

UNIVERSITY OF CALGARY

The Sieve Problem in One- and Two-Dimensions

by

Kjell Wooding

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF MATHEMATICS AND STATISTICS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

April, 2010

© Kjell Wooding 2010

Abstract

This thesis is concerned with the development of tools and techniques for solving systems of simultaneous congruences—the congruential sieve problem—in both one- and two-dimensions. Though many problems in number theory can be reduced to an instance of the congruential sieve problem, one problem in particular—that of *primality proving*—will be examined in detail.

In previous work [WW06] this author provided numerical evidence for a conjecture that primality may be proved with complexity $(\log N)^{3+o(1)}$ using quantities known as pseudosquares and pseudocubes. This thesis examines an alternate definition of pseudocube—the Eisenstein pseudocube—which leads to a more efficient primality proving method for primes $p \equiv 1 \pmod{3}$. In particular, in this thesis, we:

- develop the notion of an Eisenstein pseudocube, and an associated primality proving algorithm;
- reduce the problem of finding Eisenstein pseudocubes to an instance of the two-dimensional sieve problem;
- extend the Calgary Scalable Sieve (CASSIE) toolkit to solve instances of a two-dimensional sieve problem;
- develop a general-purpose hardware framework for implementing custom computing devices on Xilinx Field Programmable Gate Array (FPGA) devices;
- design and implement FPGA-based sieve device using this framework;
- evaluate the performance of the prototype hardware for solving two-dimensional sieve problems; and
- enumerate Eisenstein pseudocubes using these tools.

Acknowledgements

There are so many people to thank for this thesis, I hardly know where to start.

Of course, thank-you to my supervisors: Hugh Williams and Vassil Dimitrov. Thanks for your insight, your guidance, your input, and your inspiration. Hugh, an extra thank-you for putting up with me through two degrees, a ridiculous amount of interdisciplinary paperwork, and a lot of really silly questions. Thank-you also to our amazing support staff: Helen, Yanmei, Betty, Joanne, Freddy, and Marc.

To Evan, the Muffin & Cupcake crews, the Pint Day crowd (462 Tuesdays, and counting), Aberzombies, Yoda, Anonymous Faxer, and everyone else who gave us reasons to keep the pade going. What fun would by Tuesdays be without you?

Thank-you Amy, and your many heads. You are amazing—just amazing.

Thank-you to everyone who helped along the way: Aaron, Alan (T_EXmaster—one day you'll be the T_EXdoctor), Andrew, Arundel, Betty, Chris, Christine, Clifton, Colin, Collette, Danny, Dallas, Dave, Eric (Broseph!), Eryn, Heather, Joanne, Jon, Karin, Kristine, Mark, Mat, Matt (G and Mathlete), Mike, Mo, Nada, Nick, Pieter, Renate, Richard, Sarah, SCUM, Steph, Tina, Tinke, Tracy, Tyler, Veronique, and the countless names I have accidentally forgotten to include.

Thank-you to the funding agencies that helped me pay the bills: Alberta Inequity, iCore, NSERC, and the University of Calgary.

Finally, I wish to thank my wonderful family for supporting me in every conceivable way. Thank-you for the Gussysitting, the vehicles, the moral support, the financial support, the cupcakes, the phone calls, the breakfasts, the dinners, and every other kind of support imaginable. I couldn't have done it without you.

Dedication

For Dad—the original perpetual student—and Mom, my biggest fan.

Table of Contents

Abstract	ii
Acknowledgements	iii
Dedication	iv
Table of Contents	v
List of Tables	ix
List of Figures	x
1 Motivation and Background	1
1.1 Proving Primality	1
1.2 Pseudosquares and Primality Proving	3
1.2.1 Structure of the Thesis	6
2 The One-Dimensional Sieve Problem	8
2.1 The Generalized Sieve Problem	8
2.2 Transforming Sieve Problems	10
2.2.1 Sieve Rotation	10
2.2.2 Normalization	11
2.2.3 Combining Transformations	12
2.2.4 Parallellizing the Sieve Problem	13
2.3 Doubly-Focused Sieving	13
2.3.1 The Doubly-Focused Transformation	15
2.3.2 Parallellizing Doubly-Focused Sieve Problems	16
2.4 Sieve Devices	16
2.4.1 Estrin’s idea and the SRS-181	18
2.4.2 Hardware-based Sieve Devices	19
2.4.3 FPGA-based Sieves	19
2.4.4 Software Techniques	20
2.4.5 Calgary Scalable Sieve (CASSIE)	21
2.5 Enumerating Solutions via CRT	22
3 Classical Pseudosquares and Pseudocubes	25
3.1 Pseudosquare Growth	25
3.1.1 Pseudosquare Results	28

3.2	Pseudocubes and Primality Testing	31
3.2.1	Growth Rate of the Pseudocubes	32
4	Eisenstein Pseudocubes	36
4.1	Euclidean Domains and the Euclidean Algorithm	36
4.2	The Eisenstein Integers	39
4.2.1	Norms in $\mathbb{Z}[\omega]$	40
4.2.2	The Units of $\mathbb{Z}[\omega]$	41
4.2.3	Primes in $\mathbb{Z}[\omega]$	42
4.2.4	Primary elements in $\mathbb{Z}[\omega]$	44
4.3	Cubic Residue Character	46
4.3.1	The Cubic Jacobi Symbol	49
4.4	Eisenstein Pseudocubes	52
4.5	Eisenstein Pseudocubes and Primality Testing	53
4.6	Primality Proving in the Integers	55
5	How to Solve a Certain Diophantine Problem	57
5.1	Simple Continued Fractions	59
5.2	The Hermite-Serret Algorithm	66
5.2.1	Adapting Hermite-Serret	67
5.2.2	Further Generalizations	68
5.3	Cornacchia's Algorithm	69
5.3.1	Terminating the Algorithm	76
5.4	Analyzing and Optimizing Cornacchia's Algorithm	78
5.4.1	Further Improvements	81
5.5	Primality Proving using Eisenstein Pseudocubes	85
5.5.1	Computational Complexity of the EPPP Algorithm	86
6	Computing Eisenstein Pseudocubes	87
6.1	Rejecting Perfect Powers in $\mathbb{Z}[\omega]$	87
6.2	Sieve Criteria for Eisenstein Pseudocubes	88
6.3	Summary	96
7	Extending CASSIE to Two-Dimensions	97
7.1	Representing a Two-Dimensional Sieve Problem	97
7.2	Sieving in a Norm-Bounded region	98
7.2.1	Complex Conjugate and Sieve Bounds	99
7.2.2	Number of Solution Candidates	100
7.3	Normalization in Two-Dimensions	102
7.3.1	Computing Normalized Bounds	104

7.3.2	Integer Functions	105
7.3.3	Computing Sieve Bounds Using Integer Operations	108
7.4	Filtering and Output Processing	113
7.5	Summary	113
8	Open Hardware for Mathematical Research (OHMR)	114
8.1	Overview	114
8.1.1	Input/Output (I/O) Architecture	115
8.2	The PicoBlaze Embedded Soft Processor	116
8.2.1	PicoBlaze Input / Output	117
8.2.2	Interrupt Facility	119
8.3	OHMon and the OHMR Architecture	121
8.4	OHMR Register Architecture	121
8.4.1	8-bit registers	121
8.4.2	Multi-byte Registers	123
8.5	Control Banks	124
8.6	Parameter Storage	125
8.7	Serial Communication in OHMon	126
8.7.1	Receive Path	127
8.8	The Completed OHMR Design	130
9	Sieve Computing for the OHMR Toolkit (SCOT)	131
9.1	Device Parameters	131
9.2	Overview of SCOT	132
9.3	SCOT State Machine	134
9.4	Control Signals and Inputs	137
9.5	Counter Design	138
9.5.1	Loading Counter Data using OHMR Monitor (OHMon)	140
9.6	Shift Register Architecture	140
9.6.1	Additional Sieve Register Requirements	142
9.6.2	Loading Shift Register Data via Banks	143
9.6.3	Operational Cyclic Shift Register	144
9.7	Address Bus Decoding	145
9.8	Sieve Output	146
9.8.1	State Vector	147
9.9	Checkpoint/Debug Information	148
9.9.1	Comb Units	148
9.10	Sieve ROM	150
9.11	Solution Count Mode	151
9.12	Summary	151

10 Results and Analysis	152
10.1 Performance Metrics	152
10.2 Evaluating Software Sieve Performance	154
10.2.1 Number of Moduli	154
10.2.2 Number, Type, and Order of Filters	155
10.3 CASSIE Performance	156
10.3.1 Solution Counts and Parameter Choices	156
10.3.2 Adding Exclusion Moduli	157
10.3.3 Raw Canvass Rate	159
10.4 SCOT Performance	160
10.4.1 Deviation from Maximum Sieve Rate	160
10.4.2 Communication Bandwidth in SCOT	162
10.4.3 Cost of Hardware Sieving	163
10.5 Eisenstein Pseudocube Results	164
10.6 Eisenstein Pseudocube Growth	166
10.6.1 A Caveat	169
11 Future work	172
11.1 Deploying the Hardware Sieve	172
11.1.1 Decoupled Solution Decoding	172
11.1.2 Improving Clock Rate	173
11.1.3 Multi-Chip Sieve Devices	175
11.2 Solving the Communications Bottleneck	175
11.3 Improving Software Sieve Rates	176
11.4 Doubly-focusing in Two Dimensions	177
12 Conclusions	179
Bibliography	181
A Algorithms	191
A.1 Basic Algorithms	191
A.2 Computing Modular Square Roots	192
A.2.1 Cipolla-Lehmer	192
A.2.2 Tonelli-Shanks	192
A.2.3 Analysis	193
A.2.4 Efficiency	194
A.3 Sieve Algorithms	195

List of Tables

2.1	Comparison of Recent Sieve Devices	24
3.1	Recent Pseudosquare Results	31
3.2	Least Pseudocubes for Primes $q \equiv 1 \pmod{3}$	33
8.1	Error Rates in Baud Generation	129
9.1	SCOT Device Parameters	133
9.2	States in the Sieve FSM	136
9.3	Control Lines in SCOT	138
9.4	Arranging Sieve Moduli in Banks	143
9.5	Sieve Output Select	146
10.1	Candidate and Solution Counts by Sieve Bound	157
10.2	Number of Solutions with Varying Moduli	159
10.3	Small Eisenstein Pseudocubes	165
10.4	Eisenstein Pseudocubes	166

