Constraints

A post-partitioning multi-personality node can be described as a 3-tuple: for a node defined by parameters {*i, k, p*}, let *i* specify the node's identity in the graph, *k* denotes the partition that currently contains node *i*, and *p* denotes the personality selected for node *i.* Thus, each node can occupy one of $K \times P_i$ possible states, where K is the number of partitions and P_i is the number of personalities for the node. For brevity we will consider the basic bipartitioning problem where $k \in \{1, 2\}$ in the examples that follow, but the same approach can be applied to general K-way partitioning

To translate state selection into a linear constraint, we define a binary variable for each possible state. Thus, given a node, v_i, and two possible partitions, we must define $2P_i$ binary state variables of the form $v_{i,1,1}$ to $v_{i,1,P}$, and $v_{i,2,1}$ to $v_{i,2,P}$. Formally,

$$v_{i,k,p} \in \{0,1\} \qquad \forall\ i \in \{1..I\},\ k \in \{1,2\},\ p \in \{1..P_i\} \tag{13}$$

If the variable $v_{i,j,k} = 1$, then node *i* is located in partition *j* with personality *k.* If it is 0, the node is in a different partition, has a different personality, or both. Since the node must be in exactly one state, its partition and personality mapping can then be expressed as a set partitioning problem. This can be decomposed into linear constraints for an ILP solver by adding, for each node, one constraint of the form:

$$v_{i,1,1} + v_{i,1,2} + \ldots + v_{i,1,p} + v_{i,2,1} + v_{i,2,2} + \ldots + v_{i,2,p} = 1 \tag{14}$$

and an additional $2P_i$ constraints of the form:

$$v_{i,k,p} \geq 0 \tag{15}$$

4.7.2. Resource Constraints

The partitioning must satisfy a maximum imbalance constraint for each resource, as described in (10). To translate this into linear constraints, separate equations are needed for each resource. Equation (10) can be reformulated as:

$$\left|W_{V1,r} - W_{V2,r}\right| \le I_r (W_{V1,r} + W_{V2,r}) \quad \forall \quad r \in 1..R \tag{16}$$

where W_{Vi},r is the total weight in resource r of all nodes in partition V_i. The total partition weight can be expressed as an linear equation using the binary variables defined in (13) and adding constant coefficients of the form $W_{i,p,r}$ (the weight of node i in resource r when personality p is selected):

$$W_{V_k,r} = \sum_{i \in V_k} \sum_{p \in P_i} W_{i,p,r} * v_{i,k,p} \tag{17}$$

To eliminate the absolute value operation from (16), it must be split into two paired knapsack constraints, which take the form:

$$\begin{aligned} (I_r + 1)W_{V1,r} + (I_r - 1)W_{V2,r} \le 0, \\ (I_r - 1)W_{V1,r} + (I_r + 1)W_{V2,r} \le 0 \end{aligned} \tag{18}$$

Thus, the resource imbalance constraint can be expressed as integer linear equations. To expand support from bipartitioning to multi-way partitioning, similar balance constraints must be constructed for every combination of two partitions; the number of imbalance constraints then increases according to the scaling factor:

$$\binom{K}{2} = \frac{K(K-1)}{2} \tag{19}$$

4.7.3. Objective Function

The last step required to adapt the partitioning problem is to express the cost function as a linear or quadratic equation of the constraint variables. If the cost function is the cut size, then this function can be expressed as:

$$\text{Minimize: } \sum_{\forall e} C(e) * E_e \tag{20}$$

where *C(e)* is the cost of hyperedge *e* that spans partitions, and E_e is a binary variable indicating whether the edge spans partitions. In Boolean form:

$$E_e = T_{e,1} \;\&\&\; T_{e,2}, \quad \text{where} \quad T_{e,k} = v_{1,k,1} \,||\ldots|| \, v_{i,k,p} \tag{21}$$

for all v_i connected to *e*. The Boolean equations are converted into sets of 0-1 linear constraints. For multivariate optimization, RUR and other cost metrics can be linearly incorporated into the ILP objective function as needed, or an IQP solver can be used if non-linearity is desired.

4.7.4. Computational and Spatial Complexity

Although optimal partitioning is NP-Complete, ILP solvers use generic heuristics such as branch-and-cut to more efficiently narrow the solution space, and may produce feasible, non-optimal solutions within a fixed computational budget. However, spatial complexity may still pose a significant barrier for large problem sizes. Approximately $O(V + K \cdot \overline{P_v} \cdot D \cdot E)$ constraint equations are required, where *V* is the number of nodes, *E* is the number of hyperedges, *K* is the number of partitions, $\overline{P_v}$ is the average number of personalities per node, and *D* is the average hyperedge degree. In the worst case, *D* = *V*, though in many cases edge degree has a relatively small upper bound due to practical design limitations such as wire fanout constraints. The number of variables required is approximately $O(K \cdot \overline{P_v} \cdot V + E)$. Since the size of ILP's coefficient matrix is the product of the number of rows and columns, it scales quadratically with the graph size, number of partitions, and the prevalence of node personalities. The number of non-zero entries scales with *V*, $\overline{P_v}$, and *D*, but generally the matrix is sparse. As we will discuss in our experimental results, the spatial complexity of the ILP led to unfeasibly high memory requirements for larger graphs.

4.8. KLFM-Based Partitioning Algorithm Design

Because of the complexity of general optimization algorithms, it is worthwhile to consider whether algorithms designed specifically for partitioning produce better results within the same constraints or more easily scale to larger problem sizes. We develop a heuristic-driven multi-personality partitioning algorithm based on a well-known single-personality partitioning algorithm that can scale to large problem sizes: the multi-level KLFM algorithm [61] [69]. KLFM is a popular and well-studied algorithm, and has been adapted for multi-resource, single-personality partitioning in the past, therefore it serves as an ideal basis to evaluate the impacts of integrating personality selection into the partitioning algorithm. In this section, we outline the key features of the basic KLFM algorithm and discuss the modifications to the algorithm needed to support multi-personality graphs.

4.8.1. Base KLFM Algorithm

KLFM is a neighborhood-search-based algorithm for graph partitioning that is capable of finding local optima, but may not necessarily find a globally optimal solution. We describe the KLFM algorithm in pseudo-code in Algorithm 1. To roughly summarize: starting from an initial solution, it performs several passes where, in each pass, it moves all nodes from their starting partition to another partition. Nodes are moved in a priority order determined by how the move will improve the cut size and whether moving that node would violate the constraints on imbalance in the size of the partitions. The cost is updated after each move, and the algorithm keeps track of the best solution it has created thus far. At the end of a pass, the best solution is used as the starting point for the next pass. The algorithm continues in this manner until no improvement is seen over the best solution from the previous pass. The original KLFM formulation only considered a single resource constraint, but it can be extended to use multiple constraints to model heterogeneous circuit resources without significant changes to the algorithm [71].

The algorithm uses a special data structure referred to as a *gain bucket*, which selects the nodes to move between partitions. A gain bucket in each partition maintains a set of priority queues that contain

the 'unlocked' nodes that in that partition that are eligible to switch partitions (i.e., the nodes that have not already been moved in the current pass of the algorithm). Nodes are prioritized based on their *gain* − the reduction in the graph's cost which would result from moving that node. The “top” node in a gain bucket is always the highest-gain node that can be moved without violating the maximum imbalance constraint for the graph's partitions. Moving a node between partitions may cause the gains of neighboring nodes to change, so the gain buckets must be updated after each move. Efficiently updating the node gains is essential to the computational complexity of the algorithm [61]; it relies on the fact that it is only necessary to update the gains of unlocked nodes that are neighbors to the node that most recently moved. KLFM is very efficient for circuit partitioning because circuit nodes tend to have few neighbors due to wire fanout limitations [92]. Many gain bucket structures have been studied by researchers, but the gain bucket is commonly implemented as an array of queues with last-in/first-out ordering [62]. This gives KLFM's computational complexity an upper bound of O(*Q*), where *Q* is the sum of the number of ports (node-edge connections) on all the nodes in the graph. This allows KLFM to run relatively quickly for most netlists, which is particularly significant because it is common to run the algorithm several times per partitioning problem and select the best result. Spatial complexity is also O(*Q*), a feature that will become important when comparing the memory requirements of our ILP and KLFM-based MPP algorithms for large graphs.

4.8.2. Multi-Personality KLFM

Multi-Personality KLFM builds on the KLFM algorithm by adding the ability for node personalities to be changed as part of the partitioning process. The algorithm is given the freedom to change node personalities in order to make it easier to meet balance constraints (allowing higher gain moves) or to control for resource utilization ratios when using multivariate optimization. The simplest way to convert an existing single-personality algorithm to support multiple personalities is by using static personality mapping to assign each node a single personality *a priori*. A prototypical static multi-personality partitioning algorithm is outlined in Algorithm 2. As shown in the pseudocode, it is also possible for a

statically-mapped algorithm to readjust node personalities after partitioning for reasons such as optimizing resource utilization. We discuss alternative strategies for initial personality assignment and partition post-processing in detail in Section 4.10.2.

Algorithm 1. KLFM Partitioning for Single-Personality Graphs

Input: A single-personality graph, containing nodes with weights in one or more resources, and a set of maximum imbalance constraints for each resource.
Output: A single-personality partitioning solution, consisting of K disjoint subsets, containing all the nodes from the input graph.

```
{current_solution, current_cost} = GenerateRandomInitialSolution(graph, max_imbalances);
gain_buckets.SetMaximumResourceImbalances(max_imbalances);
repeat
    {new_solution, new_cost} = {current_solution, current_cost};
    gain_buckets.AddAllNodes(current_solution);
    while gain_buckets.HaveNodes() do
        node = gain_buckets.RemoveTopNode(new_solution.ResourceImbalances());
        new_solution.MoveNode(node);
        new_cost = new_cost - node.Gain();
        gain_buckets.UpdateNodeGains();
    end
until new_solution == current_solution;
```

Algorithm 2. Statically-Mapped Partitioning for Multi-Personality Graphs

Input: A multi-personality graph, containing nodes with weights in one or more resources, a set of maximum imbalance constraints for each resource, and a target RUR.
Output: A multi-personality partitioning solution, consisting of K disjoint subsets, containing all the nodes from the input graph, with one personality assigned to each node.

```
single_personality_graph = AssignInitialPersonalities(graph, target_RUR);
solution = PartitionSinglePersonalityGraph(single_personality_graph);
// Optional Post-Processing Step
for each partition in solution do
    partition.Readjust(max_imbalances, target_RUR);
done
```

Adding dynamic personality mapping to change node personalities during the KLFM process requires several changes, as shown in Algorithm 3. Dynamic personality selection gives rise to many important questions about when and how to change node personalities, leading to more complex implementation options than are present when using static mapping. Some of the issues we will discuss include deadlock prevention and computational complexity (Section 4.8.3), fine-grained changes to node personalities on a

per-move basis (Section 4.8.4), and coarse-grained global approaches that rebalance node personalities for the entire graph (Section 4.8.5).

Algorithm 3. Dynamically-Mapped KLFM Partitioning for Multi-Personality Graphs

Input: A multi-personality graph, containing nodes with weights in one or more resources, a set of maximum imbalance constraints for each resource, and a target RUR.
Output: A multi-personality partitioning solution, consisting of K disjoint subsets, containing all the nodes from the input graph, with one personality assigned to each node.

```
{current_solution, current_cost} =
   GenerateRandomInitialSolution(graph, max_imbalance, target_RUR);
gain_buckets.SetMaximumResourceImbalances(max_imbalances);
gain_buckets.SetTargetRUR(target_RUR);
repeat
      {new_solution, new_cost} = {current_solution, current_cost};
      gain_buckets.AddAllNodes(current_solution);
      rebalance_available = TRUE;
      while gain_buckets.HaveNodes() do
          if gain_buckets.HaveFeasibleMove() do
                node = gain_buckets.RemoveTopNodeAndAssignNewPersonality(
                   new_solution.Imbalance(), new_solution.RUR());
                new_solution.MoveNode(node);
                new_cost = new_cost - node.Gain();
                gain_buckets.UpdateNodeGains();
           end else if rebalance_available do
                new_solution.RebalancePersonalities(target_RUR);
                rebalance_available = FALSE;
           end else do
                Quit();
           done
      end
until new_solution == current_solution &&
       new_solution.RURDeviation ≥ current_solution.RURDeviation().;
```

4.8.3. Deadlock Prevention for Multi-Personality Graphs

Dynamic personality selection also introduces a new challenge to ensuring algorithm correctness: the potential for deadlock. In traditional KLFM, as long as no node's individual weight exceeds the maximum imbalance, it is always possible to find a node that can be moved without violating constraints. This does not change for multi-resource graphs when using static personality mapping. Previous work has shown that deadlock can be avoided in multi-resource single-personality graphs by using separate gain buckets for each resource [73], though this restriction disallows coarse-grained nodes that use multiple resource

types − a limitation that makes it difficult to employ multi-level partitioning and would be incompatible with mixed resource designs such as those we developed for high-energy physics systems in Section 3.4.

Avoiding deadlock is significantly more complex in dynamic multi-personality partitioning, because the algorithm cannot guarantee that there will always be a move available that does not violate any balance constraints. Depending on the personalities selected during previous moves, it may become impossible to empty the gain bucket without violating balance constraints. Furthermore, we also do not disallow personalities that use multiple resource types, meaning that the algorithm cannot work with each balance constraint in isolation of the others. Instead, we employ several other strategies. First, we incorporate mechanisms to locate nodes with given resource mixtures (Section 4.8.4). Second, we use the RUR metric (Section 4.3) to maintain the global balance of resources. Third, we allow for on-demand reselection of personalities for nodes that have already been moved (Section 4.8.5). Finally, we modify KLFM to allow temporary balance constraint violations in the hope that future moves eliminate the violation. However if constraints are still violated at the end of a pass, the solution is rolled back to the last non-violating solution. The approach of temporarily ignoring balance constraints is often effective. We use a modified version of this strategy to improve partitioning results when using clustered multi-level KLFM, as described further in Section 4.8.6.

4.8.4. Dynamic Gain Buckets

In single-resource KLFM, the gain buckets are used to find the highest-gain node that can be moved without violating balance constraints. Allowing the gain bucket to also dynamically select node implementations has several potential advantages:

- Changing a node's implementation might allow it to move without violating balance constraints (when it otherwise could not), increasing the partitioner's ability to move higher-gain nodes and escape local minima.
- A node's implementation can be changed as it moves to improve the overall resource balance across partitions.

- A node's implementation can be changed to improve other resource-related quality metrics, such as RURs.

Because multi-personality partitioning must minimize the total cut size while also controlling the overall resource utilization of each partition, a more complex policy is needed for selecting which node to move, and new policies are needed for selecting node personalities. We experimented with two different personality-aware gain buckets: *Multi-Personality Buckets* and *Resource-Affinity Buckets*.

Multi-Personality Buckets

Multi-Personality Buckets use a single gain-sorted priority queue of nodes, similar to the basic gain bucket. Unlike the basic gain bucket, each node entry includes a set of resource costs for each of its possible personalities. When selecting the next node to move in KLFM, we examine each gain bucket, and check whether the top node has at least one personality that would allow it to be moved without violating balance constraints. If not, we move on to the next node in the bucket, continuing until we find a node that can be moved or we exceed the maximum search depth. Once we have the best node from each bucket, we choose the one with the highest gain. As described in Section 4.8.2, it may not be possible to find a non-violating move; therefore, we set a maximum search depth to maintain the algorithm's asymptotic complexity.

In addition to choosing which node to move, the partitioner also needs to choose a personality for that node. The change in resource balance across partitions depends on both the movement of the node and the (potential) change in its personality. Personalities may be selected either to reduce the degree of resource imbalance between partitions (part of the partitioning constraints) or to reduce deviation from the target RUR (part of the optimization goals). If any of the resources are within 10% of violating their imbalance constraints, the algorithm prioritizes alleviating the imbalance when selecting the personality. For all of the node's potential personalities, the partitioner computes an *imbalance score*, defined as the root-mean-square (RMS) of the fractional imbalances in all resources. The RMS value tends to emphasize large imbalances in a single resource over small imbalances in many resources, which is desirable since large

imbalances are more likely to prevent future high-gain moves. The partitioner then chooses the personality with the lowest imbalance score.

If the partitions are not near their balance limits, the partitioner also selects personalities based on the resource utilization goals. To incorporate the target RUR goal into move decisions, it computes a similar *resource-ratio score* for each possible personality. This score is the RMS of the percent deviation from the target utilization for each resource, based on the total utilization of all resources in a partition and the target utilization ratio. The partitioner then selects the personality that minimizes a weighted sum of the imbalance and resource-ratio scores.

Resource-Affinity Buckets

Although using Multi-Personality Buckets allows us to consider heterogeneity and multiple personalities when making node moves, gain bucket queues are prioritized only by a move's effect on cut size. Sometimes it may be advantageous to instead choose moves based primarily on their effect on resource balance and utilization. For instance, if a large imbalance exists in the hypothetical Resource X, we may be more interested in re-balancing Resource X than reducing cut size; yet the highest-gain nodes may not have personalities that use Resource X. Resource-Affinity Buckets are designed to allow the partitioner to effectively "peek" deeper into the buckets to find nodes that may have a slightly lower gain, but a greater impact on imbalanced resources.

Resource-Affinity Buckets use multiple priority queues per partition, with each queue containing only nodes with personalities that match a specific resource-use profile. Before partitioning, we calculate the percent of each resource required for each node personality and assign an "affinity" for the resource which it uses most. It is inserted into the queue associated with that resource. We also create mixed resource affinities to allow for nodes that using multiple resources in a single personality. When making moves, the algorithm examines the top node in each of the affinity queues of the gain bucket. These queues provide access to nodes with specific resource mixtures, so it may be able find a node that can relieve resource imbalances or improve resource utilization ratios.

Because an individual node can have multiple personalities, the same node may appear in multiple different affinity queues. When moving a node selected from one queue, the partitioner must also purge its alternate personalities from any other queues. Thus, each node entry in a queue contains pointers to any other entries for that node in other queues. Although this complicates the implementation, it does not increase the asymptotic complexity of the algorithm. However, accesses to these references may exhibit poor spatial locality, leading to increased run times. We study the computational cost of different MPP algorithms in Section 4.10.6.

Hybrid Buckets

Multi-Personality Buckets are designed to simplify finding the highest-gain moves when resource imbalance is not critical, whereas Resource Affinity Buckets are designed to simplify finding good moves in situations where resource imbalance constraints make it difficult to find valid moves. The two bucket types can be combined into Hybrid Buckets. When using Hybrid Buckets, the partitioner maintains both gain bucket data structures and selectively chooses which structure to use based on the current resource imbalance conditions. This approach leverages the strengths of both methods; however, both sets of buckets must be updated each move, leading to greater computational overhead than using a single bucket type.

4.8.5. Pass-Level Implementation Remapping

We employ a global remapping operation that allows the partitioner to adjust all node personalities in the graph to reduce resource imbalance and/or improve RURs. This is a relatively expensive operation compared to the gain buckets' local, move-based personality selection. To avoid increasing the overall complexity of the algorithm, we globally remap only once per KLFM pass. In our experiments, the number of passes required for the algorithm to terminate scaled sub-linearly with graph size, and can also be limited as a parameter of the algorithm. Thus it is possible to use O(V) global remapping algorithms at per-pass frequency while maintaining the overall linear complexity of the base KLFM algorithm. We examined two different methods for global remapping: multi-phase greedy and fractured ILP.

Multi-Phase Greedy Personality Remapping

The greedy remapping algorithm performs a random walk of the nodes within each partition, alternating between partitions. For each examined node, the algorithm selects the personality that minimizes the chosen resource cost. Since the results are dependent on the order that nodes are visited, and because the greedy algorithm is relatively cheap, we experimented with running it multiple times with different node orders; we observed that mappings generally remained stable after 2-3 passes.

Fractured ILP Personality Remapping

The integer program described in Section 4.7 can be adapted into a simpler form that only handles personality remapping, and thus exhibits better scaling with increasing graph size. As we will show a little later in Section 4.10.1, ILP performs well when we are able to keep problem sizes small. To avoid excessive memory requirements, we limit its use to graphs of less than 10,000 nodes. For larger graphs, we arbitrarily fracture each partition into subsets of roughly 10,000 nodes, and run the ILP remapper on these sub-partitions. We do not require an optimal solution, and use a branch-and-bound heuristic with limited depth to bound complexity. In our experiments, ILP-based remapping performed marginally better than greedy remapping, but was also slower. Thus we only used the ILP remapper in our high-effort partitioner configurations.

4.8.6. Multi-Level Partitioning

KLFM's partitioning quality tends to scale poorly with the graph sizes needed to represent large-scale modern circuits, motivating the creation of multi-level KLFM [69]. In multi-level partitioning, nodes are clustered hierarchically into larger and larger supernodes before partitioning. After partitioning the resulting graph of supernodes, the graph is de-clustered by one level, and the previous result is used as the initial starting point for the next round of partitioning. This continues until the graph has been de-clustered back to its original form. There are several challenges to supporting multi-level partitioning with multi-personality nodes.

Supernode Clustering

A key aspect of multi-level algorithms is the method used for forming the supernode clusters. Because partitioning is focused on minimizing the number of edges that span partitions, clustering is often based on connectivity and density-based node agglomeration [93] [94]. For single-personality heterogeneous graphs, it has been argued that forming clusters with resource weights in a similar ratio to the total weights in the graph may make it easier to move nodes without causing balance violations [72].

The addition of multiple node personalities increases the complexity. Although a single node may have a small number of possible personalities, a supernode can take on all possible combinations of the personalities of its subnodes. These combinations scale as described in (11), thus evaluating all personalities rapidly becomes computationally intractable, and it is necessary to limit the number of personalities.

Our clustering algorithm is based on greedy modularity [95], but forms clusters based on node personalities. To select the personalities to include in the subsets, we attempt to achieve similar ratios to the target RUR. We also include personalities similar to those used by the Resource Affinity Buckets to provide the partitioner with the greatest ability to adjust resource balance. In our experiments, we limit each supernode to 16 personalities; our tests found no significant improvement beyond this number.

Multi-Level Constraint Relaxation

During experimentation, we observed that the algorithm rarely took advantage of personality changes at the coarsest levels of multi-level partitioning. Very large supernodes have large weights in every resource for each of their potential implementations; once the partitions are close to the maximum imbalance constraint in multiple resources, it is difficult to change implementations without causing a balance violation in at least one resource. Thus we relax resource constraints based on the coarseness of the partitioning level to increase freedom and improve results at the coarsest levels. As the algorithm progresses to less coarse graphs, constraints are tightened. Global remapping and move-based dynamic personality selection often allow the partitioner to meet tighter constraints at finer levels of partitioning.

We found that it was particularly effective for FPGA netlists to use multi-level constraint relaxation for DSPs and BRAMs, but maintain strict enforcement on the CLBs. Most logic can be converted into CLBs, thus it tends to be easier to adjust the balance of CLB utilization during partitioning than it is for other resources.

4.9. Implementing the Multi-Personality Partitioning Algorithms

We implemented several different optimization algorithms designed to handle MPP, including an optimal algorithm using brute-force search (infeasible for any realistic problem sizes), general-purpose heuristic-based algorithms using 0-1 integer linear programming (ILP) and quadratic integer programming (QIP), and several algorithms based on the Kernighan-Lin/Fiduccia-Mattheyses (KLFM) approach [61] that is commonly used for partitioning of single-personality partitioning.

A key limitation of adapting general-purpose optimization algorithms is the difficulty in scaling such algorithms to large problem sizes. As we noted in Section 4.7.4, the spatial complexity of ILP scales super-linearly with the size of the graph, the degree of potential heterogeneity, and the number of partitions. As a consequence, the memory requirements make an ILP-based approach infeasible for the larger benchmarks, such as those for high energy physics, Monte-Carlo simulation, or fluid dynamics. Using the SCIP solver [96] on our benchmark suite, we observed that the working set exceeded the 32 GBs physical memory available in our test system for graphs larger than 30,000 nodes. This is reflected in our later results.

The high memory requirements help to motivate the development of algorithms designed specifically for efficient partitioning. We developed multiple variants of the KLFM partitioning algorithm capable of performing MPP in order to better understand how to effectively use personality information to obtain high-quality partitions. They are described in brief below. See [97] and [98] for more details on implementation and terminology of these algorithms and their underlying data structures.

4.9.1. Partitioning Algorithms Using Static Personality Mapping

Map-First KLFM (MF)

Map-First KLFM (MF) uses static personality mapping, where personalities are mapped to meet the target RUR prior to partitioning. It is modeled after the existing approach of performing resource mapping first, then performing partitioning on the pre-mapped netlist used in existing CAD flows for FPGAs. We therefore use MF as a baseline in our algorithm comparisons and normalize the results of other partitioners to those of MF.

Our implementation of MF is based on the Native Multi-Constraint Refinement algorithm [73], developed for partitioning circuits for heterogeneous FPGAs. MF is a single-personality partitioning algorithm; it supports nodes with heterogeneous resources, but does not recognize the concept of multiple personalities. To apply the MF algorithm to a multi-personality graph, all nodes' personalities must be selected and locked prior to partitioning. We accomplish this using our ILP-based global personality mapping algorithm, described in Section 4.8.5. Personality mappings are chosen to achieve the global RUR target, but the mapping does not incorporate netlist topology information; when choosing personalities to set the global resource utilizations, it does not consider the difficulty of partitioning the graph in a way that achieves the same utilization ratios for each partition.

Partition-First KLFM (PF)

Partition-First KLFM (PF) also uses static personality mapping, but personalities are mapped after partitioning. It is based on the existing multi-level KLFM algorithm for homogeneous graphs [69] with the addition of post-partitioning personality remapping. Although the final node personality mapping is done after partitioning, the initial partitioning algorithm needs some concept of node weight to use when determining if partitions are balanced, otherwise it might produce solutions that fail to meet resource constraints under any possible personality mapping. Our algorithm converts as many nodes as possible into the most commonly-usable resource to provide greater flexibility in moving nodes without violating the maximum resource utilization. For FPGA-based netlists, it is often possible to convert almost all

nodes into personalities that use only CLBs, as the look-up tables in CLBs can implement arbitrary logic functions and CLBs include storage elements. The netlist is then partitioned using only the weights from that common resource. Because the algorithm does not consider the weights in other resources, the result may initially contain resource balance violations that must later be fixed. After partitioning, the ILP-based personality remapping algorithm attempts to fix any balance violations and improve RURs. Afterward, if balance violations still exist, nodes are greedily moved between partitions until the balance violations are resolved. This process alters the partitioning solution and typically increases the cut size.

PF is conceptually similar to the Multi-Constraint Iterative k-way Balancing method for heterogeneous graphs [72], in that it attempts to achieve better cut size results by avoiding multi-resource constraints as much as possible during partitioning, and tries to enforce balance afterwards. However, unlike that algorithm, we attempt to leverage personality remapping to fix balance constraints, exploiting nodes' multiple personalities. This has the potential to fix violations without increasing cut size.

4.9.2. Partitioning Algorithms Using Dynamic Personality Mapping

Dynamic Multi-Personality KLFM (DMP)

Dynamic Multi-Personality KLFM (DMP) combines mapping and partitioning into a single process that allows personalities to be remapped multiple times to find the most efficient partitioning. Using the techniques described in Section 4.8, DMP uses move-based personality selection using Resource-Affinity Buckets and pass-based global remapping using the multi-phase greedy algorithm. Implementation selection is based on the imbalance score. Finally, it uses global remapping using fractured ILP to improve the RUR after partitioning.

High-Effort Dynamic Multi-Personality KLFM (HDMP)

High-Effort Dynamic Multi-Personality KLFM (HDMP) is a modified version of the DMP algorithm that uses some of the more sophisticated but computationally-intensive personality mapping methods, trading off algorithm speed for potentially better results. HDMP uses Hybrid Buckets for move-based

selection and fractured ILP-based remapping for pass-based personality reselection. When performing multi-level partitioning, it first attempts multi-level constraint relaxation for non-CLB resources. If this fails to produce a result that does not violate resource constraints, it rolls back to the start of the pass and retries with normal constraints.

DMP/HDMP with Fine-Grained Ratio Control (DMP-FR / HDMP-FR)

Fine-grained Ratio Control (FR) is a modification to the DMP and HDMP algorithms that increases the priority of RUR when the gain bucket decides which node to move next. In the non-FR algorithms, a node is selected based on its gain and whether it has any personality capable of meeting balance constraints. After the node has been selected, the gain bucket selects the personality that minimizes deviation from the target RUR. With FR enabled, node selection is based on both gain and the node's best-case change in RUR, using a cost function of the form:

$$\text{move cost} = \alpha\left(\frac{gain}{cutsize}\right) + \beta\Delta R_{dev} \tag{22}$$

where R_{dev} is the current deviation from the target resource utilization ratio, and α and β are experimentally-derived scaling constants. The addition of RUR into the cost function allows moves that produce larger improvements in resource utilization, even when they are not the highest-gain move available.

Integer Linear Program (ILP)

We implemented the 0-1 linear program described in Section 4.7 using a cut-size-based objective function. Because personality selection is incorporated into the constraints, the ILP algorithm uses dynamic personality selection. Initially we ran the ILP solver without providing an initial solution. In several cases, we observed that the ILP solver had difficulty finding a feasible solution, so we also experimented with providing the ILP solver with a feasible solution as a starting point. Different methods for constructing the initial solution will be compared in Section 4.10.2.

4.10. Results

Before comparing the KLFM variants, we first examine how a general-purpose optimization heuristic, such as an ILP branch-and-cut solver, performs compared to a partitioning-specific heuristic like KLFM. We then examine how the implementation options we previously discussed impact the performance of KLFM partitioners.

4.10.1. Integer-Linear Programming

We evaluated the efficacy of the ILP-based partitioning approach using the SCIP ILP solver [96]. We found that although ILP could find good solutions when partitioning small graph sizes, its performance scaled poorly when used on large graphs, such as those representing high-energy physics systems. Figure

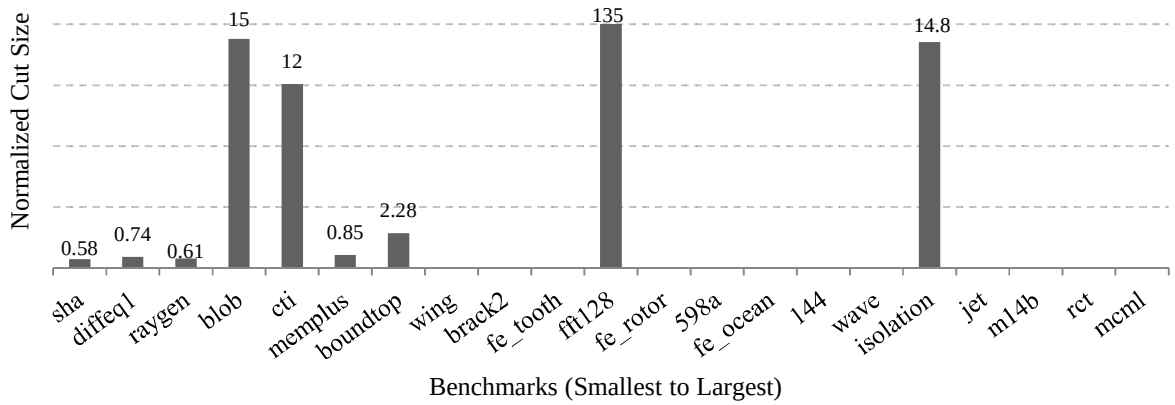

Figure 10. Cut sizes for ILP-based partitioning with a run time of 48 hours, normalized to MF statically-mapped partitioning. Missing bars indicate that ILP could not find a solution.

10 shows the cut size results of the ILP partitioner, normalized to the results of the KLFM-based MF partitioner. These results represent the best solution found by the ILP partitioner when given 48 hours of run time. This 48-hour run time is significantly longer than the time it takes to complete the KLFM algorithm, but these results are included to provide a sense of the bounds of ILP performance when not time-constrained. When the ILP partitioner's run-time is equalized to that of the MF partitioner its cut size results fall behind even for most small graphs, as shown in Figure 11.

The high spatial complexity of multi-personality ILP problems is at least partially responsible for the inability of the solver to find solutions for large graphs. Memory use scaled super-linearly with graph size

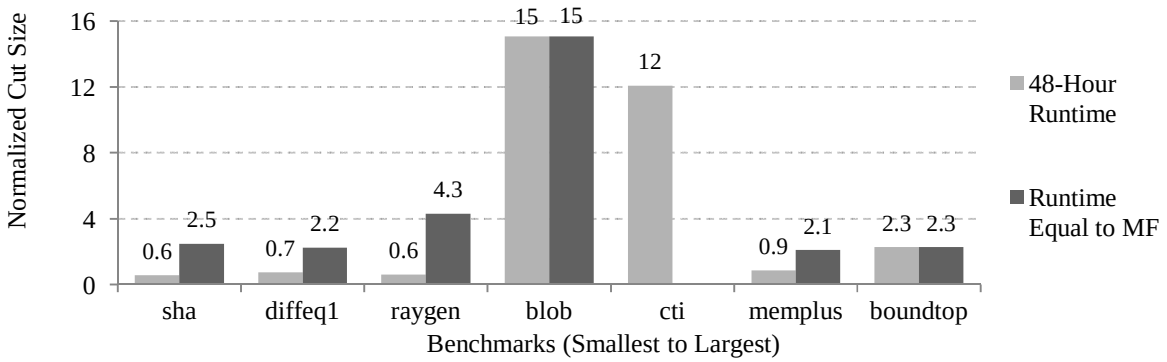

Figure 11. ILP cut sizes (normalized to MF) for graphs with less than 30,000 nodes for different amounts of runtime. ILP was unable to find a solution for cti when given runtime equal to MF.

and heterogeneity, and exceeded the 32 GB of physical memory in our test system for most graphs with more than 20,000 nodes. On its own, ILP is not effective as a multi-personality partitioning algorithm, but may have some utility when applied to smaller sub-problems; it is used to perform intra-partition personality remapping in the HDMP algorithm.

We were also interested in whether we might chain the KLFM and ILP algorithms together to get better results than running either algorithm by itself. To this end, we converted the solutions produced by the MF partitioner to provide an initial starting point for the ILP partitioner. However, the ILP partitioner was not able to improve upon the initial solutions even when given 48 hours of run time. We are not able to draw definitive conclusions on why the ILP solver was not able to improve these solutions. Given that the KLFM solution already represents a local minimum for the cost function, it may be that this initial solution is not necessarily any closer to a globally better solution than a random starting point would be. It may also be that these initial solutions do not provide much information for the branch-and-bound or branch-and-cut heuristics used by the SCIP solver to narrow their solution space.

4.10.2. Evaluating KLFM Parameters

We developed several alternative approaches for supporting multiple personalities in the KLFM algorithms' key data structures. To determine how to configure our KLFM-based partitioners, we implemented and evaluated each of these options.

Gain Buckets

We compared the average cut size achieved across all benchmarks by our DMP partitioning configuration when using it with the Multi-Personality, Resource-Affinity, and Hybrid gain buckets we described in Section 4.8.4. These results are normalized to those of multi-personality buckets and shown in Figure 12. The MP Bucket, a standard gain bucket modified to support multiple personalities, is outperformed by the Resource-Affinity Bucket, which sorts a node's personalities into different queues based on the personality's resource requirements. This suggests that being able to locate and move a node with the right resource mixture may often be more important than simply moving the node with the highest gain. The Hybrid Bucket, which combines Multi-Personality and Resource-Affinity Buckets, achieved a slight improvement over the Resource-Affinity Bucket, but at the expense of some increase in algorithm run-time, as we will discuss further in our later results.

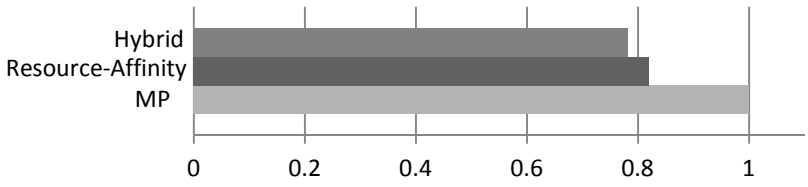

Figure 12. Average of the cut size results for all benchmark graphs using DMP and different gain bucket types, normalized to the Multi-Personality (MP) bucket type. Lower values are better.

Initial Partition Formation

We examined the impact of different methods for selecting the initial node personalities for the multi-personality algorithms by comparing the Random and Proportional-Selection methods using the DMP and DMP-FR algorithms across our collection of benchmark graphs. The results are summarized in Table V.

Neither method for selecting initial node personalities showed a statistically significant advantage for our benchmarks. The random selection policy resulted in a slight decrease in average cut sizes, of approximately 1.5%. This may be a result of the additional variability introduced into the algorithm's starting point. Although one's intuition might be that Proportional Selection should lead to lower

Table V. Normalized cut size and deviation from target resource utilization ratio for different initial node personality selection methods (Random vs. Proportional).

		Norm. Cut Size	RUR deviation
DMP	Random	1.000	22%
	Proportional	1.017	24%
DMP-FR	Random	1.087	8%
	Proportional	1.101	8%

deviation from the target resource utilization, it is important to remember that these dynamic algorithms can perform global personality remapping every pass, therefore the initial personality mapping is relatively short-lived.

4.10.3. Performance Validation of Base KLFM Algorithm

The multi-resource implementation of hMETIS is not publicly available and the benchmarks that were used to evaluate it were not disclosed [73]. The authors also declined to make the implementation or benchmarks available to us for comparison, so we cannot directly compare the performance our MF algorithm (which is based on hMETIS's multi-resource partitioner) to multi-resource hMETIS's published results. However, the single-resource version of hMETIS is publically available [99] and can be used to determine whether our KLFM implementation has competitive performance when partitioning single-resource graphs. Moreover, for the benchmark graphs from the Walshaw partitioning archive (see Section 4.5.2) we can compare the results to the best known results from 40 different partitioning algorithms [100]. Since it is generally computationally infeasible to obtain the optimal partitioning results, the Walshaw results provide a good substitute for gauging the absolute performance of our algorithm.

We ran our MF implementation and hMETIS 1.5.3 for Windows on the single-resource version of our set of CHACO graphs with an imbalance margin of 1%. hMETIS was run using its default parameters.

Table VI. Cut size comparison of MF, hMETIS, and the best of the 40 partitioners in the Walshaw Partitioning Archive for the single-resource versions of our CHACO-based benchmark graphs. Lower values are better.

	MF Cut Size	hMETIS Cut Size	Walshaw Cut Size	MF Norm. to hMETIS	MF Norm. to Walshaw
144	6738	6682	6478	1.01	1.04
598a	2420	2572	2388	0.94	1.01
brack2	708	948	708	0.75	1.00
cti	318	1064	318	0.30	1.00
fe_ocean	396	220	387	1.80	1.02
fe_rotor	2039	2152	2031	0.95	1.00
fe_tooth	4243	5052	3814	0.84	1.11
finan512	310	292	162	1.06	1.91
m14b	3927	3652	3826	1.07	1.03
memplus	7013	5012	5457	1.39	1.29
wave	8930	9358	8657	0.95	1.03
wing	935	2818	784	0.33	1.19
geomean	1762	2060	1579	0.86	1.12

The cut size results for MF, hMETIS, and the best result from the Walshaw archive are compared in Table VI. When comparing the values of MF and hMETIS to the Walshaw result, it is important to remember that the Walshaw result is the best result from many different partitioners, with no constraint on the run time of the algorithms. As such, it is not surprising that the Walshaw result is better than the MF and hMETIS results for most benchmarks. Our MF implementation had a geometric mean cut size that was 14% lower (better) than hMETIS and 12% higher than the best results from the Walshaw archive. From this we can conclude that the base KLFM algorithm we use in MF, our multi-resource partitioning algorithm with static personality mapping, has competitive performance to other KLFM algorithms − at least when run on the single-personality versions of our benchmark graphs.

4.10.4. Cut Size Results for KLFM-Based Algorithms

A comparison of the cut size results for the KLFM-based partitioners is given in Figure 13. Benchmarks are split to contrast the benchmarks derived from HDL designs for FPGAs, which are

representative of real-world heterogeneous circuit partitioning problems, and graphs taken from partitioning benchmark sets in the Walshaw archive, which may be more representative of the structure of task graph partitioning problems. In both cases, selecting node personalities prior to partitioning (as done

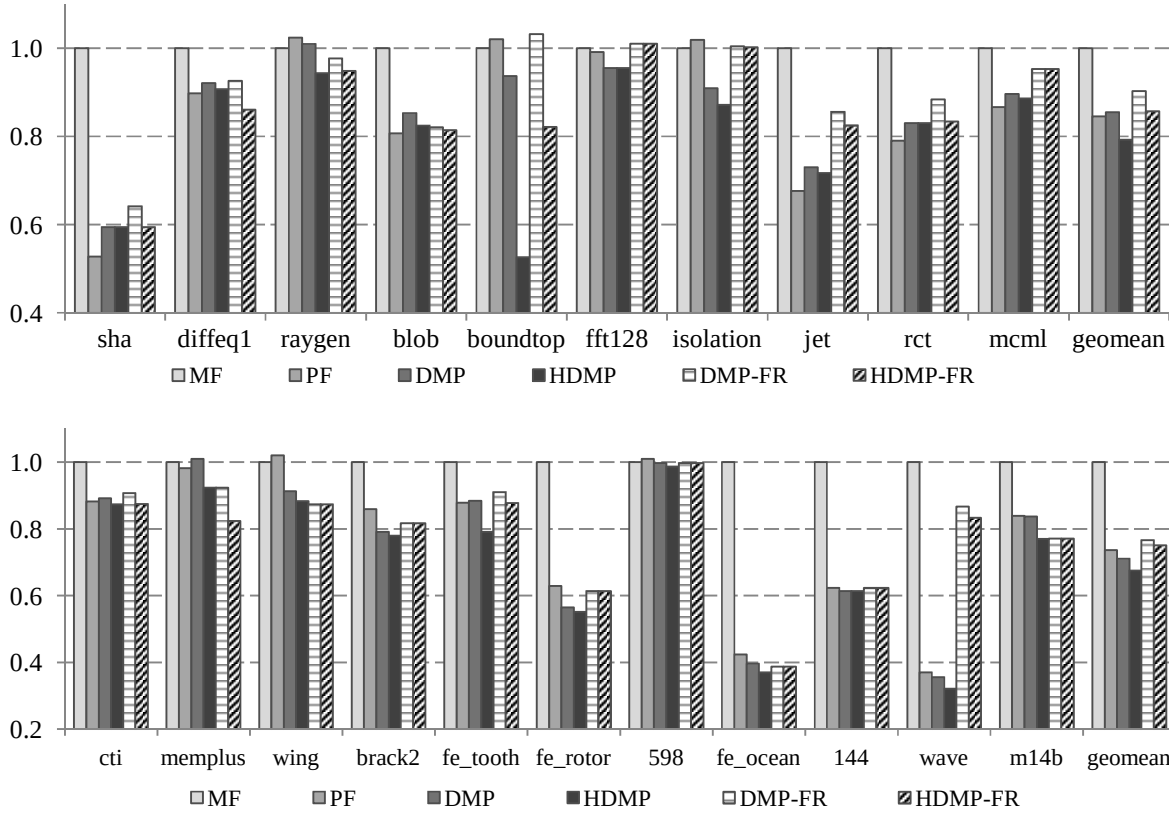

Figure 13. Cut size results for KLFM-based partitioners, normalized to map-first, and presented separately for HDL-derived graphs (top) and graphs from partitioning benchmark sets (bottom).

by the *MF* algorithm and the approach currently used for structural netlist partitioning in existing CAD tools) produced significantly worse cut size results than deferring personality assignment to after or during the partitioning process. HDMP produced the best cut size results of any algorithm.