List of Figures

2.1	A Simple Sieve Device	17
3.1	Bounds on Pseudosquare Growth	29
3.2	Pseudosquare Growth	30
3.3	Growth Rate of Pseudosquares vs. Pseudocubes	35
7.1	Region of Bounded Norm: $N(\alpha) = a^2 + b^2 - ab \leq H$	101
7.2	Estimating Integer Lattice Points	102
8.1	OHMR Architecture	115
8.2	PicoBlaze in OHMR	117
8.3	Port Timing for OUTPUT instruction	118
8.4	Port Timing for INPUT instruction	119
8.5	Interrupt Timing	120
8.6	Handling Interrupts using a Dedicated Flip-Flop	120
8.7	Registers in the OHMR architecture	122
8.8	OHMR Multibyte Registers	123
8.9	Control Bank Detail	125
8.10	UART Macros	126
8.11	I/O Path for Serial Communications	128
8.12	Schematic View of OHMR device	130
9.1	OHMR Implementation of the SCOT Design	132
9.2	Sieve State Machine Controller	134
9.3	Sieve State Machine	135
9.4	Control Bank 1 Mappings in SCOT	137
9.5	Counter Design in SCOT	139
9.6	CSRK: Cyclic Shift-by- K Register (Right)	141
9.7	OCSSR: Operational Cyclic Shift Register	144
9.8	State Decoding for DX register	147
9.9	COMB: Comb Unit	148
9.10	Address Register Decoding for ROM Use	150
9.11	Mode Register Use	151
10.1	Raw Canvass Rates vs. Number of Moduli	158
10.2	Raw Canvass Rates, Normalized Problem	161
10.3	Growth Rate of Eisenstein Pseudocubes vs. Classical	170

List of Acronyms

ACL Advanced Cryptography Laboratory	2
AMD Advanced Micro Devices	22
ASIC Application Specific Integrated Circuit	153
BSD Berkeley Software Distribution	22
CASSIE Calgary Scalable Sieve	2
CLB Combinational Logic Block	20
CPU Central Processing Unit	23
CRT Chinese Remainder Theorem	9
CUDA Compute Unified Device Architecture	176
DCM Digital Clock Manager	130
EPPP Eisenstein Pseudocube Primality Proving	85
ERH Extended Riemann Hypothesis	26
F+V fixed-plus-variable	18
FIFO First-In First-Out	127
FPGA Field Programmable Gate Array	3
FSM Finite State Machine	134
GCD Greatest Common Divisor	37
GPGPU General Purpose Graphics Processing Unit	176
GSP Generalized Sieve Problem	6
HDL Hardware Description Language	131
IC Integrated Circuit	153
I/O Input/Output	115
ISR interrupt service routine	119
LED Light Emitting Diode	115
LSB Least Significant Bit	127
MSSU Manitoba Scalable Sieve Unit	19
MUX Multiplexer	123
OASiS Open Architecture Sieve System	19
OHMR Open Hardware for Mathematical Research	114
OHMon OHMR Monitor	115
OpenCL Open Computing Language	176

PCI Peripheral Component Interconnect	116
PID Principal Ideal Domain	36
QNR Quadratic Non-Residue	192
RAM Random Access Memory	16
ROM Read Only Memory	115
TTL Transistor-Transistor Logic	19
SCOT Sieve Computing for the OHMR Toolkit.....	131
SIMD Single Instruction Multiple Data	21
SPI Serial Peripheral Interface	130
SR Set-Reset	120
SRS-181 Shift Register Sieve.....	19
SWAC Standards Western Automatic Computer	21
UART Universal Asynchronous Receiver Transmitter	114
UFD Unique Factorization Domain	38
UMSU University of Manitoba Sieve Unit	19
USB Universal Serial Bus	116

Chapter 1

Motivation and Background

The last thing one knows in constructing a work is what to put first.

—Blaise Pascal

1.1 Proving Primality

In August 2002, Agrawal, Kayal, and Saxena described an unconditional, deterministic algorithm for proving the primality of an integer N . In its original form [AKS02], this algorithm ran with time complexity¹ $(\log N)^{12+o(1)}$. A great deal of analysis followed: Macaj [Mac02], and independently Agrawal improved this complexity to $(\log N)^{10.5+o(1)}$, an analysis which was eventually published in [AKS04]; Lenstra (unpublished, described in [Ber02]) improved the algorithm to run with time complexity $(\log N)^{8+o(1)}$; and Lenstra and Pomerance (unpublished, most recent revision in [HWLP09]) improved the algorithm to run with time complexity $(\log N)^{6+o(1)}$.

In [Ber05], Berrizbeitia improved this algorithm for a large class of integers, obtaining a time complexity of $(\log N)^{4+o(1)}$. Independently, Cheng [Che03], Bernstein [Ber07], and Mihăilescu and Roberto M. Avanzi [MA] then generalized this result to produce a random-time primality proving algorithm with the same complexity.

These results raise an obvious question: how far can the time complexity of un-

¹Complexity results in this thesis refer to bit operations, unless otherwise indicated.

conditional, deterministic primality proving be improved?

In [Pom87], Pomerance proved the *existence* of a short primality certificate for N that requires only $(5/2 + o(1)) \log_2 N$ multiplications² modulo N . This leads to a primality proof requiring only $(\log N)^{2+o(1)}$ operations. Unfortunately, this result is not constructive, and Pomerance makes the observation that actually finding this certificate could well take exponential time.

In previous work [WW06] this author provided numerical evidence for a conjecture that primality may be proved with complexity $(\log N)^{3+o(1)}$ using quantities known as pseudosquares and pseudocubes. These results were achieved through a combination of improved congruential sieve techniques, and the development of a software toolkit, the Calgary Scalable Sieve (CASSIE) [Woo04] running on the University of Calgary’s Advanced Cryptography Laboratory (ACL) Beowulf cluster.

This thesis is concerned with the further development of tools and techniques for solving the congruential sieve problem—in both one- and two-dimensions—with the intent of offering further computational evidence for the conjectured complexity of primality proving.

To achieve this aim, this thesis examines an alternate definition of pseudocube: the Eisenstein pseudocube, which leads to an investigation of the congruential sieve problem in two-dimensions. In particular, we:

- develop the notion of an Eisenstein pseudocube, and an associated primality proving algorithm;
- reduce the problem of finding Eisenstein pseudocubes to an instance of the

²The actual complexity of multiplication depends greatly on the size of the inputs used. The best-known bit complexity results for multiplication, *e.g.* [SS71], require $O(n \log n \log \log n)$ operations for an input length of n .

two-dimensional sieve problem;

- extend the CASSIE software toolkit to solve instances of a two-dimensional sieve problem;
- develop a general-purpose hardware framework for implementing custom computing devices on Xilinx Field Programmable Gate Array (FPGA) devices;
- design and implement a prototype FPGA-based sieve device using this framework;
- evaluate the performance of the prototype hardware for solving two-dimensional sieve problems; and
- enumerate Eisenstein pseudocubes using these tools.

Before proceeding further, we should review the problem that motivated our approach.

1.2 Pseudosquares and Primality Proving

The *pseudosquare problem*, first considered by Kraitchik [Kra24] in 1924, is characterized in the following manner:

Definition 1.1. *Given an integer x , a pseudosquare $M_{2,x}$ is defined as the least positive integer satisfying:*

1. $M_{2,x} \equiv 1 \pmod{8}$,
2. The Legendre symbol $\left(\frac{M_{2,x}}{q}\right) = 1$ for all odd primes $q \leq x$,
3. $M_{2,x}$ is not a perfect square.

In other words, the pseudosquare, $M_{2,x}$ behaves (locally) like a perfect square modulo all primes $q \leq x$,

One of the most interesting applications of pseudosquares is in the area of primality

proving. In [Hal33], Marshall Hall was the first to demonstrate a primality test involving the pseudosquares. This test was based on a formalization of some fallacious ideas ([Leh30]) originally put forth by Seelhoff in [See86], and later espoused by Cole [Col03]³ and Kraitchik [Kra29].

The main idea of his method involved what Hall termed *apparent residues*,⁴ defined as follows.

Definition 1.2. (Apparent Residues and Non-residues)

Let $N \in \mathbb{Z}$ be odd, and $p, q \in \mathbb{Z}$ be odd primes. Furthermore, define

$$p' = \left(\frac{-1}{p} \right) p = (-1)^{\frac{p-1}{2}} p$$

If $\left(\frac{N}{p} \right) = 1$, then p' is said to be an apparent residue of N . If $\left(\frac{N}{q} \right) = -1$ then $q' = (-1)^{\frac{q-1}{2}} q$ is said to be an apparent non-residue of N .

The apparent residue character of 2 and -1 were defined in a similar manner. Hall then proved the following theorem.

Theorem 1.3. *If all the (not necessarily prime) factors of N are less than L_p , and if the primes $-1, 2, -3, 5, \dots, (-1)^{\frac{p-1}{2}} p$ can be divided into two classes: the apparent residues of N , $\mathcal{A} = \{a_1, a_2, \dots, a_s\}$ and the apparent non-residues of N , $\mathcal{B} = \{b_1, b_2, \dots, b_r\}$, such that every member $a_i \in \mathcal{A}$ is a true quadratic residue of N , and the product of every pair of elements $b_i b_j \in \mathcal{B}$ is also a true quadratic residue of N , then N is either a prime or a power of a prime.* \square

³In an editorial in *Notices of the AMS* [Kna99], the story is told of Cole's 1903 address to the Society, entitled *On the Factoring of Large Integers*. It is said the lecture was "met with a standing ovation after he lectured without saying a single word, multiplying two large integers and verifying that their product was $2^{67} - 1$." The description given in [Col03] is somewhat more verbose than this, however, and includes the discussion of Seelhoff's idea.

⁴This was a translation of Kraitchik's "résidues éventuelles." A better term would likely be *potential residue*.

Though actually using this method as a primality test is difficult,⁵ Beeger successfully used it twice, first in [Bee39] to prove the 12-digit cofactor of $2577687858367 = 17 \cdot 151628697551$ prime,⁶ then in [Bee46] to prove a 13-digit factor of $12^{45} + 1$ prime.

Dan Shanks, in correspondence with D. H. Lehmer [Wil98] implied a different test involving the pseudosquares when he noted the following theorem:

Theorem 1.4. *If $N \equiv -1 \pmod{4}$ is a base- q probable prime, that is, if $q^{N-1} \equiv 1 \pmod{N}$ for all primes $q \leq p$, then any prime divisor $P \equiv 1 \pmod{4}$ of N must satisfy $P \geq L_p$. \square*

The implied test is as follows: if it is known that $N \equiv -1 \pmod{4}$ is the product of at most two primes, and if $N < L_p$, then N is a prime if $q^{N-1} \equiv 1 \pmod{N}$ for all primes $q \leq p$. Unfortunately, like Hall's test, this algorithm is of limited practical use. The first practical primality test to make use of the pseudosquares was proposed by Selfridge and Weinberger [Wil78a], with modifications by Lukes *et al.* in [LPW96]. Their algorithm to determine the prime character of an integer $N \in \mathbb{Z}$ was based on the following theorem:

Theorem 1.5. *If*

1. $N < M_{2,x}$ for some integer x ,
2. $p_i^{\frac{N-1}{2}} \equiv \pm 1 \pmod{N}$ for all primes p_i , $2 \leq p_i \leq x$,
3. $p_j^{\frac{N-1}{2}} \equiv -1 \pmod{N}$ for some odd $p_j \leq x$ when $N \equiv 1 \pmod{8}$, or $2^{\frac{N-1}{2}} \equiv -1 \pmod{N}$ when $N \equiv 5 \pmod{8}$,

then N is a prime or a power of a prime.

By this theorem, determining the prime character of N requires $\pi(x)$ modular ex-

⁵To say the least. If N is not known to be prime and the Jacobi symbol $\left(\frac{m}{N}\right) = 1$, the problem of determining whether a given integer m is a quadratic residue modulo N is believed to be as difficult as the problem of factoring integers ([MvOV96], pp. 99).

⁶This is the numerator of the 34th Bernoulli number.

ponentiations.⁷ If $M_{2,x}$ grows sufficiently quickly—*i.e.* if $\pi(x) < c(\log M_{2,x})^k$ for fixed constants c, k —then Theorem 1.5 offers an unconditional, deterministic polynomial-time primality test for integers $N < M_{2,x}$.

The problem of finding the integers $M_{2,x}$ is actually quite difficult. In [Pat92, §3.5], Patterson proved that when reduced to a decision problem, solving systems of simultaneous congruences—the Generalized Sieve Problem (GSP)—is NP-complete. In practice, solving such systems has required the development of an interesting set of tools and techniques, ranging from algorithmic improvements to the construction of a wide variety of interesting computing devices.

1.2.1 Structure of the Thesis

In Chapter 2, we describe the one-dimensional sieve problem in its most general form. Chapter 3 reviews historical approaches—both mathematical and technological—for finding pseudosquares and pseudocubes. Chapter 4 introduces the notion of an Eisenstein pseudocube, culminating in the development of a primality proving algorithm for integers $m \equiv 1 \pmod{3}$. Chapter 5 examines a key component of this algorithm: solving a Diophantine equation of the form $fx^2 + gy^2 = m$ for coprime positive integers f, g, m , detailing the history of the problem, and culminating with an improved algorithm for doing so. Chapter 6 describes how to describe the search for Eisenstein Integers as an instance of a two-dimensional sieve problem. Chapter 7 examines the implementation of the two-dimensional sieve using CASSIE. In Chapters 8 and 9, we describe the design and development of a special-purpose hardware device for solving instances of the sieve problem. Finally, in Chapter 10, we compute a table of

⁷The cost of rejecting perfect powers is ignored in this analysis.

Eisenstein pseudocubes, analyze the performance of our approach, and compare the effectiveness of primality proving using Eisenstein pseudocubes to existing methods.

Chapter 2

The One-Dimensional Sieve Problem

It's very esoteric, of course, and since I am practically the only man working in this field you can see how widespread the interest in it is.

—D. H. Lehmer, on Sieves

2.1 The Generalized Sieve Problem

Definition 2.1. *Define a sieve instance (ρ_i) to be a modulus M_i together with a set of j acceptable residues, $\mathcal{R}_i = \{r_{i,j} \mid 0 \leq r_{i,j} < M_i\}$. Given*

1. $H \in \mathbb{Z}$, $H > 0$ (the sieve bound),
2. $k \geq 1$ sieve instances ρ_1, \dots, ρ_k whose moduli M_1, \dots, M_k are relatively prime in pairs,

the Generalized Sieve Problem (GSP) is the problem of finding all $x \in \mathbb{Z}$ such that $0 \leq x < H$ and

$$x \pmod{M_i} \in \mathcal{R}_i \text{ for all } i = 1, \dots, k.$$

These solutions, x , are called the solutions admitted by the sieve problem, and k is called the width of the sieve problem.

Equivalently, a sieve problem \mathcal{S} may be expressed in terms of sets:

$$\mathcal{S} = \bigcap_{i=1}^k \{x \in \mathbb{Z} \mid x \pmod{M_i} \in \mathcal{R}_i, \quad 0 \leq x < H\}.$$

Definition 2.2. *The sieve problems*

$$\mathcal{S}_1 = \bigcap_{i=1}^r \{x \in \mathbb{Z} \mid x \pmod{M_i} \in \mathcal{R}_i, \quad 0 \leq x < H\}$$

$$\mathcal{S}_2 = \bigcap_{j=1}^s \{x \in \mathbb{Z} \mid x \pmod{M_j} \in \mathcal{R}_j, \quad 0 \leq x < H\}$$

are equivalent if and only if $\mathcal{S}_1 = \mathcal{S}_2$ for all $A, B \in \mathbb{Z}$; i.e. for any choice of bounds the set of solutions admitted by each of the sieve problems is the same.

With this notion of equivalence, the following theorem may now be demonstrated.

Lemma 2.3. *Given a sieve problem,*

$$\mathcal{S} = \bigcap_{i=1}^k \{x \in \mathbb{Z} \mid x \pmod{M_i} \in \mathcal{R}_i, \quad 0 \leq x < H\}$$

of width $k > 1$, an equivalent sieve problem of width $k - 1$ can be formed.

Proof. Consider any two sieve instances, $\rho_1 = \{M_1, \mathcal{R}_1\}$ and $\rho_2 = \{M_2, \mathcal{R}_2\}$. Clearly, any solution $x \in \mathcal{S}$ satisfies both $x \pmod{M_1} \in \mathcal{R}_1$ and $x \pmod{M_2} \in \mathcal{R}_2$.

Define the set \mathcal{R} to be the Chinese Remainder Theorem (CRT) combination of all residues from the sets \mathcal{R}_1 and \mathcal{R}_2 ; i.e. let $M = M_1 \cdot M_2$, $N_i = \frac{M}{M_i}$, $\xi_i \equiv N_i^{-1} \pmod{M_i}$. Then

$$\mathcal{R} = \{r \mid r \equiv \xi_1 N_1 r_1 + \xi_2 N_2 r_2 \pmod{M}, \quad r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2\}.$$

Now, form a new sieve problem, replacing the sieve instances $\{M_1, \mathcal{R}_1\}$ and

$\{M_2, \mathcal{R}_2\}$ with the newly constructed instance $\{M, \mathcal{R}\}$. The width of this new sieve problem is $k - 1$. By the CRT, $x \pmod{M_1 M_2} \in \mathcal{R}$ if and only if $x \pmod{M_1} \in \mathcal{R}_1$ and $x \pmod{M_2} \in \mathcal{R}_2$. Equivalence of the sieve problems follows from Definition 2.2. \square

Theorem 2.4. *A sieve problem,*

$$\mathcal{S} = \bigcap_{i=1}^k \{x \in \mathbb{Z} \mid x \pmod{M_i} \in \mathcal{R}_i, \quad 0 \leq x < H\}$$

consisting of k sieve instances can be replaced by an equivalent sieve problem consisting of a single sieve instance:

$$\mathcal{S} = \{x \in \mathbb{Z} \mid x \pmod{M} \in \mathcal{R}, \quad 0 \leq x < H\},$$

where $M = \prod_{i=1}^k M_i$, and \mathcal{R} consists of the CRT combinations of the sets \mathcal{R}_i for $i = 1, 2, \dots, k$.

2.2 Transforming Sieve Problems

Before designing a sieve device, there are a number of optimizations and parallelization techniques that may be employed, independent of the underlying sieve implementation. We will now discuss these techniques in some detail.

2.2.1 Sieve Rotation

In certain cases, we may wish to consider a sieve problem with a nonzero lower bound; *i.e.* solutions x lying in the interval $A \leq x < B$. It should be noted that a

sieve problem of this type can always be translated to an equivalent problem in the interval $0 \leq x < B - A$ by adjusting the acceptable residues, $r_{i,j} \in \mathcal{R}_i$; *i.e.* taking $s = A$ in the (bijective) map

$$\begin{aligned} \sigma[s] &: r_{i,j} \mapsto (r_{i,j} - s) \pmod{M_i} \\ &(A, B) \mapsto (A - s, B - s). \end{aligned}$$

The $\sigma[s]$ map is called a *rotation* of the sieve problem.

2.2.2 Normalization

When a sieve problem contains a modulus for which there is but a single acceptable residue, a greater savings of computational effort may be achieved. In [Leh28], D. H. Lehmer described a technique for eliminating single-valued congruences from sieve problems. In [Leh54] he gave this technique a name: *normalization*.¹

Observe that all single-valued residue conditions in a given sieve problem may be combined (via the CRT) into the single congruence

$$x \equiv r \pmod{m}. \tag{2.1}$$

Any solution to the given sieve problem must therefore be of the form $x = r + ym$ for some y . Rather than searching for solutions x that satisfy all sieve congruences,

¹To be more explicit, the term *normalization* was used by Lehmer in the case described—to combine and eliminate moduli with only a single acceptable residue. Later authors, such as Lukes [Luk95] and Bernstein [Ber04b] use this technique to iterate over (or parallelize) all acceptable residues for a given modulus, calling the technique *multiple residue optimization* and *singly-focused enumeration* respectively. In this thesis, we use the original term for any use of this map.

it is more efficient to search for acceptable values of y by using the relationship in Equation 2.1 to translate the remaining sieve criteria; *i.e.* for all $r_{i,j} \in \mathcal{R}_i$, and all $\mathcal{R}_i \in \mathcal{S}$, apply the (bijective) map:

$$\begin{aligned} \eta[m, r] : r_{i,j} &\mapsto (r_{i,j} - r)m^{-1} \pmod{M_i} \\ (0, H) &\mapsto \left(0, \left\lceil \frac{(H - r)}{m} \right\rceil\right). \end{aligned} \tag{2.2}$$

In [Luk95, Algorithm 4.1], Lukes gives an explicit algorithm for transforming a sieve ring using this map. His approach is given as Algorithm A.2.

An equivalent technique, dubbed *denormalization*, may be developed to reverse this operation. Here the inverse permutation is generated by first locating the bit position of the final $(M - 1)^{th}$ entry in the sieve, and then repeatedly subtracting the normalization modulus. An explicit version of this technique is also described in Appendix A.

2.2.3 Combining Transformations

In practice, most sieve problems are transformed to both rotated to a specific start position, and normalized to eliminate any single-residue congruence conditions. Since rotation affects all sieve instances including those which give rise to the normalization map, the result of applying rotation to a sieve problem, \mathcal{S} , with acceptable residues $r_{i,j} \in \mathcal{R}_i$ and normalization $\eta[m, r]$ is given by the map:

$$\begin{aligned} \sigma[s] : \eta[m, r] &\mapsto \eta[m, (r - s) \bmod m] \\ r_{i,j} &\mapsto (r_{i,j} - s) \pmod{M_i}. \end{aligned} \tag{2.3}$$

Definition 2.5. Given a sieve problem \mathcal{S} operating over the arbitrary interval $A \leq x < B$ for which the solutions satisfy $x = my + r$, we define its equivalent normalized sieve problem to be $\eta[m, r](\mathcal{S})$, which operates over the interval $0 \leq y < \lceil \frac{H-r}{m} \rceil$. Here $H = B - A$.

2.2.4 Parallellizing the Sieve Problem

An additional application of the normalization transformation arises if multiple computational units are employed in parallel. Consider a sieve instance $\rho = \{M, \mathcal{R}\}$ with $|\mathcal{R}|$ acceptable residues. The sieve problem containing this instance may be partitioned into $|\mathcal{R}|$ parallel problems by applying a normalization, $\eta[M, r_j]$, for each of the acceptable residues, $r_j \in \mathcal{R}$, for a particular modulus M . Each of these normalized problems is then assigned to one of the available sieve devices. The solution set \mathcal{S} is simply the union of the results for each of the $|\mathcal{R}|$ parallellized sieve problems.