4.10.5. RUR Deviation Results for KLFM-Based Algorithms

A comparison of the RUR deviation for the KLFM-based algorithms is shown in Figure 14. In contrast to the cut size results, waiting until after the partitions have been determined (the *PF* algorithm, and the approach currently used for functional netlist partitioning) produces far worse deviation from the desired ratio of resource utilization in the partitions than the algorithms that assign personalities prior to or during

partitioning. Incorporating resource ratios into the cost function in the '-FR' algorithm variants can significantly reduce RUR deviation.

4.10.6. Run-Time Cost

Dynamic personality mapping increases the number of computations that must be performed during KLFM. To evaluate the computational cost of each partitioning strategy, we measured their wall-clock run times when run on an Intel Core i7 processor using a single thread in isolation. The results were

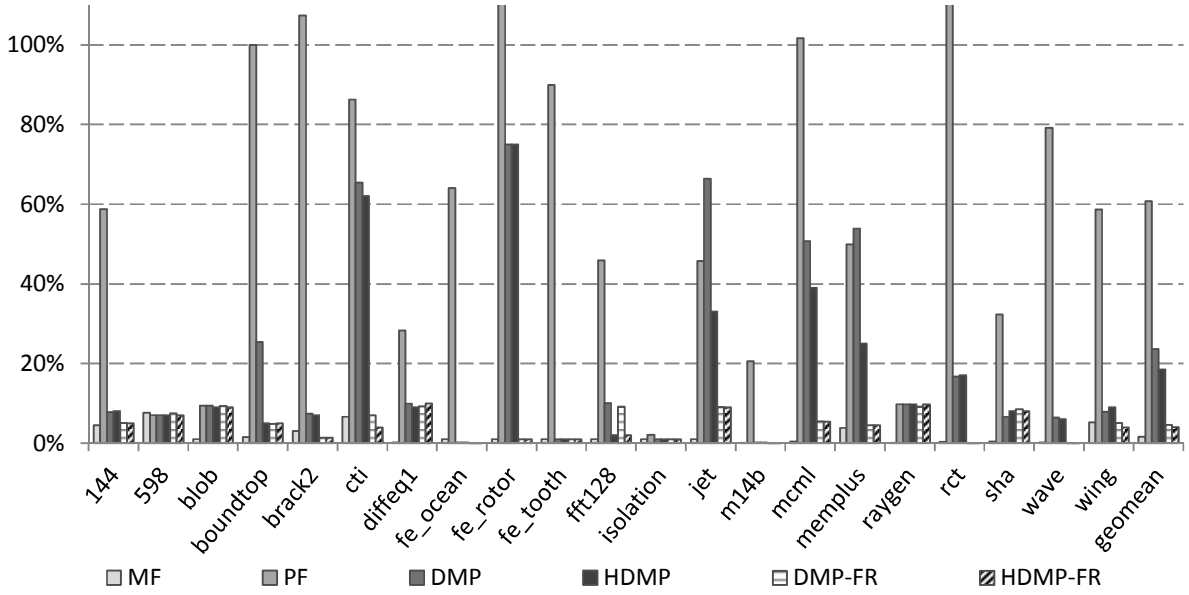

Figure 14. RUR deviation results for all benchmarks. Lower values are better.

normalized to that of MF and are given in Table VII.

PF is fastest, requiring 40% less run time than MF. Because it converts almost all nodes to a single resource, it uses simple gain buckets that only need to satisfy a single resource constraint when choosing the next node to move. The tradeoff for this speed is the worst RUR deviation of all tested algorithms. DMP and HDMP 30% and 70% slower respectively, but produce better RUR deviations and cut sizes. Adding Finegrained Ratio Control to the dynamic algorithms significantly improves RUR deviation without increasing run time cost. Absolute times for a 50-run benchmark set were on the order of 6 hours for the largest circuits using HDMP.

Table VII. Geometric means of the normalized cut size, absolute RUR deviation, and normalized wall-clock run times for the six KLFM partitioning strategies.

	MF	PF	DMP	HDMP	DMP-FR	HDMP-FR
Normalized Cut Size	1.00	0.79	0.78	0.73	0.83	0.80
RUR Deviation	1.6%	61%	24%	19%	4.6%	4.0%
Normalized Run Time	1.0	0.6	1.3	1.7	1.3	1.7

To assess the importance of run-time on solution quality, we measured the time needed for HDMP to complete 50 runs, then ran all the algorithms for that fixed time period. The extra time produced no significant improvement for the faster algorithms. In many partitioning scenarios, a moderate increase in run time is worthwhile if it provides significant improvements in cut size or resource utilization. However, if time budgets are tightly constrained, such as for a rapid design exploration process, then algorithm speed might motivate the use of DMP over HDMP.

4.10.7. Comparison of Pareto Optimality

The tradeoffs between cut size and RUR deviation when performing multivariate optimization are

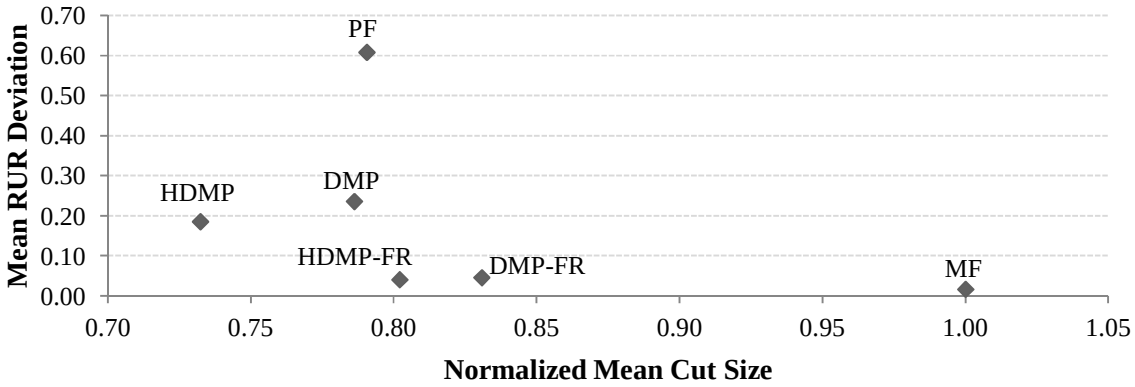

Figure 15. Pareto comparison of cut size vs. RUR deviation for all KLFM-based partitioners. Positions lower and further left on the chart represent better results.

highlighted in Figure 15, which highlights the Pareto optimality of our experimental algorithms. Compared to partitioning algorithms that use static personality assignment, the dynamic algorithms offer significant advantages. DMP and HDMP are strictly better than PF, providing marginally lower cut sizes and 70% lower RUR deviation. Although MF still achieves the lowest RUR deviation of any algorithm, DMP-FR and HDMP-FR are only slightly higher while also achieving up to a 20% lower cut size.

4.11. Performance Consistency of KLFM Algorithms

In addition to considering the mean performance of the KLFM algorithms across a set of diverse benchmarks, it is also interesting to examine the relative variability of their performance on individual

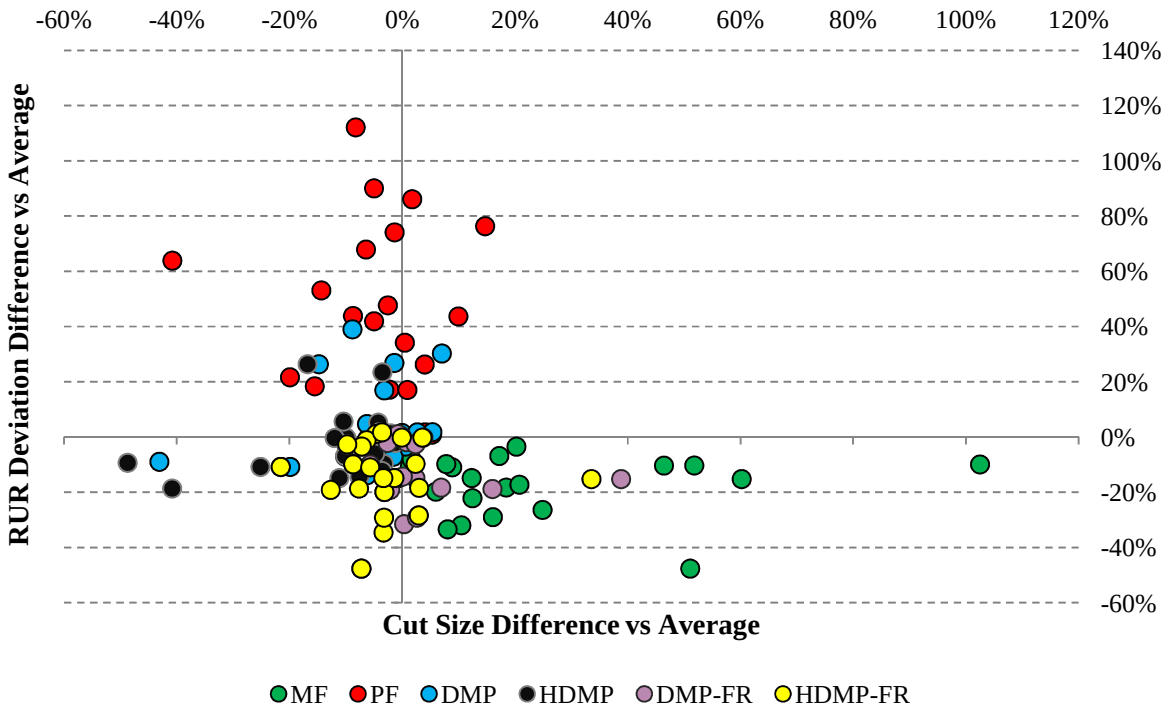

Figure 16. Performance consistency of the multi-personality KLFM algorithms. Each point represents a single benchmark and the algorithm's relative performance (in terms of RUR deviation and cut size) relative to the average result of all algorithms on that benchmark. For both performance metrics, lower (more negative) values are better.

benchmarks. For instance, if an algorithm consistently performs well relative to other algorithms, it may be appropriate for an end user CAD flow to use only that algorithm for all input circuits. On the other hand, if an algorithm performs very well for some inputs and very poorly for others, it may be more appropriate to use multiple algorithms and select the best result.

In Figure 16 we demonstrate the performance of each algorithm, as measured relative to the mean performance of all of the algorithms, plotted for all of the benchmarks. Lower values are better for both RUR deviation and cut size, therefore a data point at a negative percentage indicates that the algorithm out-performed the average for that benchmark. Thus an algorithm that consistently outperformed the

others in both performance metrics would have most of its points in the lower left quadrant of the graph. From this data we observe that HDMP and HDMP-FR exhibit the greatest consistency in performance, being near or better than the average in both cut size and RUR deviation for almost all of the benchmarks. In contrast, MF and PF both demonstrate consistency in one metric and a high degree of variability in the other. MF for example, outperforms the RUR deviation average on every benchmark, but its cut size results vary from being even with the average to being more than twice as high. Therefore, it is probably not appropriate to use MF as the sole partitioning algorithm in a CAD flow.

4.12. Contributions

Our work is the first to introduce and formally define a novel graph partitioning problem − multi-personality partitioning. Multi-personality partitioning allows us to model key properties of flexible-resource heterogeneous devices, such as modern FPGAs, that are not addressed in previously existing partitioning problems. We describe the new challenge of controlling the ratio of resource utilization within a partition and argue for a new resource utilization-based quality metric for partitioning heterogeneous systems with malleable personalities. In response to the new multi-personality partitioning problem, we provide the first optimal and heuristic-based algorithms for multi-personality partitioning, including the first formulation as an integer programming optimization problem, and demonstrate how to modify the widely used KLFM partitioning algorithm [61] to support multiple logic personalities. We show that custom heuristics like KLFM can offer significant performance advantages over generalized heuristics like ILP branch-and-bound, particularly as problem size increases. We experimentally evaluate several different algorithms for multi-personality partitioning, and demonstrate that dynamic, personality-aware algorithms can achieve an average of 27% improvement in partition cut size or a 57% decrease in deviation from partition resource utilization targets when compared to algorithms that perform partitioning and personality assignment separately. We demonstrate that integration of resource utilization heuristics into the cost function of KLFM as a multivariate optimization problem produces new points of Pareto optimality when compared to solely cut size-based heuristics [98] .

5. Information-Aware Partitioning

In this chapter, we discuss information-aware partitioning a new paradigm for circuit partitioning that leverages concepts from information theory in order to minimize the amount of information transmitted between partitions rather than the number of wires that span partitions.

After presenting a motivation for this work in Section 5.1, we provide an overview of work related to this area in Section 5.2 and a conceptual introduction to the application of concepts from information theory to digital circuits in Section 5.3. To illustrate the benefits of information-aware partitioning, we begin in Section 5.4, where we perform a proof-of-concept analysis based on the firmware we designed for the CMS Regional Calorimeter Trigger. Later, in Sections 5.5 and 5.6, we introduce a broader set of benchmarks circuits and associated data sets that we will use to evaluate information-aware partitioning, and present entropy-based partitioning results for these benchmarks. In Section 5.7, we discuss the challenges of gathering entropy data for large and complex designs, introduce several techniques for measuring and estimating the entropy of signals within circuits and compare their efficacy, cost, and design impact. Finally, in Section 5.8 we examine some of the practical challenges of exploiting low entropy partitions in the design of the CML Level-1 Trigger Emulator, a system for validating some of the high energy physics systems described in this document.

5.1. Motivation

Traditionally, circuit partitioning algorithms − including the multi-personality partitioning algorithm we described in Chapter 4 − have tried to optimize their partitioning solutions by reducing the number of wires that cross between partitions. Minimizing the wire cut size makes sense when the partitions represent chips that are connected using a large number of parallel I/O pins; reducing the wire cut size in that context may simplify board-level routing or increase performance if pins are time-multiplexed [101]. However, as data rates and inter-partition communication have increased, multi-chip designs are increasingly moving away from using a large number of slow parallel I/O connections and instead are

using a smaller number of high-speed serial I/O connections [9]. This is also true for the CMS Regional Calorimeter we described in Chapter 3, which uses multi-gigabit fiber optic links connected to FPGAs [50]. Instead of routing inter-partition signals via many parallel wires, these signals are buffered for transmission, serialized, sent across the high-speed links (sometimes using shared networks), and then buffered and de-serialized on the receiving end. Within this context, the cost of communication may be viewed in terms of factors such as the bandwidth required by the serial link, the total transmission time, and the injection rate into the network. Reducing the amount of data that must be transmitted can reduce transmission time (potentially improving performance) or allow us to use lower-bandwidth networks that are simpler, cheaper, and more reliable. For instance, IBM's recent Twinstar logic emulator, which uses serial communication between FPGAs, observed up to a 5X improvement in emulation speed based on reducing cut size [102].

Although there is some relationship between the number of signals that span between partitions and the amount of data that is transmitted, these metrics are not interchangeable. In particular, individual signals may have different entropy − a measure of the amount of information they carry − based on the structure of the circuit and the type of data it processes. In particular, if we produce partitions with low entropy, it may be possible to compress the cut set information to reduce the amount of data that must be transmitted. That is, if we treat the current values of all of the signals in the cut set as a data set that must be transmitted to the other partition (we will refer to this data set as a 'frame'), then we may be able to compress the frame prior to transmitting it and decompress the frame after it has been received in order to decrease transmission time and/or the required communication bandwidth. Because some popular stream compression methods (e.g. Huffman coding [103] and arithmetic coding [104]) have compression ratios that are based on entropy, reducing the entropy of the frames may improve the opportunity for compression.

Entropy-based signal coding has been successfully employed in many application-specific systems [105] [106] [107], including high energy physics [108]. However, identifying such opportunities during the design process, and partitioning a design in an efficient way to take advantage of those opportunities,

can pose a significant challenge. To understand patterns or redundancy in application-specific data, the circuit designer must possess expert knowledge in the application domain. Even then, it may be difficult to exploit this knowledge when using design automation techniques such as high-level synthesis [109].

In this chapter, we investigate automated algorithms for minimizing information cut sizes in large circuits, in order to create opportunities for entropy-based optimization. We will discuss further motivation for entropy-based partitioning that was inspired by our work on the design of the CMS Regional Calorimeter Trigger in Section 5.4, but first we will introduced related work in this area and discuss of the fundamentals of information theory as it applies to digital circuits.

5.2. Related Work

Information-aware Partitioning combines the concepts we developed in our earlier graph partitioning work with research related to Information Theory and its applications to circuit design. A wealth of research has focused on various optimization methods for partitioning based on wire cut size, as discussed previously in Section 4.4. For an introduction to the concept of information entropy that will be used throughout this chapter, refer back to our introduction to information theoretical concepts in Section 2.5.

A component of our research focuses on the distribution of entropy in wires within a circuit. Landauer was among the first to adapt theories about entropy production and stability originally developed by Prigogine and Glansdorff for the statistical mechanics domain [110] [111] and show that they could be applied to the domain of electrical circuits [112]. This also built on Landauer's own earlier work in establishing a relationship between logically reversible computations in digital circuits and reversible reactions in thermodynamics [113]. Macii and Poncino defined a formalized method for computing the entropy of signals in combinational logic defined at the RTL level using arithmetic decision diagrams [114]. We will implement an analogous entropy computation method as part of our Structural Entropy Estimation method in Section 5.7.2. Cheng and Agrawal found a relationship between the complexity and logic depth of Boolean functions (a concept they referred to as "Computational Work") and the entropy of their outputs for randomly-generated multi-output functions [115]. They also demonstrated that assigning

specific values for "Don't Care" inputs to a function could reduce the entropy of its outputs. We make use of this in our work when computing entropy values in our circuits.

In part, our analysis will focus on the opportunity for designs to exploit low entropy in inter-partition signals to optimize inter-partition communication. Designers commonly use domain-specific knowledge about data entropy to include entropy-based data compression when creating accelerators for high-throughput computing. For instance, Microsoft's Catapult leverages the frequency of search terms in website scoring [116] and CMS's RCT takes advantage of the low probability of concurrent high-energy collisions in particle triggering [46].

Compression methods have also been proposed for communicating between logic partitions on the same chip with on-chip serial buses; Morganshtern proposed a compression scheme for serial links that takes advantage of leading zero suppression [117]. These compression methods currently rely on circuit designers to manually identify signals they believe will have low entropy based on their domain knowledge and manually partition the circuits to ensure that inter-partition signals are compressible. Deployment of automated partitioning that takes advantage of entropy-based compression in application-specific circuit design is limited by two important factors. First, in order to achieve good compression ratios in entropy-based coding schemes such as Huffman coding or arithmetic coding, it is necessary to know the symbol probabilities (the likelihood that different data values will occur) [104]. Although, it is possible to update the probabilities dynamically [118], choosing a low-entropy partition requires *a priori* knowledge of the relative entropies of the wires in the circuit. Second, once we have entropy information, we need partitioning algorithms that can minimize the compressed information size between partitions. We propose solutions for both of these problems in this work.

Although we use compression as a motivating example, there are other optimization approaches for inter-partition communication that can also take advantage of low-entropy signals. Rather than compressing a low entropy signal, we can inject error correcting codes for more reliable transmission [119]. Benini noted that spatial locality in memory traffic creates predictable data relationships on processor address buses, and that coding schemes which reduced bit transitions for such patterns could

reduce power dissipation [120]. Yang observed that some data words occurred more frequently on processors' parallel data buses and could similarly be coded to reduce transitions [121]. Lee adapted these schemes for transmission on high-speed, on-chip serial buses [122] based on bit transitions.

Some potential applications of content-aware signal manipulation in circuits have previously been limited by the potential increase in latency that might be introduced. Traditionally, circuit partitions communicated primarily through a large number of parallel links, where the latency of transmitting a cut set frame was only limited by the parallel I/O multiplexing ratio [123]. This latency could be as low as a single cycle. Recently, inter-partition I/O is transitioning from the use of large parallel buses to multi-gigabit serial transceivers (see Section 2.4). This trend is particularly prominent on FPGAs, where high-speed serial links have seen a 20X increase in total serial bandwidth from 2007 to 2013 [124] [125], and are becoming a much more attractive option for routing inter-chip signals as the cost and complexity of increasing the number of parallel links becomes prohibitive [56]. Serial communication necessitates an increase in latency, due to the requirement that data must be serialized and de-serialized. Additional latency overhead is introduced for synchronization and line coding, resulting in minimum latencies of 10+ cycles on both the transmitting and receiving sides for current implementations on FPGAs [126]. This added latency provides an opportunity to hide the cost of additional signal processing operations, which might otherwise negatively impact performance when using parallel I/Os.

Other researchers have proposed using entropy in the circuit design process for reasons other than optimizing communication. Hwang and Wu developed an entropy-based approach for high-level power estimation of combinational logic based on Boolean functions [127]. Marculescu similarly developed entropy-based models for power [128] and energy [129] estimation at the circuit RTL level. Marculescu also considered the impact that highly-correlated input streams have on power consumption, and studied the problem of conditional dependence between signals in a circuit when estimating entropy [130]. We will revisit this problem when we discuss our approaches for entropy estimation in Section 5.7. Agarwal proposed a relationship between entropy and the difficulty of digital fault testing, based on the concept that the change in entropy from inputs to outputs is a measure of the complexity of a circuit [131].

5.3. Entropy in Digital Logic Circuits

Within the field of information theory, the amount of information in a signal is quantified as its entropy (see Section 2.5 for more background discussion on the concept of entropy for digital signals; we will expand on how entropy is applied to digital logic circuits in this section). There are many different ways to compute entropy for digital signals. When we refer to entropy in this chapter, we will specifically refer to Shannon Entropy, which is computed based on the probability that a signal will take each of its possible values (also referred to as 'symbols'). The maximum entropy of 1 bit per wire occurs when all possible symbols are equally likely. In other words, at maximum entropy, the wire width and entropy of a signal are equal.

In this research, we develop partitioning methods that use entropy information to optimize the amount of information that must be transferred between partitions. If all of the signals in the circuit had maximal entropy (where the entropy of each wire equals 1 bit), this would be no different from traditional partitioning that minimizes the wire cut size. Therefore, entropy-based partitioning is only of use if at least some of the internal signals within a circuit have lower entropy than others. In this section, we will describe several major causes of entropy reduction for signals within circuits.

5.3.1. Low-Entropy Input Data

If the entropy of the internal signals in a circuit is a measure of the amount of information passing through the circuit, then intuitively, the entropy of the internal signals may be related to the amount of data entering the circuit − that is, the entropy of the inputs. A circuit's inputs may have low entropy for a variety of reasons. For applications that process inputs coming from sensor data, like our high energy physics circuits, not all sensors may produce data at the same time. For applications that process multimedia inputs, like audio, video, still images, or text, low entropy may result from natural redundancies in the input. Such inputs often exhibit spatial or temporal correlations. For instance, one frame of video is often similar to the next—as are adjacent pixels within a single frame. Inputs may also

not use all of their dynamic range, and some input values may be much more common than others; Shannon famously calculated that English-language text (circa 1950) had approximately 50% redundancy [132].

5.3.2. Non-Reversible Computations

Any logical operation that is irreversible − that is, any operation where it is not possible to compute the values of the input when given only a description of the operation and the output values − results in a decrease in entropy [133]. A simple example of an irreversible logical operation is a 2-input OR function. Given that the output is '0', we can deduce that the inputs were both '0'. However, if the output is '1', we cannot determine the value of the inputs, because there are three possible input combinations that could result in that output. A simple example of a reversible computation is the NOT function. Given the function's output value, we can always determine its input. We will discuss a method for computing the effect of irreversible computations on entropy later in Section 5.7.2 when we discuss entropy estimation based on a circuit's structure.

There are active research efforts focused on synthesizing circuits that primarily use reversible logic [134] [135] [136], but that work is primarily focused on quantum computing, and is not a focus of commercial circuit synthesis tools for traditional computing devices. Furthermore, since our goal is to make use of lower entropy to improve information cut size, we focus instead on irreversible logic.

5.3.3. Redundancy / Reduction in Boolean Complexity

When considering larger logic operations within a circuit, there is some evidence that there is a relationship between the complexity of multi-input / multi-output Boolean functions and the difference in entropy between their inputs and outputs. Cook and Flynn hypothesized that the complexity of synthesizing logical networks could be directly related to the entropy of their outputs [137]. Later, this concept was modified to describe a relationship between the "computational work" performed by a Boolean function in terms of the entropy change between its inputs and its outputs and the number of

logic gates and gate levels in the synthesized circuit based on experimental synthesis of random functions [115].

Viewed from the opposite perspective, circuit transformations performed at either the design or logic synthesis stages may reduce the entropy of a logic block's internal signals. For instance, logic and signals may be replicated in order to decrease critical paths [138], reduce power [139], improve fault tolerance [140], or make the circuit easier to test [141]. Computational redundancy (and the associated reduction in the entropy of outputs) may also simply result from a lack of complete optimization during design and synthesis.

5.4. Entropy-Based Partitioning in the Regional Calorimeter Trigger

Our work designing the high-luminosity upgrade for the CMS experiment's Regional Calorimeter Trigger (RCT) system (see Chapter 3) provided an early motivating application for our investigation into partitioning circuits based upon the data carried by their internal signals. One of the important challenges we identified in the design of high energy physics systems is the large amount of information that must be shared between chips. Because the physics algorithms have very high throughput requirements and cannot fit on a single device, they are partitioned across multiple chips. Sharing this information requires a large number of multi-gigabit, inter-chip communication links that present significant implementation challenges [142] [143]. As described in Section 3.3.2, the RCT system receives inputs from over 4000 sensor regions; however the hadron collisions in the experiment typically only produce around 20 sub-particles per input data set [5]. Each particle contributes data to 2-6 sensor regions [42]. From this, one can deduce that on average only 1-2% of the input sensors will be contributing meaningful data to the RCT algorithms. Moreover, the RCT includes computations to filter noise and quantize data. Based on this knowledge, one may hypothesize that the signals within the RCT are information-sparse. Specifically, we expect that at any given time, a large number of the signals in the RCT's datapath will be carrying values that are equal to or close to zero. Another way of viewing this is that each individual signal in the datapath only spends a small fraction of its time with non-zero values. This can be experimentally

quantified by simulating the RCT circuit using representative inputs and calculating the Shannon entropy of each signal. We synthesized a subsection of the RCT for a 15x15 sensor grid on a Virtex-7 FPGA and simulated the synthesized netlist using input data generated by the CMS experiment's physics emulator

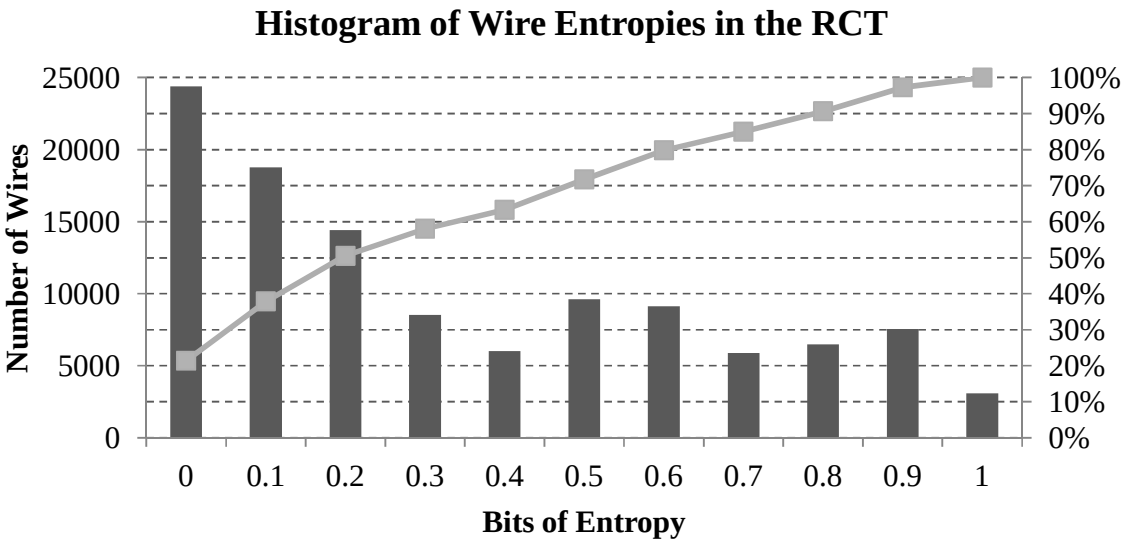

Figure 17. Histogram of the bits of Shannon entropy per wire in a subsection of the RCT, measured during post-synthesis simulation. The line represents the cumulative percentage of wires (as labeled on the right axis) that have at least the amount of entropy indicated on the y axis.

[48]. Since the RCT uses the same algorithms for all sensor regions in the detector barrel, the entropy measurements should be representative of the full system. A histogram of the entropy per individual wire in the circuit is shown in Figure 17.

The entropy measurements confirm the hypothesis about the information sparseness of the RCT. A wire that fully utilizes its information carrying capability would have an entropy of one bit. Less than 5% of the wires in the RCT had entropies in the range of 0.9-1.0 bits, half of the wires had entropies below 0.3 bits per wire, and more than one fifth of the wires had below 0.1 bits of entropy. When viewing these data points purely from a standpoint of information efficiency, the wires in the RCT appear to be poorly utilized. Given that the RCT's high degree of interconnection and communication costs were one of the major challenges in the design of the original system, we were motivated to consider whether we could leverage the low entropy of RCT's internal signals to reduce the amount of intercommunication needed in the RCT hardware. Ideally, our goal would be to use resource sharing to increase the information

utilization of each wire while decreasing the number of wires. Doing so within an individual FPGA presents significant challenges; the amount of information on a given wire cannot be predicted at any one time and the resource overheads for internal routing and switching of signals could be significant enough to negate the savings in reducing the number of wires in the original datapath. Exploiting entropy at the

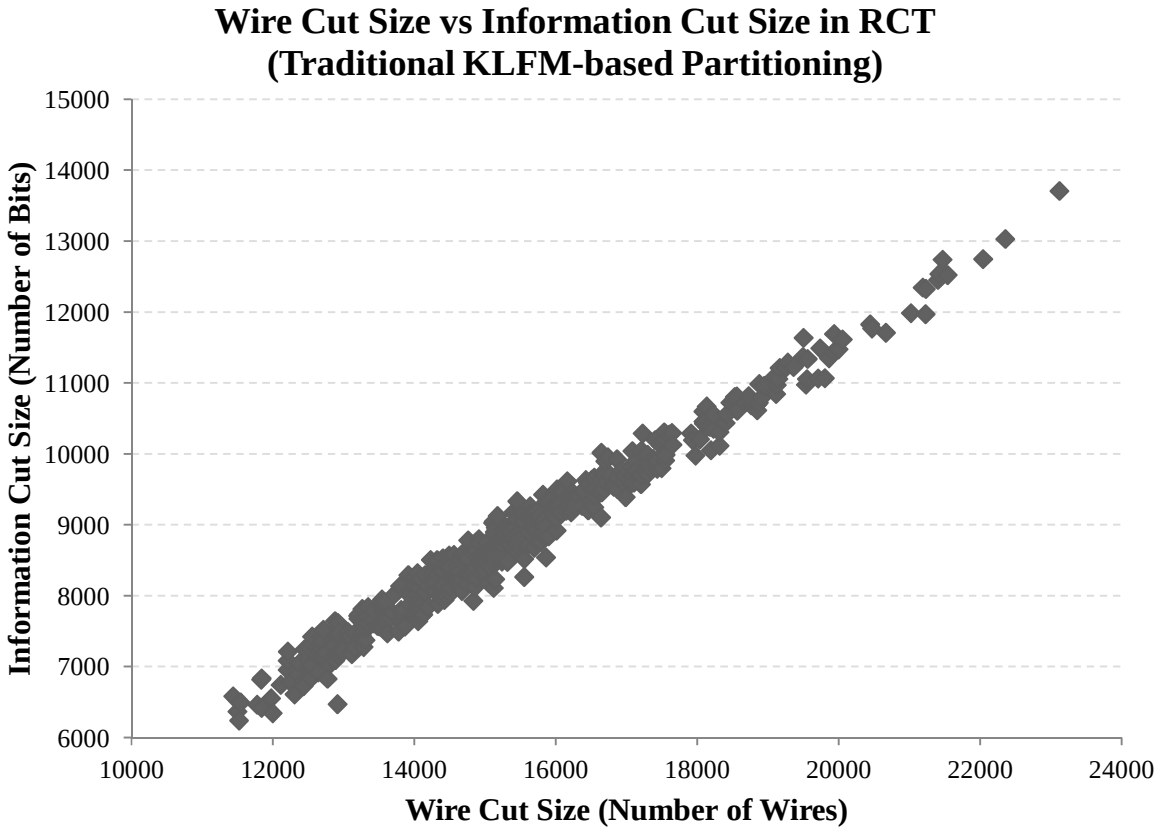

Figure 18. Plot of wire cut size versus information cut size in the RCT for 500 runs of a KLFM algorithm using a traditional wire-minimization objective function.

chip boundaries − that is, the cut sets of our logic partitions − provides a much more attractive option. Signals must already be routed and multiplexed at the chip boundary in order to make use of the FPGA's high speed serial links, and the buffering and queuing of data necessary for serialization lends the opportunity to hide the latency overhead of entropy-based coding. Research into entropy-based compression of physics data using results from the TPC trigger used in the NA49 and ALICE experiments have shown compression factors of 2-3X are possible using arithmetic and Huffman coding [144], with a hardware overhead equivalent to 0.05% of the logic resources on a Virtex-7 2000T FPGA

[145]. Compressing the data transmitted between partitions could meaningfully reduce the cost and complexity of the RCT design.

Thus far, we have only presented data on the entropy of the signals in the RCT as a whole. Although it may be tempting to conclude that the presence of a large number of low-entropy signals in the overall circuit means that our circuit partitions will be similarly low entropy, this is not necessarily the case. Partitioning algorithms attempt to balance the logic in partitions while minimizing the number of cut wires, and it is possible that the wires forming minimal cuts tend to have a different entropy distribution than the overall circuit. Significantly, the majority of logic operations in the constituent circuits are non-reversible, and Landauer's Principle of computational thermodynamics dictates that entropy therefore decreases within a circuit based on logic depth [112] [146] [147]. From this, we might hypothesize that lower entropy signals would be located on the periphery of a circuit, closer to the outputs.

To better understand the relationship between partition cuts and signal entropy, we partitioned the RCT circuit using a KLFM-based partitioner (see Section 4.8). We ran the partitioning algorithm 500 times using random starting points. We then took the entropy data we collected earlier and assigned it to the cut sets produced by the partitioning algorithm. We computed an “information cut size” − the joint Shannon entropy of the cut set − by summing the individual wire entropies in the cut set. By summing the entropy of individual wires to produce the joint entropy, we are making the simplifying assumption that each wire in the cut set can be treated as independent from one another. This will tend to overestimate the information cut size (refer back to Section 2.5 for more details), but makes the makes the problem computationally tractable. We plotted the wire cut size (the number of wires in the cut set) versus the information cut size. This data is given in Figure 18. We observe a high degree of linearity between the wire and information cut sizes. There is surprisingly little variation in the entropy per wire between the best and worst results in terms of wire cut size; the standard deviation is only 2.3%. The cut set shows a much lower information cut size than wire cut size, but it is more information-dense than the circuit as a whole. The average entropy of wires in the cut sets is 0.57 bits per wire − nearly double the 0.30 bits per wire average in the entire circuit. This suggests that partitioning using an algorithm with a traditional

objective function based on wire count consistently selects wires with above average entropy in the RCT circuit. It also provokes an interesting question: Could an algorithm that favors cutting low-entropy signals produce a balanced partitioning result with a lower information cut size at the expense of minimizing the raw wire count?

To answer this question, we modified the KLFM's objective function to use the information cut size rather than wire cut size as the partitioning cost. We then ran the new algorithm 500 times and compared the 10 best results produced by the algorithm with the entropy-based cost function to those produced using the traditional wire-based cost function. The results are shown in Figure 19. Since our method for

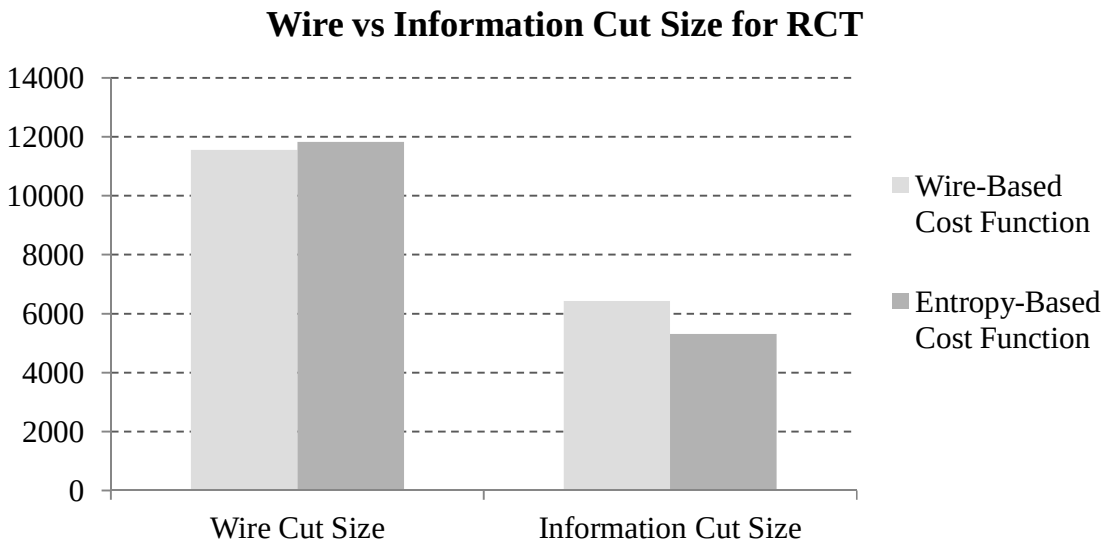

Figure 19. Wire cut size vs. Information cut size in the RCT when using KLFM algorithms with Wire-based and Entropy-based cost functions.

computing the joint entropy will tend to overestimate the information cut sizes, these should be viewed as conservative results. We see that for the RCT, using an entropy-based cost function during partitioning is able to reduce the information cut size by 20% compared to the wire-based cost function. Moreover, it only incurs a 2% overhead in wire cut size. This reflects a reduction in the average entropy per wire in the cut sets from 0.57 bits to 0.43 bits. The decrease in entropy may be significant if we wish to perform optimizations to our inter-partition communication. For compression, Shannon's source coding theorem

[12] allows us to compute an upper bound on the efficiency of per-symbol lossless compression using entropy coding if we assume that wire values can be treated as independent random variables. The

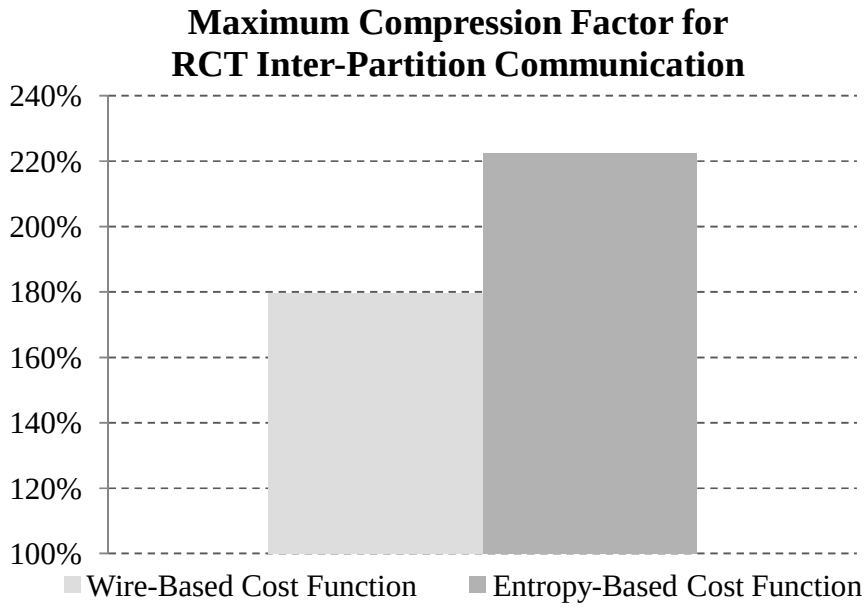

Figure 20. The maximum lossless compression factor achievable for compression using entropy coding on inter-partition cuts produced by wire-based partitioning and entropy-based partitioning.

maximum possible compression factors for the cuts obtained from wire-based and entropy-based partitioning are compared in Figure 20. Entropy-based partitioning allows us to increase the maximum compression by 40% over wire-based partitioning and the 2.2X compression factor over the wire cut size presents compelling evidence for the value of implementing compression using entropy coding in the RCT. We will revisit discussion of the practical issues of using entropy-based partitioning to implement compression for the CMS L1 Trigger Emulator in Section 5.8.