This optimization can be useful even if the normalized sieve problems are solved consecutively. As demonstrated by Lehmer [Leh28], an effective speedup of $\frac{M_i}{|\mathcal{R}_i|}$ is achieved by executing each of the normalized sieve problems in series. Lukes [Luk95] called this technique *multiple residue optimization*. Bernstein [Ber04b] called it *singly-focused enumeration*.

2.3 Doubly-Focused Sieving

Doubly-focused sieving, first described by Bernstein [Ber04b] makes use of an explicit form of the Chinese Remainder Theorem to map a sieve problem, \mathcal{S} into two smaller sieve problems whose solutions may then be combined to retrieve all $x \in \mathcal{S}$. To illustrate this technique, we first require a lemma.

Lemma 2.6. *Every x in the range $0 \leq x < H$ may be expressed as the difference $t_p M_n - t_n M_p$, where M_p, M_n are relatively prime, $0 \leq t_p < \left\lceil \frac{H + M_n M_p}{M_n} \right\rceil$, and $0 \leq t_n < M_n$.*

Proof. Consider the arithmetic progression obtained by fixing t_n and varying t_p in the expression²:

$$x = t_p M_n - t_n M_p.$$

This progression is capable of producing any $x \equiv -t_n M_p \pmod{M_n}$. Thus, if $t_n M_p \pmod{M_n}$ is made to range over all residue classes $\{0, 1, 2, \dots, M_n - 1\}$, the resulting arithmetic progressions can be used to produce all possible integers x in an interval $[0, H)$ by varying t_p .

Consider $t_n \in \{0, 1, 2, \dots, M_n - 1\}$. Since $\gcd(M_n, M_p) = 1$, it is straightforward to show that the set $\{t \mid t \equiv t_n M_p \pmod{M_n}, 0 \leq t_n < M_n\}$ forms a complete reduced residue system. If not, then for some $0 \leq i, j < M_n, i \neq j$, the congruence: $i \cdot M_p \equiv j \cdot M_p \pmod{M_n}$ would hold. Multiplying both sides by $M_p^{-1} \pmod{M_n}$, however, gives $i \equiv j \pmod{M_n}$, a contradiction.

Hence, it is sufficient to consider $0 \leq t_n < M_n$ to produce the necessary arithmetic progressions. Since t_n is always non-negative, choose $t_p \geq 0$, and as $t_n < M_n$, it follows that $t_p < \left\lceil \frac{H + M_n M_p}{M_n} \right\rceil$.

□

²Subtraction is used in the CRT decomposition of x instead of the more traditional addition to allow both sieve problems in the doubly-focused enumeration to operate in the same direction. See Appendix A for a description of the algorithm.

2.3.1 The Doubly-Focused Transformation

The doubly-focused transformation $\delta[M_n, M_p]$ works as follows. Consider a sieve problem:

$$\mathcal{S} = \bigcap_{i=1}^k \{x \in \mathbb{Z} \mid x \pmod{M_i} \in \mathcal{R}_i, \quad 0 \leq x < H\}. \quad (2.4)$$

Partition the moduli M_1, M_2, \dots, M_k into two distinct sets, \mathcal{M}_n and \mathcal{M}_p , and define the quantities M_n and M_p as the products of the moduli in these sets: $M_n = \prod_{i=1}^s M_i$ and $M_p = \prod_{j=s+1}^k M_j$ respectively.

From Lemma 2.6, a solution $x \in \mathcal{S}$ can be written $x = t_p M_n - t_n M_p$. Reducing this expression modulo M_n and M_p respectively, it is clear that any solution x must satisfy the congruences:

$$\begin{aligned} x &\equiv -t_n M_p \pmod{M_n} \in \mathcal{R}_n \\ x &\equiv t_p M_n \pmod{M_p} \in \mathcal{R}_p. \end{aligned}$$

Thus, rather than sieving for solutions x in the original problem, we can search instead for solutions t_n, t_p in two translated sieve problems, and recombine these solutions to obtain solutions in the original interval as follows. Set $\mathcal{T}_p = \eta[M_n^*, 0](\mathcal{R}_p)$, $\mathcal{T}_n = \eta[M_p^*, 0](\mathcal{R}_n)$, $M_n^* \equiv M_n^{-1} \pmod{M_p}$ and $M_p^* \equiv (-M_p)^{-1} \pmod{M_n}$. The doubly-focused map is given by:

$$\delta[M_n, M_p] : \mathcal{S} \rightarrow (\mathcal{S}_p, \mathcal{S}_n),$$

where

$$\mathcal{S}_p = \bigcap_{i=1}^s \left\{ t_p \in \mathbb{Z} \mid t_p \pmod{M_p} \in \mathcal{T}_p, \quad 0 \leq t_p < \left\lceil \frac{H + M_n M_p}{M_n} \right\rceil \right\} \quad (2.5)$$

$$\mathcal{S}_n = \bigcap_{i=s+1}^k \{ t_n \in \mathbb{Z} \mid t_n \pmod{M_n} \in \mathcal{T}_n, \quad 0 \leq t_n < M_n \}. \quad (2.6)$$

We call these new problems the *positive* and *negative* sieve problems, respectively. The solutions of these new problems may be converted back into solutions to the original sieve problem (2.4) either by maintaining an array of the entire solution space [Sor06], or by implementing an enumeration algorithm such as those in Appendix A.

2.3.2 Parallellizing Doubly-Focused Sieve Problems

There are several techniques that may be used to parallelize a doubly-focused sieve problem. As in Section 2.2.4 normalization may be used to partition the problem by residue class. In [Sor10], Sorenson describes an efficient parallelization technique that can be employed when large amounts of Random Access Memory (RAM) are available.

2.4 Sieve Devices

The basic design of a one-dimensional sieve device is as follows. Consider, for example, the following system of congruences:

$$\begin{aligned} x \pmod{3} &\in \{0, 2\} & x \pmod{4} &\in \{2, 3\} \\ x \pmod{5} &\in \{1, 2, 4\} & x \pmod{7} &\in \{0, 1, 4, 6\}. \end{aligned}$$

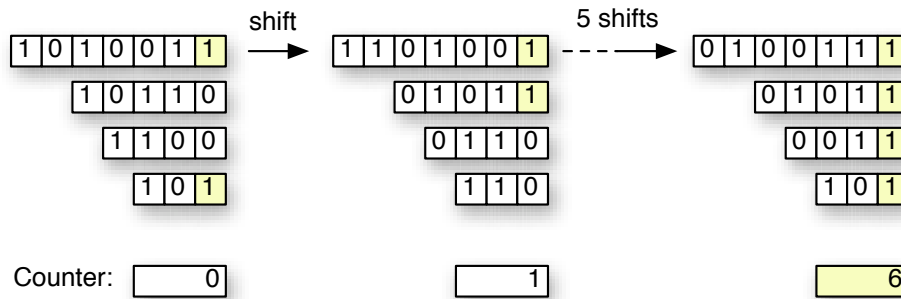


Figure 2.1: A Simple Sieve Device

To implement this sieve problem, imagine that we construct a cyclic shift mechanism to represent the possible residues for each modulus; *e.g.* 4 cyclic shift registers of size 3, 4, 5, and 7 bits respectively. We call this mechanism a *sieve ring*. For each sieve ring, we initialize bit positions corresponding to acceptable residues to 1, and all other bits to 0. Next, we adjust these shift registers so that their least significant bits are aligned. Finally, we initialize a counter to zero.

To evaluate the next candidate solution, we advance the sieve ring contents (circularly) one position to the right while simultaneously advancing the counter. If at any time the least significant position of the sieve rings all indicate an acceptable solution, then this solution is given by the counter contents. The complete process is illustrated in Figure 2.1

This example illustrates that a simple sieve device can be constructed with little more than cyclic shift registers and a counter. The solution detection logic—typically called a *solution tap*—for this arrangement consists of little more than a large AND-gate wired to the least significant bit of each register.

Of course, there is no requirement for the sieve to be electronic in nature. In fact, most early automated sieve designs were mechanical in nature [Woo04]; however shift-

register-based sieves have dominated the last 30 years of congruential sieve design. In fact, most modern sieve devices are based on an architecture first proposed by Estrin *et al.* — the fixed-plus-variable (F+V) design. We will now turn our attention to this design, and its successors.

2.4.1 Estrin’s idea and the SRS-181

In 1962 [CEFT62], Cantor, Estrin, Fraenkel, and Turn described a new architecture for solving the sieve problem. This revolutionary idea—the F+V design—incorporated both a fixed, general-purpose component, and a variable, custom component to produce a device capable of sieving at rates of up to 10^{10} numbers per minute. Their proposal outlined three main ideas. First, it described an efficient algorithm for implementing the sieve problem using shift registers on reconfigurable hardware, including a method for constructing the solution detection mechanism that eliminated the need for the largest modulus.

Second, it described a technique for implementing additional (virtual) exclusion moduli in software on the attached general purpose computer by carefully matching the predicted output rate of the shift-register sieve with the arithmetic capabilities of the host machine.

Finally, the paper made the important observation that by employing r solution taps in parallel, and relabelling the bit positions of the 1-bit circular shift registers (effectively converting them to r -bit circular shift registers), the parallelism of the sieve device could be increased by a factor of r at the expense of little more than additional solution detection circuitry.

Revolutionary as it was, the Estrin proposal was not acted on for almost 13 years.

The first to do so was D.H. Lehmer, inventor of virtually all previous sieve devices.³ Though never completed (the device was mistakenly removed from the lab and sold for scrap while its inventors were away at a conference), the device, referred to as the Shift Register Sieve (SRS-181), used cyclic shift registers to implement congruence classes, and was intended to sieve at speeds approaching 20,000,000 trials per second. The device had 42 hardware rings (sieve instances), implemented using 8-bit shift register Transistor-Transistor Logic (TTL) integrated circuits, and representing the primes (or prime powers) from 2, 3, . . . , 181 [Pat92].

2.4.2 Hardware-based Sieve Devices

In 1983, Hugh Williams and Cam Patterson began a reimplementaion of the SRS-181 design [Pat83]. The completed device, dubbed the University of Manitoba Sieve Unit (UMSU) was implemented on a set of 3 wire-wrap boards using 500 integrated circuits. It featured 8 solution taps, and a shift rate of 16.67 MHz for an overall canvas rate of 1.33×10^8 trials/second.

Hugh Williams and his collaborators went on to design two additional hardware-based sieve units: Open Architecture Sieve System (OASiS) [Ste89], and the Manitoba Scalable Sieve Unit (MSSU) [Luk95], the latter capable of sieve rates of up to 6.144×10^9 trials per second.