We have established that information-based partitioning is worthwhile for partitioning the *rct* circuit. In the remainder of this chapter, we will demonstrate that information-based partitioning can be applied more broadly to a diverse set of large-scale heterogeneous circuits and show that it provides tangible benefits in real applications.

5.5. Benchmarks for Information Partitioning

In order to determine the information cut size of a circuit, it is necessary to be able to observe value changes over time for all of the signals in a circuit. We will examine options for relaxing this constraint through estimation techniques later in Section 5.7; however, exact measurements of signal entropy will still be needed to evaluate the accuracy of estimation. We gather this data by synthesizing HDL descriptions of our benchmark circuits and performing cycle-accurate simulation of the resulting netlists. This simulation processes introduces several additional requirements for our benchmark circuits compared to those used in creating the benchmarks graphs in our Multi-Personality Partitioning research described in Section 4.5:

- HDL circuit descriptions must be available for all benchmarks.
- Benchmarks must be synthesizable and functionally verifiable in post-synthesis simulation.
- Input data sets must be available that accurately represent the input data that would be used in normal use of the circuit.

Like the MPP benchmarks discussed in Section 4.5, we also favored using large circuits that are publicly available. However, because this research does not focus on personality mapping, we did not require that the circuits be capable of using multiple types of resources on FPGAs. Because of the new constraints on our benchmarks, only three of the benchmarks used in our MPP research (jet, isolation, and rct − circuits developed as part of the high-energy physics research described in Chapter 3) were appropriate for use in this research. We were also unable to directly use other pre-existing circuit benchmark suites created for CAD research such as VPR [148], ISPD [149] [150] [151], DAC [152], RAW [101], or Titan [153] as they do not include input data with their benchmarks. Moreover, they often do not contain any functional documentation of the circuits, making it difficult even to construct a representative input data set. In general, we observe that previous CAD research in partitioning, floorplanning, placement, and routing has rarely considered optimizations based on signal content and this is reflected in the difficulty in accurately simulating the available benchmarks. Research within the

field of automated testing requires circuit simulation, producing benchmark suites such as ACM/SIGDA [154], ITC [155], and FITTest [156], but these suites are targeted at minimizing input patterns for defect coverage rather than inputs representative of real world use.

Lacking a suitable existing benchmark suite, we compiled our own set of benchmark circuits, using circuits from other suites and publicly-available circuit designs. In some cases, we modified the circuits to make them synthesizable, created simulation test fixtures for them, and compiled input data sets. The benchmarks and their associated data sets are described in the following section.

5.5.1. Benchmarks and Data Sets

The benchmark set used for our Information-Aware Partitioning experiments is summarized in Table VIII; detailed descriptions of each benchmark follow. The resource costs for each benchmark (in LUTs and flip flops) are based on synthesis for the Xilinx Virtex-7 FPGA using the Vivado 2014 synthesis engine.

Table VIII. Summary of benchmark circuits used in Information-Aware Partitioning.

Name	LUTs	Flip Flops	Description
adpcm_dec	195	58	Audio Decoding
adpcm_enc	303	85	Audio Encoding
bitcoin1	57,928	55,906	Cryptocurrency
bitcoin4	36,936	26,086	Cryptocurrency
gsm_switch	x	x	Networking
isolation	4,251	288	High-Energy Physics
jet	19,245	7,745	High-Energy Physics
mkjpeg	42,732	3,774	Image Encoding
orsoc_gfx	40,988	48,993	3D Graphics Rendering
rct	33,079	29,702	High-Energy Physics
stereo_vision	21,562	16,657	3D Vision
sudoku	14,535	5,851	Artificial Intelligence
warp	4,551	3,588	3D Texturing

adpcm_dec, adpcm_enc

An audio-processing circuit that performs encoding/decoding of audio files encoded using Adaptive Differential Pulse Code Modulation (ADPCM) based on the circuit implementation by Moti Litochevski

available from OpenCores [157]. The input data set includes nine public domain classical music recordings from OnClassical [158].

bitcoin1, bitcoin4

An open-source circuit implementation [159] for mining Bitcoin [160], a popular cryptocurrency [161], that is targeted at the Altera and Xilinx FPGA platforms. The algorithm relies on repeated computations of secure hash codes based on the SHA-256 algorithm. The implementation can be customized, and we use two different configurations: *bitcoin1* uses a fully-unrolled hashing circuit whereas *bitcoin4* reuses some logic and thus requires 1/4 as much hashing logic as bitcoin4. The bitcoin circuits only require a 256-bit seed value as input; we use the default seed provided in the included testbench.

gsm_switch

A high-radix, 10-gigabit on-chip networking fabric for routing and switching of GSM [162] packets for mobile communications included with the Titan23 benchmark suite [153]. We simulate the circuit using a synthetically-generated traffic pattern consisting of 10 million packet transmissions based on the traffic generator included with the benchmark.

isolation

A high-energy physics circuit used to determine whether energy deposits associated with reconstructed electromagnetic or tau particles are positionally isolated from other particles produced by a hadron collision under high pile-up conditions [43]. Designed for the high-luminosity upgrade of the Large Hadron Collider particle accelerator. The input data set consists of 10,000 test patterns produced by the CMS SLHC Calorimeter Trigger emulator [48].

jet

A high-energy physics circuit used to reconstruct multi-particle jets and assign particles to jets in order to create energy maxima. Designed for the high-luminosity upgrade of the Large Hadron Collider. Uses the same input data set as the *isolation* benchmark.

mkjpeg

An open source circuit for encoding and manipulating digital images in the JPEG format [163] implemented by Michal Krepa and available from OpenCores [164]. The input data set consisted of the top 50 photographs available from photo-sharing website Flickr [165] on December 15th, 2014. The images were converted to the bitmap format with a resolution of 640x480 pixels and a 24-bit color depth.

orsoc_gfx

An open-source graphics core included as part of an embedded system-on-chip design from the OpenRISC project [166] and targeted at FPGAs. We use the included GFXBench validation benchmark as our input data set.

rct

A high-energy physics circuit implementing the high-luminosity Regional Calorimeter Trigger, as described in Chapter 3. We use the same input data source as in *isolation* and *jet*.

stereo_vision

An open-source 3D vision compositing, rectification, correspondence, and post-processing framework designed for FPGAs by Dan Strother [167]. The input data set consists of 30 images and disparity maps from the Middlebury Stereo 2005 & 2006 Datasets for stereo vision research [168] [169] [170].

sudoku

An open-source circuit for solving puzzles from the game Sudoku, designed for FPGAs by Altera [171]. The input data set consists of the first 5000 puzzles from the included puzzle bank.

warp

An open-source circuit for mapping textures in an audio visualizer designed by Sebastien Bourdeauducq and available from OpenCores [172]. The input data set consists of the same images used with *mkjpeg*.

5.6. Entropy-Aware Partitioning Results for All Benchmarks

Using the benchmarks described in the previous section, we will evaluate whether the initial results gathered for the *rct* circuit when developing our entropy-aware partitioning process are representative of what can be achieved with a broader sample of circuits. We are also interested in trying to obtain insight into what features of a circuit may lead it to benefiting from entropy-aware partitioning over wire-based partitioning algorithms.

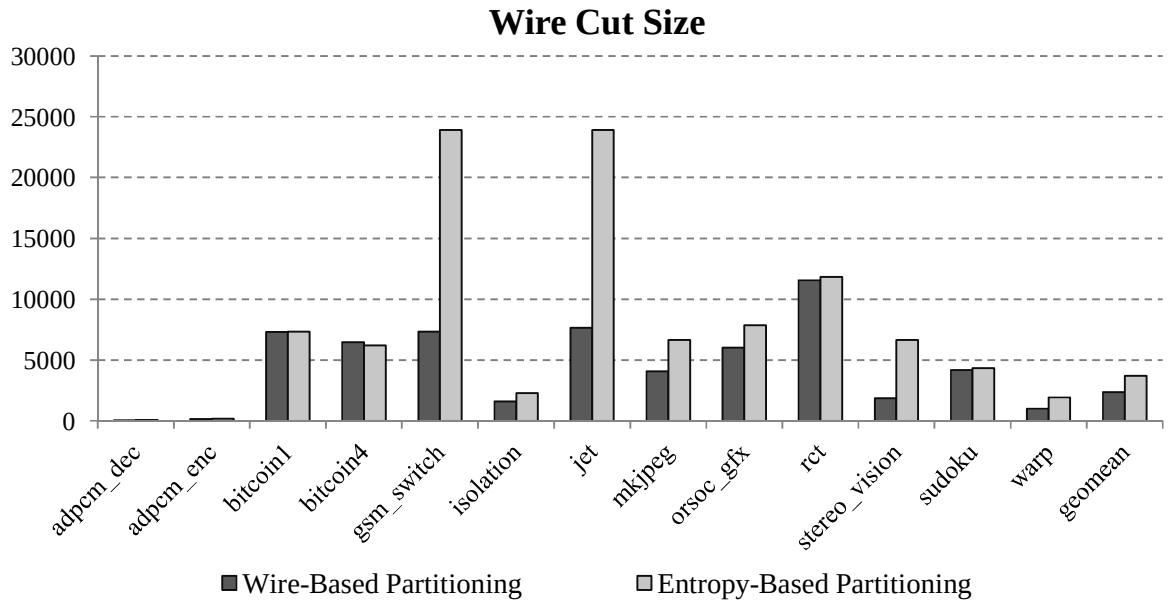

Figure 21. Absolute wire cut sizes using wire-based and entropy-based partitioning.

5.6.1. Wire Cut Size Results

We begin by examining how both algorithms perform when considering the traditional wire cut size. Intuitively, we would expect the wire-based algorithm to perform better in terms of wire cut size, as that

is its optimization goal. Ideally, an entropy-based algorithm would still perform well in wire cut size, as this would allow a single algorithm to be used in the CAD flow without needing to consider in advance whether the circuit will use entropy coding on its inter-partition communication. Absolute wire cut sizes and normalized wire cut sizes are shown in Figure 21 and Figure 22.

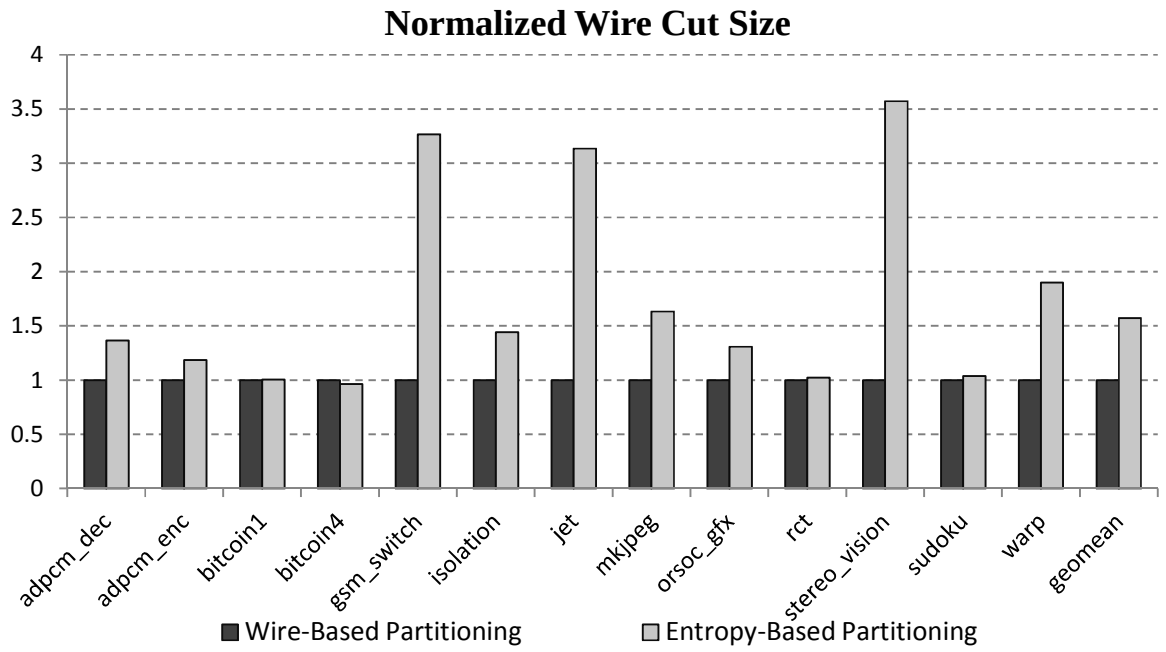

Figure 22. Normalized wire cut sizes using wire-based and entropy-based partitioning.

From the absolute cut sizes, we see that most of the benchmarks have very high amounts of interconnection, with only *adpcm* having a cut size less than 1000 wires, and two of the three high-energy physics benchmarks (*jet* and *rct*) are among the benchmarks with the most inter-partition connections. We showed previously that *rct* showed a negligible overhead in wire cut size when using the entropy-based partitioning cost function. This observation does not, however, hold true across all benchmarks. Although *adpcm_enc*, *bitcoin1*, *bitcoin4*, and *sudoku* also show little increase in wire cut size, the geometric mean of all benchmark results is 50% higher for the entropy-based algorithm compared to the wire-based algorithm. A few benchmarks (*gsm_switch*, *jet*, and *stereo_vision*) exhibit very high wire cut sizes when using entropy-based partitioning − 3X the result obtained with wire-based partitioning. Later in our

results when we further examine the type of wires included in their cuts we will gain greater insight into why some benchmarks exhibit such significant difference in wire cut size.

5.6.2. Information Cut Size Results

Next, we consider the difference in information cut sizes, which served as our motivation for investigating entropy-based partitioning. The absolute and normalized information cut size results are given in Figure 23 and Figure 24.

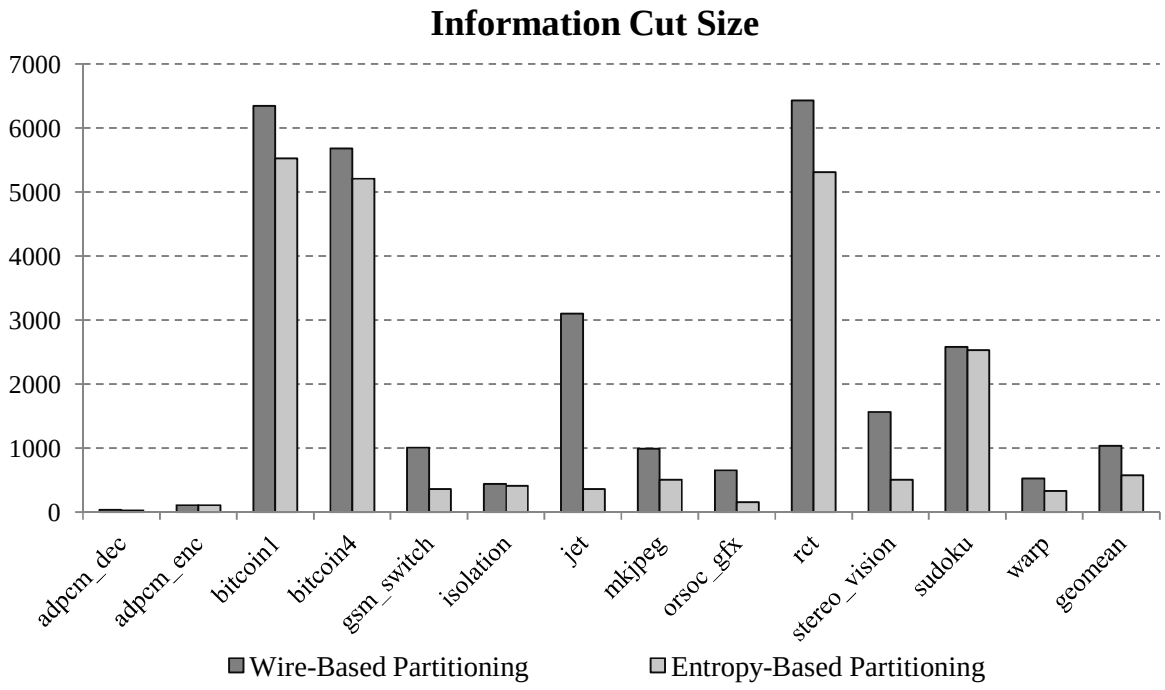

Figure 23. Absolute information cut sizes for wire-based and entropy-based partitioning.

rct exhibited good opportunities for compression in our initial experiments in Section 5.4; it also has one of the highest information cut sizes among our benchmarks. This suggests that there are even greater opportunities for entropy coding in some of our other benchmarks; we quantitatively evaluate this in Section 5.6.4. The other high information cut sizes come from our *bitcoin* benchmarks. We will discuss the reason for this further when we examine the information density of the wires in the benchmark circuits.

The normalized results demonstrate that the three benchmarks which performed poorly in terms of their wire cut size when using the entropy-aware algorithm (*gsm_switch*, *jet*, and *stereo_vision*) are among the benchmarks that show the greatest advantage in terms of the information cut size that can be obtained with the entropy-aware algorithm. From this, we may deduce that these circuits have a large

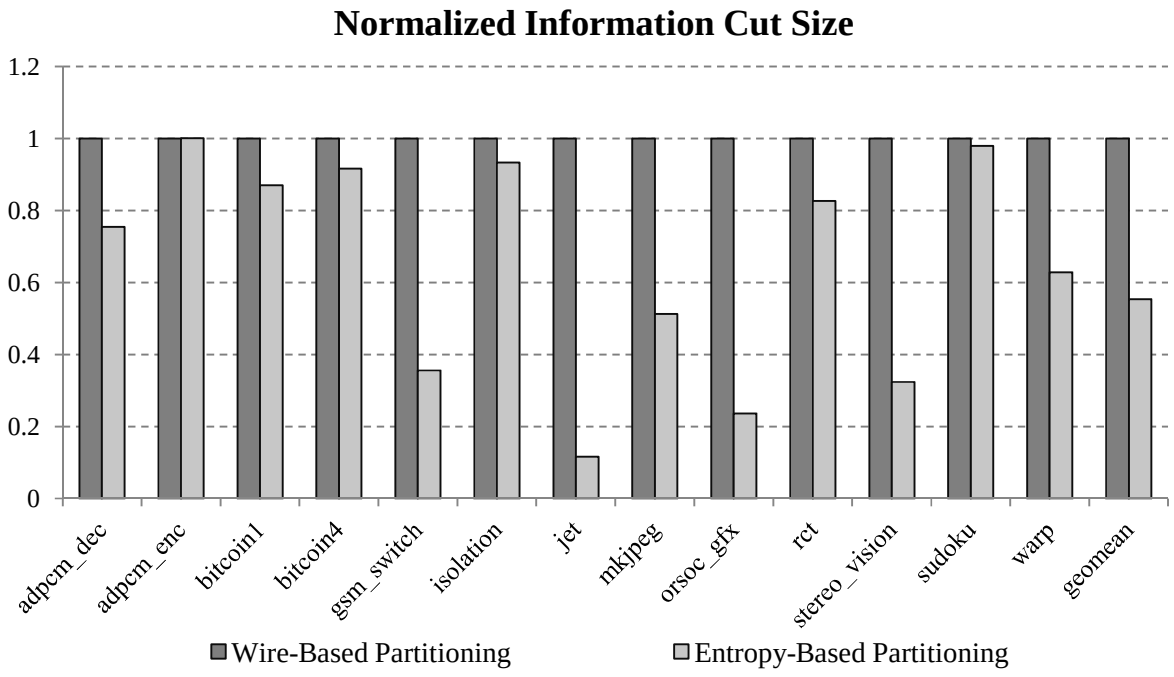

Figure 24. Normalized information cut sizes for wire-based and entropy-based partitioning.

number of very low-entropy signals included in their cuts. They also demonstrate cases where wire-driven and entropy-driven partitioning diverge the furthest. The two approaches produce significantly different cuts and that the quality of these cuts varies dramatically depending on whether entropy coding will be used on the inter-partition communication. Some benchmarks, like *sudoku* and *adpcm_enc*, show relatively little difference in their information cut size. *sudoku* also showed negligible difference in its wire cut size. One possible explanation for why *sudoku* may see no difference in both information and wire cut sizes from the two algorithms is that it may have high information density, where signals have close to one bit of entropy per wire, which would make the wire-based and entropy-based cost functions identical. We will experimentally explore this possibility later in our results.

Overall, the geometric mean of our normalized information cut size results shows that the entropy-aware algorithm is capable of producing cuts that contain 45% less information than those produced by traditional partitioning algorithms.

5.6.3. Information Density Results

In our previous discussion of wire and information cut sizes, we speculated about the impact of information density − that is, the average number of bits of entropy per wire in the circuit − on the relative performance of the wire-based and entropy-based partitioning algorithms. We thus compared the partitioning results from both algorithms, and computed the average number of entropy bits per wire in each cut set. The absolute and normalized data is shown in Figure 25 and Figure 26.

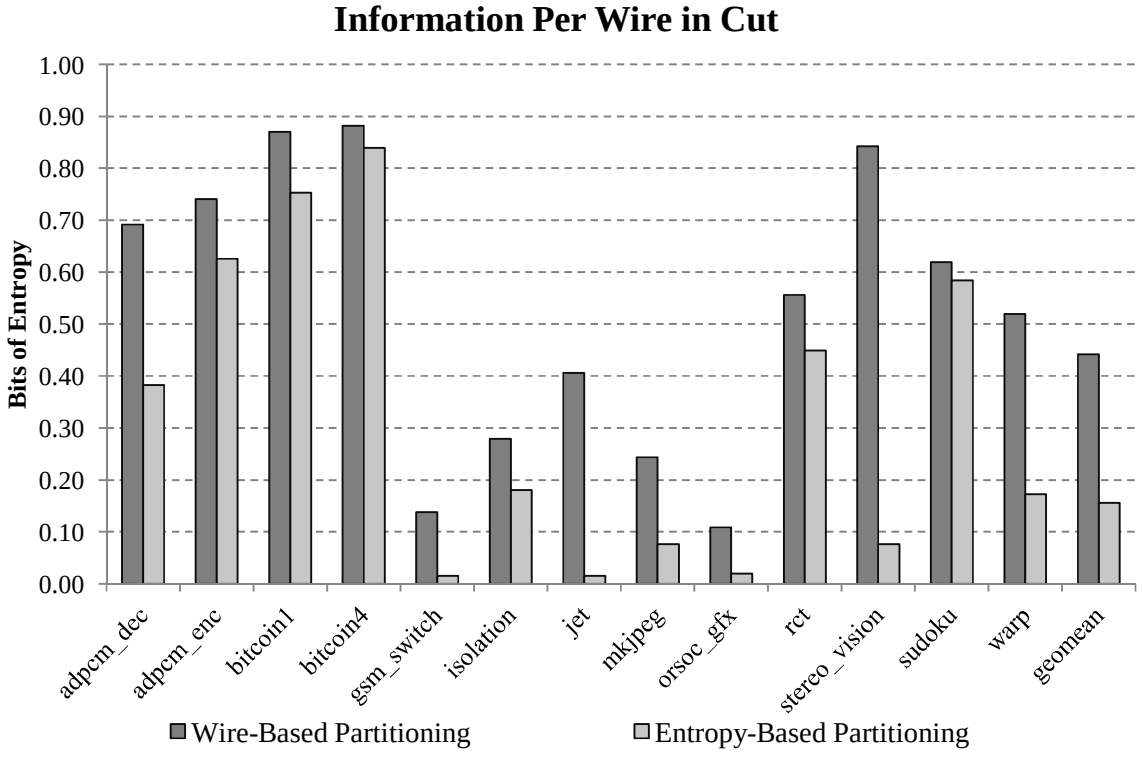

Figure 25. Information, in terms of bits of Shannon Entropy per wire for wire-based and entropy-based partitioning.

The information density results confirm several of our earlier hypotheses. The two *bitcoin* benchmarks, which exhibited high information cut sizes, are indeed among the most information-dense in

our benchmark set, averaging between 0.8 and 0.9 bits of entropy per wire, where the maximum possible is 1 bit per wire. There is also relatively little difference in information density in the cuts produced by the wire-based and entropy-based algorithms for these circuits, suggesting that the circuits contain few low-entropy wires available in the circuit. Considering that there are several factors that reduce entropy in

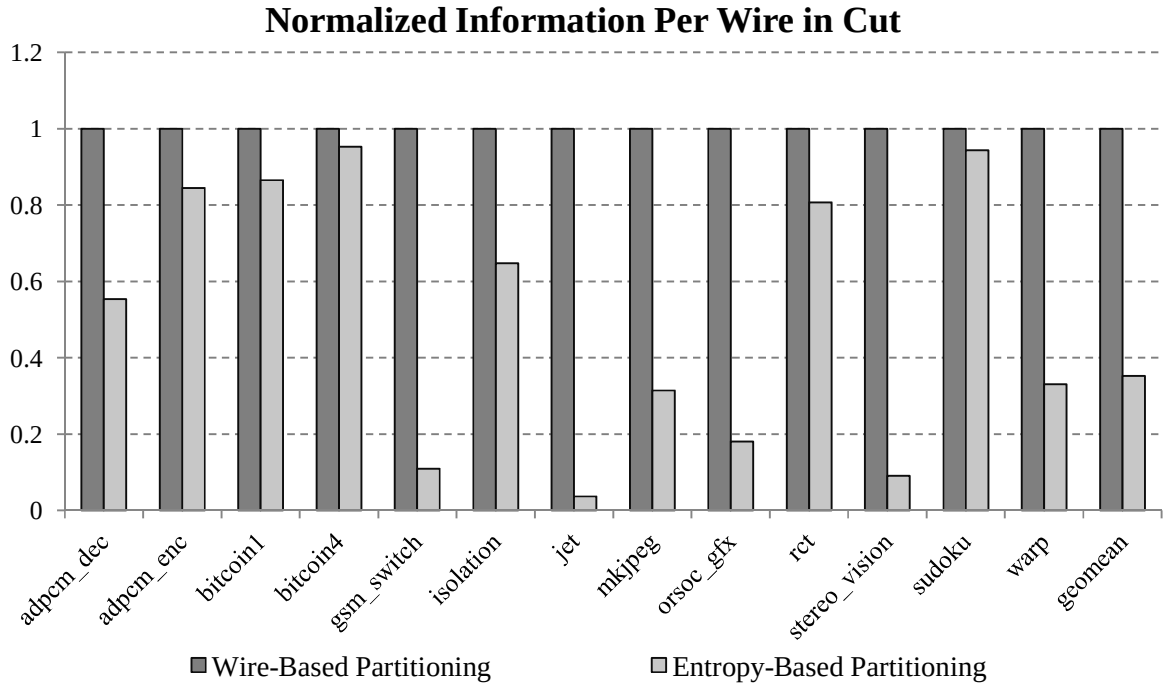

Figure 26. Normalized information, in terms of bits of Shannon Entropy per wire for wire-based and entropy-based partitioning.

circuits (as described in Section 5.3), one may wonder what properties of *bitcoin* contribute to its unusual information density. One key is the type of logic operations performed in the circuit. Much of the logic in the *bitcoin* circuits is used to perform SHA-256 secure hashes, and SHA-256 relies on logical shift, rotation, and XOR operations, which do not reduce the information density of their outputs [173]. Also, unlike most of the other benchmarks, *bitcoin* also does not take an input data set and instead relies on a single seed value. Therefore information density cannot be affected by low entropy inputs.

The benchmarks that showed the greatest improvement in information cut size when using the entropy-aware algorithm, such as *gsm_switch*, *jet*, *orsoc_gfx*, and *stereo_vision*, exhibit very low information density. *jet* is another high-energy physics circuit (see Section 3.3.2) that shares properties

with *rct* − low-entropy inputs and lots of filtering operations. *gsm_switch* is a 2-dimensional mesh networking circuit. From inspection of its individual signal entropies, the low density of *gsm_switch* comes from a combination of lack of full link utilization at any given time (full network utilization is generally not practical [174]), commonly repeated fields in packets (such as the source address), and non-uniform routing (higher utilization in the center of the mesh than the periphery). *stereo_vision* generates 3D images by comparing two images from slightly different perspectives. Since these two images are largely similar, it is not surprising to see some amount of data redundancy.

orsoc_gfx deserves special notice, as it highlights one of the limitations of entropy-based partitioning: the partitioner relies on the entropy data gathered from simulation to be representative of what will be encountered in actual use. This is why the majority our benchmarks use multiple real-world data sets to characterize their entropy. *orsoc_gfx* is one of the few benchmarks we evaluated with a purely synthetic input data set − one that was used for hardware correctness validation. This validation test was not necessarily designed to replicate conditions of maximum logic utilization, so although the partition generated for *orsoc_gfx* may be efficient when running such tests (i.e. if the circuit is used as an emulator to validate the design), but may not be for other use cases.

Finally, we consider the differences between benchmarks that were intentionally chosen to be similar. Both *adpcm_dec* and *adpcm_enc* manipulate digital audio signals however the lower information density in *adpcm_dec* reflects the fact that it operates on decompressed data. *bitcoin1* and *bitcoin4* perform the same task, but *bitcoin1* uses more hardware duplication, whereas *bitcoin4* reuses the same logic for multiple cycles per computation. As one might expect, the duplicated logic shows a slightly lower information density.

5.6.4. Maximum Theoretical Compression Factors

In this section we examine compression with entropy coding as a straightforward motivating example for the possible benefits of partitioning with lower information cut sizes. We will consider the maximum effective compression possible for each benchmark, where the "effective compression" refers to the ratio

between the smallest possible wire cut size (achieved by either algorithm) to the information cut sizes of each algorithm. This provides a fairer measure of compressibility because it avoids inflating the compression ratio for benchmarks that had poor wire cut sizes for the entropy-based algorithm. The absolute and normalized effective compression ratios are presented in Figure 27 and Figure 28 respectively. Because a few benchmarks have very high compression ratios, we also provide zoomed in versions of these data in Figure 29 and Figure 30.

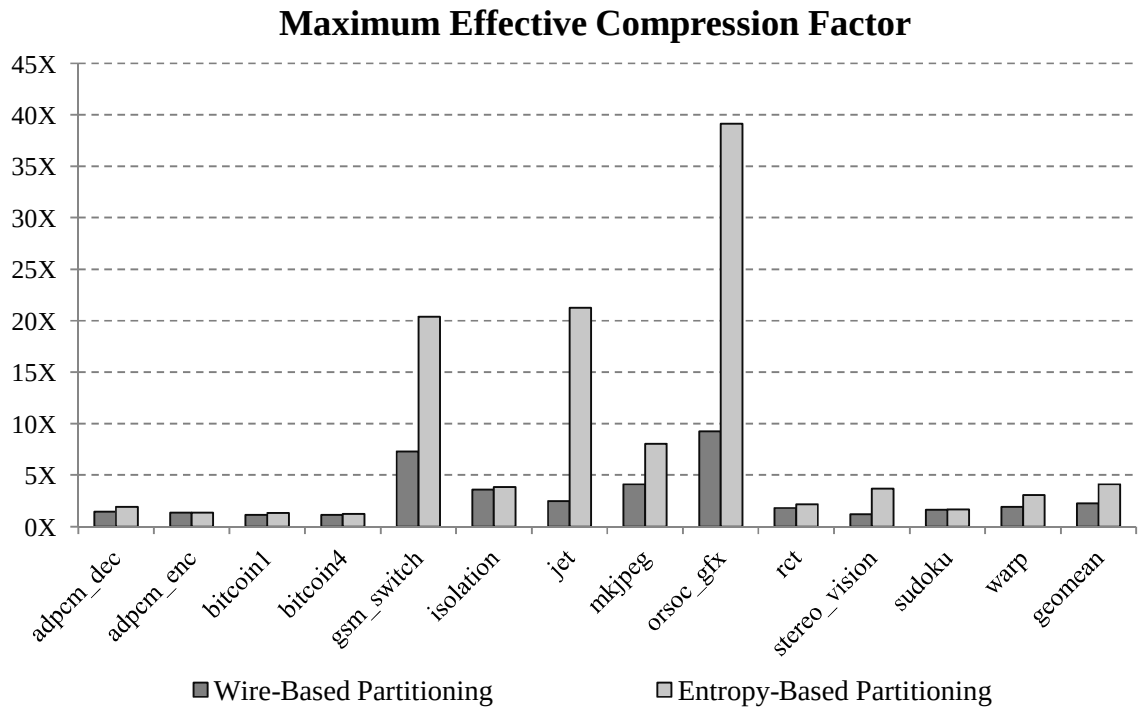

Figure 27. Absolute maximum effective compression factors for wire-based and entropy-based partitioning.

The absolute compression figures show that several benchmarks have inter-partition cuts that could be compressed very effectively. *gsm_switch*, *jet*, and *orsoc_gfx* have maximum compression ratios greater than 20X when using the cuts produced by the entropy-aware algorithm. This follows from the very low information density in the cuts we saw for these benchmarks in Section 5.6.3. Conversely, the benchmarks with information-dense cuts − *adpcm_enc*, *bitcoin1*, and *bitcoin4* − are not likely to benefit from compression. They only show a theoretical maximum reduction in data size of around 5-10% when using compression, regardless of the partitioning algorithm employed. It is worthwhile to note that many

benchmarks can significantly benefit from compression even without using an entropy-driven partitioning algorithm, though there is more potential benefit available when entropy-driven partitioning is used.

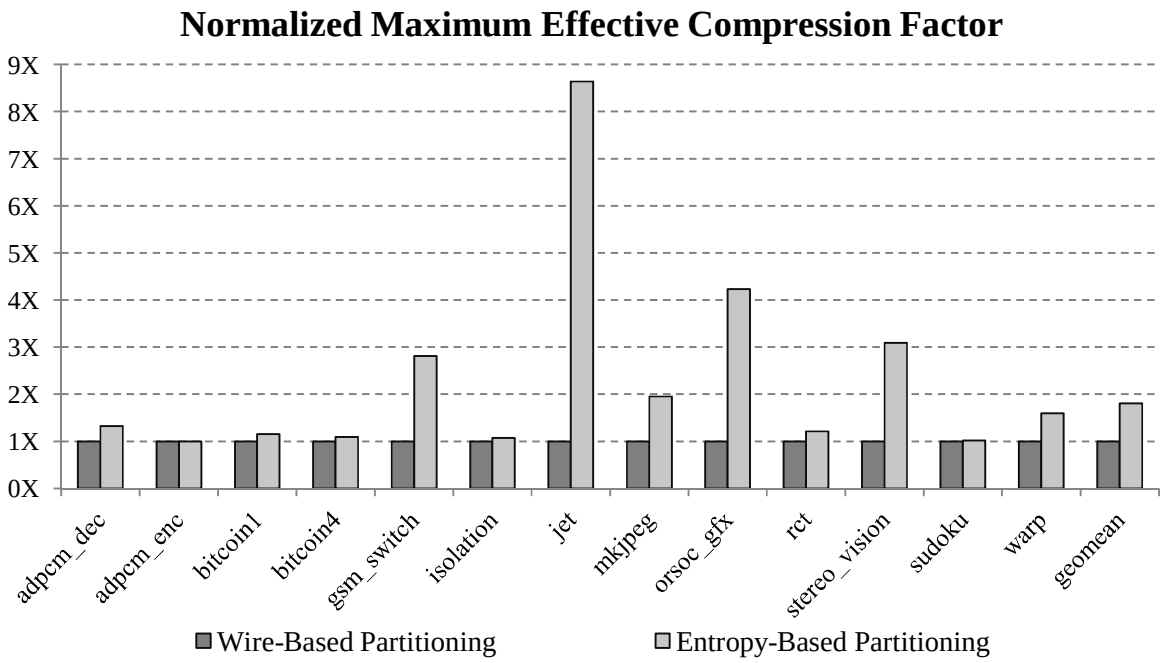

Figure 28. Normalized maximum effective compression factor for wire-based and entropy-based partitioning.

The normalized values allow us to compare the wire-based and entropy-based algorithms more directly. We again see the benchmarks that exhibited the greatest difference in information cut density benefit the most from entropy-based partitioning. In our preliminary study of the *rct* benchmark, we saw a 20% improvement in compression ratios. For our entire benchmark suite, entropy-based partitioning outperforms wire-based partitioning by a much larger 80% margin.

5.6.5. Discussion

In considering whether entropy-based partitioning is worthwhile, we can separate our analysis to two questions: First, is there enough of a difference between the wire cut size and the information cut size in large, application specific circuits to make it worthwhile to employ some entropy-based optimization on the cut set?

Our results have shown that even without using an entropy-driven partitioning scheme, the information cut size of the partitioning solutions produced by our traditional algorithm is half the size of its wire cut size. When considering compression using an entropy coding scheme, we saw an average maximum compression factor of 2X across our benchmarks using wire-based partitioning. Given the

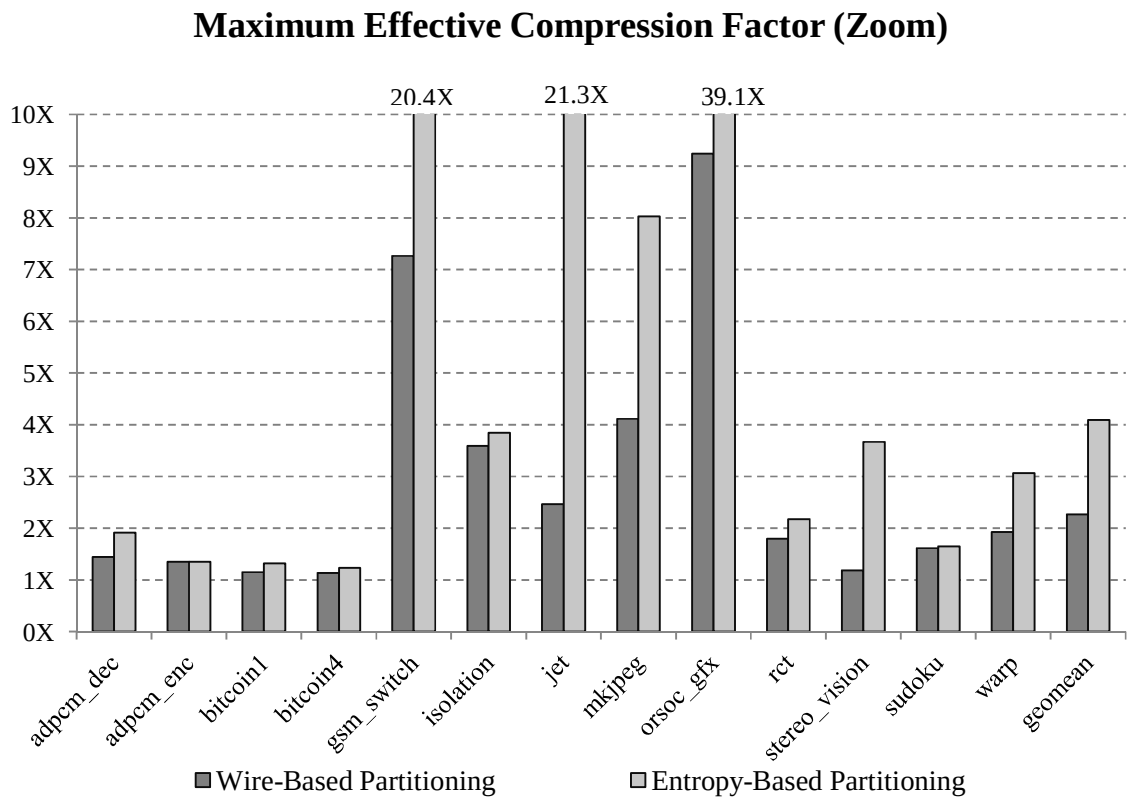

Figure 29. Absolute maximum effective compression factors for wire-based and entropy-based partitioning. The chart scale is zoomed in to illustrate scale of smaller values.

transition to the use of high-speed serial data connections between partitions, where communication costs are based on the required bandwidth rather than the number of wires and where serialization processes can be used to hide compression latency, there is a clear motivation to take advantage entropy-based optimization for applications where decreasing transmission time or network bandwidth are important. Such applications include the high energy physics systems we have used as motivation throughout this research. Beyond compression, there may also be opportunities for useful applications in areas such as fault tolerance and testability.