2.4.3 FPGA-based Sieves

In 1994, Bronson and Buell described the first FPGA-based sieve device [BB94]. Their device, implemented on a hardware platform called *Splash*, consisted of a linear array

³See [Woo04] for a brief history of Lehmer's contributions to the field.

of 32 Xilinx 3090 FPGA chips, each containing a grid of 16×20 configurable logic blocks. Each Combinational Logic Block (CLB) could be programmed to assume one of a variety of configurations, including a pair of flip-flops, any single 5-input, 2-output combinatorial function, or two 4-input, 1-output combinatorial functions. For the purposes of the sieve problem, one prime shift register was implemented per FPGA chip. To maximize performance, a 64-bit solution tap was employed, and Estrin's optimization was employed to convert the 1-bit cyclic shift registers to 64-bit cyclic shift register. Using a master clock rate of 16 MHz, this sieve device was capable of achieving a total sieve rate of 1024×10^6 trials per second.

In 2003, Wake and Buell described a sieve design targeted at their latest generation of F+V hardware: the Star Bridge Hypercomputer 36m [WB03]. This machine employed a dual 2.4 GHz Intel Xeon machine as its fixed host hardware. The variable component consisted of 7 Xilinx Virtex FPGAs, four of which (Xilinx xc2v6000's) were available as programmable computing resources. Using synthesis models, they conjectured that it was possible to implement 12 sieves per Virtex FPGA, each sieve handling the 37 primes up to and including 157. By employing the most naive form of parallelism (each sieve configured at a different start point), the HC 36m device was capable of 48-bit parallelism. Combined with the 64-bit solution tap, the Star Bridge architecture offered theoretical sieve rate of 192×10^9 trials per second.

2.4.4 Software Techniques

Over the years, the sieve problem has been implemented on a wide variety of computing devices, including general purpose computers. The first software implementation was made by Lehmer and Selfridge in 1954, on the first large electronic computer to

operate in the western United States: the Standards Western Automatic Computer (SWAC) [Leh54]. Other implementations are described in [BS67] and [Pat92, §4.5]. Unfortunately, due to the inherently serial nature of these machines, software implementations tended to lag far behind special-purpose devices in terms of speed.⁴ Thus, it came as somewhat of a surprise when D. J. Bernstein announced in 2000 that he had succeeded in extending the table of pseudosquares—last extended by a 180-day computation by the MSSU—using software running for 10 days on a single-processor general purpose computer, a Pentium IV running at 1406 MHz. His solution was the first to make use of doubly-focused enumeration, and breathed new life into implementation on general purpose machines.

2.4.5 CASSIE

In [Woo04], we developed a software-based toolkit called CASSIE for representing and solving congruential sieve problems. The toolkit implements the transformations of Chapter 2 as a set of extensions to the scripting language Tcl [Ous94]. To implement parallel sieving as described in Section 2.2.4, CASSIE was implemented on the University of Calgary’s Advanced Cryptography Lab (ACL)—a Beowulf cluster consisting of 152 dual-Xeon Pentium IV processors running at 2.4 GHz.

CASSIE is written in portable C and was compiled using GCC (2.96 and later) under Red Hat Linux 7.3 (kernel 2.4.18-27.7) on the Intel Pentium IV architecture. It has

⁴One notable exception occurred in 1976 [Leh76], when Lehmer wrote an implementation for the ILLIAC IV—the first Single Instruction Multiple Data (SIMD) architecture computer. The unique parallel architecture of ILLIAC IV machine allowed Lehmer’s sieve implementation to operate with a degree of parallelism not usually possible on general-purpose computers. As a result, Lehmer’s sieve implementation was able to reach speeds of 15 million trials per second, making it the second-fastest sieve device Lehmer ever devised. Unfortunately, the experimental nature of the ILLIAC IV machine prevented Lehmer from using it on a long-term basis [SW90].

since been ported to the 64-bit Advanced Micro Devices (AMD) Opteron architecture, and the Berkeley Software Distribution (BSD)-derived OpenBSD operating system.

In this thesis, we extended the CASSIE toolkit to incorporate both the two-dimensional sieve techniques described in Chapter 6, and the ability to use special-purpose hardware devices, such as the one described in Chapter 9.

2.5 Enumerating Solutions via CRT

In [Sor06], Sorenson described an alternate technique for solving simultaneous congruences on a general purpose computer. Rather than using the method of exclusion, Sorenson’s approach makes use of a data structure known as a *wheel* to efficiently enumerate all possible combinations of acceptable residues, producing outputs via the Chinese Remainder Theorem.

Sorenson’s approach required one interesting modification to the doubly-focused technique described in Section 2.3. Since solutions emerge from the wheel structure in essentially random order, the DFSIEVE enumeration technique of Algorithm A.6 cannot be used. Instead, Sorenson makes use of the availability of large amount of RAM, as follows.

1. First, fix an array size for the outputs of the positive sieve, \mathcal{S}_p . This limit should be chosen to ensure that all solution arrays fit into physical RAM. Sorenson, for example, chose 30 million.⁵
2. Using a density prediction, choose a wheel interval $t_{min} \leq t_p < t_{max}$ that should obtain this many solutions. Fill the positive solution array by sorting the solu-

⁵30,000,000 64-bit integers \times 16 cores per node = 3.84 GB; *i.e.* slightly less than half of the physical memory available on the compute node.

tions emerging from the positive wheel.

3. Using t_{min} and the sieve bound, H , compute a range for the candidate solutions, t_n . As each t_n emerges from the negative wheel, find all t_p values (via interpolation search on the positive solution array).
4. Compute candidate solutions $x = t_p M_n - t_n M_p$. Solutions may be further filtered by using an exclusion sieve.

In a long-term computation on Butler University's supercomputer cluster, *Big Dawg*, Sorenson was able to produce two new pseudosquares and ten new pseudocubes over a period of 16 months.

Though a precise canvass rate is difficult to quantify, Sorenson estimates that filling and sorting of the positive solution array required 2.5 Central Processing Unit (CPU)-minutes over an interval of 4.7667×10^{11} candidate solutions [Sor09]. This corresponds to a raw canvass rate of 3.1778×10^9 trials/sec.

Though this technique has produced several record-setting results (see, *e.g.* Tables 3.1 and 3.2) this approach will not be further examined at this time.

Raw canvass rates for the sieve devices discussed in this section are summarized in Table 2.1.

Sieve	Moduli	SD width	Clock(MHz)	Raw Canvass Rate
UMSU	32	8	16.67	1.333×10^8
OASiS	RAM ^[1]	16	13.3	2.128×10^8
OASiS-II	RAM ^[2]	16	13.3	2.128×10^8
MSSU	30	8	24	1.92×10^8
Splash	21	64	16	1.024×10^9
Wake/Buell ^[3]	37	64	66	4.224×10^9
CASSIE ^[4]	16 ^[5]	32	[7]	2.5×10^8
Sorenson ^{[4][6]}	13	[7]	[7]	3.2×10^9

¹ 16 x 8192-word RAM

² 32 x 8192-word RAM

³ Conjectured. Device was never built.

⁴ Software implementation. Canvass rates are approximate.

⁵ Parameters were configurable. Chosen values described in Chap. 7.

⁶ Implementation used wheel data structure, not method of exclusion.

⁷ There is no equivalent notion in this implementation.

Table 2.1: Comparison of Recent Sieve Devices

Chapter 3

Classical Pseudosquares and Pseudocubes

If it looks like a duck, and quacks like a duck, we have at least to consider the possibility that we have a small aquatic bird of the family anatidae on our hands.

—Douglas Adams, *Dirk Gently's Holistic Detective Agency*

Recall the definition of a pseudosquare: given an integer x , a *pseudosquare* $M_{2,x}$ is defined as the least positive integer satisfying:

1. $M_{2,x} \equiv 1 \pmod{8}$
2. The Legendre symbol $\left(\frac{M_{2,x}}{q}\right) = 1$ for all odd primes $q \leq x$
3. $M_{2,x}$ is not a perfect square.

The growth rate of pseudosquares has several key applications in Number Theory.¹ In Section 1.2, we described a method for efficient primality proving using the pseudosquares, where the efficiency of the result was dependent on a growth rate of the form

$$\pi(x) < c(\log M_{2,x})^k.$$

We will now formalize this prediction.

3.1 Pseudosquare Growth

In [Hal33], Hall proved the following result:

¹See, for instance the applications described in [Woo04, §5.1.1].

Theorem 3.1. *If N is a quadratic residue of all but a finite number of primes then N is a perfect square.*

Since the pseudosquares are by definition both nonsquare and nondecreasing, a natural consequence of this theorem is the following:

Corollary 3.2. *The values of $M_{2,x}$ tend to infinity with x .*

A lower bound, conditional upon the Extended Riemann Hypothesis (ERH), can be derived from a theorem of Bach:

Theorem 3.3. *Let \mathcal{G} be a nontrivial subgroup of $(\mathbb{Z}/m\mathbb{Z})^*$ (the group of reduced residues modulo m) such that $n \in \mathcal{G}$ for all positive $n < x$. Then $x < 2(\log m)^2$.*

Proof. [Bac90]. □

Consider the subgroup \mathcal{G} of $(\mathbb{Z}/M_{2,p}\mathbb{Z})^*$ consisting of all g such that $\left(\frac{g}{M_{2,p}}\right) = 1$. Since $M_{2,p}$ is a nonsquare, there must be an odd prime q such that $q^\alpha \parallel M_{2,p}$ with α odd. Let t be a quadratic nonresidue of q , i.e. $\left(\frac{t}{q}\right) = -1$, and set:

$$\begin{aligned} r &\equiv t \pmod{q^\alpha} \\ r &\equiv 1 \pmod{M_{2,p}/q^\alpha}. \end{aligned}$$

By the CRT, $r \in (\mathbb{Z}/M_{2,p}\mathbb{Z})^*$ and by the properties of the Jacobi symbol, $\left(\frac{r}{M_{2,p}}\right) = -1$. Thus, \mathcal{G} is a nontrivial subgroup of $\mathbb{Z}/(M_{2,p})^*$. Also, $n \in \mathcal{G}$ for all $0 < n < p$. By Theorem 3.3, $p < 2(\log M_{2,p})^2$, and hence

$$\log M_{2,p} > \sqrt{\frac{p}{2}}. \tag{3.1}$$

Schinzel, [Sch97], produced both ERH-conditional and unconditional bounds on

$M_{2,p}$ of

$$(1 - o(1))\sqrt{p} < \log M_{2,p} < (\log 4 + o(1))\frac{p}{\log p}, \text{ and} \quad (3.2)$$

$$(4\sqrt{e} + o(1)) \log p < \log M_{2,p} < (1/4 + o(1))p \quad (3.3)$$

respectively.

Pomerance and Shparlinski [PS09], citing a pigeonhole argument of Soundararajan, later showed that the upper bound of (3.2) was in fact unconditional. They went on to show that the pseudosquares are equidistributed in fairly short intervals, a fact that lends credence to the following heuristic argument of Lukes [Luk95, pp. 111].