Given that we have identified the advantages of leveraging information cut sizes in our partitions, the second key question is whether an entropy-aware partitioning algorithm can produce information cut sizes that are significantly smaller than those produced using the wire-based partitioning method. Our results

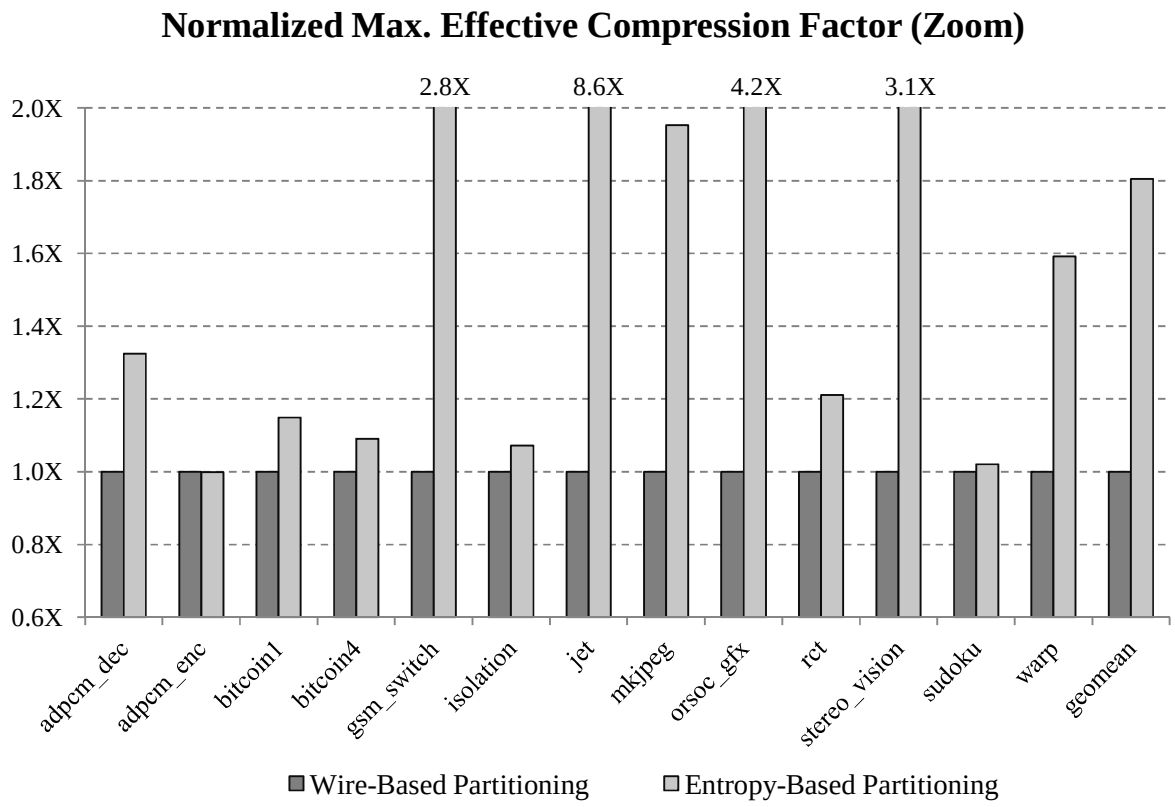

Figure 30. Normalized maximum effective compression factor for wire-based and entropy-based partitioning. The chart scale is zoomed in to illustrate scale of smaller values.

show that an entropy-driven algorithm can produce information cut sizes that are more than 40% smaller than traditional algorithms on average and as much as 90% smaller in 3 of the 13 benchmark circuits we used in our evaluations. From an applications standpoint, this could result in a compression factor that is 80% higher than what was achieved using wire-based partitioning.

Entropy-based partitioning provides greater opportunities for optimization in a bandwidth-centric partitioning model. However, using entropy-based partitioning requires us to possess additional information about the characteristics of the circuit's behavior, as we will discuss in the next section.

5.7. Methods for Collecting Entropy Data in Circuits

In order to use the entropy-based partitioning algorithm we developed earlier in this chapter, we need to know the entropy of the internal signals of the circuit prior to partitioning it. For our partitioning results from Section 5.6, we collected this data by synthesizing the circuit then simulating the post-synthesis netlist using a representative input data set. There are, however, limitations to this approach for gathering entropy data. Post-synthesis simulation can be expensive for large circuits. The simulation process may consume a large amount of CPU time, and recording every value change on all signals in the circuit can produce a large amount of data. For instance, completely simulating the 3D conversion process for just the first image in the dataset used for the *stereo_vision* benchmark required approximately 2 hours of CPU time and 18 hours of wall clock time on a 2.4 GHz quad-core Intel i5 test system. It also produced 36.2 GB of value change data for *stereo_vision*'s 30,000 internal post-synthesis signals. Given that the *stereo_vision* dataset contains 30 input images, both the computation time and the data size are not insignificant. This is particularly true if we want to make entropy-based partitioning feasible for use as part of a rapid design exploration process.

In Section 5.3, we described the role of a circuit's logic topology and the entropy present in its input data set in influencing the entropy of the circuit's internal signals. In this section, we will explore the ideas for using information about the circuit's structure and inputs to compute entropy estimates for the circuit's internal signals in a more computationally efficient manor than by performing post-synthesis simulation.

5.7.1. Evaluating Entropy Estimation Methods

The goal of creating different entropy gathering methods is to reduce the time it takes to collect entropy data we need for partitioning. However, when these methods include estimation, we must also determine the accuracy of the data. As we examine different entropy estimation methods, we will quantify their accuracy by computing the absolute differences between the calculated entropy for each signal in a given benchmark circuit and the actual entropy, as measured using full post-synthesis simulation with our

representative input sets. Although this comparison allows us to gauge the accuracy of each estimation approach, we are also interested in the practical implications the variations in accuracy have on the results we can achieve using entropy-based partitioning. Therefore, we also compare the information cut size of the partitions produced by the entropy-based algorithm when using the different entropy data sources. To do this, we take top 10 partitioning results from each, and then compute the true information cut sizes produced using estimated data by applying the true entropy values.

For our experiments, we use the benchmarks and data sets described in Section 5.5.1, with the exception of excluding the *gsm_switch* benchmark because its circuit design uses proprietary black-box logic created using the Xilinx Core Generator [175] that is not compatible with all of our estimation techniques.

5.7.2. Structural Estimation

Structural estimation is a method of estimating signal entropies based purely on the logic structure of the circuit. To accomplish this, the circuit is decomposed into a directed graph of Boolean logic functions. Recalling that the Shannon entropy of a signal is based on the probabilities of each possible data symbol (0 or 1 for a single-bit Boolean variable), to compute the entropy, we must obtain the symbol probabilities. If we possess the symbol probabilities of the inputs to a Boolean equation, we can compute

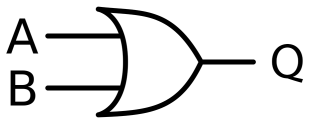

Figure 31. A 2-input OR gate with inputs A & B and output Q.

the probability of each output symbol by computing the probability of each possible input combination and then summing the probabilities for the input combinations that produced each output symbol. We can illustrate this with a simple example. Consider the OR gate pictured in Figure 31. Symbolically, let *P(X)* represent the probability that signal 'X' has a value of 1. For this example, let P(A) = 0.5 and P(B) = 0.7. In Table IX, we show the truth table for the OR gate, along with the probability of each input

combination. If we sum the combinations that produce an output value of 1, we see that P(Q) = 0.85 in our example. This would result in a Shannon entropy of 0.61 bits for the outputs.

Our example assumed that we knew the input symbol probabilities for the Boolean function. These input probabilities can be computed by recursively applying the symbol probability computation to each input until we reach a primary input of the circuit. For our purely structural estimation method, we assume that each input has an entropy of one bit per wire. This method is conceptually very similar to that

Table IX. Truth table with probabilities for OR gate entropy-computation example.

A	B	Q	Probability
0	0	0	0.15
0	1	1	0.35
1	0	1	0.15
1	1	1	0.35

proposed by Macii and Poncino [114]. This method requires us to compute the probabilities of each input combination, which means that the number of computations scales with 2^N, where N is the number of inputs. This might seem problematic; however complex Boolean functions can be decomposed into smaller Boolean operations. Moreover, this process is already performed by the logic synthesis tool in order to map logic to gates ASICs or LUTs in the case of FPGAs. In particular, the Xilinx Virtex FPGAs used in our tests use 6-input LUTs in their slice logic [176], so Boolean operations must be decomposed to operations using 6 or fewer inputs. In practice, structural entropy computations are very fast, even for large circuit designs, as we will see later in our results.

The computation method described can exactly compute entropies of combinational logic; however, most circuits also include synchronous elements such as flip-flops and memories. We make a few assumptions when dealing with synchronous logic. For flip-flops, we ignore control signals like enable, clear, or preset and simply set the flip-flop's output entropy based on its data input entropy. For large memories, it is difficult to determine output symbol probabilities because they depend not only on data inputs, but also on addresses. As such, we do not attempt to compute entropies for memory outputs and simply assume an entropy of 1 bit per wire.

There is one additional important implicit assumption being made in our structural entropy computation: that inputs can be treated as independent variables. The input probabilities we computed for the example summarized in Table IX assumed that signals A and B could be treated as independent random variables. However, depending on the function of A and B in the circuit and the logic that generated them, this may not be the case. In fact, there are clearly some examples of signals in a circuit

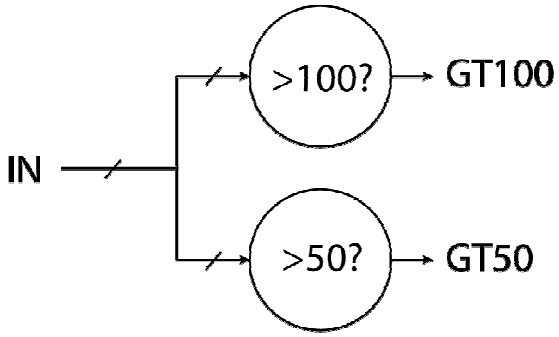

Figure 32. An example circuit that uses two comparators to produce two single-bit outputs (GT100 and GT50) that have correlated values due to the logic of the circuit.

which are correlated. Consider the example given in Figure 32 which shows signals generated by two different comparator circuits. These signals are clearly correlated, as GT100 = 1 also implies that GT50 = 1. Now let us assume that GT100 is signal A of our OR gate from Figure 31 and GT50 is signal B. Knowing the correlation between signals, the new P(Q) is 0.7, not 0.85 as previously computed. Determining whether signals in a circuit are independent is a difficult problem, particularly when the circuit inputs may be correlated. Marculescu et al. argue that this problem, which they refer to as 'conditional signal independence', is NP-Complete under general assumptions [177]. Therefore, we continue with the assumption that signals are independent when making structural entropy estimates, and assume that signal correlation may be one contributor to error in our estimations.

To summarize, the structural estimation approach has three significant limitations that may impact the accuracy of its entropy calculations:

- It assumes that all primary circuit inputs make use of their full information carry ability (1 bit of entropy per wire).
- Entropy computations for synchronous logic outputs are limited.
- All signals are assumed to be independent.

To evaluate structural entropy estimation, we will first compare its computation cost to that of gathering entropy information using post-synthesis netlist simulation. Table X contains the CPU time required to perform both entropy measurement via post-synthesis simulation and entropy estimation using the structural estimation method. The data show that structural estimation is dramatically faster than post-synthesis simulation. Structural estimation is also extremely fast in absolute terms, requiring no more than 2.3 seconds for any given benchmark circuit. This not only indicates that structural estimation can be scaled up to very large designs, but also suggests that it might be combined with other entropy estimation methods without incurring much additional cost − an approach we will explore further later in this section.

Table X. Comparison of CPU time required for entropy gathering using Post-Synthesis Simulation and Structural Estimation.

	Post-Synthesis Simulation CPU Time (s)	Structural Estimation CPU Time (s)	Speedup for Structural Estimation
adpcm_dec	39,302	0.08	490,000X
adpcm_enc	5,379	0.02	350,000X
bitcoin1	3,887	2.30	1,700X
bitcoin4	1,629	1.50	1,100X
isolation	3,007	0.19	16,000X
jet	17,951	1.07	17,000X
mkjpeg	84,500	0.42	200,000X
orsoc_gfx	2,210	2.15	1,000X
rct	1,825	2.06	890X
stereo_vision	201,660	0.36	560,000X
sudoku	216	0.49	440X
warp	629	0.16	3,900X
geomean	5,442	0.43	13,000X

The speedup provided by structural estimation is only useful provided that the entropy values it provides are suitably accurate. To understand the accuracy, we first consider a detailed look at the

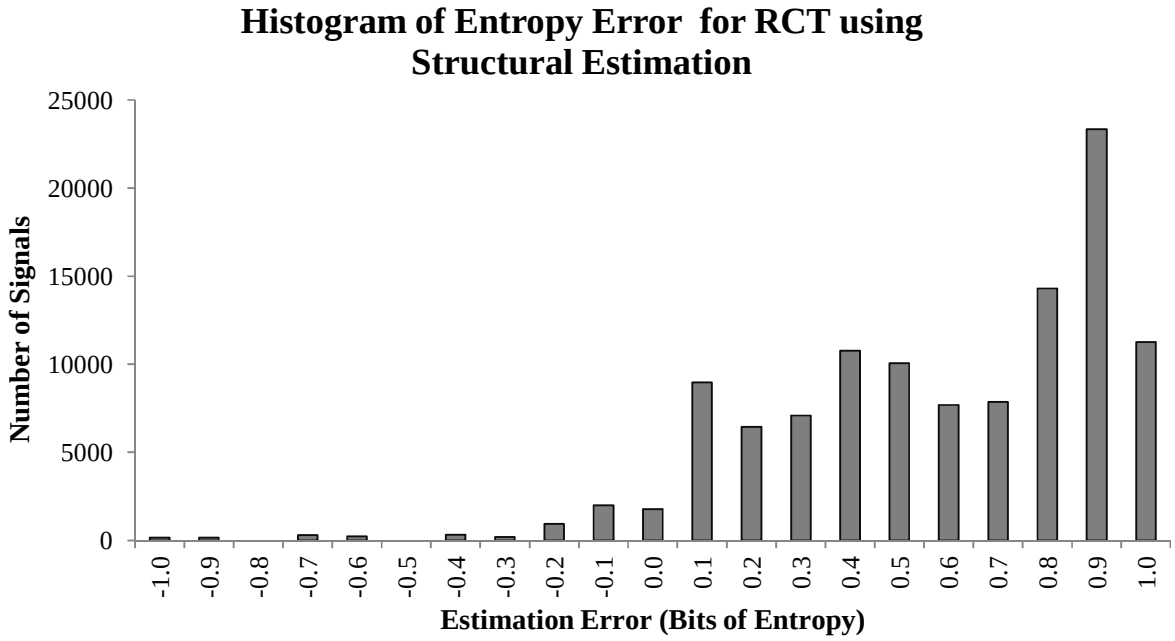

Figure 33. Histogram of the error (in bits of entropy) for all signals in the *rct* circuit when using structural entropy estimation.

accuracy for individual wires in a single benchmark. A histogram of the estimation error when using structural estimation on the *rct* circuit is presented in Figure 33. In this figure positive error values indicate the entropy of the wire was overestimated and negative values indicate the entropy was underestimated.

In evaluating the data, we are interested both in the absolute error and in how that error is distributed. Specifically, it is important to determine whether the estimation error is systematically biased toward overestimation or underestimation. If the error is unbiased − that is, if it were equally likely to overestimate entropy as it were to underestimate entropy for any given wire selected at random − then it may be possible to model it as random error, and we might expect the error to be canceled out in when we compute the information cut size across the entire wire cut set [178]. The estimation error for structural estimation has a mean absolute error of 0.65 bits/wire and a bias of +0.63 bits. Given that the maximum error is 1 bit, both of these values appear to be relatively high. We hypothesize that the fact that structural

estimation assumes that primary circuit inputs and the outputs of synchronous logic have the full 1 bit of entropy per wire may be the cause of its bias toward overestimation.

The average absolute error for all benchmarks is shown in Figure 34. We see the least error when

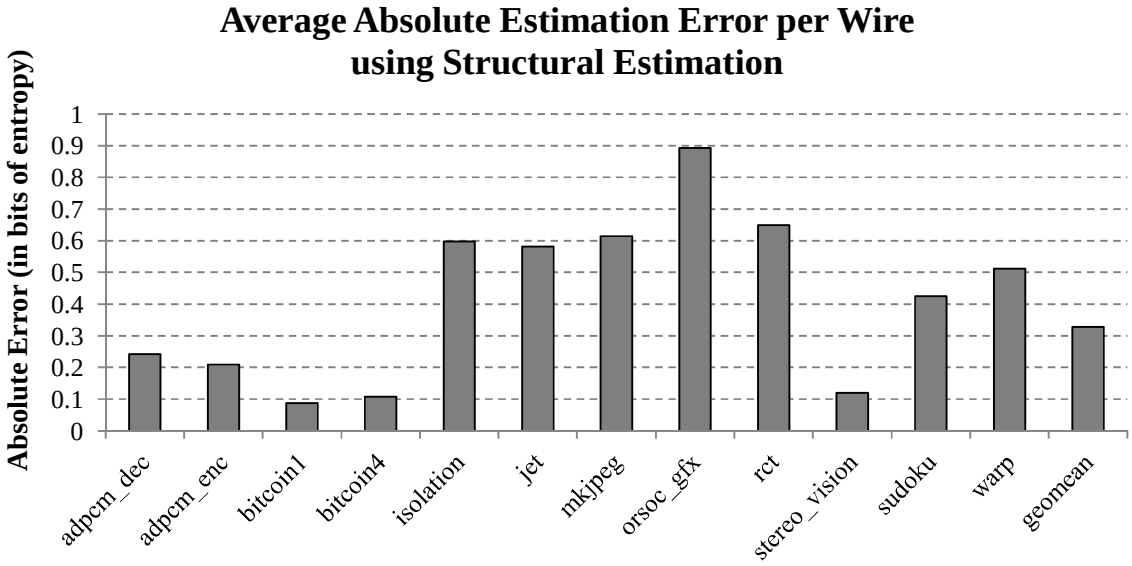

Figure 34. Average absolute estimation error for all benchmarks using Structural Estimation.

using structural estimation with the *bitcoin* benchmarks. This is due in part to the lack of inputs to the *bitcoin* circuit and also due to the fact that the *bitcoin* benchmarks' internal signals had relatively high information density (see Section 5.6.3). Structural estimation assumes high entropy for signals it cannot calculate, and this causes less error in circuits that have high information density.

5.7.3. Input-Based Estimation

One of the major limitations of structural entropy estimation is that it assumes that circuit inputs all have full entropy. However, as we discussed in Section 5.3, some input data sets possess low entropy, and this may contribute to low entropy within a circuit's internal signals. To compensate for this, Input-Based Estimation computes the input symbol probabilities for all of the circuit's primary inputs and then applies structural estimation to determine the entropy for the internal signals in the circuit.

We compare the CPU time required for input-based estimation to that of post-synthesis simulation in Table XI. Input-based estimation is slower than purely structural estimation, but it still shows a

Table XI. Comparison of CPU time required for entropy gathering using Post-Synthesis Simulation and Input-Based Estimation.

	Post-Synthesis Simulation CPU Time (s)	Input-Based Estimation CPU Time (s)	Speedup for Input-Based Estimation
adpcm_dec	39,302	748	53X
adpcm_enc	5,379	748	7X
bitcoin1	3,887	3.3	1200X
bitcoin4	1,629	2.5	650X
isolation	3,007	60	50X
jet	17,951	430	42X
mkjpeg	84,500	7,550	11X
orsoc_gfx	2,210	9	250X
rct	1,825	56	33X
stereo_vision	201,660	2,880	70X
sudoku	216	53	4X
warp	629	10	63X
geomean	5,442	97	54X

substantial speed-up over post-synthesis simulation, with a geometric mean speedup of 54X. Significant variation exists among the benchmarks, due to the amount of input data present and method needed to compute its entropy. The *bitcoin* benchmarks only input a seed value, and thus their time varies little from that of structural estimation. On the other end of the spectrum, *adpcm*, *mkjpeg*, and *stereo_vision* see smaller speedups. This is because the input data for these benchmarks is read asynchronously and the read order is based on feedback signals from the circuit. This means that in order to accurately compute the input symbol probabilities, we must simulate the circuit. However, since we do not need to gather internal signal information, it is sufficient to perform faster functional simulation. For the remainder of the benchmarks, data is read synchronously and in order, allowing input entropies to be computed without any simulation.

A histogram of the per-signal entropy estimation error is included in Figure 35. By comparing the error distribution to that of pure structural estimation from Figure 33, we can see the effect of including accurate input symbol probabilities in our estimation process. The peak located near +0.9 bits of error in

structural estimation has been eliminated, and we now see a peak of signals that exhibit an absolute error of 0.1 bits or less. This is a consequence of the fact that we are no longer over-estimating the entropy of the primary inputs and signals nearby. As a result, the absolute mean error has decreased from 0.65 to 0.36 bits of entropy per wire, and the bias has been reduced from +0.63 to +0.30 bits. After removing the signals that have low error (between -0.1 and 0.1), we also observe that the error appears to be more normally distributed than before, albeit with a positive bias of +0.3 to +0.4 bits.

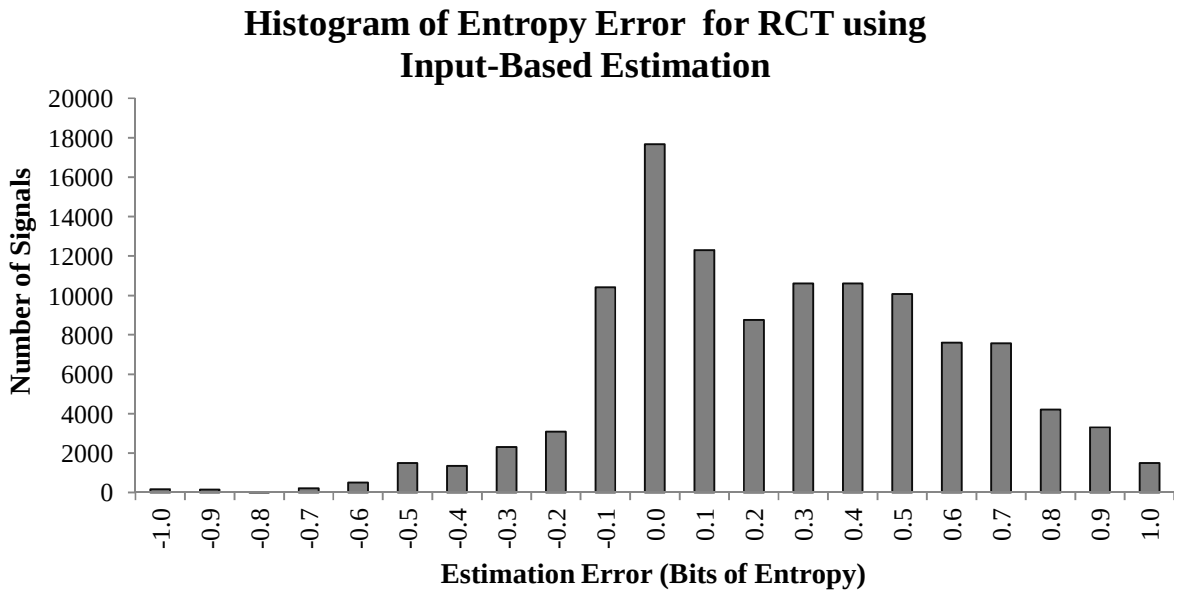

Figure 35. Histogram of the error (in bits of entropy) for all signals in the *rct* circuit when using input-based entropy estimation.

The absolute error for all benchmarks is given in Figure 36. If we consider the benchmarks that still have high estimation error after incorporating the input data (*mkjpeg*, *orsoc_gfx*, *rct*, *sudoku*, and *warp*), one element they share is that they all include memories. Accurately characterizing the information context of memory outputs requires observation of changes in value over time − something that is difficult to achieve with estimation methods that do not simulate the circuit.

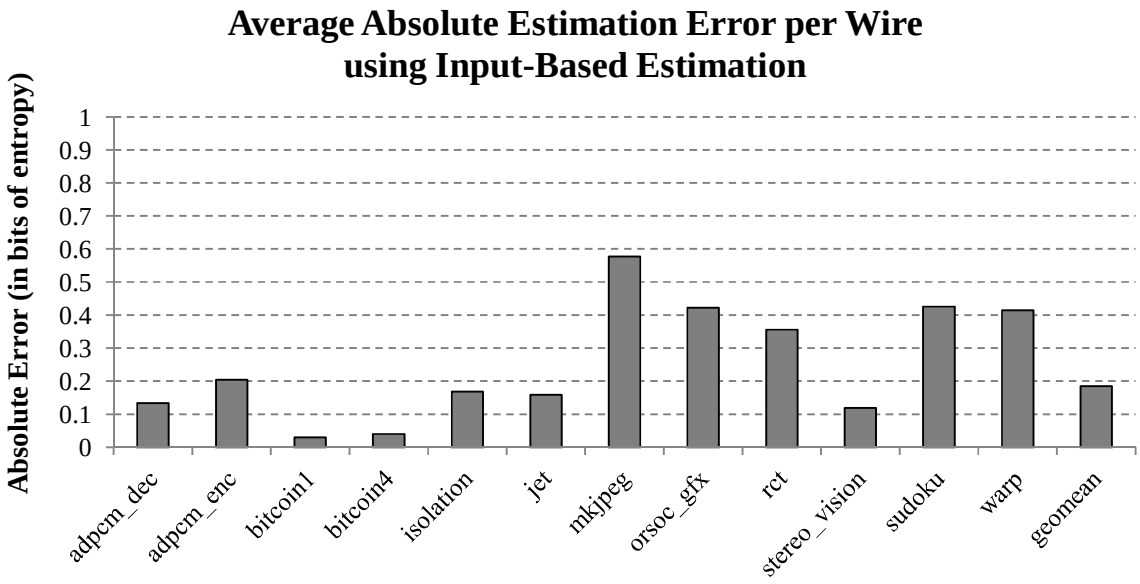

Figure 36. Average absolute estimation error for all benchmarks using Input-Based Estimation.

5.7.4. Functional Simulation-Based Estimation

Input-based simulation allows us to eliminate one of the major limitations of structural estimation, but we are still left with the difficulty of not being able to correctly compute the entropy of synchronous logic outputs and also being unable to account for correlation between signals. Although we can eliminate these issues with simulation, the cost of performing simulation on a post-synthesis netlist is high. An alternative to simulating the post-synthesis netlist at a structural level is to simulate the original RTL description of the circuit. We will refer to this as 'functional simulation'.

Functional simulation is performed at a higher level of abstraction than post-synthesis simulation. For instance, when simulating a multiplication operation at the RTL level, it can be done simply by multiplying the operands together directly, whereas when simulating at the post-synthesis level, it may be necessary to synthesize a multiplier from many smaller logic elements like gates or look-up tables and individually simulate the behavior of each of these elements. As a consequence, functional simulation is often much faster than post-synthesis simulation. We compare the CPU costs of post-synthesis and

functional simulation for our benchmark set in Table XII. The mean speedup is about 10X, and every benchmark sees at least a 3X speedup when using functional simulation.

Although functional simulation is faster than post-synthesis simulation, it does not measure the entropy of every signal in the post-synthesis netlist, which represents that actual circuit that will be partitioned. Internal signals that are not visible at the RTL level − for instance, those signals that internally connect the smaller logic elements used to synthesis the RTL functions. Structural estimation is used to compute the entropy for those post-synthesis signals that are not visible at the functional level. In a general sense, we can view both the input-based and functional simulation-based methods as using a mixture of sampling and estimation. The input-based method samples measurements only from primary inputs and computes the remaining signal entropies. The functional approach increases its sampling to include all signals that are observable at the RTL level. However, there is a key limitation associated with trying to sample all of the RTL-visible signals: we must guarantee that these signals still exist in the post-synthesis netlist.

Table XII. Comparison of CPU time required for entropy gathering using Post-Synthesis Simulation and Functional Simulation-Based Entropy Estimation.

	Post-Synthesis Simulation CPU Time (s)	Functional Simulation CPU Time (s)	Speedup for Functional Simulation
adpcm_dec	39,302	13,145	3.3X
adpcm_enc	5,379	1,727	3.1X
bitcoin1	3,887	88	44.2X
bitcoin4	1,629	35	46.5X
isolation	3,007	340	8.8X
jet	17,951	1,300	12.8X
mkjpeg	84,500	7,550	11.2X
orsoc_gfx	2,210	149	14.8X
rct	1,825	320	5.7X
stereo_vision	201,660	54,660	3.7X
sudoku	216	66	3.3X
warp	629	21	31.5X
geomean	5,442	552	9.9X

In a normal synthesis process, the synthesis tool is free to perform optimizations to the RTL description, such as refactoring Boolean equations, which may mean that the signals present in the RTL description no longer exist in the post-synthesis netlist. Furthermore, synthesis tools are often able to identify and remove unused or redundant logic that may be present in an RTL design. To ensure that our entropy-sampled signals continue to exist, we annotated benchmarks' RTL to instruct the synthesis tool to

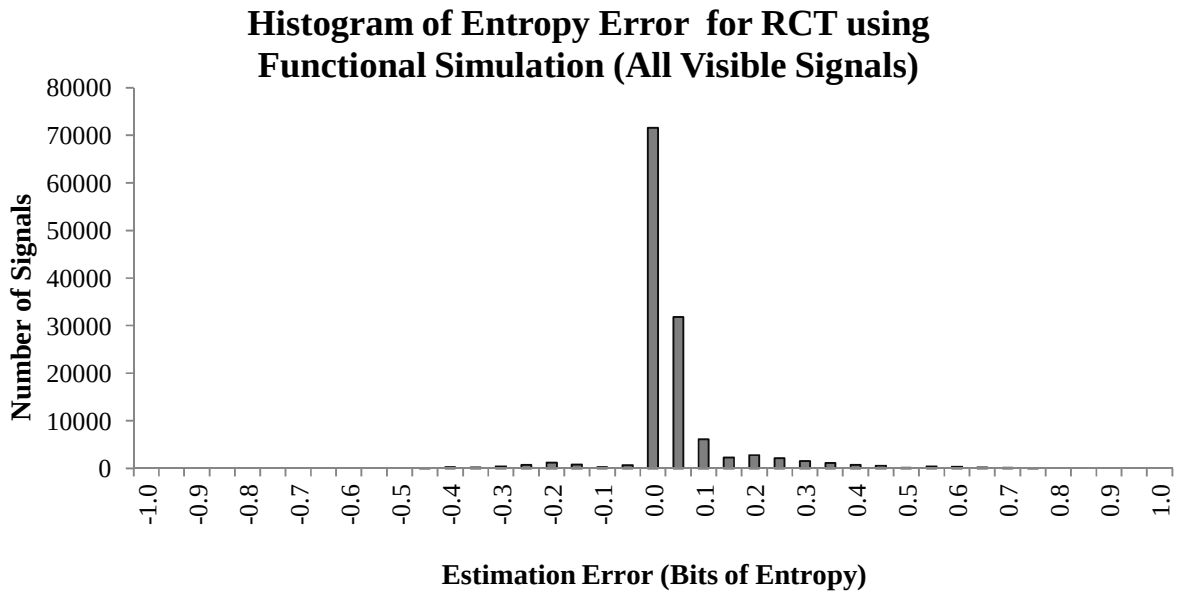

Figure 37. Histogram of the error (in bits of entropy) for all signals in the *rct* circuit when using functional simulation of the circuit and including data from all signals visible in functional simulation.

preserve all of the RTL-visible signals. A consequence of this is that the post-synthesis netlist we will use when evaluating the accuracy of functional simulation may not be identical to the one used when evaluating structural and input-based estimation. This histogram of entropy estimation errors for every signal in *rct* when using functional simulation-based estimation is given in Figure 37. Please note that due to the much narrower error distribution, this histogram uses bin sizes of 0.05 bits rather than the 0.1 bit bin size used in the previous histograms.

The error present when using functional estimation is significantly lower than the previous methods we have discussed. The mean absolute error has been reduced to 0.02 bits and the bias to 0.04 bits. Given that our earlier results from Section 5.6.2 demonstrated that entropy-based partitioning produces mean

information cut sizes that are approximately 45% smaller than those produced using traditional partitioning, a 2% error in signal entropies seems unlikely to have a major impact on its efficacy.

Although functional simulation appears to perform well both in terms of CPU cost and accuracy, it possesses one important drawback. As stated earlier, we ensure that all signals visible during functional

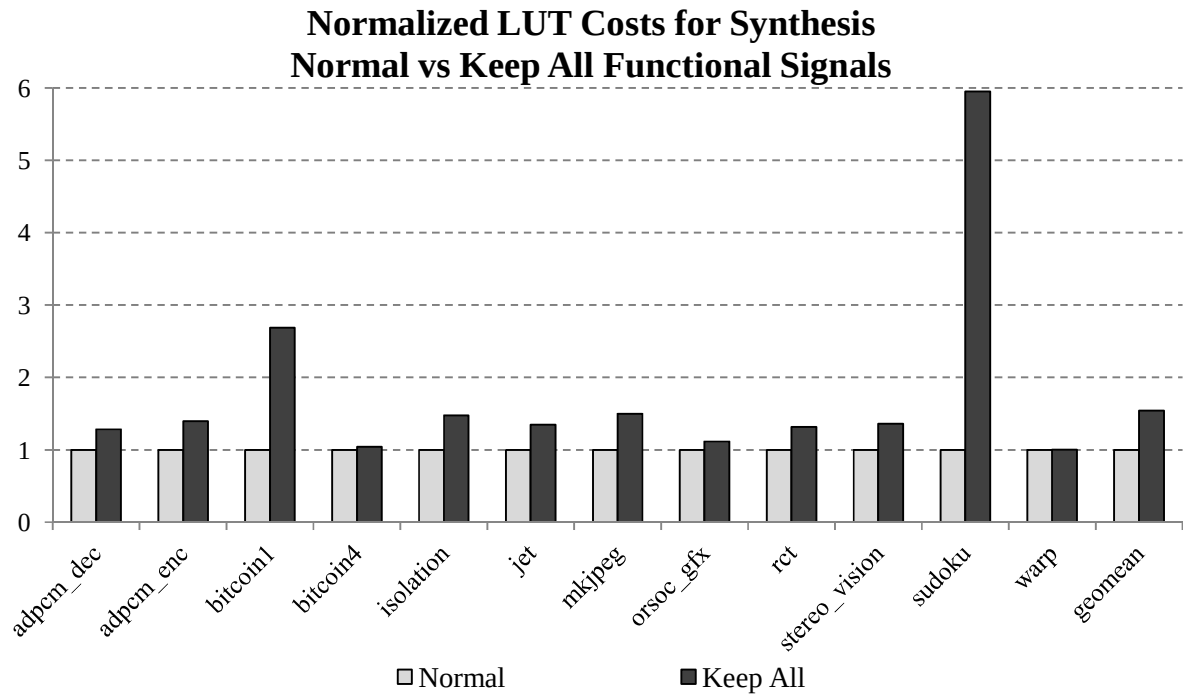

Figure 38. Normalized LUT synthesis cost for the benchmark circuits when synthesizing normally and requiring that the synthesis tool keep all functional signals for use with functional-simulation-based entropy estimation.

synthesis persist in the post-synthesis netlist, but by doing so, we limit the ability of the synthesis tool to perform certain logic optimizations. This may lead to a circuit that requires more logic resources and contains more signals, which in turn may increase resource requirements and/or partition cut size. To quantify these effects, we synthesized each benchmark circuit using the Xilinx Vivado synthesis engine both with and without preserving this type of signal and recorded the resulting logic costs. To ensure that implementations were directly comparable, we also disabled the use of BRAM and DSP blocks and constrained the synthesis engine to only use LUTs and flip-flops. The normalized results for flip-flop cost are given in Figure 39 and the normalized LUT cost is given in Figure 38.

The mean flip-flop overhead is 6.5%, and aside from *sudoku* (which has a 29% overhead), the impact on flip flop costs is not high. For the featured *rct* benchmark, it is only 0.5%. For LUTs, the overhead is much higher. The average is 54%, with a 31% increase for *rct*. Conceptually, the flip-flops stores state

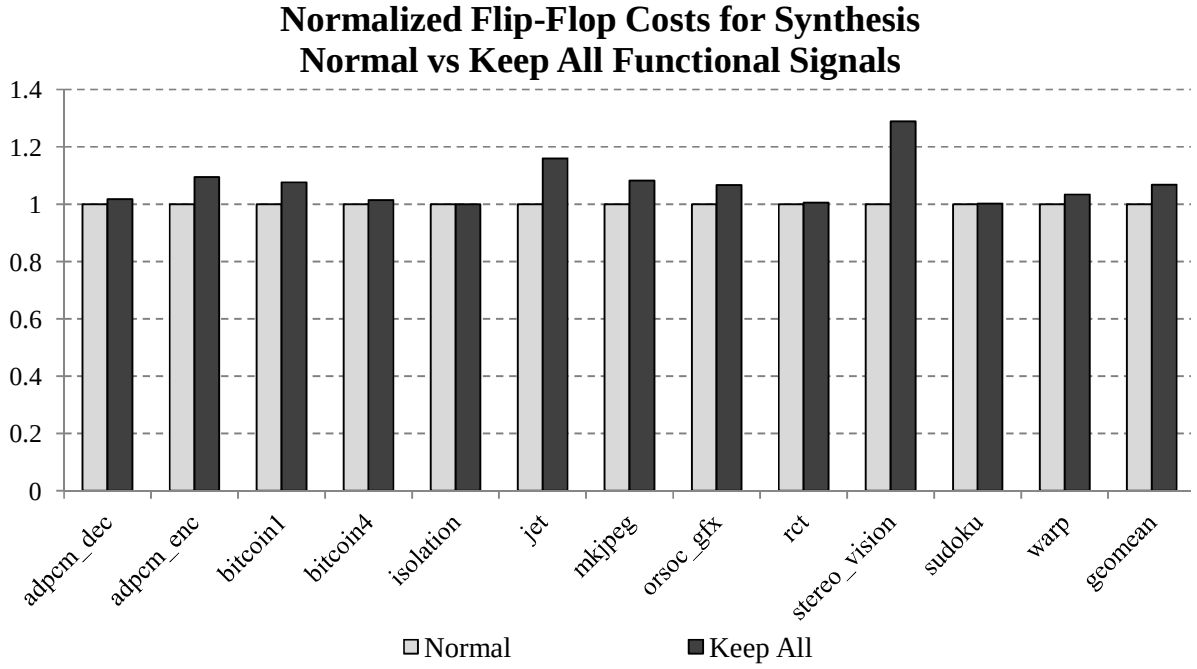

Figure 39. Normalized flip-flop synthesis cost for the benchmark circuits when synthesizing normally and requiring that the synthesis tool keep all functional signals for use with functional-simulation-based entropy estimation.

information. Storage can only be optimized away if it is unused or redundant. On the other hand, LUTs implement Boolean logic, which can be optimized in a variety of ways such as decomposition [179], factoring [180], flattening, LUT packing [181], constant propagation, and others. The 50% LUT overhead is likely to make entropy estimation using functional simulation of all RTL signals unsuitable for most applications that require partitioning. The impact on cut sizes due to the increased number of wires in the circuit also poses a major problem. For *rct*, the netlist generated when keeping all signals had an information cut size that was 40% higher when partitioned than that of the normal netlist, making it worse than the wire-based partitioning algorithm.

Although functional simulation has high accuracy, its negative impact on circuit optimization during synthesis makes it unsuitable for partitioning applications. However, if we can modify the functional simulation method to reduce its impact on synthesis, it may become an attractive alternative to post-synthesis simulation.

5.7.5. State-Sampled Functional Simulation-Based Estimation

In our discussion of estimation based on functional simulation, we described it as a form of sampling, where we measure the actual entropy of a subset of signals within the circuit and use structural

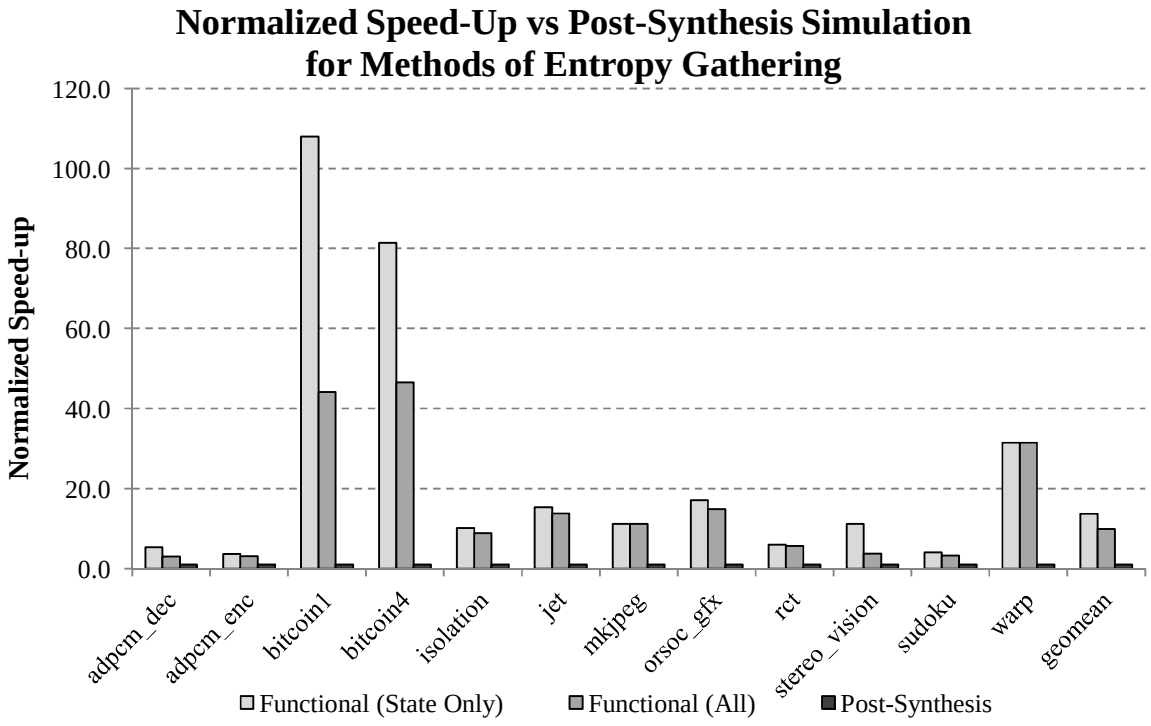

Figure 40. Speed-up of all simulation-based entropy gathering methods, normalized to the CPU time required for post-synthesis simulation.

information to estimate the entropy for the non-sampled signals. However, ensuring that all of the signals that are visible in functional simulation are carried over to the post-synthesis netlist severely limits the synthesis tool's opportunities for optimizing Boolean logic. The natural extension of this line of thought is to ask whether we might choose to sample some subset of the signals visible in functional simulation such that we use signals that we keep in the post-synthesis netlist without limiting logic optimization.