Let p_i denote the i^{th} prime ($p_1 = 2$), and consider all solutions satisfying the residue conditions for the n^{th} pseudosquare, M_{2,p_n} . Since there are $\frac{p_i-1}{2}$ acceptable residues modulo p_i , CRT tells us that there are $S = \prod_{i=2}^n (p_i - 1)/2$ solutions in the interval $0 < x < H$ where $H = 8p_2p_3 \cdots p_n$. Assuming the equidistribution of these solutions, we would expect the least pseudosquare to be given by:

$$M_{2,x} \approx \frac{H}{S} = 2^{n+1} \prod_{i=1}^n \frac{p_i}{p_i - 1}.$$

If we invoke Mertens's theorem [HW79, pp. 351], $\prod_{p \leq x} \frac{p-1}{p} \sim \frac{e^{-\gamma}}{\ln x}$ as $x \rightarrow \infty^2$, so

$$M_{2,x} \sim 2e^\gamma 2^n \log x. \quad (3.4)$$

Recall from the Prime Number Theorem [HW79, pp. 9] that $n \sim \frac{p_n}{\log p_n}$. Substituting this result into Equation (3.4) produces a growth estimate of:

² $\gamma = 0.57721$ is Euler's constant.

$$M_{2,p} \sim 2e^\gamma 2^{p/\log p} \log p.$$

Writing $\log p = 2^{\frac{\log \log p}{\log 2}}$, we obtain the estimate:

$$M_{2,p} \approx 2^{(1+o(1))n} \log p. \tag{3.5}$$

Empirical evidence seems to support this prediction, as will now be shown.

3.1.1 Pseudosquare Results

In [Luk95], Lukes extended the table of known pseudosquares by computing the pseudosquares for all primes $q \leq 277$. In [Ber04b], Bernstein extended this result to $q \leq 281$. Using CASSIE, we extended the table of pseudosquares to include all primes $q \leq 359$ [WW06]. In [Sor10], Jon Sorenson further extended this table to $q \leq 373$ using a software technique originally described in [Sor06]. All these recent results are summarized in Table 3.1.

Figure 3.1 examines the growth rate of the pseudosquares vs. Schinzel's predicted bounds (Equation 3.2). Figure 3.2 plots pseudosquare growth using Lukes' growth predictions. In both cases it appears that the conjectured growth rate of Equation 3.5 holds; hence, using the primality test implied by Theorem 1.5 primality proving may be done using $(\log N)^{1+o(1)}$ modular exponentiations. Since performing each exponentiation (using, for instance, the techniques of Schönhage and Strassen [SS71]) incurs a complexity of $(\log N)^{2+o(1)}$, it may therefore be conjectured that the primality of an arbitrary integer N may be proved with $(\log N)^{3+o(1)}$ operations.

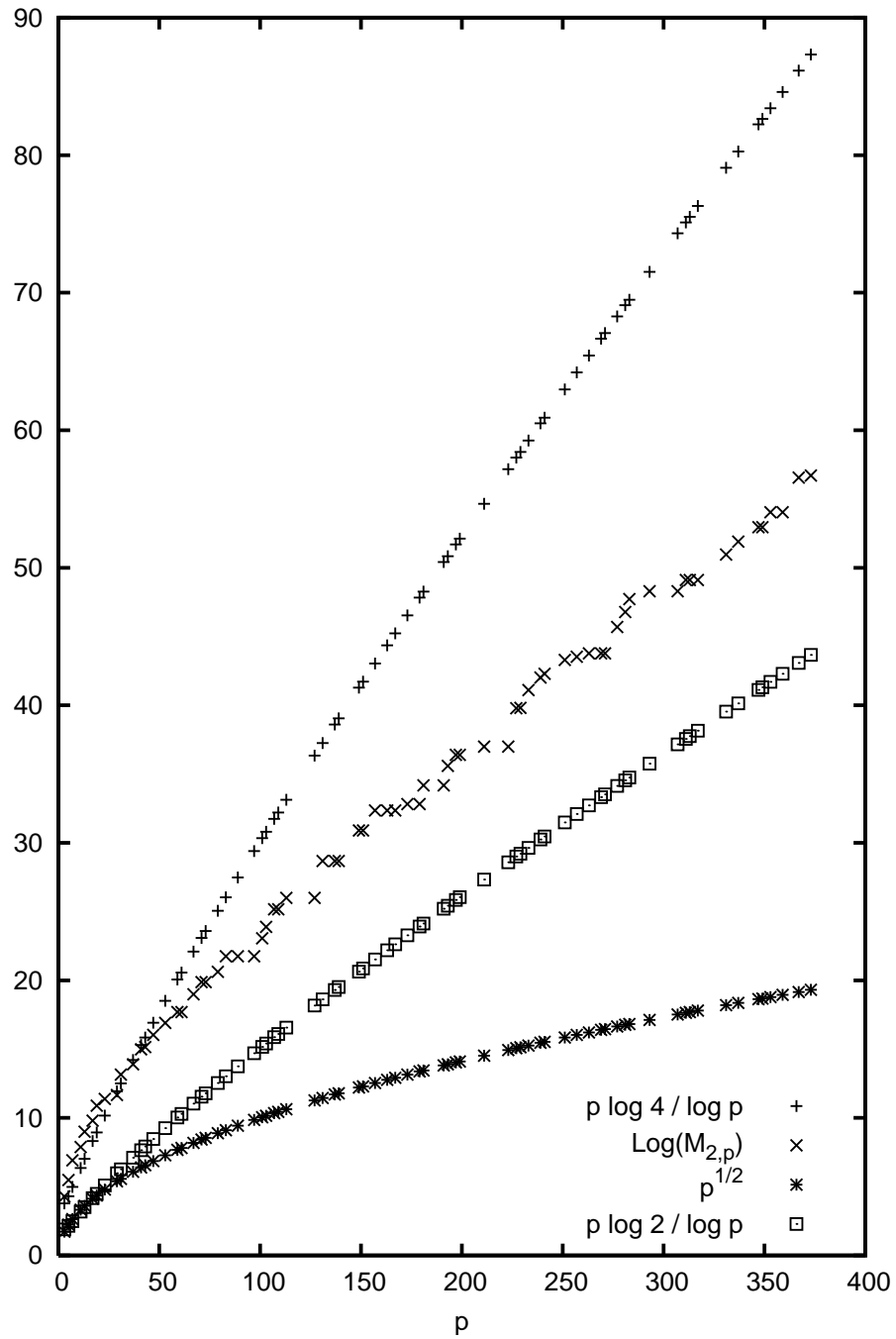


Figure 3.1: Bounds on Pseudosquare Growth

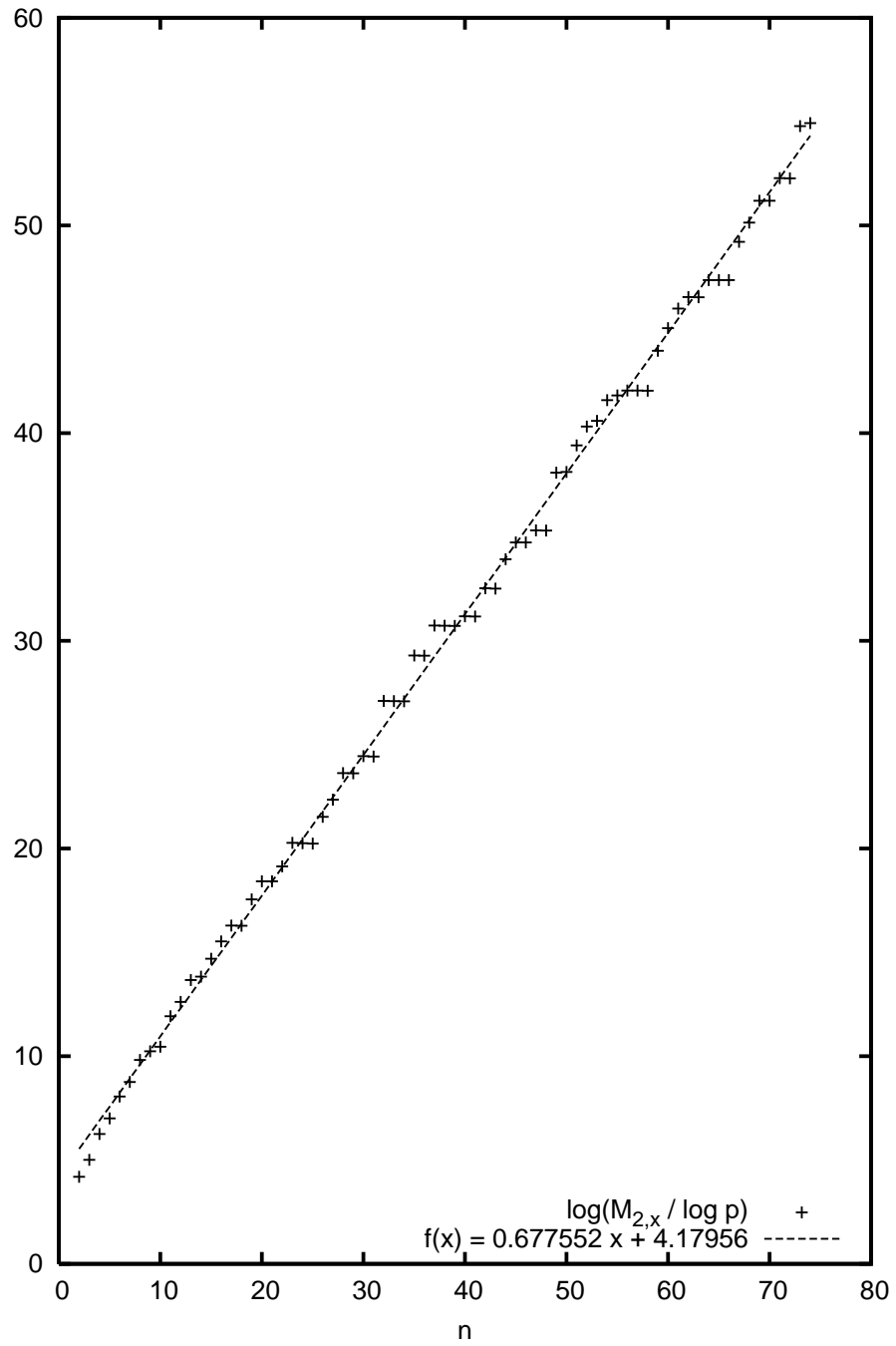


Figure 3.2: Pseudosquare Growth

p	$M_{2,p}$	Source
281	2 08936 36579 90449 75961	Bernstein (2001) ¹
283	5 33552 66333 98282 03681	Wooding, Williams (2003) ²
293, 307	9 36664 07926 67146 97089	
311, 313, 317	21 42202 86037 02699 16129	
331	136 49154 49155 82988 03281	
337	345 94858 80167 01277 78801	Wooding, Williams (2003) ³
347, 349	994 92945 93047 92133 34049	
353, 359	2953 63487 40090 03108 80401	
367	36553 34429 47705 74600 46489	Sorenson (Aug 2008) ⁴
373	42350 25223 08059 75035 19329	
383	> 10 ²⁵	

¹ Software, doubly-focused enumeration [Ber04b]

² 2-processors, software on CASSIE/ACL [Woo04]

³ 180-processors, software on CASSIE/ACL [Woo04]

⁴ Software wheel. [Sor09]

Table 3.1: Recent Pseudosquare Results

3.2 Pseudocubes and Primality Testing

The pseudocube problem, first examined by Cobbham in [Cob66], may be defined in a manner analogous to that of the pseudosquares.