In our discussion of functional simulation, we noted that keeping signals did not produce a large increase in flip-flops. Furthermore, our inability to accurately estimate the entropy for the outputs of synchronous logic was one of the major deficiencies of our estimation methods that do not use simulation.

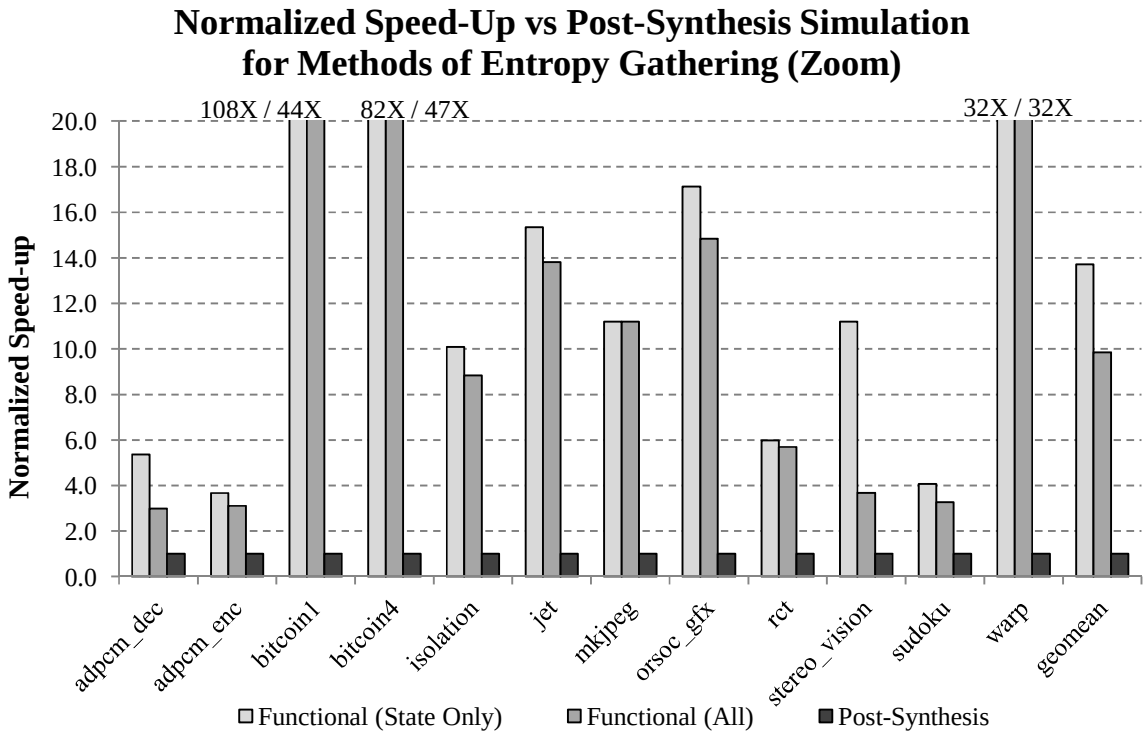

Figure 41. Speed-up of all simulation-based entropy gathering methods, normalized to the CPU time required for post-synthesis simulation. The chart is zoomed to illustrate the scale of smaller values.

Based on this, we hypothesized that if we sampled only the signals in the RTL description that represented the outputs of synchronous logic − that is, the state of the circuit − we may be able to get some of the accuracy benefits of simulation without incurring a high logic overhead.

Using state sampling has a beneficial secondary effect on simulation time, as shown in Figure 40. A zoomed in version of this data is presented in Figure 41. By sampling only the state signals, simulation time is accelerated by an average of 40% over functional simulation using all signals. This speedup can be attributed to the dual effects of having fewer signals to record and by having a smaller circuit to simulate.

The smaller circuit size is due to more efficient optimization of the logic, as demonstrated by the comparison of resource requirements after synthesis, as shown in Figure 42 and Figure 43. The flip-flop overhead decreases from 6.7% to 3.7% when using state sampling. More significantly, the LUT overhead

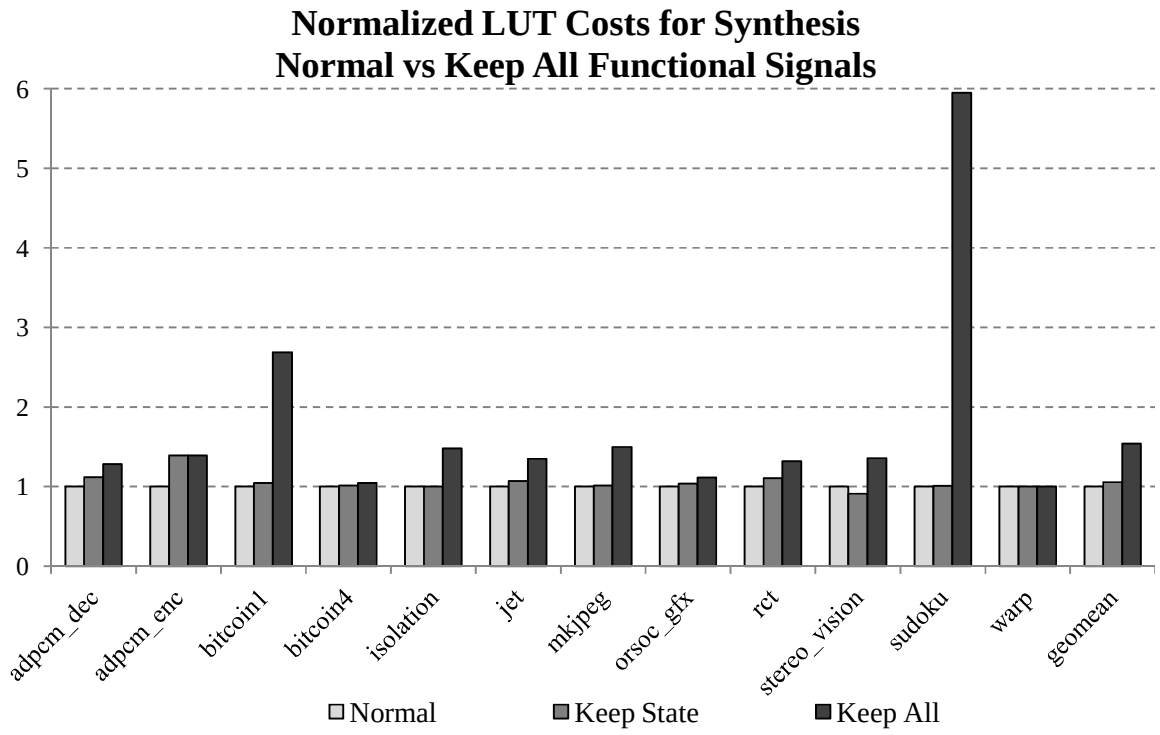

Figure 42. Normalized LUT synthesis cost for the benchmark circuits when synthesizing normally, keeping all function signals, or keeping only state signals.

decreases from 54% to 5%. This confirms our earlier hypothesis that by only forcing the synthesis tool to keep signals that correspond to the outputs of synchronous logic it is still free to perform optimization on the circuit's combinational logic.

Forcing the synthesis tool to retain synchronous logic may prevent some optimizations to state logic, such as state reassignment, but this appears to have a much smaller effect than retaining the signals for combinational logic. The remaining overhead is likely due to the circuit design specifying more synchronous logic than is actually used in the circuit. Based on our inspection of the synthesized circuits, this seems to most commonly occur when the designer specifies a register that is larger than the maximum number of bits needed to store a result. In such a case, the synthesis tool would normally resize

the register, reducing the number of flip-flops and any associated Boolean logic that might be connected to its outputs. It is important to note that these are overheads that could be eliminated, either through optimization of the RTL design to remove redundant state logic or through integrating the synthesis and simulation tools, such that only the outputs of non-redundant synchronous logic are sampled.

Reducing the number of signals that are sampled produces a modest decrease in the accuracy of the entropy estimation, as shown in Figure 44. The mean absolute error using state sampling is 0.12 bits per wire, with a bias of +0.09 bits for the *rct* circuit. Across all benchmarks, the mean error is 0.06 bits per wire, as shown in Figure 45. Given that our simulation method accounts for the entropy of the inputs and for the outputs of synchronous logic, we can conclude that this is the error introduced by our assumption that signals are uncorrelated. In future, it may be possible to further improve on these results by including sampling of some non-state signals in the circuit in an attempt to strike the best balance between sampling all signals as opposed to sampling only the state outputs.

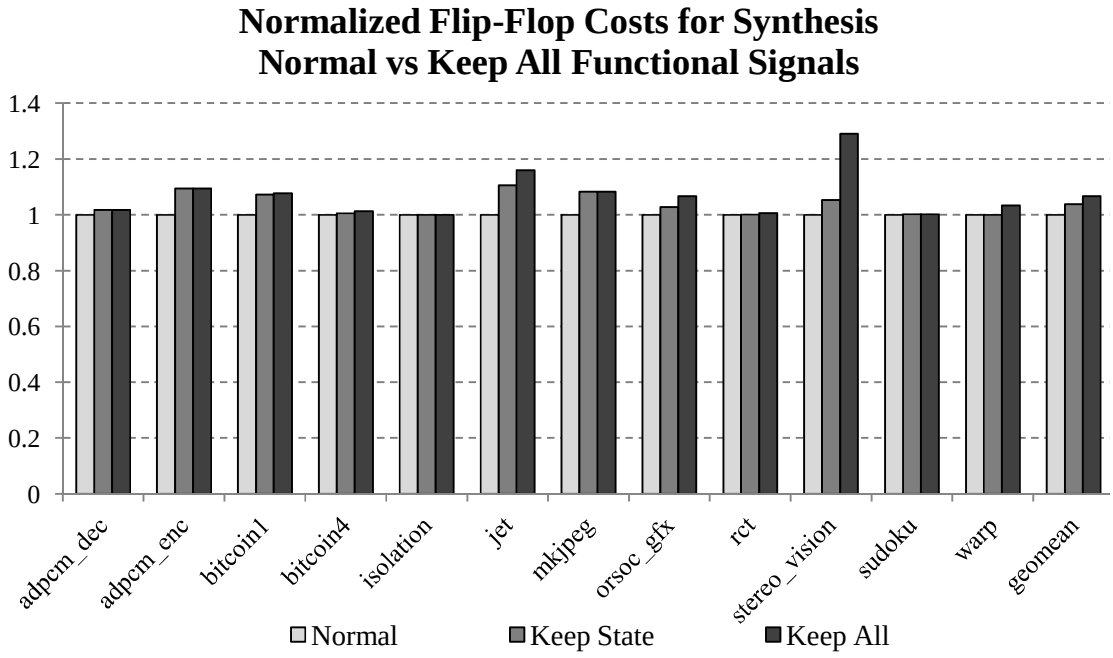

Figure 43. Normalized flip-flop synthesis cost for the benchmark circuits when synthesizing normally, keeping all function signals, or keeping only state signals.

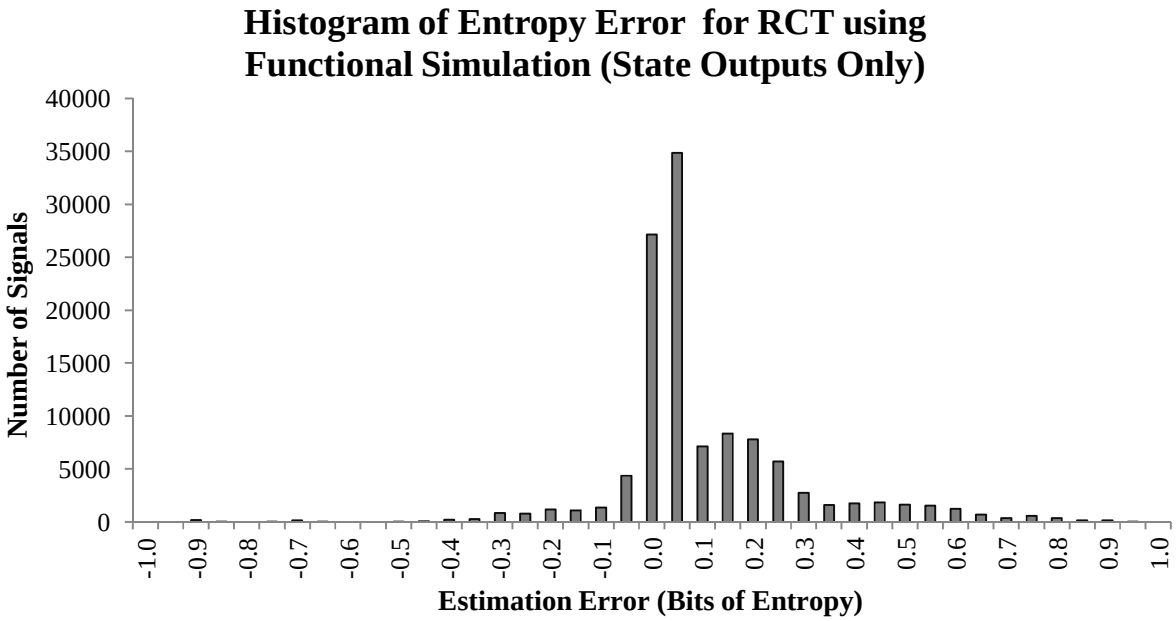

Figure 44. Histogram of the error (in bits of entropy) for all signals in the *rct* circuit when using functional simulation of the circuit and including data from only state outputs in functional simulation.

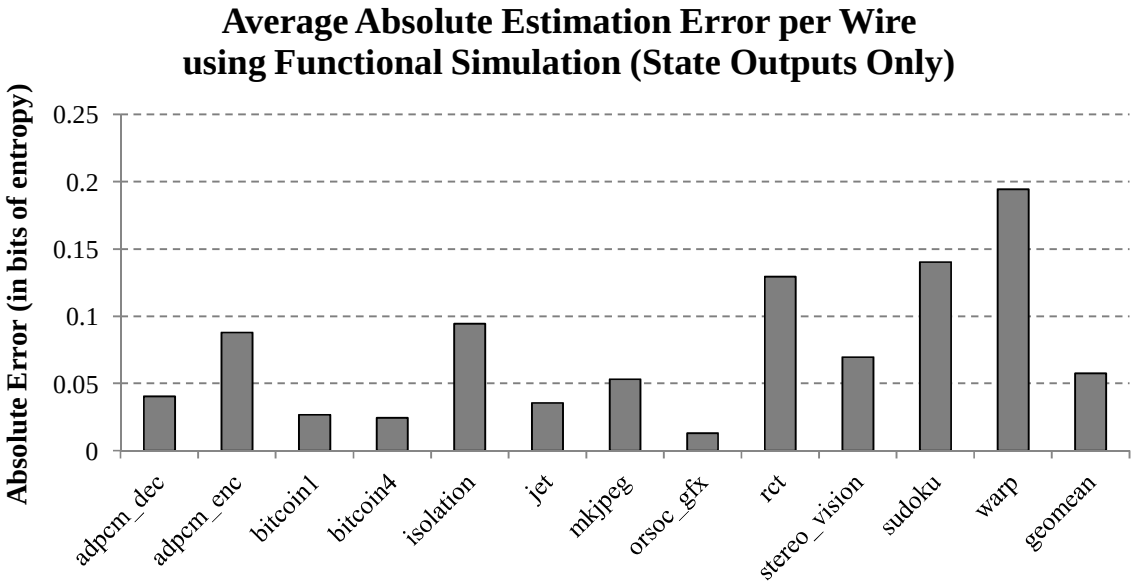

Figure 45. Average absolute entropy estimation error for all benchmarks when using functional simulation of the circuit and including data from only state outputs in functional simulation.

5.7.6. Impact of Entropy Accuracy on Partitioning

Within the context of this research, the significance of the entropy estimation error arises from the impact it has on entropy-based partitioning. For our uses, increasing the accuracy of entropy estimation

only has value insofar as it allows the entropy-based partitioning algorithm to achieve lower information cut sizes. To evaluate this impact, we ran the entropy-based partitioner using the entropy data generated from each of the estimation methods. We then took the best partitioning result produced with each

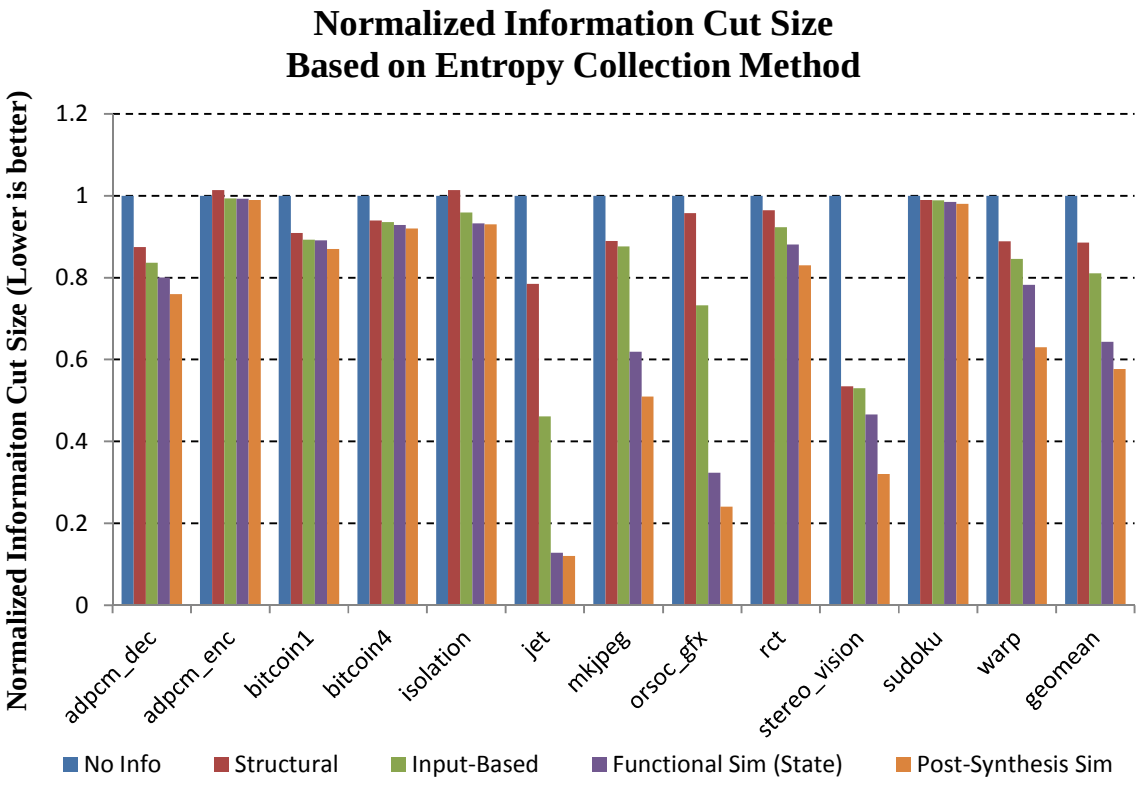

Figure 46. Normalized information cut sizes achieved when using entropy-based partitioning with different methods of collecting the wire entropy data. Values are normalized to partitioning with no entropy info (wire-based partitioning). Lower values are better.

estimation method, used the entropy measurements from post-synthesis simulation to compute their true information cut sizes, and then compared these cut sizes to the cut sizes produced when using no entropy data (equivalent to wire-based partitioning) and when using entropy data measured from post-synthesis simulation. The results are normalized to the wire-based partitioning case to demonstrate the amount of improvement in information cut size that is achievable using each source of entropy data. These results are presented in Figure 46.

The data shows that entropy estimation accuracy has a meaningful impact on the information cut sizes achieved by entropy-based partitioning. Intuitively, this makes sense. Given that signal entropy values are

used in the cost function and in making the decisions of which nodes to move during the algorithm, errors in those entropy values could lead to less optimal moves. The data also shows that in most cases, some entropy information is still better than no information; even structural estimation, with its relatively low accuracy, performs around 12% better on average than partitioning with no entropy information at all, though it did result in slightly worse cut sizes for *adpcm_enc* and *isolation*. There appears to be some correlation between the value of accurate entropy estimation and the average information density results from Section 5.6.3. The benchmarks with low information density (*jet*, *orsoc_gfx*, *stereo_vision*) were among those that were most sensitive to accurate entropy estimation. This finding is consistent with our earlier conclusion that benchmarks with lower information density are more amenable to entropy-based partitioning. Since our estimation methods were all biased toward over-estimating the entropy, the greater the estimation error, the more similar the entropy-weighted graph became to the wire-weighted graph.

5.7.7. Summary of Entropy Estimation Methods

In this section, we have developed and evaluated several methods for estimating the entropy of the internal wires in a circuit, as required for use in entropy-based partitioning. Aside from the functional simulation method that measured all signals, which was unsuitable for use due to its negative impacts on synthesis optimization, each entropy gathering method we discussed has its own set of tradeoffs that may make it useful in different partitioning scenarios. Post-synthesis simulation provides the greatest accuracy when measuring entropy, leading to the best improvement in cut size (43%), but is the slowest method and may not be feasible when using very large circuits and large input data sets. For applications where post-synthesis simulation is prohibitively expensive but where reducing information cut size is still very important, state-sampled functional simulation-based estimation gives a 14X speedup over post-synthesis simulation while still improving cut sizes by 36%. However, it does produce a slight increase in logic utilization. In cases where the run-time of the CAD flow is very important, structural estimation still achieves a 12% reduction in cut size at almost no computational overhead. It also has the advantage of not

requiring a representative input data set to be available. If an input data set is available, input-based estimation is still extremely fast and increases the reduction in cut size to 19%.

We have discussed several software-based options for gathering entropy data in this section, but other options exist outside of software. Hardware-based signal analyzers and probes may be used to measure entropy directly after a design has been deployed to hardware [182], particularly when combined with the sampling techniques we have described. It may also be possible to gather entropy using existing signal monitoring logic included for test purposes. For instance, security researchers have noted that it is possible to obtain internal entropy information of circuits using JTAG connections [183]. However detailed analysis of these hardware-based methods is beyond the scope of our work.

5.8. Application Case Study: Entropy-Based Partitioning in the CMS Level-1 Trigger Emulator Prototype

Earlier in this chapter, we used the Regional Calorimeter Trigger (whose design is described in Chapter 3) and its implementation in the *rct* benchmark circuit to motivate the value of entropy-based partitioning. In this section we discuss some of the challenges and practical benefits of applying compression based on entropy-based partitioning to the RCT firmware. We base our analysis on the challenges of implementing the firmware in prototype of the CMS Level-1 Trigger Emulator (L1TE), a scaled down version of the high energy physics electronics meant for use in testing and validating the physics algorithms and their circuit implementations (including the *rct*, *jet*, and *isolation* circuits used in our benchmark set earlier in this chapter).

5.8.1. Prototype Architecture

The prototype hardware system for the L1TE is under construction at the University of Wisconsin. Based on current prototypes, the full system is planned to contain at least 288 FPGAs (the prototype currently uses Xilinx Virtex-6 XC6VHX250T or XC6VHX350T [3] models, but will likely use Virtex-7 FPGAs in the final system). In addition to partitioning at the FPGA-level, pairs of FPGAs are grouped

onto cards, referred to as Calorimeter Trigger Processors (CTPs), which communicate via high-speed serial I/O links each operating at 4.8 Gbps. CTPs are partitioned into 'crates', which allow connections between 8 CTPs via a custom backplane utilizing a star topology. Communication between crates is performed using gigabit Ethernet. Additional details on the prototype hardware can be found in the related publication [46].

5.8.2. Implementing Compression

Many different options are available for compressing signals. Berger studied compression mechanisms for trigger primitive data in the NA49 high energy physics experiments [144] and compared 11 compression algorithms including zero suppression, run length encoding [184], vector quantization [185], and several variants of static and dynamic Huffman coding [103] and Lempel-Ziv [186]. Based on those findings, we selected Huffman coding using multiple code trees (one tree for each serial link). We chose Huffman coding for several reasons:

- It is an entropy-based coding method, allowing us to use the entropy gathering methods we detailed earlier to estimate its maximum compression factor.
- We can reuse the entropy information gathered for partitioning to populate the code tree.
- Because the statistical distribution of data for high energy physics experiments has been well characterized in research and is time-invariant, it is possible to use static code trees. This allows the encoder to be implemented using a simple look-up table, which is well-suited for FPGAs.
- The hardware cost is relatively low − Berger's implementation requires the equivalent of ~10,000 gates to handle 200 million samples per second, which is less than 0.2% of the logic area of a Virtex-7 2000T FPGA [125].

Huffman compression has also been shown to be feasible in other trigger applications, such as ALICE's Time Projection Chamber [187] and in ATLAS's Level-1 Calorimeter Trigger (using ASICs) [188]. The serialization process required for Huffman encoding, which is sometimes viewed as a

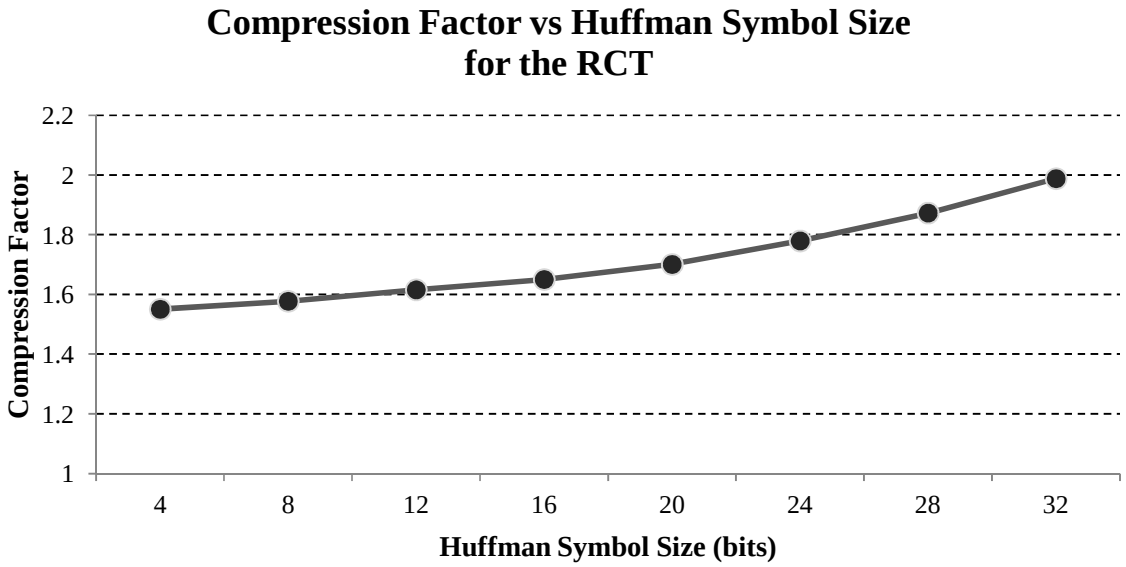

Figure 47. Compression factors for various Huffman symbol sizes in the RCT.

limitation of Huffman coding, has little impact in our application as the data must be serialized for transmission over the high-speed I/O links anyway. Similarly, there is no need to implement more costly parallel decoders in our FPGAs [189] since the data arrives serialized.

To evaluate the efficacy of the compression, we generated a static Huffman tree for the best cut set produced for the *rct* benchmark from our entropy-based partitioner, partitioned the circuit based on this cut set, and compressed the data being transmitted between partitions.

As a first experiment, we compared different possible symbol sizes (the number of wires in the cut to include in a group that will be mapped into a codeword) to determine the impact of the symbol size on the compression factor. We measured the compression factor for symbol sizes ranging from 4 to 32 wires per symbol. The results shown in Figure 47 demonstrate that there is an approximately linear increase in compression factor with increasing symbol size across the range of sizes we tested. Although it is not

evident from the figure, the maximum theoretical compression factor for this partition based on its entropy is 2.3X, so we can expect the improvement to level off for larger symbol sizes.

From a practical implementation standpoint, when using a table-based encoder with a symbol size of N, the encoder requires a code table with 2^N entries, so the symbol sizes is limited by hardware cost. Tree-based encoders can eliminate the code table scaling problem, but their encoding latency scales with tree depth (which itself scales linearly with the symbol size in the best case) rather than remaining constant [190]. Larger symbol sizes also require longer codewords, which increases the decoding time. To keep hardware and latency overheads low, we choose an implementation with an 8-bit symbol size. This achieves an average compression factor of 1.58X, with a hardware cost equivalent to 0.2% of the logic slices per link on our target FPGA and a coding/decoding latency of 33 ns per symbol/codeword. Since we can overlap coding and decoding with serialization and de-serialization, this has a relatively minor impact on end-to-end transmission latencies.

Efficiency in all areas might be improved if FPGA vendors included dedicated hardware to perform encoding/decoding of data sent or received using their serial I/Os; specialized hardware, could allow for larger symbol sizes and improved compression factors. Alternatively, if the inter-partition network fabric is the communication bottleneck rather than the I/O bandwidth of the FPGA, it could be possible to use external ASICs to perform coding and decoding at the FPGA boundaries. However, our analysis is focused on prototyping and design exploration, so we will focus on implementing this logic within the FPGAs.

5.8.3. Temporal Entropy Variation

In our previous discussions of entropy in circuits, we have characterized each wire with a single entropy value. This represents the entropy of the signal over the entire input data set. It can also be viewed as the average amount of information carried by the signal. However the amount of information transmitted on a given signal may not be uniformly distributed over time. An application may go through distinct phases in which different sets of signals are more utilized or variations in the input data set may

change the rate at which information enters the circuit. Selecting a minimal-entropy cut and compressing inter-partition signals based on their average entropy minimizes the total data transmitted, but if we divide the applications run time into different time windows, the amount of information that is transmitted during each windows may vary.

To examine this possible behavior more closely, we divided our simulation of the internal signals of the rct benchmark into 1 µs time windows and computed the entropy for each signal in each time windows and compared the entropy of the signal in each window to the average entropy of the same signal over all time windows. A histogram of the standard deviation in entropy (as a percentage of the average entropy) for each signal is presented in Figure 48. The majority of wires have standard deviations of 10-30% of their average entropy, but there are thousands of wires that have standard deviations of

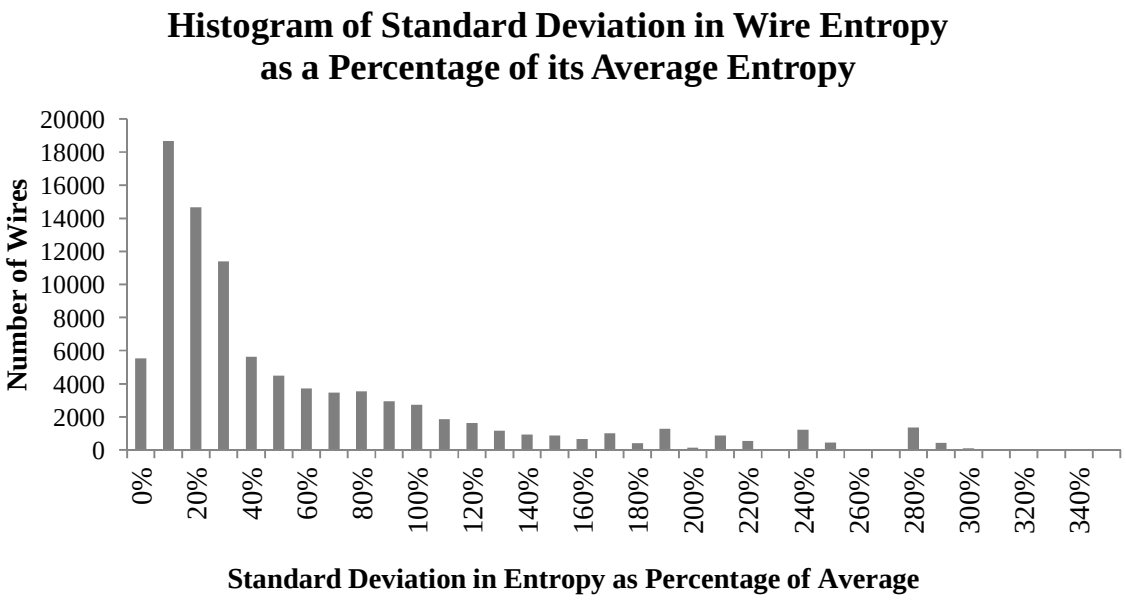

Figure 48. Histogram of the standard deviation in wire entropy (as a percentage of each wire's average entropy) for the *rct* benchmark when computing entropy over 1 µs time windows.

more than 100% of their average entropy, demonstrating significant variation in behavior over time. However, the actual volatility in information cut size is not as large as this data might suggest, because increases in entropy in some wires are often offset by decreases in others. When we consider the standard

deviation in entropy for the entire cut set of the best partitioning result for the RCT, the total entropy variation is 9% when using a window of 1 µs.

The time window size used to compute the entropy variation has an impact on the amount of variation; larger windows allow us to average the entropy of the signals over more clock cycles. Earlier, we

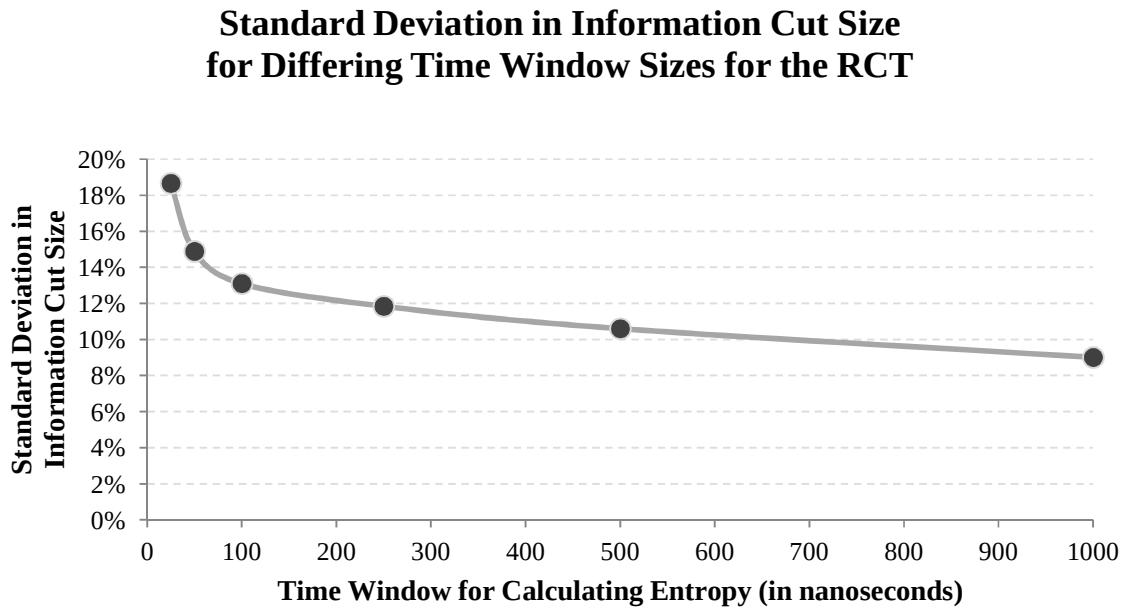

Figure 49. Standard deviation in information cut size for the RCT using different entropy collection time windows.

discussed entropy variation using a 1-µs window size. A 1-µs window is consistent with the planned computation budget for the high-luminosity RCT [53], and is therefore representative of the entropy for the time it takes for a single data set to be processed as well as being representative of number of data sets that would be present in the RCT's pipeline at any given time. Reducing the window size to 25 ns allows us to capture the entropy from a single data set for each wire. We will refer to the data for one cycle of our cut set as a 'frame'. Figure 49 includes a comparison of the standard deviation in entropy for the wires in the cut set using different window sizes. The amount of variation in entropy is significant in that it allows us to make a first-order approximation of the variation in compressibility of the data (since the actual Huffman compression ratio depends on the overall symbol frequency, it is not an exact measure of compression variation). The data show that much of the possible decrease in variation can be achieved by

using a window size of 100 ns, or 4 frames. From an implementation standpoint, this would require the sender and receiver to add storage to buffer 4 frames of data, additional logic for multiplexing frames, and would increase the transmission latency.

In order to ensure all of the data can be losslessly transmitted between partitions, the variation in transmission times can be handled in one of two ways:

- Keep the communication bandwidth available for inter-partition communication constant. This causes transmission time to vary.
- Keep the maximum transmission time for each cut set constant. This causes the required communication bandwidth to vary.

The appropriate approach depends on the application. For circuits used to implement real-time applications with firm deadline, it may be necessary to ensure that the transmission time is bounded to ensure that data is transmitted between partitions rapidly enough to meet computation deadlines. However, this may require us to allocate enough bandwidth to handle the worst-case (least compressible) time window for our application, which may eliminate some of the potential advantages of using compression to reduce network costs. For circuits performing batch operations, it may be preferable to allow transmission time to vary, which can increase performance and/or reduce network costs.

Our high-energy physics application highlights both use cases. The generation of the input data can be viewed as a stochastic process. The average entropy of the input towers is approximately constant over long periods of time, but can significantly vary over a small number of bunch crossings. The real RCT system has firm deadlines, as the LHC produces new experimental data sets every 25 ns, and has a maximum trigger decision computation budget that must be met due to limitations in sensor data buffers [5]. On the other hand, the L1TE system need not operate in real time in order to validate the correctness of the physics algorithms, because the input data sets are not being produced in real time. For the sake of comparison, we will evaluate the implementation tradeoffs of using both constant bandwidth and constant transmission time with the L1TE.

5.8.4. Synchronous L1TE with Bounded Transmission Time

The real Level-1 Trigger system operates with strict deadlines based on limitations in the amount of input data it can buffer. The time it takes to transfer data between FPGAs is included in the deadline, so if we wish to model deadlines in the L1TE while also compressing the data that is transmitted between FPGAs, we must still ensure that there is an upper bound on this transmission time in order to guarantee that the deadline will not be violated. We will refer to this deadline-driven model as the 'synchronous' L1TE, since it processes input data sets and transfers data sets between partitions at a constant rate. In order to determine how much inter-partition bandwidth is needed to set the upper bound on transmission time, we need a better understanding of how the time-variation of the entropy in the cross-partition cut set impacts the amount of data that is transmitted.

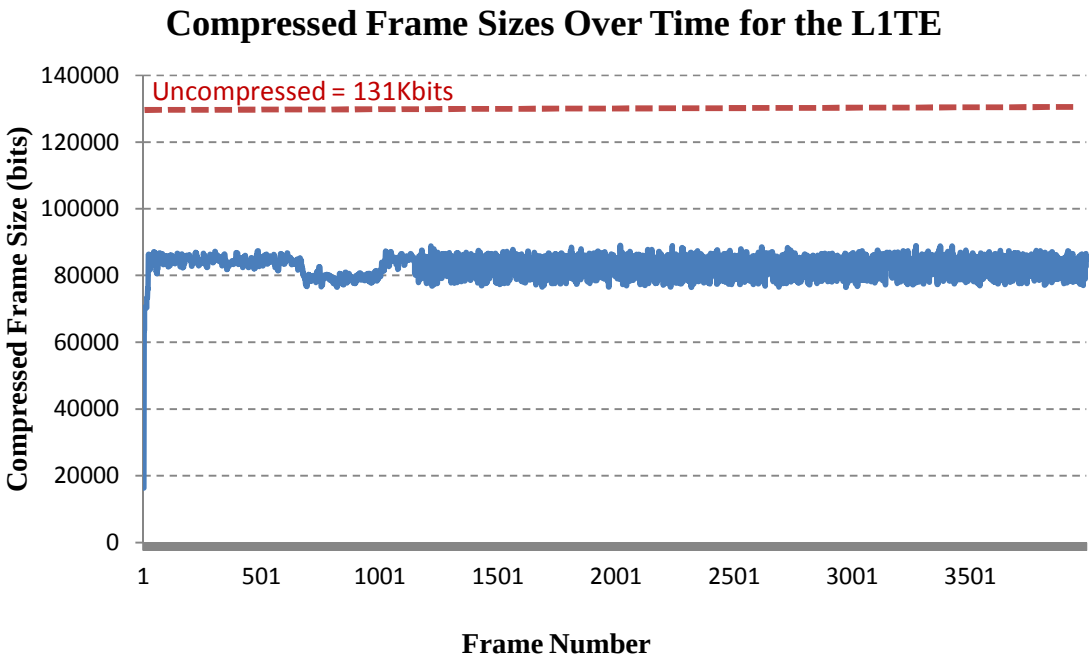

Figure 50. Huffman compressed frame sizes for the first 4000 input data sets for the RCT circuit, partitioned for the L1TE.

We can determine the exact compressed frame sizes by simulating the partitioned *rct* benchmark on the L1TE using the Huffman coder we described in Section 5.8.2. We graphed the compressed frame sizes for the first 4000 frames (representing 1 ms of real time for the hardware system). The results are

shown in Figure 50. We see some start-up behavior reflected in the first 1000 frames, including a decrease in frame size during the period of time when programming dynamic look-up tables and thresholds (refer to Section 3.4 for a description of the use of dynamic look-up tables in the RCT). After this, the frame size maintains a consistent average, of ~83 Kilobits (Kbits), which is consistent with the predicted compression factor of 1.6X for an 8-bit Huffman symbol size from our earlier experiments. Frame-to-frame variation from that average has the appearance of randomness, which matches our assumptions about the RCT's input data.

The maximum compressed frame size in our simulation is 89,024 bits. We will assume a transmission deadline of 1 µs for synchronous operation of the L1TE (the total end-to-end deadline is 3.2 µs [5], so this allows a 1-µs computation budget for each partition with the remaining 200 ns reserved for additional latency costs inherent to serial I/O technology, such as data marshalling, alignment, serialization/de-serialization, channel bonding, and line coding [126] [53]). This represents a 40X reduction versus the maximum throughput of the final Level-1 Trigger system; the full system will use much more hardware running in parallel. We deduct the latency of encoding/decoding (33 ns) from the transmission budget for compressed transmission to allow for a fairer comparison to uncompressed transmissions. To transmit the 89-Kbits frame within our deadline requires an average data rate of 92 Gigabits per second (Gbps). However, because the FPGA's serial I/O links use 8b/10b line encoding [191] − which has a 25% bandwidth overhead − to enable low-voltage differential signaling [192], it actually requires a total link bandwidth of 115 Gbps between the partitions. With the 5-Gbps data rate per link that has been validated on the L1TE, this requires 23 links per FPGA operating at full capacity. In comparison, without compression it would require 164 Gbps using at least 33 links per FPGA. These numbers only represent inter-partition communication do not include the links used for the circuit's primary inputs and outputs. Given that the number of number of serial I/O transceivers on Virtex-6 family FPGAs varies from 12 to 48 (with associated increases in cost for models with more transceivers) [3], this can be viewed as a significant savings.

In Section 5.8.3, we introduced the concept of frame combining as a method of reducing the variation in the size of each frame. We can use frame combining to reduce the maximum bandwidth requirement

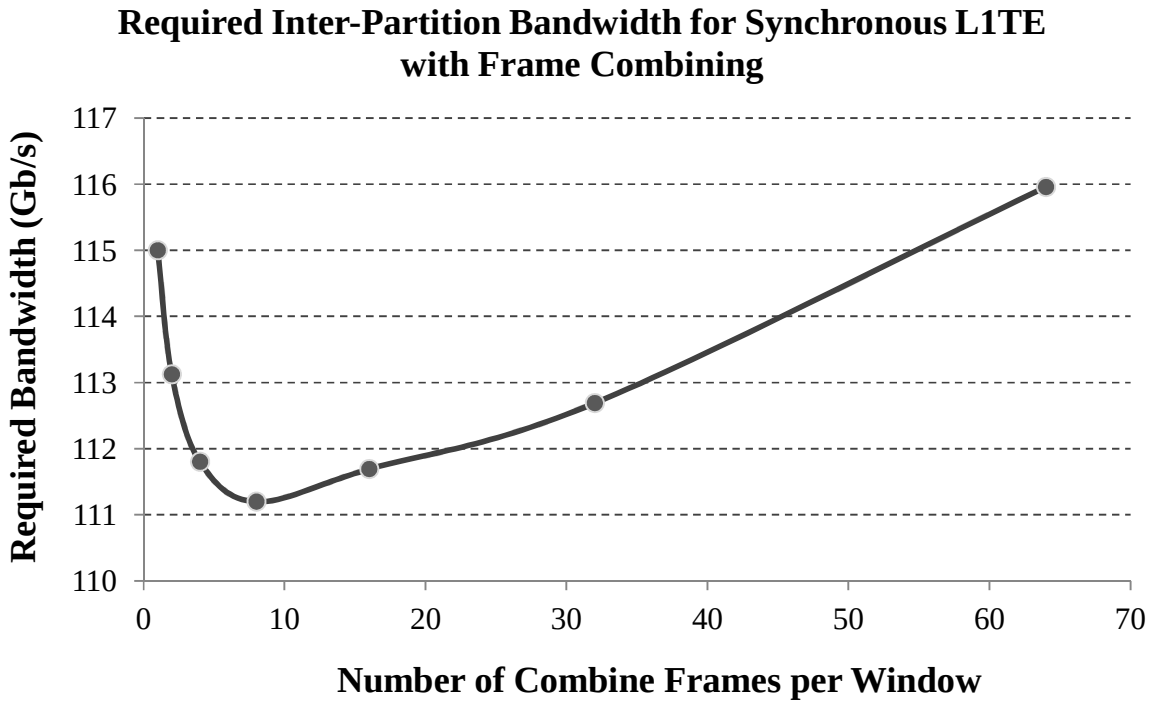

Figure 51. The reduction in the required inter-partition bandwidth for the synchronous L1TE when using frame combining to reduce the variation in the data that must be transmitted in each time window.

for the synchronous L1TE. However, introducing the additional latency required to buffer multiple frames also reduces the transmission time budget. As shown in Figure 51, an 8-frame window size reduces the required bandwidth by 4 gigabits per second. For larger window sizes, the frame buffering latency begins to dominate and the required bandwidth increases. For the *rct* circuit using our input data sets, the value of frame combining is relatively small. This is a consequence of the fact that there is only an 8% difference between the average and worst-case frame sizes in our simulation data. For cut sets with a larger variation in frame sizes, frame combining may be more valuable. To demonstrate this, we randomly generated normally distributed frame sizes with increasing standard deviation in frame size and applied frame combining with an 8-frame time window. The results are shown in Figure 52. When frame

sizes are normally distributed with a 50% standard deviation, frame combining can reduce the required bandwidth by 45%.

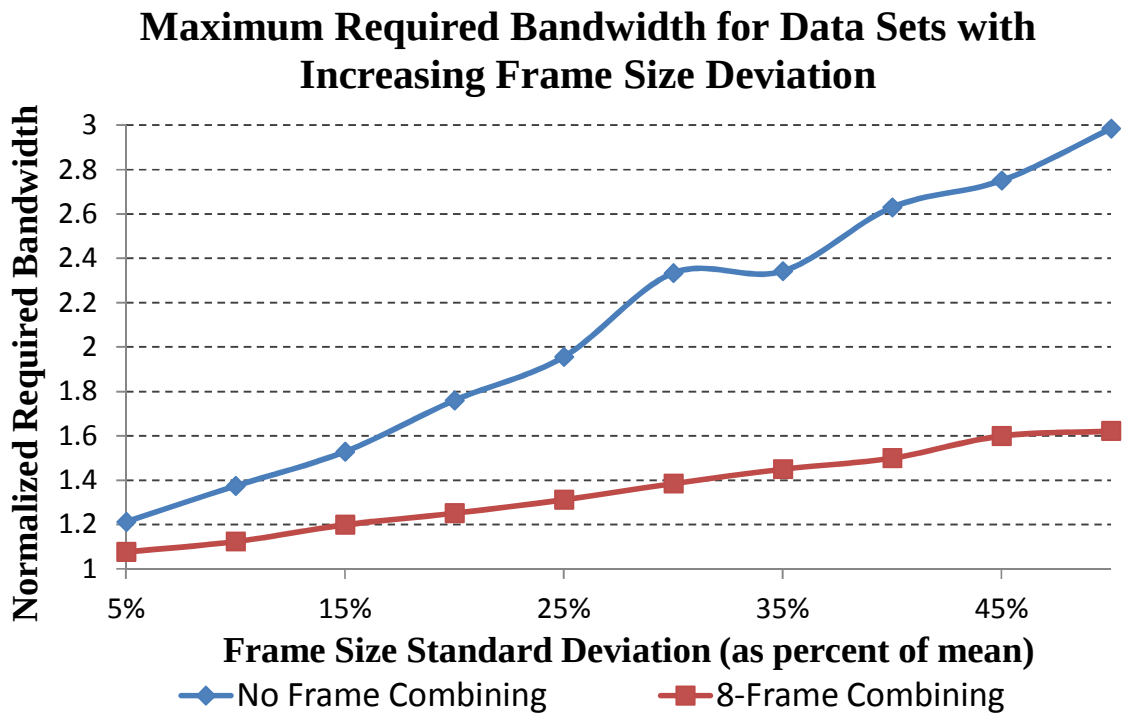

Figure 52. Normalized reduction in required bandwidth when using frame-combining with a time window of 8 frames for normal distributions with increasing deviation in frame sizes.

The 89-Kbit worst-case frame size from our simulation is not necessarily the worst-case frame size that might be encountered in real use. It is difficult to determine the worst-case frame size. From the counting argument, we know that there are some frames that are incompressible [193] − which would suggest a worst-case size of at least 131 Kbits − but it is not clear whether it is actually possible to produce such frames based on the logic of the circuit and the statistical distribution of potential input values, which are constrained by physical properties of the experiment. Such an analysis of the physics are beyond the scope of this work, and providing formal guarantees that transmission deadlines will be met are not necessary for the L1TE. A simpler engineering approach might be to add some additional margin to the maximum value seen in simulation, without providing a guarantee. For instance, adding a 10% margin would provide enough bandwidth to handle frame sizes up to four standard deviations above

the average frame size we observed in simulation. As we have shown, we can use frame combining to further reduce the effect of higher deviations in frame sizes However, without providing a strict guarantee, it is possible that deadline violations will occur. We will discuss the implications of failing to meet deadlines further in Section 5.8.6.

5.8.5. Asynchronous L1TE with Bounded Bandwidth

Although synchronous operation is necessary for the real L1 Trigger hardware system, but is not a requirement for the emulator system. When performing timing validation it may be desirable to operate the L1TE synchronously, but for other cases where we are more interested in maximizing the throughput of the system − such as for testing and validation of the physics algorithms being implemented in the

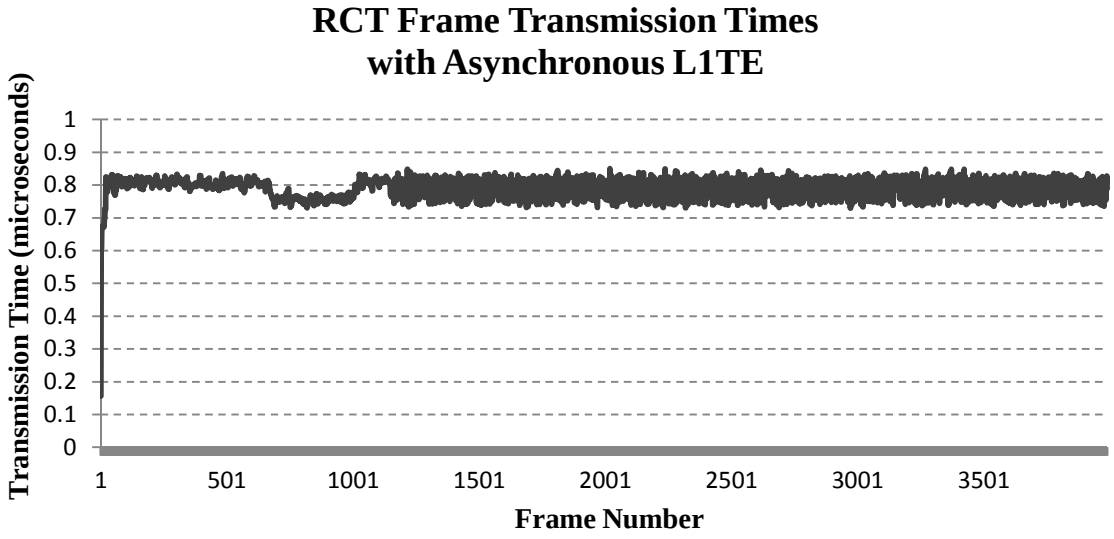

Figure 53. Frame transmission times (in µs) for the first 4000 frames on the asynchronous L1TE when using the same bandwidth as the synchronous L1TE.

system − it may be more desirable to operating the L1TE asynchronously. In synchronous operation, any time difference between the transmission time and the maximum transmission deadline would result in the communication links being under-utilized. In asynchronous operation, the L1TE processes new input data sets as fast as possible.

If we run the asynchronous L1TE using the same bandwidth as in the synchronous L1TE (with a 10% extra bandwidth margin), we achieve the frame transmission times shown in Figure 53. Naturally this distribution mirrors that of the compressed frame sizes. Recall that the transmission deadline for the synchronous trigger was 1 µs. In order to provide enough bandwidth to meet the worst-case deadline, the communication links would be unutilized more than 20% of the time for most frames. Overall, asynchronous operation results in a 26% improvement in throughput compared to the synchronous system when using compression. If we use the same bandwidth as required for the uncompressed synchronous system, the throughput is 52% higher.

5.8.6. Inexact Synchronous Operation

Although we can use asynchronous operation with the L1TE, the final trigger system must operate synchronously. Although we demonstrated significant benefits to using compression of inter-partition communication frames with the L1TE, providing more 10% bandwidth than needed to transmit the largest compressed frame from our simulation seems inefficient, and does not actually provide a formal guarantee that all deadlines will be met. From an engineering standpoint, it is worthwhile to speculate about the consequences of violating deadlines and the potential implementation advantages of designing the system with the understanding that it may miss some deadlines.

Missing a deadline in the Level-1 Regional Calorimeter Trigger system means that we cannot produce calorimeter-based triggering decisions for at least one bunch crossing. There are other sources of triggering decisions such as the Muon Trigger and the Tracker [5] which could be used if the RCT triggering decisions are not available, but in the worst case, we can assume that we are not able to acquire data for that bunch crossing. If this is the case, it is important to understand what type of bunch crossings would lead to deadline violations. Specifically, we are interested in whether incompressible bunch crossings are likely to include data that is more or less valuable to the experimenters than data from compressible bunch crossings.

The ability to achieve good compression ratios in the RCT come from the fact that only a certain number of particles are produced in each bunch crossing, and as a consequence the majority of the sensor inputs have low energy values. This may suggest that collisions that produce an unusually large number of high energy particles would be the ones that could not be efficiently compressed and would fail to meet frame transmission deadlines. This seems problematic, as such collisions might be more interesting to physicists. One potential solution to this problem is to detect when a frame is too large to transmit and 'steal' the transmission time from the next bunch crossing − that is, use the time window that would belong to the next bunch crossing to finish transmitting the frame. This would allow the system to produce triggering decisions for the large frame at the cost of failing to produce decisions for the following frame. Since the contents of bunch crossings are not temporally correlated [194], this is no worse than dropping a frame at random. This is conceptually similar to the concept of 'dead time', a technique used by many trigger designs that samples bunch crossings rather than processing all of them in order to ensure deadlines can be met [195]. Unlike dead time in a conventional trigger system, our proposed approach would drop bunch crossings intelligently based on actual data rates rather than to meet worst-case scenarios.

Other options for inexact transmission include the use of lossy compression methods. However as with other applications of inexact computing [196] [197], applicability depends on the specific application. As with our earlier discussion of lossy look-up table compression in Section 3.4.2, the physics implications of lossy frame compression are beyond the scope of our work and are left to future study.

5.8.7. Summary of L1TE Case Study

Using the L1 Trigger Emulator as an example, we demonstrated entropy-based partitioning, along with data compression, can bring tangible benefits for real applications. We demonstrated that a Huffman coder could produce compression ratios of ~1.6X with low hardware and latency costs when implemented on an FPGA. When operating the L1TE in synchronous mode, we could reduce the amount of inter-partition bandwidth by more than 30% compared to a system that does not use compression when using

the same frame transmission deadline. When operating the L1TE in asynchronous mode, we achieved 52% higher throughput than the uncompressed system when using the same bandwidth. The use of compression introduces variation in the frame transmission time that complicates our ability to provide hard guarantees of transmission times in systems with real-time constraints. In such scenarios it may be worthwhile to consider the opportunities to make use of opportunistic trigger dead time or lossy compress, however further study of the impact of such techniques on the physics performance of the experiment are needed.

5.9. Contributions

In this project we introduced information-based partitioning, a new circuit partitioning paradigm that focuses on the information content of the signals within a circuit rather than just their number of wires. We developed and evaluated several methods for measuring the entropy of real circuits based on their logic structure, characteristic input sets, and simulation. Using this entropy data, we developed an entropy-based partitioner that is capable of producing circuit partitions with significantly lower information cut sizes that those produced by traditional partitioning algorithms. We demonstrated the practical benefits that can be achieved from low-information partitions using the Level-1 Trigger Emulator and showing that a combination of entropy-based partitioning and compression could significantly reduce the bandwidth required to meet deadlines when operating synchronously or increase the processing throughput when operating asynchronously. Our results also show that the opportunities for optimization based on entropy-based partitioning extend to a wide-variety of application specific circuit designs beyond those in high energy physics.

6. Future Directions

Our research has focused on improving methods for designing large-scale, distributed, heterogeneous computing systems. The design of such systems remains a significant challenge, and many avenues are open for further progress, both for high-energy physics systems and more general computing systems.

In our discussions of compressed look-up tables for high-energy physics architectures and cut set compression when using entropy-based partitioning, we focused on using lossless compression algorithms. Lossy compression algorithms might allow for even greater optimization opportunities while also introducing new challenges. With lossless compression, the circuit's functionality is unchanged. However, using lossy compression alters the function of the circuit, as the data received by one partition after decompression may not be exactly the same as the data that was sent by another partition prior to compression. The rapidly growing field of Inexact Computing focuses on similar problems of how relaxing constraints on the exactness of computations can be used to optimize the performance or energy consumption of computing systems. Because the triggering systems used in high-energy physics are heuristic algorithms, we believe that there may be a significant opportunity to apply techniques from inexact computing. However, in order to enable such optimizations, research is needed to quantify the impact that relaxing various correctness constraints in the computations impacts the quality of the solutions produced by the physics algorithms. It would also require the development of methods to identify the most significant bits in a cut set from an exactness perspective and design compression methods that preserved the most significant bits.

Our entropy-based partitioning algorithm uses a relatively simple method to compute the total entropy of a cut set − the sum of the Shannon entropy of individual wires. This represents an upper bound on the true joint entropy; if we were capable of computing the correlation between wires in our cut set, we could potentially obtain smaller cut sizes and achieve better compression ratios. In our study of cut set compression for the L1 Trigger Emulator, we discussed the problem that temporal variation in information cut sizes poses for systems that have firm deadlines on cut set transmission time. It may be

possible to develop more sophisticated entropy-aware partitioning algorithms that minimize the worst-case information cut size over all time windows. Incorporating signal correlations and temporal entropy variation into a KLFM-based partitioner is not possible using a traditional gain bucket data structure, and further exploration of alternative data structures or partitioning algorithms are needed to make these techniques feasible.

Our research was motivated by the development of high performance application-specific computing systems, thus our experiments focused on the partitioning of specialized hardware systems. However, general-purpose high-throughput computing is increasing incorporating heterogeneous computing elements to improve performance or reduce energy consumption. Cloud computing services have begun to offer large-scale distributed computing platforms to a much wider audience. Efficiently partitioning software applications across these systems will be important for the operators of these platforms. Multi-Personality Partitioning and Information-Aware Partitioning may both be valuable in such systems, where CPUs, GPUs, and other accelerators offer the capability for software to be flexibly mapped to different resource types and the hierarchal network designs make minimizing inter-partition communication important. However, compared to partitioning circuits for heterogeneous logic devices, the problem of partitioning software task graphs for heterogeneous cloud computers introduces interesting new challenges. Running applications from multiple users incorporates scheduling problems into partitioning. Moreover, such applications may be interested in optimizing other properties, such as the total energy consumption, which are related to the type of resource that is used. This motivates us to explore different ways of incorporating energy cost metrics into our algorithms for dynamic personality mapping. Furthermore, since software partitioning may need to be done on-demand, there will be greater emphasis on developing very fast heuristics.

7. Conclusion

We have presented three research projects focused on addressing the challenges of efficiently designing large-scale, distributed, heterogeneous computing systems. Our research has focused on design methods that can make a system more robust to change and on methods for efficiently automating key design tasks that might otherwise only be possible through time-consuming manual optimization by specialized domain experiments.

First, we examined the domain of triggering architectures for high energy physics experiments, including hands-on prototyping work for the high luminosity upgrade of the Compact Muon Solenoid experiment. In collaboration with physicists from that experiment, we developed flexible and hardware-efficient implementations of new high-luminosity particle triggering algorithms. As part of this process, we demonstrated techniques for implementing logic in FPGAs using coarse-grained look-up tables in order to reduce area and delay variation, and consequently make it easier for the experimenters to update their algorithms without violating design constraints. Our designs served as a basis for prototyping the Level-1 Regional Calorimeter Trigger computing system for the upgraded experiment.

Second, based on the challenges of efficiently partitioning large physics systems for distributed FPGAs we introduced Multi-Personality Partitioning, a novel form of the graph partitioning problem. Multi-Personality Partitioning is capable of addressing the unique challenges of partitioning logic for heterogeneous devices with flexible resources, where the decision of how to map computations to logic resources can have a major impact on the partitioning process. We also establish the concept of multivariate optimization for multi-personality graphs − an approach that attempts to improve both the cut size and the resource utilization of heterogeneous devices during partitioning. We created an integer linear programming model to describe multi-personality partitioning, and modified the popular KLFM heuristic algorithm to support multi-personality graphs. We developed a multi-personality graph partitioning benchmark set consisting of large-scale circuit designs that targeted heterogeneous FPGAs and used it to evaluate several variants of partitioners based on our multi-personality algorithms. Our

results show that the dynamic personality-mapping algorithms we invented provide new Pareto optimal solutions when compared to prior techniques that relied on static personality mapping.

Finally, we extended our innovations in partitioning for application-specific circuits by introducing the idea of applying an information-theoretic approach to the process of circuit partitioning. Information-based partitioning leverages knowledge of a circuit's functionality and input data sets in order to optimize inter-partition communications based on the data being transmitted. We discovered that by modifying a partitioning algorithm to use entropy in its cost function, we could obtain significantly lower information cut sizes than with traditional wire-based partitioning, allowing us to apply optimizations based on entropy coding. To make entropy-based partitioning feasible for very large circuits, we created and analyzed several different methods for efficiently estimating the entropy of a circuit's internal signals. Using an entropy-based partitioning algorithm, we demonstrated that we could efficiently compress the data transmitted between partitions in the Level-1 Trigger Emulator, a system for validating new high energy physics algorithms, improving its performance. Together, we hope the results of these projects will help to improve the quality and reduce the design cost of large heterogeneous systems.

8. Publications

Table XIII. Publications derived from projects in described in this document.

Title	**Venue**
Publications as primary author	
FPGA Design Analysis of the Clustering Algorithm for the CERN Large Hadron Collider [50]	FCCM '09
Advances in Architectures and Tools for FPGAs and their Impact on the Design of Complex Systems for Particle Physics [198]	TWEPP '09
High Energy Physics, *in Handbook of Signal Processing Systems, (2010)* [199]	Springer Publishing
High Energy Physics, 2nd Edition, *in Handbook of Signal Processing Systems, 2nd Edition (2013)* [51]	Springer Publishing
Multi-Personality Partitioning for Heterogeneous Systems [97]	FPT'13
Multi-Personality Mapping and Partitioning for Heterogeneous Devices and Systems	TODAES (accepted pending revision)
Publications as a contributing author	
Model-based DSP Implementation on FPGAs [200]	RAPID'10
Modular High-Throughput and Low-Latency Sorting Units for FPGAs in the Large Hadron Collider [52]	SASP'11
First Measurement of the Cross Section for Top-Quark Pair Production in Proton-Proton Collisions at $\sqrt{s}$ = 7 TeV [201]	Physics Letters B
CMS Level-1 Upgrade: Calorimeter Trigger Prototype Development [46]	Journal of Instrumentation
CMS Calorimeter Trigger Phase I Upgrade [53]	Journal of Instrumentation

9. Appendix A: A Brief History of HEP Triggering Systems

High-energy physics (HEP) is a field of theoretical physics that studies particle interactions at high energy levels. Some of the primary work in the field has been in the discovery of fundamental particles and the development of theories that explain the fundamental forces that govern particle interactions.

To test HEP theories, physicists must be able to make measurements and observations of particle behavior at high energy levels – often higher than those that occur naturally on Earth. While some HEP experiments are based around the study of high-energy particles produced from sources elsewhere in the galaxy, such as cosmic rays, most are based around particle accelerators. These accelerators are huge machines that accelerate small particles to very high speeds, collide them with other particles, and observe the results on sophisticated groups of sensors and detectors.

Data from the detectors is used to reconstruct information about what happened during and shortly after the collision process. High-energy collisions may produce rare, unstable particles that rapidly decay into lower-energy, more-stable particles. To understand the production and decay of these rare particles, physicists must use the detector data to locate, isolate, count, measure, and identify the post-decay particles, and reconstruct tracks of their trajectories. Almost all collider experiments have detectors that absorb particles to measure their energy (calorimeters) or detect the passage of a particle through a particular location in order to reconstruct its trajectory (scintillators, Cerenkov rings, silicon trackers, multi-wire chambers, muon chambers).

Over the course of HEP experiments, theories that are testable at low energy levels are exhausted first, and further progress requires even higher energy collisions to produce even rarer particles or physical phenomena. Successfully collecting enough data to test theories based on rare phenomena requires the production of a very large number of collisions and more accurate and sensitive detectors. The need for high collision rates and high-resolution detectors has led to a situation in which experiments produce far more data than it is reasonable to store and process. For example, experiments at the recently activated

Large Hadron Collider accelerator, discussed later in the paper, are capable of producing *millions of Gigabytes of data each second*.

To deal with this problem, the data acquisition (DAQ) systems of the experiments employ 'triggers': high-performance, real-time embedded processing systems that are responsible for identifying a small subset of data that is most likely to produce to represent the rare phenomena that scientists are trying to discover. Data from the trigger systems guide the DAQ to save a small, manageable fraction of the total data and discard the rest. Because the trigger system is responsible for determining what data is actually recorded, the effectiveness and efficiency of the algorithms they employ is critical to the success of an experiment. At the same time, the extremely high data rates that must be handled by trigger systems means that they must be large, parallel systems with high processing performance and fast response times. Meeting these strict requirements has led trigger system designers to rapidly exploit technological advances and employ complex systems that are often a hybrid of high-performance embedded hardware and software designs. This paper will examine the architecture of trigger systems at a variety of different accelerators and experiments spanning that past three decades.

9.1. Early Trigger Systems

Compared to today's HEP systems, early accelerators had lower luminosities and smaller event rates; therefore the amount of data being handled by the DAQ systems was much lower than in modern systems. However, the ability to store experimental data for offline analysis was limited both by the slow speed of archival data tapes and by the high expense of computing resources. For these reasons, the development of trigger systems was still important. Trigger system for particle accelerators have existed in some manner since the early development of accelerators in the 1930's and 1940's. Prior to the development of fast, high-density integrated circuit technology, accelerators relied on analog devices and signaling to trigger their DAQ systems. A common technique in the 1950s was to use plastic scintillation counters capable of detecting neutrons, gamma rays, and other ionizing particles emitted in collisions [202]. Scintillators are materials that luminesce when hit by an ionizing energy. When paired with a

photodetector, which emits electrons when exposed to light, scintillators can be used to form simple counters of high-energy particles. These counts were used to guide triggering decisions. The early origins of modern trigger systems can be traced to the early 1970s when multi-wire chambers began being used at the ADONE electron-positron collider at Frascati National Laboratory [31]. These systems applied the concept of track finding that was originally developed in the form of the 'spiral reader' [32] for bubble physics. This system featured on-line processing using software running on computers. The introduction of computer-based track reconstruction formed the basis of modern triggering systems.

9.2. Trigger Systems of the 1980s

The 1980s was an important decade for the development of digital electronics, and the advancements of the decade are reflected in the triggering systems of the era. Although Application-Specific Integrated Circuits(ASICs) and RAM-based triggering systems were most prevalent at the hardware level, microprocessors, introduced in the early 1970s, began to work their into trigger design. Near the end of the decade, the invention of FPGAs started to make ripples in trigger architectures.

9.2.1. Axial Spectrometer Experiment at the Intersecting Storage Rings (1971-1984)

CERN's Intersecting Storage Rings (ISR) was the first particle accelerator to achieve proton-proton collisions by colliding separate beams [33]. Previously, colliders had used a single beam and a fixed target. The development of colliding beam accelerators had several impacts on trigger and DAQ systems that increased the background event rate that the trigger systems had to handle. First, the presence of multiple beams made the likelihood of beam-gas interactions higher. With more circulating particle beams, the odds that collision events might occur between a particle beam and molecules of a contaminant gas increase. Such "background events" do not have high enough energy to produce interesting physical phenomena and can therefore be considered similar to noise. Second, the higher collision energy provided by two beams also allowed for scaling to high luminosities.

The Axial Field Spectrometer (AFS) was a multi-purpose experiment at the ISR that was designed to study the behavior of proton-proton and proton-antiproton collisions characterized by high transverse momentum (pT) and transverse energy (ET) [203]. The experiment used a large number of detectors including multiple calorimeters, multi-wire chambers (MWCs), a drift chamber, central detector, and Cerenkov counter. The presence of a large number of detectors and an interaction rate greater than 800 kHz made an efficient online trigger system vital to operations.

The AFS Trigger system uses two levels of processing. The first level (L1) has a latency deadline of 1µs. It uses simple processing based on 'coincidence counting' and pre-computed RAM look-ups [34]. Signals from the MWCs, scintillators, and barrel counter are checked for matched hits. Each wire is assigned an address. If there are enough hits from the corresponding detectors, the wire address is used to index into the RAM-based look-up-table (LUT) that holds pre-computed momentum thresholds. The process of passing through barrel counters, coincidence counting, and passing the LUT thresholds filters out many of the less interesting events. This reduces the event rate from 700 kHz to 10 kHz.

Events that pass the first trigger level are passed to the second level (L2), composed of ESOP processors. ESOP is a custom, 16-bit programmable DSP processor designed in-house at CERN for use in triggering applications [35]. The 80-MHz ESOP processor has a 250-µs deadline to reduce the 10-kHz event rate to a few Hz. ESOP uses a cache-less Harvard architecture with multiple ALUs and hardware support for zero-overhead loops to achieve high-speed, deterministic stream processing. ESOP's primary purpose is track finding, an algorithm that takes a set of time-stamped interaction points in 3D space and attempts to construct them into linear and lines flight paths that point back to the beam collision point. Each potential input event is described by 24 bytes of data. For a 10-kHz input event rate, the ESOP processors must handle a data rate of 240 kB/s.

9.2.2. The CDF Experiment at the Tevatron (ca. 1987)

CDF is an experiment at Fermilab created to study the behavior of high-energy proton-anti-proton collisions. It was designed for the first run of the Tevatron collider. CDF used a 3-level trigger system,

with hardware-based L1 & L2 triggers and a software-based L3 running on a computer farm [36]. One of the important goals of the design of CDF's trigger and DAQ were to provide flexibility to allow the systems to be modified after construction. This proved to be a valuable strategy, as the Tevatron has remained in use through various upgrades for over 20 years. To meet the goal of flexibility, the CDF trigger system was designed with modularity in mind, and individual trigger functions were implemented on separate circuit boards.

The L1 trigger generated triggering decisions based on 5 components:

- Electromagnetic (ECAL) energy, hadronic (HCAL) energy, and total ET sums
- Imbalance in energy between ECAL and HCAL
- The existence of non-deflected particle tracks and muons
- Presence of beam-beam or beam-gas interactions
- Hits recorded in small-angle silicon counters

Most of the L1 systems were implemented using emitter-coupled logic (ECL) ASIC technology. The exception was for the adder circuitry. Because calculation of sums was a time-critical step, adders were implemented using analog designs, which were capable of lower latency at the time. The results of the various L1 components were used to index an LUT that provided the final L1 decision. The L1 system had a deadline of 3.5 μs.

The L2 trigger system was based on a group of Mercury processors connected via a shared processor bus. Unlike the ESOP processors, the Mercury modules were generally non-programmable, fixed-function ASICs, however modules could be swapped or added to extend the function of the trigger. The L2 added track-finding processing to its algorithms and had an extended time budget of 10 μs.

An important aspect of CDF's trigger and DAQ architecture was its communication bus. CDF's DAQ system was based on FASTBUS, an IEEE standard bus technology developed in the mid-1980s. FASTBUS was built on a segmented serial architecture and used high-speed ECL logic rather than CMOS or TTL. FASTBUS was a significant advancement in DAQ bus architecture for several reasons. First, the use of segmentation allowed for scaling to higher frequencies, meaning faster data acquisition

and higher resolution detectors were possible. Second, the segmentation made the bus modular, allowing for future expansion as needed for the experiment and its upgrades. Finally, the use of a standard bus interface rather than a proprietary design made it easier to reuse communication electronics in other experiments.

9.2.3. The DELPHI and L3 Experiments at the Large Electron-Positron Ring (ca. 1989)

The Large Electron-Positron storage ring (LEP) was a particle accelerator built at CERN in the late 1980s to study electron-positron annihilations at center-of-mass energies between 80 and 200 GeV. LEP hosts four experiments, ALEPH, DELPHI, OPEL, and L3. Each of these experiments were designed to investigate similar physics, however each uses a different set of detectors. Because all the experiments use the same accelerator and have similar goals, their trigger systems have many similarities, therefore not all aspects will be discussed in depth. Each of them have a 45 kHz bunch crossing rate[1], each features a segmented processing with parallel hardware systems set up to process different physical segments of the detector, and each has a trigger architecture with 3 or 4 levels before DAQ readout. The frequency and properties of background events caused by beam-gas interactions and off-momentum collisions with the vacuum chamber were not well known prior to the start of real experiments, therefore the designers of the trigger systems needed to incorporate a great deal of flexibility so they could readjust the trigger parameters after collecting the first experimental data without having to completely redesign the trigger [37].

[1] The particle beams in an accelerator are not are not uniformly distributed. Rather, the beams contain separated groups of tightly packed particles – or 'bunches'. The bunch crossing rate refers to the frequency at which bunches from one beam collide with bunches from another beam. The product of the bunch crossing rate and the number of collisions per bunch (determined by luminosity) gives the collision rate of the accelerator.

The L3 experiment's charged particle trigger was composed of three stages. Due to the nature of the L3 experiment's trigger and DAQ system, it had very stringent latency deadlines for its L1 trigger

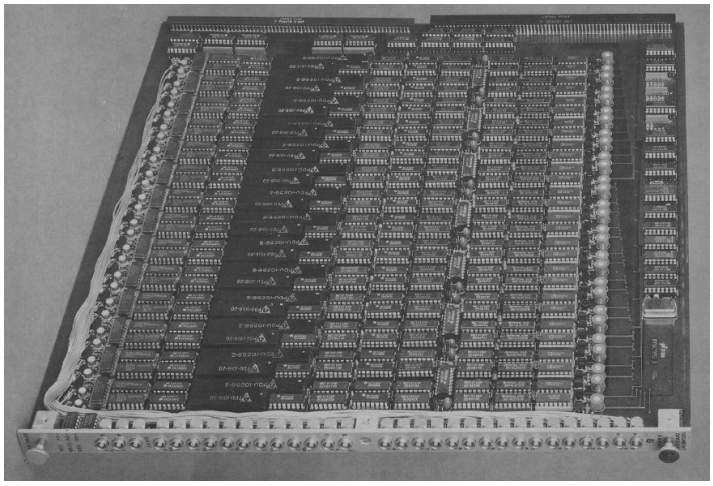

Figure 54. One of the many L1 processing boards from the L3 experiment [14].

decision. Total computation time was less than 1 µs. The charged particle trigger incorporated LUTs and FPGAs that were used to perform track finding and counting of co-planar tracks, making it one of the earlier systems to incorporate FPGA technology in trigger processing [38]. A picture of a FASTBUS-based L1 trigger board from the L3 experiment is shown in Figure 54.

The L1 trigger in the DELPHI experiment was primarily based on RAM-based LUTs. To help reduce the amount of memory needed to implement primary processing in LUTs, the DELPHI system arranged LUTs into a tree structure and multiplexed access to RAM modules [204]. A diagram of this configuration is shown in Figure 55. The use of multi-stage look-ups as a means to reduce LUT size is a technique that is still used in modern systems [50]. The L1 trigger was responsible for reducing the event rate from 90 KHz to 700 Hz in 3.2 µs [205]. DELPHI implemented three more trigger levels: an L2 trigger based on simple Boolean equations applied to the results of the L1 decisions from each detector. The L3 and L4 triggers were based the Motorola 68020 processor, also discussed in the description of the trigger for the SLD experiment [206].

9.3. Trigger Systems of the 1990s

Improvements in processing power in the 1990s allowed for increased event rates and higher resolution detectors. Many triggering systems began adopting standard, off-the-shelf components as the capabilities of digital signal processing (DSP) chips increased.

9.3.1. The SLD Experiment at the SLAC Linear Collider (ca. 1991)

The Stanford Large Detector (SLD) was the main detector at SLAC's Linear Collider (SLC), built to detect Z bosons created in electron-positron collisions. The SLD had a few interesting features that impacted the design of its DAQ process. It was the first detector to implement track finding using a large-

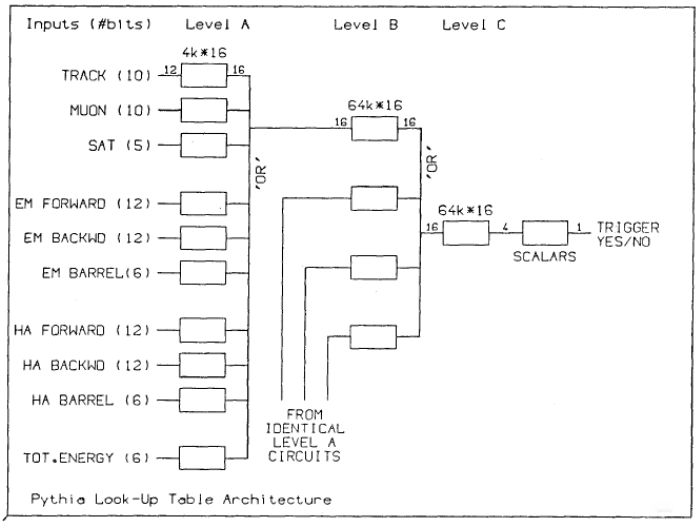

Figure 55. DELPHI's Tree-based Look-up Table [135].

scale CCD-based vertex detector in a particle accelerator [207]. Instead of using multi-wire chambers or scintillators to detect particle flight, vertex detectors use charge-coupled devices (CCDs). CCDs accomplish a similar task as scintillators-photodetector counters; when a particle contacts a CCD, it leaves behind a charge in its wake, allowing hit points to be directly digitized. CCDs are the same technology implemented in many cameras and imaging devices. The transition to CCDs allowed for faster response times and higher resolution. The SLD's vertex detector was capable of producing a 120 Mega-

pixel image for each bunch crossing. However, this higher resolution also means more data to process and higher bandwidth in the trigger and DAQ.

The SLC featured an unusually slow bunch crossing rate of 180 Hz, but events were considered to span the entire detector. A single event thus required a readout of the full detector, which featured an 8000-wire drift chamber, a Cerenkov ring, and multiple calorimeters in addition to the large vertex detector. Despite the low bunch crossing rate, the resulting data for each event were 89 MB (65% of which comes from the vertex detector), producing a maximum data rate of 16 GB/s in the DAQ. These features resulted in atypical trigger architecture. Rather than having a hardware-based L1 trigger to reduce the event rate, the SLD's ASIC-based hardware is only used for data compression and pre-scaling. The primary trigger was implemented in software, a design that was only possible due to the low event rate.

The SLD's primary trigger used an array of commercial-off-the-shelf (COTS) Motorola 68020 processors, part of the popular 68K series of 32-bit general-purpose processors. Processing time on in the primary trigger was non-deterministic and depended on the degree of particle deposition and pile-up in the detector. In some cases, trigger processing does not finish in time to record the next bunch crossing and misses it, resulting in dead time for the DAQ; however this is limited in practical application to less than 10% [208]. Final trigger processing is done on a farm of micro-VAX processors Each event required an average of 2100 MIPS (Millions of Instructions Per Second) to process, thus the trigger would need to be capable of 378,000 MIPS to completely prevent trigger-induced dead time in a worst-case scenario. Given that high-performance VAX computing systems of the time were capable of ~40 MIPS per processor [209], this required a large computing farm.

9.3.2. The NA48, WA98, and NOMAD Experiments at the SPS (ca. 1992-1997)

The Super Proton Synchrotron (SPS) is a proton-anti-proton accelerator located at CERN. Unlike many other accelerators which are built for a small number of permanent experiments, the SPS was home to nearly 200 experiments before it was subsumed in the Large Hadron Collider (LHC). Most of SPS's experiments were designed for a single purpose and only operated for a span of several months to a few

years. As a result, flexibility was less of a concern for most SPS triggers, though architectures of previous experiments were sometimes re-used to reduce development time [210].

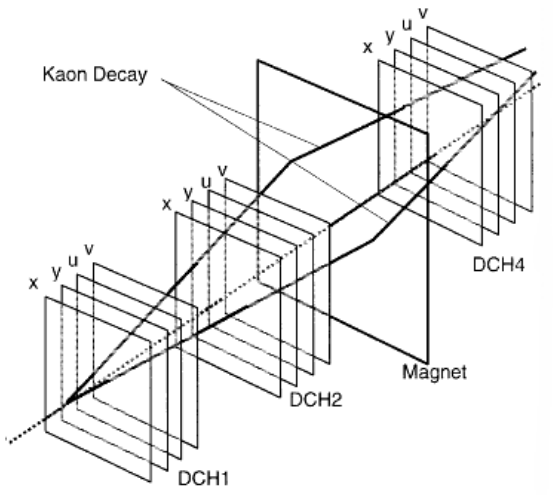

Figure 56. Tracking sensors used in the NA48 experiment [179].

The WA98 experiment was designed to measure the behavior of photons and charged particles in heavy ion reactions. It used a slightly modified version of the system created for the WA93 experiment. Compared to most of the other trigger systems discussed in this paper, its logic was quite simple, and used only ASIC-based hardware modules. An interesting facet of the WA98's system is the inclusion of a Pulse Trigger module. The Pulse Trigger was a module that could be used to generate test signals to calibrate the rest of the trigger by stimulating the detectors using LED, LASER, radioactive, and electronic sources.