Definition 3.4. *Given an integer, x , the pseudocube $M_{3,x}$ is defined as the least positive integer satisfying:*

1. $M_{3,x} \equiv \pm 1 \pmod{9}$
2. $M_{3,x}^{\frac{q-1}{3}} \equiv 1 \pmod{q}$ for all primes $q \leq x$, $q \equiv 1 \pmod{3}$,
3. $q \nmid M_{3,x}$ for all primes $q \leq x$, $q \not\equiv 1 \pmod{3}$,
4. $M_{3,x}$ is not a perfect cube.

In [BMW04], Berrizbeitia *et al.* generalized the primality test of Theorem 1.5 to involve pseudocubes. This definition makes use of the ring of Eisenstein integers, $\mathbb{Z}[\omega]$ where $\omega = e^{2\pi i/3}$. The general properties of these integers will be reviewed in Chapter

4.

Theorem 3.5. *Let N be odd, $3 \nmid N$. Define $N^* = N$ if $N \equiv 1 \pmod{3}$ and $N^* = -N$ if $N \equiv -1 \pmod{3}$. Furthermore, if $q \equiv 1 \pmod{3}$ and prime, define $\alpha_q \in \mathbb{Z}[\omega]$ by $\alpha_q \overline{\alpha_q} = q$. If*

1. $N < (M_{3,x})^{2/3}$,
2. $\left(\frac{\alpha_q}{N}\right)_3 \equiv \lambda_q^{\frac{(N^*-1)}{3}} \pmod{N}$ for all primes $q \equiv 1 \pmod{3}$, $q \leq x$,
3. $q \nmid N$ for all primes $q \equiv 2 \pmod{3}$, $q \leq x$,

where $\alpha \in \mathbb{Z}[\omega]$, $\lambda_q = \alpha_q / \overline{\alpha_q}$, then N is a prime or a power of a prime.

In [SW90], Stephens and Williams computed a list of pseudocubes $M_{3,q}$ up to $q = 313$. In [Luk95], Lukes extended this computation to $q = 349$. Using CASSIE, we were able to tabulate pseudocubes to $q = 487$ [WW06]. Most recently, Sorenson has extended this table to $q = 617$ [Sor10]. A complete list of known pseudocubes is given in Table 3.2.

3.2.1 Growth Rate of the Pseudocubes

Using arguments similar to those of Section 3.1, we can arrive at a growth prediction for the pseudocubes [BMW04]; *i.e.*

$$M_{3,q} \equiv c3^n (\log q)^2. \tag{3.6}$$

If we let p_i denote the i^{th} prime, then the pseudosquare primality test performs n modular exponentiations, where n is defined such that p_n is the first prime satisfying $N < L_{p_n}$. The pseudocube primality test, on the other hand, requires n modular exponentiations, where m is defined such that, if q_i denote the i^{th} prime congruent

q	$M_{3,q}$	Source
7	71	
13	181	
19	2393	
31	3457	
37	5669	
43, 61	74339	
67	166249	
73	2275181	
79	72 35857	
83, 89	72 98927	
97,101,103,107	87 21539	
109,113	912 46121	
127	912 46121	
139	980 18803	
151,157	16123 83137	
163,181,193	79910 83927	
199	2 03657 64119	
211	251 55987 68717	
223	644 05557 21601	
229,241,271	2913 58749 01141	
277,283,307,313	40654 06766 72677	
331,337,349	75 01762 52728 79381	Lukes, Williams (1995)
367	996 43865 13658 98469	
373	2152 98491 43899 68651	
379	12403 28486 28199 56587	
397, 409, 421	37605 27410 54792 28611	
433	2 05830 03900 63371 14403	Wooding, Williams (2005)
439	18 45193 81892 86034 36441	
457	78 54338 42538 52259 02393	
463	129 04554 92806 82688 48739	
487	133 84809 54852 12275 17303	
499	601 25695 21674 16551 89317	Sorenson (Sep 2008)
523,541	1166 14853 91487 02789 15947	
547	41391 50561 50994 78852 27899	
571,577	1 62485 73199 87995 69143 39717	
601,607	2 41913 74719 36148 42758 90677	
613,617	67 44415 80981 24912 90374 06633	Sorenson (2009)
619	$> 10^{27}$	

Table 3.2: Least Pseudocubes for Primes $q \equiv 1 \pmod{3}$

to 1 (mod 3), then $N < M_{q_n}^{2/3}$. The pseudocube test will be more efficient than the pseudosquare test if $M_{q_n}^{2/3} > L_{p_n}$ as n grows.

From Equations (3.5) and (3.6), we see that

$$\frac{M_{q_n}^{2/3}}{L_{p_n}} \approx \frac{c_2^{2/3} 3^{2n/3} (\log q_n)^{4/3}}{c_1 2^n \log p_n} > \frac{c_2^{2/3}}{c_1} \left(\frac{3^{2/3}}{2} \right)^n > 1$$

for sufficiently large n . From Figure 3.3, we can make two conclusions: first, that the growth rate of the pseudocubes matches that predicted by Equation (3.6); and second, the growth of the $2/3$ power of the pseudocubes appears such that it should eventually outpace that of the pseudosquares.

We turn our attention now to a new generalization of the pseudosquare notion to the cubic case: Eisenstein pseudocubes.

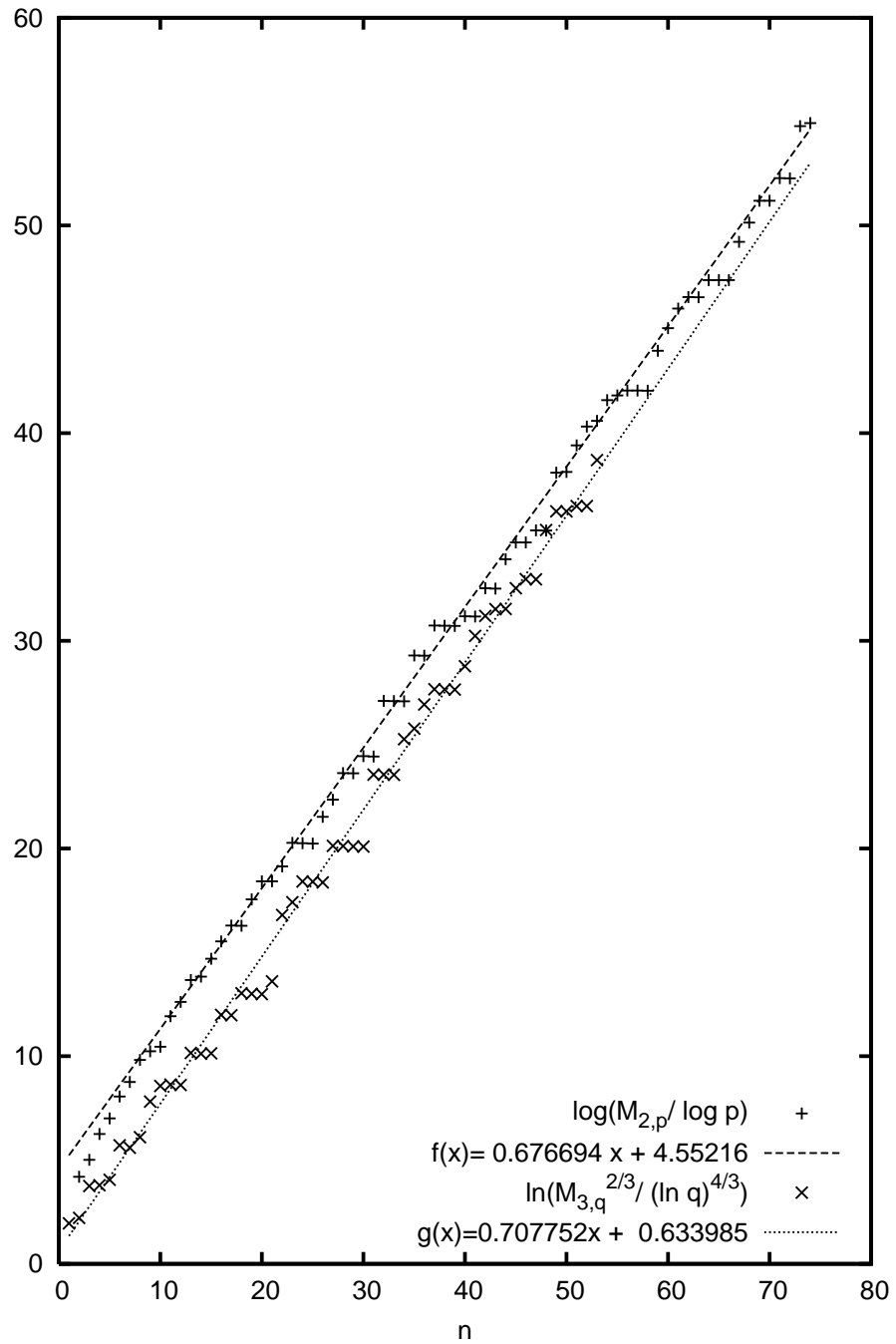


Figure 3.3: Growth Rate of Pseudosquares vs. Pseudocubes

Chapter 4

Eisenstein Pseudocubes

If you don't know where you are going, any road will take you there.

—George Harrison

4.1 Euclidean Domains and the Euclidean Algorithm

Let $\mathbb{Z}_{\geq 0} = \{0, 1, 2, \dots\}$ denote the set of nonnegative integers, \mathbb{N} the set of positive integers, $\mathbb{N} = \{1, 2, \dots\}$.

Definition 4.1. *An integral domain, D is said to be Euclidean domain if there is a function $N : D \setminus \{0\} \rightarrow \mathbb{Z}_{\geq 0}$ such that:*

1. *if $a, b \in D, ab \neq 0$ then $N(a) \leq N(ab)$, and*
2. *if $a, b \in D, b \neq 0$, there exists $q, r \in D$ with $a = qb + r$ and either $r = 0$ or $N(r) < N(b)$.*

Theorem 4.2. *Every Euclidean Domain is a Principal Ideal Domain.*

Proof. Let D be a Euclidean Domain and \mathfrak{J} a nonzero ideal of D . Choose $a \in \mathfrak{J}$ as the element in \mathfrak{J} such that $N(a)$ is minimal. Since D is a Euclidean Domain, if $b \in \mathfrak{J}$, there exists q, r such that $b = aq + r$ where $r = 0$ or $N(r) < N(a)$. But $r = b - aq \in \mathfrak{J}$, so we must have that $r = 0$. Thus, $b \in \langle a \rangle$ and $\mathfrak{J} = \langle a \rangle$. Since the zero ideal is given by $\langle 0 \rangle$, every ideal in D is principal, and hence, D is a Principal Ideal Domain (PID).