NA48 was an experiment designed to study CP violation in a decay mechanism of a kaon into a pair of pions. The most interesting aspect of NA48's trigger architecture is its level-2 trigger. The L2 trigger must reduce the event rate from 100 kHz to 10 Hz with a latency of 100 µs. Its decision process is based on the reconstruction of data from eight parallel sense wire planes to track the decay of the kaon into pions in 3-dimensional space. The sensor system used for this is shown in Figure 56. The L2 trigger reconstructs these paths based on hit data in drift chambers DCH1, DCH2, and DCH4. The architecture of

the trigger is shown in Figure 57. Data from the drift chambers is first passed to FPGA-based Coordinate Builders which collect data from neighboring wires, calculate the time-of-flight, and use this data to index into a LUT that provides a coordinate value. This data is placed in FIFO queues and sent over a fully connected optical crossbar to the first available Event Worker node. Transmission of event data between the queues was done using a standard Xon/Xoff producer-consumer communication protocol. This asynchronous communication takes advantage of the fact that event arrival times follow a Poisson distribution, allowing the processing to be spread out evenly in time, rather than over-provisioning the hardware.

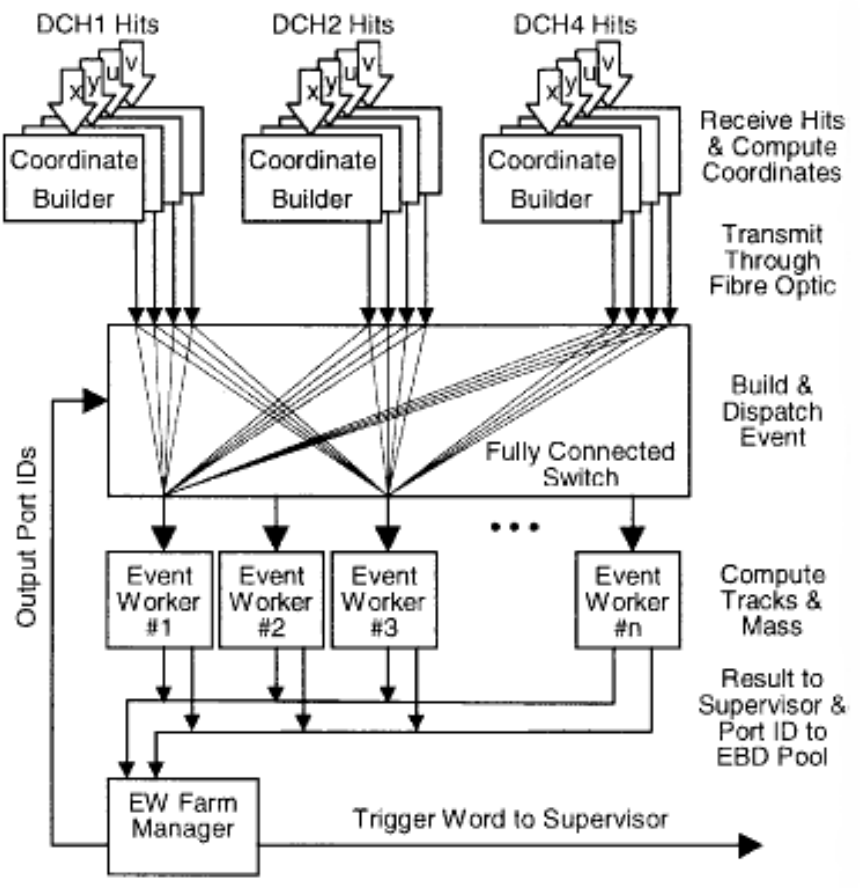

Figure 57. Architecture of the NA48 trigger [179].

Event worker nodes are based on the Texas Instruments C40 DSP processor, with up to 16 nodes supported in the system. The C40 processor was selected based on its MIMD capabilities, however when it was proposed that the event rate of the experiment was doubled, requiring upgrades to prevent the system from being overwhelmed with dead time due to insufficient processor resources, the designers argued that RISC-based CPUs would provide a more attractive platform for upgrades [211]. Their primary argument was that the programming model for a MIMD DSP made development of the L2 software too difficult and time-consuming. This move towards more programmable processors and concern for software complexity has been a consistent trend in trigger designs.

Unlike the experiments discussed so far, the NOMAD experiment was not based on electron-positron or proton-anti-proton collisions. Instead, its physics were based on interactions of a neutrino beam passing through a 3 ton target. Neutrinos rarely interact with other matter, therefore the event rates (both desired events and background events) tend to be quite low. In practice, NOMAD generated just 500,000 events per year [212]. As a result, trigger systems for neutrino-source experiments tend to be relatively simple. NOMAD also had a bunch crossing rate of only two injections every 14 s. Because of the large amount of time between bunch crossings, NOMAD is able to cope with a single-level trigger [213]. This trigger is built using programmable logic devices (PLDs) and programmable array logic (PALs) from Altera's MAX series. PALs provide some of the flexibility of FPGAs with lower device size and cost. PLDs and PALs were sufficient since the NOMAD trigger was primarily based on Boolean combinations of "veto" bits generated from the detector primitives.

9.3.3. The BaBar Experiment at the PEP-II Collider (ca. 1999)

BaBar was an experiment designed to study CP violation at the PEP-II electron-positron collider at SLAC. Compared to other accelerators, it had an exceptionally high bunch crossing rate of 238 MHz [214], delivering continuous data to the trigger and DAQ system. Accommodating this high continuous collision rate puts a lot of pressure on the L1 trigger system. The trigger systems were also designed to operate at up to 10x their nominal event rates, anticipating the potential for upgrades to the experiments

luminosity. A high-level diagram of the BaBar trigger is shown in Figure 58. The L1 trigger is a parallel, pipelined system based on FPGA hardware, designed to operate with less than 10% dead time. A trigger primitive generator (TPG) compresses data from the electromagnetic calorimeter and drift chamber and serves as the input to the L1 in the form of compressed calorimeter tower energy sums and drift chamber track segments. Data from the ECAL is formed into clusters around energy deposits representing isolated particles. The ECAL data is partitioned in segments of 5x5 calorimeter towers to the L1 FPGAs. Each L1 processor forms clusters by calculating all 2x1 tower sums within its partition. Data from the drift chamber is used to reconstruct simple particle tracks. Track reconstruction is performed by linking track segments together to form linear tracks that span the drift chamber. These processes are illustrated in Figure 59. After the initial processing, the L1 performs track counting, p_{xy} cuts, energy cuts, and track-cluster matching to make the L1 decision. The L1 trigger produces a 2 kHz event rate with a latency of 11-12 μs. Space is reserved for an optional L2 trigger to further reduce the rate of events that must be acquired by the DAQ. This optional trigger may perform more advanced track reconstruction based on a silicon vertex tracker (SVT).

Data that has been filtered by the L1 (and optionally the L2) is read out into an event builder that formats the data. Before the events are written to mass storage, they are further filtered by the L3 DAQ processor farm. Because it can take a long time to fully read out events from the detectors, the if no dedicated L2 trigger is in place, L3 system is capable of performing a fast L2 filtering while readout is still taking place, provided that L3 event rates are low enough. This fast filter operates using data from the SVT to perform primary vertex position cuts. The L3 also performs slower filters regardless of the L2 trigger mechanism based on full event data from the detector read-out.

The L3 trigger/DAQ farm is built with 10 64-bit PowerPC processors that are capable of communicating via a token-ring architecture. When designing the L3 trigger, PowerPC processors were

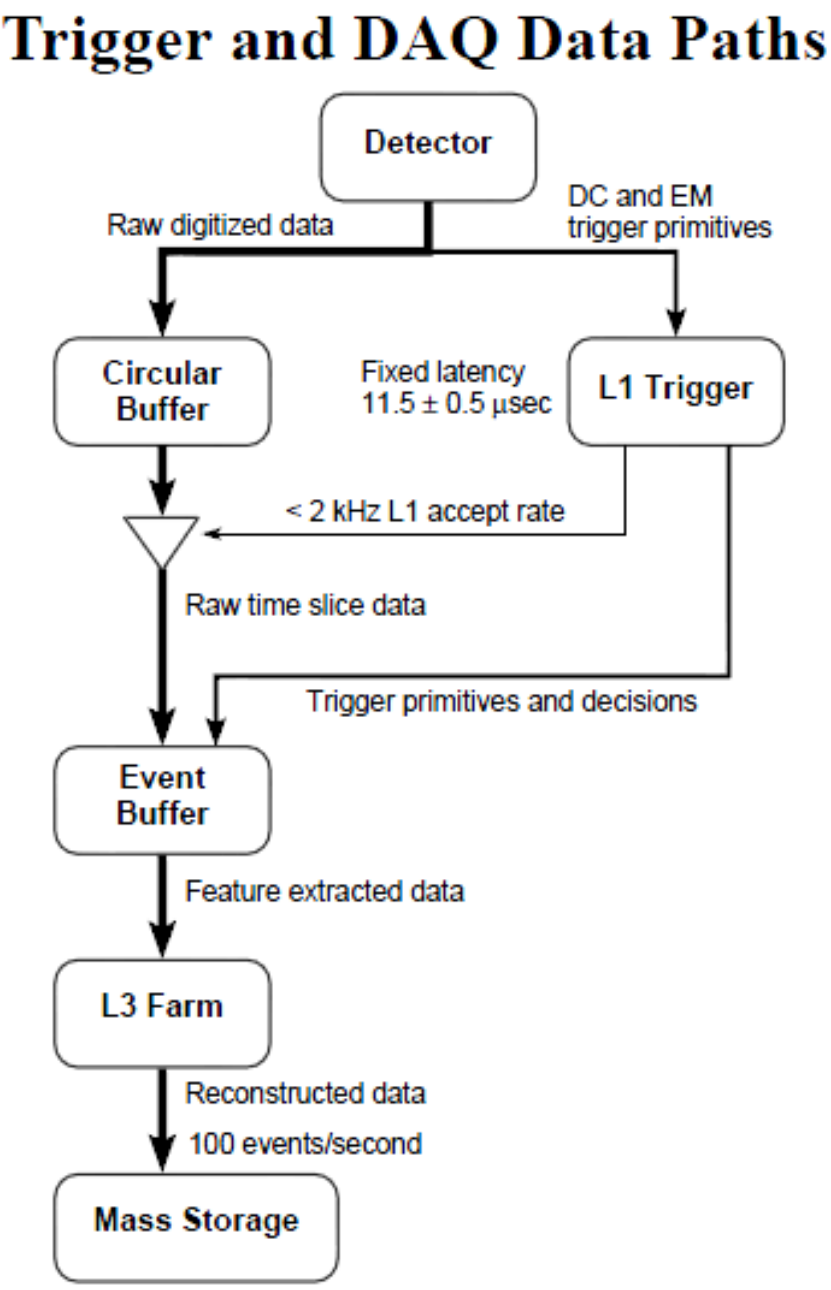

Figure 58. High-level diagram of the BaBar Trigger and DAQ system [147].

chosen over DSPs for two main reasons: the large data cache available in the PowerPC processor is useful

for the data formatting and feature extraction processes that must be performed, and the multipliers available in the processor are capable of single-cycle integer and floating point multiplication, valuable in the reconstruction algorithms.

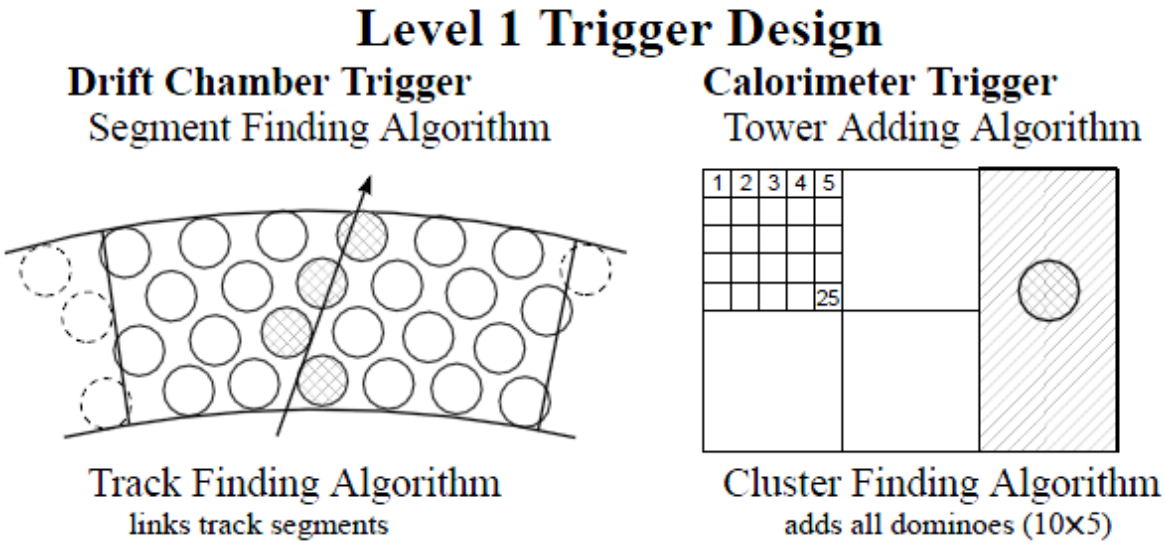

Figure 59. BaBar L1 trigger algorithms [147].

9.4. Trigger Systems of the 2000s

As physicists strove to discover more rare phenomena, the event rates needed for their experiments (and correspondingly, the data throughput of their trigger systems) increased substantially. At the same time that FPGAs were being adopted in greater numbers, ASICs were still needed to cope with the tremendous data rates. These data rates also put a greater focus on the interconnection systems used in triggers. Software triggers were being pushed from application specific instruction processors (ASIPs) and DSPs to general-purpose processors, driven by a goal of improving programmability.

The PHENIX and STAR Experiments at the Relativistic Heavy Ion Collider (ca. 2000)

PHENIX and STAR are the two largest detectors located at the Relativistic Heavy Ion Collider (RHIC). As its name suggests, RHIC is an accelerator that performs physics based on the collisions of ions. It supports many different modes of operation based on the ions being used, which can range from

simple hydrogen ion (proton) collisions up to gold ion collisions. The RHIC operates at a bunch crossing frequency of 9.4 MHz. The event rate and amount of data generated per event depends heavily on the type of collisions. This is illustrated in Figure 60. Heavier ions produce fewer collisions, however these heavy ions also have higher energy and more mass. Therefore, when heavy ions collide, they produce a very large number of post-collisions particles that must be tracked and identified. This unique feature of heavy

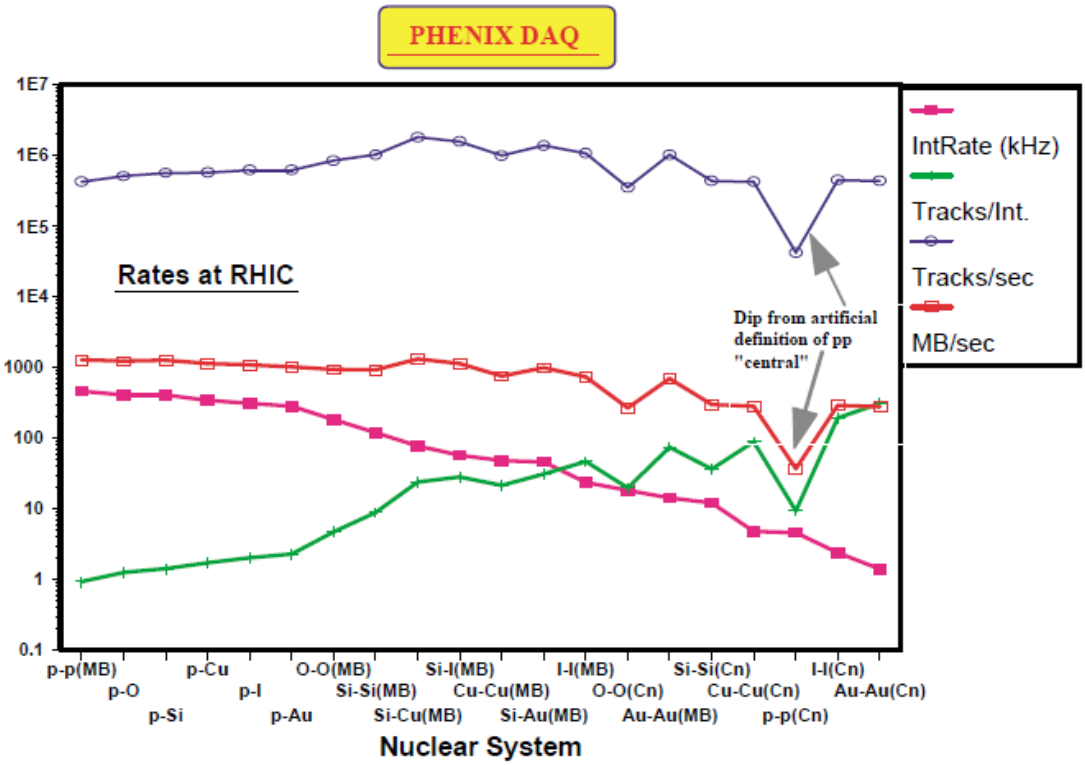

Figure 60. Event rates in the PHENIX experiment for different ions [146].

ion colliders is reflected in the design of the PHENIX and STAR triggers.

The PHENIX experiment was designed to study the transition of ordinary matter into a special state known as quark-gluon plasma. Although the RHIC has a bunch crossing rate of 9.4 MHz, the interaction event rate ranges from only 500 Hz to a few kHz. The PHENIX trigger is designed in a flexible way to handle events from different types of ion collisions. The L1 trigger has a latency of 36 µs, which is much longer than the L1 latencies in most proton-anti-proton and electron-positron colliders. The L1 is

pipelined, allowing for dead-time-free operation for most collision types and up to 5% dead time at peak luminosity [215]. The L1 modules are built on VME boards using a mixture of ASICs and Actel and Xilinx FPGAs. The L1 is broken into two levels: a local L1 that receives data from the detectors on 8 1.2 Gb/s fiber channels per board and performs compression on this data, and a global L1 that generates the triggering decisions based on the compressed data from the local L1. The global L1 implements its algorithms by indexing into pre-programmed SRAM-based LUTs that can be accessed via a programmable crossbar. SHARC 21062 DSPs were also used for data transmission.

An interesting aspect of the PHENIX trigger is that it kept copies of its decision data from each pipeline stage in the L1 trigger in a FIFO register system. These registers could be read out from any stage to analyze the trigger output and detect errors due to hardware failures. Data could also be pushed into the registers to perform testing of downstream portions of the trigger. This functionality is very similar to the concept of register scan chains used in the field of Design for Testability. Because of the

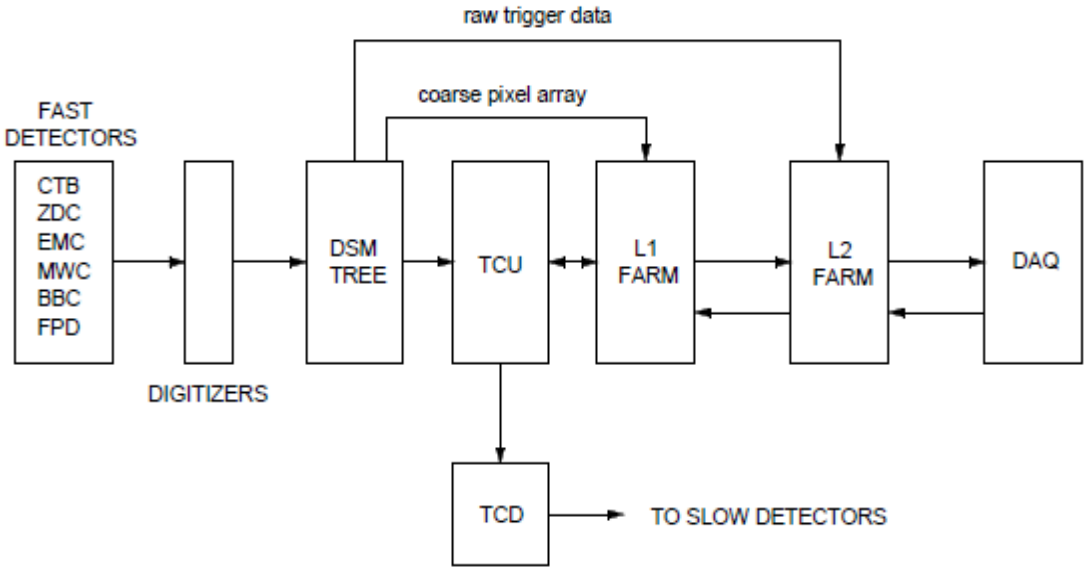

Figure 61. Data acquisition in the STAR experiment [147].

long latency of the trigger, these FIFOs held data for 1024 separate bunch crossings at any one time, which probably made this implementation costly in terms of FPGA resources.

The STAR detector is another experiment at the RHIC. The number of events per bunch crossing in STAR is very low – an average of just one event per crossing. Because of the low event rate, the STAR trigger is able to implement an atypical token-based triggering system [216]. The STAR detectors are split into two groups: fast read-out and slow-readout. It is not possible to read the slow detectors on every bunch crossing because it takes too long to extract the data. The STAR trigger analyzes data from the fast detectors and uses it to estimate which slow detectors, if any, should be read out for that bunch crossing. By selectively reading the slow detectors, dead time is reduced. The process works as follows: An L0 Data Storage and Management (DSM) board performs initial computation on digitized detector data from the fast trigger. This processing is implemented in pipelined ASICs that add the number of hits in the central trigger barrel and in multi-wire counters. Output from the DSM is fed to the Trigger Control Unit (TCU) which uses a 256K-entry LUT to select which of the slow detectors to activate, and sends a 4-bit encoded trigger command that starts readout from those detectors (if the DSM does not recognize any event, none of the detectors are selected). While amplification and digitization of the slow detector data is taking place, the L1 trigger uses a coarse pixel array from one of the trackers to attempt to determine whether the event sensed by the DSM is a real physics event or the result of a beam-gas interaction. If the latter, it can send a veto signal and abort readout of the slow detectors. The L1 makes its veto decision within 100 μs. In the next 5 ms, the L2 trigger performs reconstruction on a fine pixel array. A diagram showing the data flow in the STAR trigger is shown in Figure 61.

The L1 and L2 are both based on Motorola 2306 VME processors running the VxWorks real-time OS. To ensure that there is enough processing resources available to finish event reconstruction within the latency deadline, a token-based processor arbitration method is used. When the TCU issues a trigger command, it also assigns a token to the event associated with the command. The TCU has a limited number of tokens that can be active at any one time to avoid overloading the L1 and L2 triggers, and tokens are returned after an event has been reconstructed in the L2.

9.4.1. The H1 Experiment at the upgraded-luminosity HERA Accelerator (ca. 2001)

H1 was a detector used in proton-lepton collision experiments at the HERA accelerator at DESY. It was a general-purpose device that consisted of silicon vertex trackers, liquid argon and scintillating fiber calorimeters, jet chambers, and muon detectors. In 2001, HERA was upgraded to increase its beam luminosity by a factor of 5 [217]. To deal with the higher event rates, H1 was upgraded with a track-based trigger system called the Fast Track Trigger (FTT). The purpose of the FTT was to create high-resolution 3-dimensional tracks to aid in the efficiency of triggering on charged particle events [218]. To accomplish

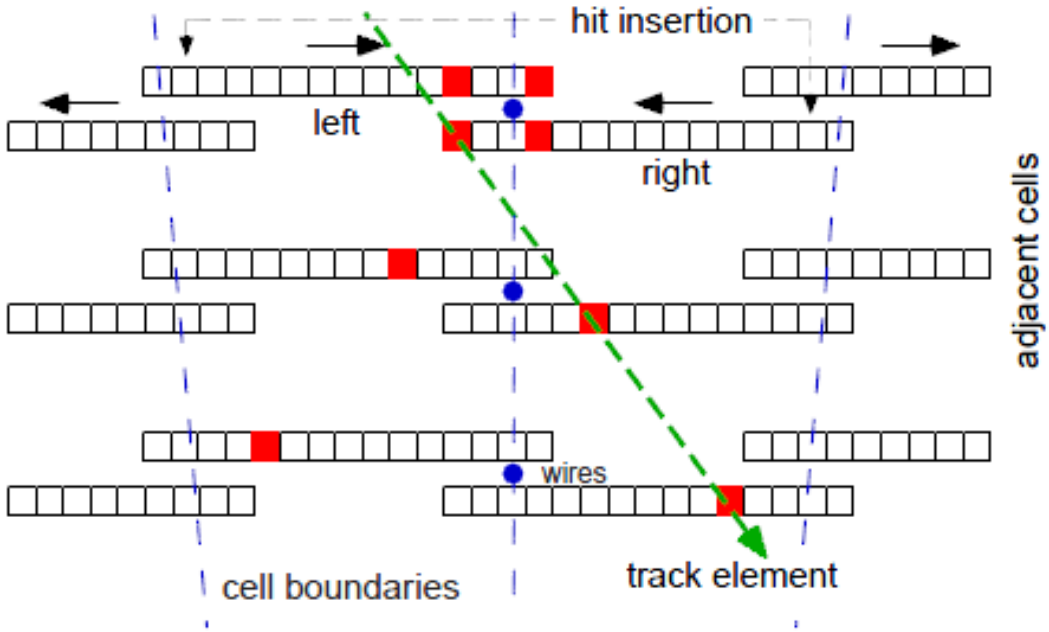

Figure 62. Segment reconstruction scheme in the H1 at HERA [151].

this, it needed to match the existing trigger's L2 latency of 22 µs. The FTT was integrated into the existing triggers 3-level trigger system. This trigger used L1 & L2 levels to select events for readout. After the L2 decision, data readout starts from the detectors. The L3 trigger operates in parallel with readout and is only capable of aborting the acquisition of L2 events.

The L1 portion of the FTT creates rough track segments based on hit data from a wire chamber. An illustration of the segment reconstruction process is shown in Figure 62. In this figure, the vertical dashed

lines represent the detector wires and the horizontal cells are shift registers. Hit data is shifted into the registers. The system then tries to find a line that passes through multiple hit points in the registers if the register cells were to be placed in 3-dimensional space according the location of the associated wire and the time the hit was shifted in. An example of a diagonal track is shown in the figure. Because it would be computationally expensive to try to calculate all possible tracks, reconstruction is done by reading out the shift register data and matching it against pre-defined hit patterns. In many other trigger systems this type of operation would be done by forming the data into an address and indexing into a LUT. A disadvantage of this approach is that it may require very large memories to hold the LUT. The FTT system instead uses content-addressable memories (CAMs). In a CAM, a data word is given instead of an address, and the memory either returns a bit to indicate that the data matched a word in memory (un-encoded mode) or returns a list of addresses that contain that data (encoded mode). This operation is useful for pattern matching applications that often appear in track finding algorithms. The FTT uses an Altera-based FPGA that featured embedded CAMs.

A higher resolution track reconstruction algorithm is performed in the L2 portion of the FTT. Each parallel board in the L2 system an APEX FPGA and SRAM-based LUT to perform initial filtering, then sends the filtered track candidates to a set of four TI 320C DSP processors. A load-balancing algorithm is used to distribute work between the four DSPs. These DSPs try to fit track candidates into planes using polar coordinate-based processing. After the L2 system signals the DAQ to acquire events, data from the L2 is passed to a processor farm of up to 16 COTS processors in the L3 trigger farm. The L3 trigger is given up to 100 μs to perform fine analysis of the L2 tracks and abort the acquisition of any tracks that are not deemed to be worthwhile.

9.4.2. Upgrades to the CDF Experiment for Tevatron Run-II (ca. 2001)

The CDF trigger system, originally commissioned in the late 1980s, was discussed earlier in the paper. In order to continue providing new physics data, the Tevatron underwent a series of upgrades to improve its beam power and luminosity. Here I will discuss some of the changes to the CDF trigger system to

illustrate the evolution of trigger architecture between the start of the experiment and the Tevatron's 'Run-II' operations.

The upgraded accelerated featured a bunch crossing rate of 7.6 MHz. The L1 trigger system was built on FPGA technology and its latency deadline was extended to 5.5 μs. The trigger was upgraded to a dead-time-free design, helping to ease the latency constraint. Dead-time-free operation was achieved by fully pipelining the trigger hardware. The L1 ran at the same clock speed as the bunch crossing rate and featured a 42-stage pipeline [219]. This meant that the DAQ system had to be capable of buffering data from at least 42 separate bunch crossings before event readout. Whereas the original L1 trigger had an output event rate of a few kHz, the upgraded L1 trigger had an output rate of 30-50 kHz. This meant that the upgraded L1 and L2 triggers needed to process an order of magnitude greater event rate than their predecessors. The upgraded L1 used a hierarchal design with separate Muon, Track, and Calorimeter Trigger modules processing data from separate detectors in parallel, then feeding their detector-specific decisions to a Global Trigger that made the final L1 decision. A block diagram of the L1 and L2 trigger architecture is shown in Figure 63.

The L2 trigger uses FPGA-based pre-processors to filter L1 trigger primitives. The L2 features a second-level calorimeter trigger and a trigger for the silicon vertex tracker (SVT). The SVT tracker is placed in the L2 level because, like most trackers, it takes a long time to be able to dump data from the tracker due to the large quantity of data produced by CCD-based detectors. Data from these two L2 triggers as well as data from the L1 trigger are collected in the L2 Global Trigger. The L2 Global Trigger is based on a DEC Alpha processor. The processors are connected with a custom MAGICBUS interface. A general-purpose processor was desirable in the L2 because it could allow for the addition of additional triggering algorithms during Run-II that were not considered before the start of the experiment.

After events pass the L1 and L2 triggers, the DAQ sends the triggered events to the L3 event builder that consists of a farm of 48 COTS processors and is responsible for formatting the event data for transfer to mass storage. Since this does not require significant processing power, a portion of the event builder processors can be used to implement the full offline reconstruction algorithms on a small subset of the

events received from the L2. Performing full online reconstruction of a small number of events can be a good way to provide rapid feedback to the operators of the accelerator and detectors to help them

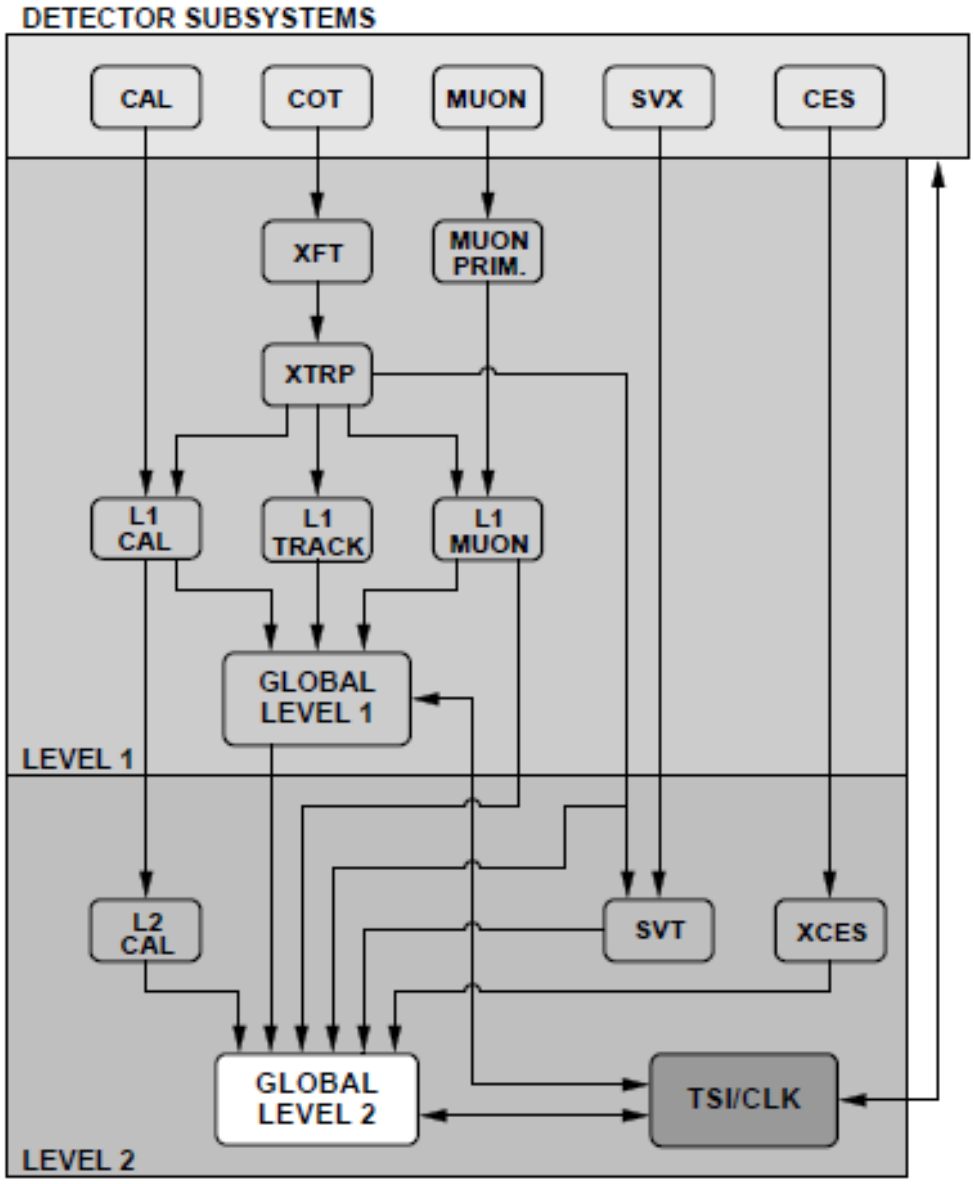

Figure 63. Level-1 and Level-2 trigger architecture for the CDF experiment [152].

calibrate the experimental parameters and trigger algorithms. COTS general-purpose processors are effective in the L3 event builder because they inexpensive and flexible. Another advantage is that

processor farms are easily expandable; the processing power of the L3 could be increased simply by adding more processors until the bus bandwidth becomes the limiting factor.

9.4.3. The CDF Upgrade for Tevatron Run-IIb (ca. 2005)

Just a few years after upgrading the CDF trigger system for Run-II, the L2 trigger (which had remained largely the same since the mid-1990s) received further upgrades. The motivation for these upgrades was only for performance to meet the higher luminosity of Run-IIb, but also to improve maintainability. The designers wanted to eliminate the obsolete DEC Alpha processor and move away from custom bus interfaces to standard-based interfaces. The MAGICBUS protocol was replaced with S-LINK and the DEC Alpha was replaced by a combination of AMD Opteron and Intel Xeon processors. The experimenters found that these new processors were able to reduce the mean event processing time from 13.6 µs for the Alpha to 1.3 µs for the Opteron – a significant improvement that could either allow for more complex reconstruction algorithms, processing of more events at the L2 level, or a reduction in the amount of buffering needed in the DAQ. A histogram comparing the event processing times for the Opteron, Xeon, and Alpha are shown in Figure 64. Despite the fact that the Xeon has a higher frequency than the Opteron, the Opteron provides significantly higher performance. This is attributed to better inter-processor communications buses, showing that data transfer into and out of the CPUs is a bottleneck to performance for when using modern CPUs. Another interesting change comes at the L2 software level. Whereas previous software triggers had used stripped down real-time operating systems to ensure that software met hard deadlines, the CDF designers found that the Linux 2.6 kernel provided good enough real-time scheduling to be used in their L2 [220]. Movement to Linux may have been beneficial since it provides a robust programming and debugging environment for the development of the L2 software.

9.4.4. The ATLAS, CMS, and LHCb Experiments at the Large Hadron Collider (ca. 2009)

At the time of this writing, CERN's Large Hadron Collider (LHC) is the most powerful particle accelerator in the world, and its experiments have some of the most demanding performance parameters for any trigger systems. The LHC is home to four major experiments –ATLAS, CMS, LHCb, and ALICE. ATLAS and CMS are complementary experiments designed with the goal of discovering the Higgs boson and studying theories that go beyond the Standard Model of particle physics. The trigger systems for

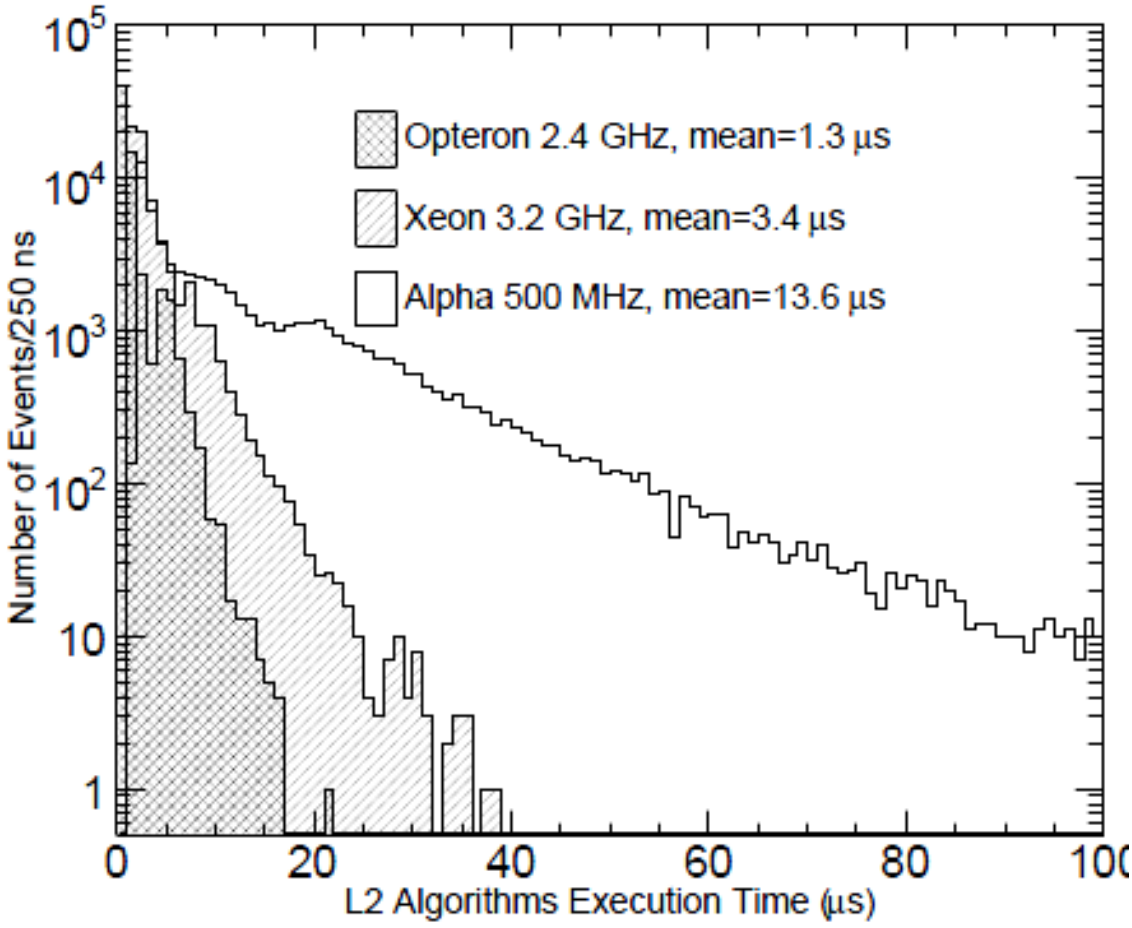

Figure 64. Event processing times for different processor architectures in CDF Run-IIb [153].

these experiments have similar parameters and general architectures. LHCb is an experiment designed to study CP-violation in interactions of particles containing beauty quarks. ALICE is an experiment designed to study the physics of heavy-ion collisions, similar to the STAR and PHENIX experiments discussed earlier in this paper.

ATLAS and CMS are characterized by event rates that dwarf other experiments. Both run at a 40 MHz bunch crossing rate and pick up 20-30 events per crossing, for an average event rate of 1 billion per second. In ATLAS the event size is 1.5 MB and in CMS it is 1 MB, giving them each a data rate of over 1 Petabyte per second [39]. Both ATLAS and CMS feature a hardware-based L1 Trigger and software-based High-Level Trigger (HLT) (ATLAS divides this into a separate software-based L2 trigger and event builder). In CMS, the L1 trigger has a decision latency of 3.2 µs – however because there are significant transmission times to get data from the detector to the trigger, the actual computation time is less than 1 µs. The L1 output event rate is about 100 kHz, and the hardware is primarily based on FPGAs (Virtex-II Pro and Virtex-4 connected by 2 Gb/s data links [40] [41]) and high-speed ASICs for addition, sorting, and table look-up. Use of RAM-based LUTs and FPGAs with programmable firmware was favored because as general-purpose experiments, the trigger algorithms may need to be changed over the course of operation. The ATLAS L1 trigger is also FPGA and ASIC-based and has an output event rate of 75 kHz. Both of the experiments have unusually high input and output event rates. This is largely due to the fact that production of the Higgs boson is predicted to be very rare, limited to about 1 in every 10 billion events [221]. As a result, the systems must process and keep a large number of candidates. The L1 triggers are also fully pipelined to ensure that no rare events are missed due to dead time. The high data rates and large number of events to be processed by the hardware systems required very large scale systems, spanning thousands of FPGAs and ASICs. In order to cope with this hardware complexity, CMS and ATLAS use a variety of software to monitor and configure the triggers [222] [223]. Because fully testing such large systems in the field is very difficult, the experiments also rely on detailed software-based emulators to test the efficiency of trigger algorithms [224].

The high output event rates from the L1 put a lot of pressure on the software-based triggers. ATLAS uses a more conventional architecture with a software-based L2 trigger followed by a switching network connected to more software-based event builders (sometimes called L3 triggers). CMS uses a less conventional approach, with the L1 trigger directly feeding the switching network, which is then connected to a unified HLT as shown in Figure 65. The CMS designers argue that this gives more

flexibility to upgrade CMS's L1 trigger hardware, since it has a flexible interface through the switching network rather than a fixed interface to an L2 trigger [225]. Both experiments have huge software systems based on general-purpose COTS processors. ATLAS has approximately 80 server racks comprising 2300 nodes. Each node uses a 2.5 GHz quad-core Intel CPU with hyperthreading, for a total of 18,400 active threads. CMS's HLT uses approximately 1000 dual-core CPUs [226].

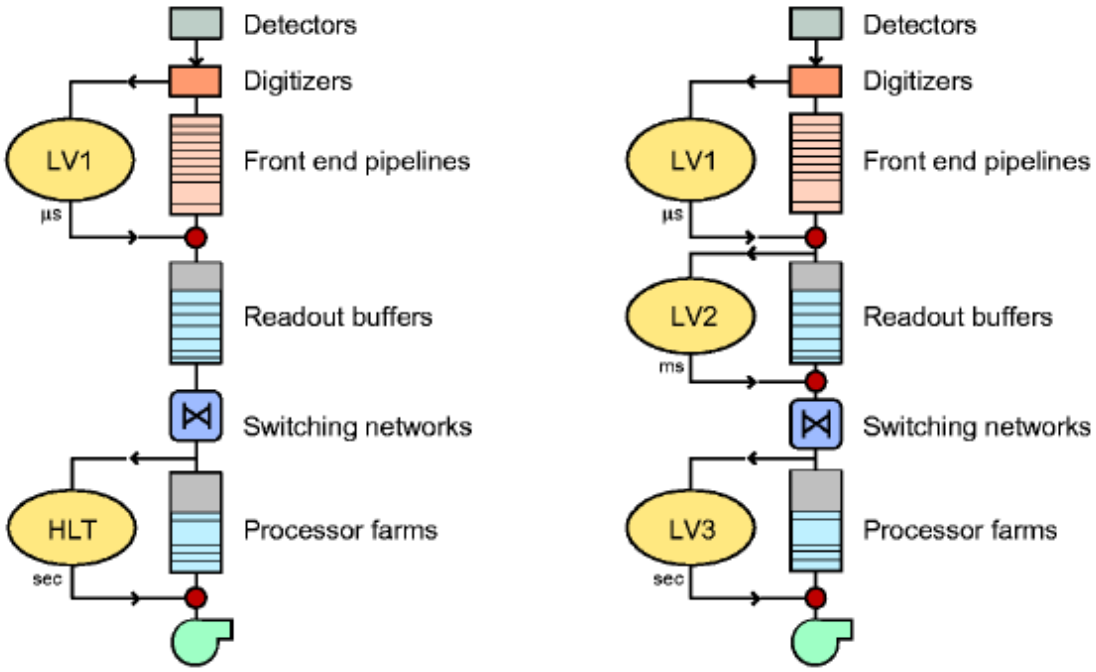

Figure 65. The CMS trigger architecture (left) compared to a traditional architecture (right) [158].

LHCb has somewhat different parameters from the two general-purpose experiments. It has a lower interaction rate of 10 million events per second. It also has a much lower event size of 35 kB, however the output event rate of its L1 trigger is 1 MHz – significantly higher than any other experiment. To reduce the amount of internal trigger data from so many events, LHCb uses an L0 system of FPGA-based readout boards that perform zero-suppression and simple compression to reduce the data rate by a factor of 4-8 [227]. Even after compression it still requires 5,000 optical and analog links with a max bandwidth of 160 MB/s each. Because of the high event rate reaching its HLT, LHCb also needs a very large computing system. The HLT is composed of about 8,000 3-GHz-equivalent CPU cores [228] [229].

9.5. Trends in Future Trigger Design

The design of trigger systems is influenced by two primary factors: the physics properties of the accelerator and experiment (bunch crossing rate, background interaction rate, types of particles that must be identified, etc) and by the engineering requirements and available technology (bandwidth, latency, computational throughput, flexibility, design time, cost).

Examination of trigger systems in this chapter shows that the physics requirements are highly dependent on the type of accelerator and experiment. In general, the strictest requirements are placed on general-purpose experiments, such as ATLAS, CMS, and CDF, because they must have great flexibility to handle multiple experimental goals. ATLAS and CMS are already gearing up for a process of upgrades to the LHC's luminosity, dubbed the Super LHC (SLHC). These upgrades will require several changes to the trigger systems [230]. Higher luminosity means a larger raw event rate from the detectors. Increasing the number of events per bunch crossing will produce greater pile-up and make it more difficult to isolate individual particles. One of the proposed solutions would be to integrate data from the silicon tracker into the L1 level of the trigger. Since trackers produce more data than most other detectors, this will increase the bandwidth requirements of the L1 hardware. It will also make it necessary to perform some coarse-grained track finding in hardware – a task which has previously been done in software. This upgrade is only one part of a planned series of upgrades to the LHC. As has been touched upon already in discussions of the CDF upgrades at Fermilab, the process of upgrading systems after they have already been deployed is a common occurrence. As the scale and cost of particle accelerators has grown very high, it is reasonable to expect that there will be even more desire to upgrade existing accelerators over longer periods of time rather than undertaking the extremely costly process of building new accelerators. This will further motivate the trend of designing flexible trigger systems. It will also encourage the use of modular architectures, such as the one described in the CMS experiment [225]. Modularity allows systems to be more easily upgraded and replaced, whether that is done to meet the needs of new physics experiments or to take advantage of faster and higher density hardware.

From a hardware perspective, one of the most prominent trends has been the move away from custom designs to off-the-shelf components. FPGAs are rapidly replacing ASICs in L1 triggers. The use of FPGAs not only grants greater flexibility due to the programmable nature of FPGA firmware, but also avoids the costly and time-consuming process of verifying and validating the fabrication of a small-run custom ASICs. Some modern FPGAs now provide embedded RISC processors. Although this technology does not seem to be in current use in trigger systems, the ability to run simple high-level algorithms on high-throughput hardware could be an advantage for systems that need to integrate some form of high-level algorithms into their L1 architecture (like the previously mentioned CMS and ATLAS upgrades). Standard bus technologies, particularly VME, are rapidly replacing custom buses, allowing for more modular technology. Some efforts are in place to further modularize systems by using standardized boards and crates to house the electronics.

General-purpose CPUs are also replacing domain-specific processors and DSPs in the software-based triggers. A primary driving force in this transition has been a desire to reduce the time it takes to design and verify the software, which can sometimes be the most time-consuming step in an upgrade process [211] [220]. General-purpose CPUs can speed the development process because they use more familiar architectures, have mature tool-chains, and may have more robust OS and infrastructure support. Another advantage is that general-purpose computing farms can be used for other tasks when the experiment is not operational. Although the prevailing trend is towards general-purpose CPUs, there are some arguments to be made for the use of more domain-specific processors. The CMS HLT, for example, spends a large fraction of its CPU time on muon processing using a Kalman filter kernel [226]. Since that the software triggers in LHC experiments each consist of 1000s of cores, there is potential for significant hardware reduction if the most expensive kernels can be accelerated through some means. Given that trigger processing is highly-parallel and tile-based on some detectors, an argument could be made that the application might map well to a GPU-like architecture, however the current complexity of GPU programming models serves as a high barrier to adoption.

10. Bibliography

[1] I. Dawson, "The SLHC prospects at ATLAS and CMS," *Journel of Physics: Conference Series*, vol. 110, 2008.

[2] P. Wells, "Physics Program of the LHC Upgrades," in *Lepton Photon Conference*, 2013.

[3] Xilinx, Inc., "Virtex-6 Family Overview," 2009.

[4] S. Fartoukh and F. Zimmermann, "The HL-LHC Accelerator Physics Challenges," CERN, 2014.

[5] CMS Collaboration, "CMS TriDaS Project: Technical Design Report; 1, The Trigger Systems," Geneva, 2000.

[6] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.

[7] E. Balas and M. Padberg, "Set partitioning: A survey," *SIAM*, vol. 18, no. 4, pp. 710-760, 1976.

[8] R. Dobkin, A. Morgenshtein, A. Kolody and R. Ginosar, "Parallel vs Serial On-Chip Communication," in *SLIP*, 2008.

[9] Xilinx, Inc., "High-speed serial I/O made simple," 2005.

[10] J. Pawlowski, "Hybrid Memory Cube," *Hotchips*, vol. 23, pp. 1-24, 2011.

[11] S. Kimura, T. Hayakawa, T. Horiyama and M. Nakanishi, "An on-chip high speed serial communication method based on independent ring oscillators," in *Solid State Circuits Conference*, 2003.

[12] C. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379-423, 1948.

[13] L. Brillouin, "Maxwell's demon cannot operate: Information and entropy. I," *Journal of Applied Physics*, vol. 22, no. 3, pp. 334-337, 1951.

[14] W. Zurek, "Thermodynamic cost of computation, algorithmic complexity, and the information metric.," *Nature*, vol. 341, no. 6238, pp. 119-124, 1989.

[15] A. Berut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider and E. Lutz, "Experimental verifiication of Landauer's principle linking information and thermodynamics," *Nature*, vol. 483, no. 1388, pp. 187-189, 2012.

[16] E. Jaynes, "Information theory and statistical mechanics," *Physics Review*, vol. 106, no. 4, p. 620, 1957.

[17] A. Ben-Naim, A farewell to entropy, World Scientific, 2008.

[18] *Scientific American,* vol. 225, p. 180, 1971.

[19] C. Chrysafis and A. Ortega, "Efficient context-based entropy coding for lossy wavelet image compression," in *Data Compression Conference*, 1997.

[20] X. Wu and N. Memon, "Context-based, adaptive, lossless image coding," *IEEE Transactions on Communications,* vol. 45, no. 4, pp. 437-444, 1997.

[21] K. Brandenburg, J. Herre, J. Johnston, Y. Mahieux and E. Schroeder, "Aspec-Adaptive spectral entropy coding of high quality music signals," in *Audio Engineering Society Convention 90*, 1991.

[22] S. Pincus and R. Kalman, "Not all (possibly) "random" sequences are created equal," *Proceedings of the National Academy of Sciences, USA,* vol. 94, pp. 3513-3518, 1997.

[23] G. Croll, "BiEntropy - The Approximate Entropy of a Finite Binary String," in *ANPA-34*, Hampshire, 2013.

[24] M. Nathanson, "Derivatives of binary sequences," *SIAM Journal on Applied Mathematics,* vol. 21, no. 3, pp. 407-412, 1971.

[25] J. McNair, "The Binary Derivative: A new method of testing the appearance of randomness in a sequence of bits," 1989.

[26] W. Chen, J. Zhuang, W. Yu and Z. Wang, "Measuring complexity using FuzzyEn, ApEn, and SampEn," *Medical Engineering & Physics,* vol. 31, no. 1, pp. 61-68, 2009.

[27] A. Kolmogorov, "Three Approaches to the Quantitative Definition of Information," *Problems in Information Transmission,* vol. 1, no. 1, pp. 1-7, 1965.

[28] G. Chaitlin, "On the Simplicity and Speed of Programs for Computing Infinite Sets of Natural Numbers," *Journal of the ACM,* vol. 16, no. 3, pp. 407-422, 1969.

[29] L. Ming and Vitanyi., "Kolmogorov complexity and its applications," in *Handbook of Theoretical Computer Science: Algorithms and Complexity*, 1990, p. 187.

[30] W. Smith, P. Chumney, S. Dasu, M. Jaworski and J. Lackey, "CMS Regional Calorimeter Trigger High Speed ASICs," in *Sixth Workshop on Electronics for LHC Experiments.*

[31] P. Waloschek, "Fast Trigger Techniques," *Physica Scripta,* vol. 23, pp. 480-486, 1981.

[32] G. Lynch, "Remarks on the use of the Spiral Reader," Berkeley, 1967.

[33] K. Johnsen, "The ISR in the time of Jentschke," *CERN Courier,* June 2003.

[34] S. Cairanti, B. Heck, J. Hooper, T. Killian, S. Nielsen, L. Rosselet and B. Schistand, "The Microprogrammable Processor ESOP on the AFS Trigger System," in *Topical Conference on the Application of Microproccessors to High-Energy Physics Experiments,* Geneva, 1981.

[35] D. Jacobs, "Applications of ESOP, a fast microprogrammable processor, in high energy physics experiments at CERN," *Computer Physics Communications,* vol. 22, no. 2-3, pp. 261-267, 1981.

[36] D. Amidei et al., "A Two Level FASTBUS Based Trigger System for CDF," Chicago, 1987.

[37] The ALEPH Collaboration, "ALEPH: A Detector for Electron-Positron Annihiliations at LEP," Geneva, 1990.

[38] P. Bene et al., "First-level Charged Particle Trigger for the L3 Detector," *Nuclear Instruments and Methods in Physics Research,* vol. A306, pp. 150-158, 1991.

[39] T. Kubes, *Data Processing in ATLAS,* 2009.

[40] J. Brooke et al., "Performance of the CMS Global Calorimeter Trigger," in *Technology and Instrumentation in Particle Physics*, Tsukuba, 2009.

[41] M. Stettler, G. Iles, M. Hansen, C. Foudas, J. Jones and A. Rose, "The CMS Global Calorimeter Trigger Hardware Design," in *12th Workshop on Electronics for LHC and Future Experiments,* Valencia, 2006.

[42] M. Bachtis, *SLHC CMS Calorimeter Trigger dataflow and algorithms,* 2008.

[43] M. Bachtis, S. Dasu and W. Smith, "SLHC Calorimeter Trigger at Phase 1," in *SLHC Upgrade Workshop,* Fermilab, 2008.

[44] G. Iles and A. Rose, "A time-multiplexed calorimeter trigger for CMS with addendum," CERN, 2011.

[45] G. Iles, J. Jones, C. Foudas and M. Hansen, "Trigger R&D; for CMS at SLHC," CERN, 2009.

[46] P. Klabbers, et al., "CMS level-1 upgrade calorimeter trigger prototype development," *Journal of Instrumentation,* vol. 8, no. 2, 2013.

[47] W. Smith, "Trigger and Data Acquisition for the Super LHC," in *Proceedings of the 10th Workshop on Electronics for LHC Experiments*, 2004.

[48] "Wisconsin CMS Repository," University of Wisconsin - Madison, 2015. [Online].

[49] M. Bachtis, *SLHC Calorimeter Trigger algorithms,* 2009.

[50] A. Gregerson, et al., "FPGA Design Analysis of the Clustering Algorithm for the CERN Large Hadron Collider," in *FCCM '09,* Napa, 2009.

[51] A. Gregerson, K. Compton and M. Schulte, "High Energy Physics," in *Handbook of Signal Processing Systems, 2nd Ed.*, Springer, 2013.

[52] A. Farmahini-Farahani, A. Gregerson, M. Schulte and K. Compton, "Modular high-throughput and low-latency sorting units for FPGAs in the Large Hadron Collider," in *SASP,* 2011.

[53] P. Klabbers, T. Gorski, M. Bachtis, K. Compton, S. Dasu, A. Farmahini-Farahani, A. Gregerson, R. Fobes and W. Smith, "CMS Calorimeter Trigger Phase I Upgrade," *Journal of Instrumentation*, vol. 7, no. 1, 2012.

[54] L. Rossi and O. Bruning, "High Luminosity Large Hadron Collider: A description for the European Strategy Preparatory Group," CERN, 2012.

[55] M. Cacciari and G. Salam, "Pileup subtraction using jet areas," *Physics Letters B*, vol. 659, no. 1, pp. 119-126, 2008.

[56] J. Jaeger, "Partitioning an ASIC Design into Multiple FPGAs: EE Times," 10 February 2010. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1274719. [Accessed 22 April 2014].

[57] Z. Marrakchi, C. Alexandre and R. Farhat, "Timing-Driven Hybrid RTL/Gate Partitioning for Predictable FPGA-Based Prototyping: EE Times," 3 March 2014. [Online]. Available: http://www.eetimes.com/author.asp?section_id=36&doc_id=1321260. [Accessed 22 April 2014].

[58] J. Stone, D. Gohara and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 3, 2010.

[59] B. Gaster, L. Howes, D. Kaeli, P. Mistry and D. Schaa, Heterogeneous Computing with OpenCL: Revised, Newnes, 2012.

[60] S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Research*, vol. 21, no. 2, pp. 498-516, 1973.

[61] C. Fiduccia and R. Mattheyses, "A linear-time heuristic for improving network partitions," in *Design Automation*, 1982.

[62] S. Hauck and G. Borriello, "An evaluation of bipartitioning techniques," *IEEE Trans. on CAD*, vol. 16, no. 8, pp. 849-866, 1997.

[63] L. Hagen, D.-H. Huang and A. Kahng, "On implementation choices for iterative improvement partitioning algorithms," *CADICS*, vol. 16, no. 10, pp. 1199-1205, Oct 1997.

[64] T. Bui and R. Byung, "Genetic algorithm and graph partitioning," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 841-855, 1996.

[65] L. Hagen and A. Kahng, "New spectral methods for ratio cut partitioning and clustering," *CADICS*, vol. 11, no. 9, pp. 1074-1085, 1992.

[66] D. Van den Bout and T. I. Miller, "Graph partitioning using annealed neural networks," *Neural Networks*, no. June, pp. 192-203, 1990.

[67] D. Johnson, C. Aragon, L. McGeoch and C. Schevon, "Opimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning," *Operations Research*, vol. 37, no. 6, pp. 865-892, 1989.

[68] P. Eles, Z. Peng, K. Kuchcinski and A. Doboli, "System Level Hardware Software Partitioning Based on Simulated Annealing and Tabu Search," *Design Automation for Embedded Systems*, vol. 2, no. 1, pp. 5-32, 1997.

[69] C. Alpert, J. Huang and A. Kahng, "Multilevel circuit partitioning," *CADICS*, vol. 17, no. 8, pp. 655-667, 1998.

[70] R. Kuznar, F. Brglez and K. Kozminski, "Cost minimization of partitions into multiple devices," in *DAC*, 1993.

[71] H. Liu, K. Zhu and D. Wong, "Circuit partitioning with complex resource constraints in FPGAs," in *FPGA*, 1998.

[72] G. Karypis and V. Kumar, "Multilevel Algorithms for Multi-Constraint Graph Partitioning," Minneapolis, MN, 1998.

[73] N. Selvakkumaran, A. Ranjan, S. Raje and G. Karypis, "Multi-resource aware partitioning algorithms for FPGAs with heterogenous resources," in *DAC*, 2004.

[74] K. Taura and A. Chien, "A Heuristic Algorithm for Mapping Communicating Tasks on Heterogeneous Resources," in *Heterogeneous Computing Workshop*, 2000.

[75] G.-M. Wu, J.-M. Lin and Y.-W. Chang, "Generic ILP-based approaches for time-multiplexed FPGA partitioning," *CADICS*, vol. 20, no. 10, pp. 1266-1274, 2001.

[76] S. Adya, et al., "Unification of partitioning, placement, and floorplanning," in *ICCAD*, 2004.

[77] T. Taghavi, et al., "Innovate or perish: FPGA physical design," in *ISPD*, 2004.

[78] R. Tessier and H. Giza, "Balancing logic utilization and area efficiency in FPGAs," in *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, Heidelberg, Springer Berlin, 2000, pp. 535-544.

[79] P. Arato, Z. Mann and A. Orban, "Algorithmic aspects of hardware/software partitioning," in *TODAES*, 2005.

[80] R. Lysecky and F. Vahid, "A study of the speedups and competitiveness of FPGA soft processor cores using dynamic hardware/software partitioning," in *DATE*, 2005.

[81] Z. Mann, A. Orban and V. Farkas, "Evaluating the Kernighan-Lin Heuristic for Hardware/Software Partitioning," *International Journal of Applied Mathematics and Computer Science*, vol. 17, no. 2, pp. 249-267, July 2007.

[82] R. Niemann and P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 165-193, 1997.

[83] A. Chadha, *Benchmark Creation for Multi-Personality and Content-Aware Circuit Partitioning Algorithms*, University of Wisconsin - Madison, 2014.

[84] D. W. Knapp, Behavioral Synthesis: Digital System Design Using the Synopsys Behavioral Compiler, Prentice-Hall, 1996.

[85] S. Palnitkar, Veriloghdl: A Guide to Digital Design and Synthesis, Prentice Hall Press, 2003.

[86] T. Feist, "Vivado Design Suite," Xilinx, Inc., 2012.

[87] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," 2001.

[88] D. Chang, C. Jenkins, P. Garcia, S. Gilani, P. Aguilera, A. Nagarajan, M. Anderson, M. Schulte and K. Compton, "ERCBench: An open-source benchmark suite for embedded and reconfigurable computing," in *Field Programmable Logic and Applications*, 2010.

[89] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. Kent, P. Jamieson and J. Anderson, "The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing," in *Field Programmable Gate Arrays*, 2012.

[90] B. Hendrickson and R. Leland, "The CHACO User's Guide: Version 2.0," Sandia National Labratories, 1995.

[91] A. Soper, C. Walshaw and M. Cross, "A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning," *Journal of Global Optimization*, vol. 29, no. 2, pp. 225-241, 2004.

[92] P. Christie and D. Stroobandt, "The interpretation and application of Rent's rule," *IEEE Transactions on VLSI Systems*, vol. 8, no. 6, pp. 639-648, 2000.

[93] U. Brandes, M. Gaertler and D. Wagner, "Engineering Graph Clustering: Models and Experimental Evaluation," *Journal of Experimental Algorithmics*, vol. 12, pp. 1-26, June 2008.

[94] S. E. Scaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27-64, August 2007.

[95] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75-174, February 2010.

[96] T. Achterberg, "SCIP: Solving constraint integer programs," in *Mathematical Programming Computation*, 2009.

[97] A. Gregerson, A. Chadha and K. Morrow, "Multi-Personality Partitioning For Heterogeneous Systems," in *ICFPT*, Kyoto, 2013.

[98] A. Gregerson, A. Chadha and K. Morrow, "Multi-personality Netlist Mapping and Partitioning for Heterogeneous Devices and System," *Submitted to TODAES.*

[99] "Obtaining hMETIS," Karypis Lab, [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download. [Accessed 10 January 2015].

[100] C. Walshaw, "The Graph Partitioning Archive," [Online]. Available: http://staffweb.cms.gre.ac.uk/~wc06/partition/. [Accessed 9 January 2015].

[101] J. Babb, M. Frank, V. Lee, E. Waingold, R. Barua, M. Taylor, J. Kim, S. Devabhaktuni and A. Agarwal, "The RAW Benchmark Suite: Computation Structures for General Purpose Computing," in *FCCM*, 1997.

[102] S. Asaad, et al., "A Cycle-accurate, Cycle-reproducible multi-FPGA System for Accelerating Multi-core Processor Simulation," in *FPGA*, 2012.

[103] D. Knuth, "Dynamic Huffman Coding," *Journal of Algorithms*, vol. 6, no. 2, pp. 163-180, 1985.

[104] A. Moffat, R. Neal and I. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems*, vol. 16, no. 3, pp. 256-294, 1998.

[105] F. Marcelloni and M. Vecchio, "A simple algorithm for data compression in wireless sensor networks," *Communication Letters*, vol. 12, no. 6, pp. 411-413, 2008.

[106] K. Chen and T. Ramabadran, "Near-lossless compression of medical images through entropy-coded DPCM," *IEEE Transactions on Medical Imaging*, vol. 13, no. 3, pp. 538-548, 1994.

[107] S.-M. Lei and M.-T. Sun, "An entropy coding system for digital HDTV applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 1, no. 1, pp. 147-155, 1991.

[108] C. Patauner, W. Pribyl and A. Marchioro, "A lossless data compression method for an application in high energy physics," in *PRIME*, 2009.

[109] P. Coussey and A. Morawiec, High-Level Synthesis, Springer Netherlands, 2008.

[110] I. Prigogine, G. Nicolis and A. Babloyantz, *Physics Today*, vol. 25, no. 11, p. 23, 1972.

[111] P. Glansdorff and I. Prigogine, Thermodynamic Theory of Structure Stability and Fluctuations, Wiley-Interscience, 1973.

[112] R. Landauer, "Stability and Entropy Production in Electrical Circuits," *Journal of Statistical Physics*, vol. 13, no. 1, pp. 1-16, 1975.

[113] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183-191, 1961.

[114] E. Macii and M. Poncino, "Exact computation of the entropy of a logic circuit," in *Sixth Great Lakes Symposium on VLSI*, 1996.

[115] K.-T. Cheng and V. Agrawal, "An entropy measure for the complexity of multi-output Boolean functions," in *DAC*, 1990.

[116] A. Putnam, A. Caufield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *ISCA*, 2014.

[117] A. Morgenshtein, A. Kolodny and R. Ginosar, "Asynchronous Bit-stream Compression (ABC)," in *EEEI*, 2006.

[118] D. Knuth, "Dynamic Huffman Coding," *Journal of Algorithms*, vol. 6, no. 2, pp. 163-180, 1985.

[119] T. Moon, "Error correction coding," in *Mathematical Methods and Algorithms*, Wiley and Son, 2005.

[120] L. Benini, G. De Micheli, E. Macil, M. Poncino and S. Quer, "Power optimization of core-based systems by address bus encoding," *Transactions on VLSI Systems*, vol. 6, no. 4, pp. 554-562, 1998.

[121] J. Yang, R. Gupta and C. Zhang, "Frequent value encoding for low power data buses," *TODAES*, vol. 9, no. 3, pp. 354-384, 2004.

[122] K. Lee, S.-J. Lee and H.-J. Yoo, "SILENT: Serialized low energy transmission coding for on-chip interconnection networks," in *ICCAD*, 2004.

[123] e. a. J. Babb, "Logic emulation with virtual wires," *CADICS*, vol. 16, no. 6, pp. 609-626, 1997.

[124] A. Gregerson, A. Farmahini-Farahani, M. Schulte, W. Plishker, Z. Xie, K. Compton and S. Bhattacharyya, "Advances in architectures and tools for FPGAs and their impact on the design of complex systems for partitcle physics," in *TWEPP*, 2009.

[125] Xilinx, Inc., "Xilinx DS180 7 Series FPGAs Overview Data Sheet v1.15," 2014.

[126] Xilinx, Inc., "Virtex-5 RocketIO GTX Transceiver User Guide," 2008.

[127] C. Hwang and A. Wu, "An entropy measure for power estimation of Boolean functions," in *ASP-DAC*, 1997.

[128] D. Marculescu, R. Marculescu and M. Pedram, "Information theoretic measures for power analysis [logic design]," *CADICS*, vol. 15, no. 6, pp. 599-610, 1996.

[129] D. Marculescu, R. Marculescu and M. Pedram, "Information theoretic measures of energy consumption at register transfer level," in *International Symposium on Low Power Design*, 1995.

[130] R. Marculescu, D. Marculescu and M. Pedram, "Efficienct power estimation for highly correlated input streams," in *DAC*, 1995.

[131] V. Agarwal, "An Information Theoretic Approach to Digital Fault Testing," *Transactions on Computers*, vol. 30, no. 8, pp. 582-587, 1981.

[132] C. Shannon, "Prediction and entropy of printed English," *Bell Systems Technical Journal*, vol. 30, no. 1, pp. 50-64, 1951.

[133] C. Bennett, "Logical reversability of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525-532, 1973.

[134] D. Miller, D. Maslov and G. Dueck, "A transformation based algorithm for reversible logic synthesis," in *DAC*, 2003.

[135] P. Gupta, A. Agrawal and N. Jha, "An algorithm for synthesis of reversible logic circuits,"

CADICS, vol. 25, no. 11, pp. 2317-2330, 2006.

[136] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," in *DAC*, 2009.

[137] R. Cook and M. Flynn, "Logical Network Cost and Entropy," *IEEE Transactions on Computers*, Vols. C-22, no. 9, pp. 823-826, 1973.

[138] R. Camposano, "Path-based scheduling for synthesis," *CADICS*, vol. 10, no. 1, pp. 85-93, 1991.

[139] K. Roy and S. Prasad, "Circuit activity based logic synthesis for low power reliable operations," *IEEE Transactions on VLSI*, vol. 1, no. 4, pp. 503-513, 1993.

[140] R. Larsen and I. Reed, "Redundancy by coding versus redundancy by replication for failure-tolerant sequential circuits," *IEEE Transactions on Computers*, vol. 100, no. 2, pp. 130-137, 1972.

[141] S. Chiu and C. Papachristou, "A design for testability scheme with applications to data path synthesis," in *DAC*, 1991.

[142] F. Vasey, V. Arbets-Engels, J. Batten, G. Cervelli, K. Gill, R. Grabit, C. Mommaert, G. Stefanini and J. Troska, "Development of radiation-hard optical links for the CMS tracker at CERN," *IEEE Transactions on Nuclear Science*, vol. 45, no. 3, pp. 331-337, 1998.

[143] A. Aloisio, F. Cevenini, R. Giordano and V. Izzo, "Characterizing jitter performance of multi gigabit FPGA-embeded serial transceivers," *IEEE Transactions on Nuclear Science*, vol. 57, no. 2, pp. 451-455, 2010.

[144] J. Berger, U. Frankenfeld, V. Lindenstruth, P. Plamper, D. Rohrich, E. Schaefer, W. Schultz and A. Wiebalck, "TPC data compression," *Nuclear Inst. and Methods in Physics Research A: Accelerators, Spectrometers, Detectors, and Associated Equipment*, vol. 489, no. 1, pp. 406-421, 2002.

[145] Xilinx, Inc, "Xilinx Virtex-7 2000T FPGA and Stacked Silicon Interconnect," 2011.

[146] N. Gershenfeld, "Signal entropy and the thermodynamics of computation," *IBM Systems Journal*, vol. 34, no. 3.4, pp. 577-586, 1996.

[147] C. Bennett, "The thermodynamics of computation - A review," *International Journal of Theoretical Physics*, vol. 21, no. 12, pp. 905-940, 1982.

[148] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Field-Programmable Logic and Applications*, 1997.

[149] C. Alpert, "The ISPD98 Circuit Benchmark Suite," in *International Symposium on Physical Design*, 1998.

[150] G. Nam, C. Alpert, P. Villarrubia, B. Winter and M. Yildiz, "The ISPD2005 Placement Contest and Benchmark Suite," in *International Symposium on Physical Design*, 2005.

[151] N. Viswanathan, C. Alpert, C. Sze, Z. Li, G. Nam and J. Roy, "The ISPD-2011 Routability-Driven Placement Contest Benchmark Suite," in *International Symposium on Physical Design*, 2011.

[152] N. Viswanathan, C. Alpert, C. Sze, Z. Li and Y. Wei, "The DAC 2012 Routability-Driven Placement Contest and Benchmark Suite," in *DAC*, 2012.

[153] K. Murray, S. Whitty, S. Liu, J. Luu and V. Betz, "Titan: Enabling large and complex benchmarks in academic CAD," in *FPL*, 2013.

[154] F. Brglez, "A D&T Special Report on ACM/SIGDA Design Automation Benchmarks: Catalyst or Anathema," *IEEE Design and Test of Computers*, vol. 10, no. 3, pp. 87-91, 1993.

[155] F. Corno, M. Reorda and G. Squillero, "RT-level ITC'99 Benchmarks and First ATPG Results," *IEEE Design and Test of Computers*, vol. 17, no. 3, pp. 44-53, 2000.

[156] T. Pecenka, Z. Kotasek and L. Sekanina, "FITTest BENCH06: A New Set of Benchmark Circuits Reflecting Testability Properties," in *Design and Diagnostics of Electronic Circuits and Systems*, 2006.

[157] M. Litochevski, "IMA ADPCM Encoder & Decoder," December 2014. [Online]. Available: http://opencores.org/project,ima_adpcm_enc_dec.

[158] "OnClassical: Quality WAV Audio Files of Classical Music," Archive.org, [Online]. Available: https://archive.org/details/onclassical-quality-wav-audio-files-of-classical-music. [Accessed 1 January 2015].

[159] GitHub, "Open-Source FPGA BitCoin Miner," 18 July 2013. [Online]. Available: https://github.com/progranism/Open-Source-FPGA-Bitcoin-Miner.

[160] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Consulted*, vol. 1, no. 2012, 2008.

[161] J. Surowiecki, "Cryptocurrency," *MIT Technology Review*, 2011.

[162] M. Mouly, M. Pautet and T. Haug, "The GSM system for mobile communications," Telecom Publishing, 1992.

[163] G. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30-44, 1991.

[164] M. Krepa, C. Gutierrez and A. Tekyildiz, "JPEG Encoder," 25 September 2014. [Online]. Available: http://opencores.org/project,mkpeg.

[165] "Flickr," Yahoo, Inc., 15 December 2014. [Online].

[166] OpenRISC, "ORSoC - FPGA, ASIC, DSP - Embedded SoC Design," 6 August 2012. [Online]. Available: http://www.orsoc.se.

[167] D. Strother, "FPGA Stereo Vision Project," 24 01 2011. [Online]. Available: http://danstrother.com/2011/01/24/fpga-stereo-vision-project.

[168] D. Scharstein and C. Pal, "Learning Conditional Random Fields for Stereo," in *Computer Vision and Patern Recognition*, 2007.

[169] H. Hirschmullter and D. Scharstein, "Evaluation of Cost Functions for Stereo Matching," in *Computer Vision and Pattern Recognition*, 2007.

[170] D. Scharstein, H. Hirschmuller, Y. Kitajima, G. Krathwohl, N. Nesic, X. Wang and P. Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," in *German Conference on Pattern Recognition*, 2014.

[171] J. Pistorius, M. Hutton, A. Mishchenko and R. Brayton, "Benchmarking method and designs targeting logic synthesis for FPGAs," *IWLS*, vol. 7, pp. 230-237, 2007.

[172] S. Bourdeauducq, "Image warping/Texture mapping core," 27 September 2010. [Online]. Available: http://opencores.org/project,warp.

[173] H. Gilbert and H. Handschuh, "Security analysis of SHA-256 and sisters," in *Selected Areas in Cryptography*, Springer Berlin Heidelberg, 2004, pp. 175-193.

[174] A. Agarwal, "Limits on interconnection network performance," *IEEE Transactions on Distributed Systems*, vol. 2, no. 4, pp. 398-412, 1991.

[175] Xilinx, Inc., "Xilinx Core Generator: Achieve high performance results while cutting your design time," 2005.

[176] Xilinx, Inc., "Virtex-5 FPGA User Guide," 2008.

[177] D. Marculescu, R. Marculescu and M. Pedram, "Information theoretic measures for power analysis," *CADICS*, vol. 15, no. 6, pp. 599-10, 1996.

[178] J. Taylor, An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements, University Science Books, 1997.

[179] J. Cong and Y.-Y. Hwang, "Structural gate decomposition for depth-optimal technology mapping in LUT-based FPGA design," in *DAC*, 1996.

[180] R. Brayton, G. Hachtel and A. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264-300, 1990.

[181] A. Taneem, P. Kundarewich and J. Anderson, "Packing techniques for Virtex-5 FPGAs," *TRETS*, vol. 2, no. 3, 2009.

[182] Altera, Corp., *Design Debugging Using the SignalTap II Embedded Logic Analyzer*, 2009.

[183] R. Buskey and B. Frosik, "Protected JTAG," in *Parallel Processing Workshops*, 2006.

[184] J. Teuhola, "A compression method for clustered bit-vectors," *Information Processing Letters*, vol. 7, no. 6, pp. 308-311, 1978.

[185] A. Gersho and R. Gray, Vector quantization and signal compression, Springer Science & Business Media, 1992.

[186] 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.

[187] M. Richter, "Data compression in ALICE by on-line track reconstruction and space point analysis," *Journal of Physics: Conference Series*, vol. 396, no. 1, 2012.

[188] V. Schatz, "Test of a Readout and Compression ASIC for the ATLAS Level-1 Calorimeter Trigger," Heidelberg, 2000.

[189] Z. Aspar, Z. Yusof and I. Suleiman, "Parallel Huffman decoder with an optimized look up table option on FPGA," *TENCON*, vol. 1, pp. 73-76, 2000.

[190] R. Freking and K. Parhi, "Low-memory, fixed-latency Huffman encoder for unbounded-length codes," in *Asilomar Conference on Signals, Systems and Computers*, 2000.

[191] A. Widmer and P. Franaszek, "A DC-balanced, partitioned-block, 8B/10B transmission code.," *IBM Journal of research and development*, vol. 27, no. 5, pp. 440-451, 1983.

[192] Texas Instruments, "LVDS Owner's Manual, Fourth Edition," 2008.

[193] D. Salomon, Data Compression: The Complete Reference, Springer Science & Business Media, 2004.

[194] I. Ross, Personal Communication, 2012.

[195] J. Adelman, "The silicon vertex trigger upgrade at CDF," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors, and Associated Equipment*, vol. 572, no. 1, pp. 361-364, 2007.

[196] F. Chaitlin-Chatelin and E. Traviesas, "Qualitative computing," in *Handbook of Numerical Computation*, 2002.

[197] J. Han and M. Orshansky, "Approximate Computing: An emerging paradigm for energy-efficient design," in *European Test Symposium*, 2013.

[198] A. Gregerson, A. Farmahini-Farahani, W. Plishker, Z. Xie, K. Compton and M. Schulte, "Advances in Architectures and Tools for FPGAs and their Impact on the Design of Complex Systems for Particle Physics," in *Topical Workshop on Electronics for Particle Physics*, Paris, 2009.

[199] A. Gregerson, K. Compton and M. Schulte, "High Energy Physics," in *Handbook of Signal Processing Systems, 1st Ed.*, Springer, 2010.

[200] W. Plishker, C. Shen, S. Bhattacharyya, G. Zaki, S. Kediaya, N. Sane, K. Sudushinghe, A. Gregerson, J. Liu and M. Schulte, "Model-based DSP Implementation on FPGAs," in *RAPID*, 2010.

[201] CMS Collaboration, "First Measurement of the Cross Section for Top-Quark Pair Production in

Proton-Proton Collisions at √s = 7 TeV," *Physics Letters B*, no. 695, pp. 424-443, 2011.

[202] F. Brooks, "A Scintillation Counter with Neutron and Gamma-Ray Discriminators," *Nuclear Instruments and Methods*, vol. 4, no. 3, pp. 151-163, 1959.

[203] H. Gordon et al., "The Axial-Field Spectrometer at the CERN ISR," in *1981 INS International Symposium on Nuclear Radiation Detectors*, Tokyo, 1981.

[204] S. Quinton, R. Lucock, L. Cerrito, P. Chapentier, K. Johansson and L. Lanceri, "An Overview of the First and Second Level Trigger of DELPHI," *IEEE Transactions on Nuclear Science*, vol. 36, no. 1, pp. 390-394, February 1989.

[205] V. Canale et al., "The DELPHI Trigger System at LEP200," Geneva, 1999.

[206] V. Bocci et al., "Architecture and Performance of the DELPHI Trigger System," Geneva, 1994.

[207] G.D. Agnew et. al., "Design and Performance of the SLD Vertex Detector, a 120 MPixel Tracking System," Stanford, 1992.

[208] D. Sherden, "The Data Acquisition System for SLD," in *Intl. Conf. on Advances in Experimental Methods for Colliding Beam Physics*, Stanford, 1987.

[209] R. Longbottom, December 2009. [Online]. Available: http://www.roylongbottom.org.uk/mips.htm.

[210] Y. Lee, "Trigger System for WA98," Geneva, 1994.

[211] S. Anvar et al., "The Charged Trigger System of NA48 at CERN," *Nuclear Instruments and Methods in Physics Research A*, vol. 419, pp. 686-694, 1998.

[212] December 2009. [Online]. Available: http://www.uniurb.it/Phys/NOMAD.

[213] J. Altegoer et al., "The Trigger System of the NOMAD Experiment," Geneva, 1998.

[214] C. Day, R. Jacobsen, J. Kral, M. Levi and J. White, "The BaBar Trigger, Readout and Event Gathering System," in *CHEP95*, 1995.

[215] S.S. Adler et al., "PHENIX On-line Systems," *Nuclear Instruments and Methods in Physics Research A*, vol. 449, pp. 560-592, 2003.

[216] F.S. Bieser, et al., "The STAR Trigger," *Nuclear Instruments and Methods in Physics Research A*, vol. 499, pp. 766-777, 2003.

[217] T. Fuljahn et al., "Concept of the First Level Trigger for Hera-B," *IEEE Transactions on Nuclear Science*, vol. 45, no. 4, pp. 1782-1786, August 1998.

[218] A. Baird, "A Fast High Resolution Track Trigger for the H1 Experiment," *IEEE Transactions on Nuclear Science*, 2000.

[219] T. LeCompte and H.T.Diehl, "The CDF and D0 Upgrades for Run II," *Annu. Rev. Nucl. Part. Sci.*,

vol. 50, pp. 71-117, 2000.

[220] K. Anikeev et al., *CDF Level 2 Trigger Upgrade,* 2005.

[221] F. Pastore, "ATLAS Trigger: Design and Commissioning," 2009.

[222] P. Bell et al., "The Configuration System of the ATLAS Trigger," Geneva, 2009.

[223] A. Sidoti, "Trigger Monitoring at ATLAS," 2009.

[224] V. Ghete, "The CMS L1 Trigger Emulation Software," in *CHEP09,* Prague, 2009.

[225] C. Seez, "The CMS Trigger System," *Eur Phys J C,* vol. 34, no. 01, pp. 151-159, 2004.

[226] R. Covarelli, "The CMS High-Level Trigger," in *CIPANP '09,* 2009.

[227] F. Alessio et al., "LHCb Online event processing and filtering," *Journal of Physics: Conference Series,* vol. 119, 2008.

[228] A. Navarro and M. Frank, "Event reconstruction in the LHCb Online cluster," Geneva, 2009.

[229] H. Ruiz, "The LHCb Trigger: Algorithms and Performance," in *TIPP09,* Nima, 2009.

[230] F. Parodi, *Upgrade of the Tracking and Trigger System at ATLAS & CMS,* 2009.

[231] W. Dwinnell, "Data Mining in MATLAB," 17 November 2006. [Online]. Available: http://matlabdatamining.blogspot.com/2006_11_01_archive.html.