□

Definition 4.3. Let R denote a commutative ring with identity. A nonzero element, $a \in R$, is said to divide an element $b \in R$ (denoted $a \mid b$) if there exists an $x \in R$ such that $ax = b$.

Definition 4.4. If $a \mid b$ and $b \mid a$ then a, b are said to be associates.

Definition 4.5. Let D be an integral domain, and suppose $r \in D$ is nonzero and not a unit. a is called irreducible if whenever $a = bc$ with $b, c \in D$, at least one of b or c is a unit.

Definition 4.6. Let D be an integral domain. A nonzero element $a \in D$ is called prime if a is not a unit and whenever $a \mid bc$, we have either $a \mid b$ or $a \mid c$.

Furthermore, recall that we can describe the notion of divisibility in terms of ideals.

Theorem 4.7. Let $a, b, e \in R$ where R is a commutative ring with identity. Then

1. $a \mid b \iff \langle b \rangle \subset \langle a \rangle$,
2. a, b are associates $\iff \langle a \rangle = \langle b \rangle$,
3. e is a unit $\iff e \mid a$ for all $a \in R$,
4. e is a unit $\iff \langle e \rangle = R$,
5. “ a is an associate of b ” is an equivalence relation,
6. If e is a unit and $a = be$, then a and b are associates. If R is an integral domain, the converse is also true.

□

Definition 4.8. Let R be a commutative ring. A Greatest Common Divisor (GCD) of a and b is a nonzero $d \in R$ such that

1. $d \mid a$ and $d \mid b$, and
2. if $d' \mid a$ and $d' \mid b$ then $d' \mid d$.

Though there exist rings for which no GCD exists, a sufficient condition for its existence is given in the following theorem.

Theorem 4.9. *Let D be a PID and $a, b \in D$. Then a and b have a greatest common divisor $d = \gcd(a, b)$, and $\langle a, b \rangle = \langle d \rangle$.*

Proof. [IR90, Prop. 1.3.2] □

Definition 4.10. *Given $a, b \in D$, a Euclidean Domain where $N(a) \geq N(b)$. If $b = 0$, return a . Otherwise, determine $q, r \in D$ such that $a = qb + r$, $N(r) < N(b)$. Replace $a \leftarrow b$, $b \leftarrow r$ and repeat the process until $b = 0$. This process is known as the Euclidean Algorithm.*

Theorem 4.11. *If d is the output of the Euclidean algorithm for inputs a and b , then $d = \gcd(a, b)$.*

Proof. Given $a, b \in D$, compute one iteration of the Euclidean algorithm; i.e. $a = q_1b + r_1$. Since we can write $r_1 = a - q_1b$, any common divisor of a and b must also divide r_1 . In general (and writing $a = r_{-1}$, $b = r_0$), we see that $\gcd(a, b) = \gcd(r_i, r_{i+1})$ for every step of the algorithm. The algorithm always terminates, as since the sequence of norms $N(r_i)$ is strictly decreasing, eventually we will have $N(r_k) = 0$, hence $r_k = 0$. So $\gcd(a, b) = \gcd(r_{k-1}, 0) = |r_{k-1}|$, and this is the output of the algorithm. □

Definition 4.12. *A Unique Factorization Domain (UFD) is an integral domain D in which every nonzero, nonunit $r \in D$ satisfies the following properties:*

1. r can be written as a finite product of (not necessarily distinct) irreducibles $p_i \in D$; i.e. $r = p_1p_2 \cdots p_n$ and

2. this decomposition is unique up to associates and ordering.

Theorem 4.13. *Let D be a PID. Let S be a set of primes in D such that every prime $\pi \in D$ is associate to a prime in S , and no two primes in S are associate; i.e. S contains exactly one prime from each class of associates. Then if $a \in D$, $a \neq 0$, we can write*

$$a = e \prod_{p \in S} p^{\alpha_p}$$

where e is a unit and p ranges over all primes in S . Furthermore, e and α_p are uniquely determined by a , and $\alpha_p = \text{ord}_p a$.¹

Corollary 4.14. *Every Principal Ideal Domain is a UFD.*

Proof. [Gal02, pp. 319–20] □

4.2 The Eisenstein Integers

Let $\omega = e^{2\pi i/3} = (-1 + i\sqrt{3})/2$, a primitive cube root of unity. Then ω has the following properties:

$$\omega^2 = \frac{-1 - i\sqrt{3}}{2} = \bar{\omega} \tag{4.1}$$

$$\omega\omega^2 = 1 = \omega\bar{\omega} \tag{4.2}$$

$$\omega + \omega^2 = -1 = \omega + \bar{\omega}. \tag{4.3}$$

We denote an arbitrary element $\zeta \in \mathbb{Z}[\omega]$ by $\zeta = a + b\omega$, with $a, b \in \mathbb{Z}$.

First, observe that $\mathbb{Z}[\omega]$ is a ring. Clearly, $\mathbb{Z}[\omega] \subset \mathbb{C}$ and is closed under addition and subtraction. Furthermore, given $\alpha, \beta \in \mathbb{Z}[\omega]$,

¹That is, α_p is the unique integer such that $p^{\alpha_p} \mid a$ but $p^{\alpha_p+1} \nmid a$. Existence and uniqueness of $\text{ord}_p a$ are proved in [IR90, p. 11].

$\alpha\beta = (a+b\omega)(c+d\omega) = ac + (ad+bc)\omega + bd\omega^2 = (ac-bd) + (ad+bc-bd)\omega \in \mathbb{Z}[\omega]$, and hence $\mathbb{Z}[\omega]$ is closed under multiplication. Thus $\mathbb{Z}[\omega]$ is a subring of \mathbb{C} (and a ring in its own right).

4.2.1 Norms in $\mathbb{Z}[\omega]$

We define a norm on these elements by $N(\zeta) = \zeta\bar{\zeta}$. Since $\zeta\bar{\zeta} = a^2 - ab + b^2 = (a - \frac{1}{2}b)^2 + \frac{3}{4}b^2$, it is clear that $N(\zeta) \geq 0$ for all $\zeta \in \mathbb{Z}[\omega]$; *i.e.* $N(\zeta) \in \mathbb{Z}_{\geq 0}$. Furthermore, $N(\zeta) = 0 \iff \zeta = 0$, so $\zeta \neq 0 \implies N(\zeta) \in \mathbb{N}$. Using this fact, it is simple to show that $\mathbb{Z}[\omega]$ has no zero divisors, and is in fact an Integral Domain.

Finally, observe that $N(\zeta\alpha) = N(\zeta)N(\alpha)$ for all $\zeta, \alpha \in \mathbb{Z}[\omega]$:

$$\begin{aligned}
N(\zeta)N(\alpha) &= (a^2 - ab + b^2)(c^2 - cd + d^2) \\
&= a^2c^2 - a^2cd + a^2d^2 - abc^2 + abcd - abd^2 + b^2c^2 - b^2cd + b^2d^2 \\
&= (a^2c^2 + abcd + b^2d^2) + (-a^2cd - abc^2) + (a^2d^2 - abd^2 + b^2c^2 - b^2cd) \\
&= (a^2c^2 - 2abcd + b^2d^2) + (-a^2cd - abc^2 + abcd + abd^2 + b^2cd - b^2d^2) \\
&\quad + (a^2d^2 + 2abcd - 2abd^2 + b^2c^2 - 2b^2cd + b^2d^2) \\
&= (ac - bd)^2 - (ac - bd)(ad + bc - bd) + (ad + bc - bd)^2 \\
&= N(\zeta\alpha).
\end{aligned}$$

Notice nothing in this derivation relies on $a, b \in \mathbb{Z}$, and so it applies equally if $a, b \in \mathbb{Q}$.

Theorem 4.15. $\mathbb{Z}[\omega]$ is a Euclidean Domain.

Proof. Recall that $\mathbb{Z}[\omega]$ forms a ring; *i.e.* is closed under addition and multiplication.

Furthermore, since $\bar{\omega} = \omega^2$, $\mathbb{Z}[\omega]$ is also closed under complex conjugation.

Let $\alpha, \beta \in \mathbb{Z}[\omega]$, and suppose $\beta \neq 0$. Now $\alpha/\beta = \alpha\bar{\beta}/\beta\bar{\beta} = \alpha\bar{\beta}/N(\beta)$. Since $N(\beta) \in \mathbb{N}$ (as $\beta \neq 0$) and $\alpha, \bar{\beta} \in \mathbb{Z}[\omega]$, $\alpha/\beta = c + d\omega$ with $c, d \in \mathbb{Q}$. Thus we can find $r, s \in \mathbb{Z}$ such that $|c - r| \leq \frac{1}{2}$ and $|d - s| \leq \frac{1}{2}$. Set $\zeta = r + s\omega$. Then $N(\alpha/\beta - \zeta) = (c-r)^2 - (c-r)(d-s) + (d-s)^2 \leq \frac{1}{4} + \frac{1}{4} + \frac{1}{4} < 1$. Set $\rho = \alpha - \beta\zeta \in \mathbb{Z}[\omega]$. Observe:

$$N(\rho) = N(\alpha - \beta\zeta) = N(\beta((\alpha/\beta) - \zeta)) = N(\beta)N(\alpha/\beta - \zeta) < N(\beta).$$

Thus, given $\alpha, \beta \in \mathbb{Z}[\omega]$, we can find $\rho, \zeta \in \mathbb{Z}[\omega]$ such that $\alpha = \zeta\beta + \rho$ where either $\rho = 0$ or $N(\rho) < N(\beta)$. Hence, $\mathbb{Z}[\omega]$ is a Euclidean Domain. \square

Thus, $\mathbb{Z}[\omega]$ is a Unique Factorization Domain. We are therefore interested in the primes and units in this domain.

4.2.2 The Units of $\mathbb{Z}[\omega]$

Theorem 4.16. $\alpha \in \mathbb{Z}[\omega]$ is a unit $\iff N(\alpha) = 1$.

Proof. If $\alpha = a + b\omega$ is a unit, then $\exists \zeta \in \mathbb{Z}[\omega]$ such that $a\zeta = 1$. Thus, $N(\alpha\zeta) = N(\alpha)N(\zeta) = 1$, so $N(\alpha) \mid 1$ and hence, $N(\alpha) = 1$. Conversely, if $N(\alpha) = 1$ then $a^2 - ab + b^2 = (a + b\omega)(a + b\omega^2) = 1$. Thus, $(a + b\omega) \mid 1$ and $a + b\omega$ is a unit. \square

As such, the units in $\mathbb{Z}[\omega]$ are given by the solutions to: $a^2 - ab + b^2 = 1$; *i.e.* $(2a - b)^2 + 3b^2 = 4$. Since $a, b \in \mathbb{Z}$, we must have $b \in \{0, \pm 1\}$. We can enumerate these possibilities as follows:

- If $b = 0$, then $4a^2 = 4$ gives $a = \pm 1$.
- If $b = 1$, then $(2a - 1)^2 = 1$ implies $a \in \{0, 1\}$.

- Finally, if $b = -1$, then $(2a + 1)^2 = 1$ gives $a \in \{0, -1\}$

So the units in $\mathbb{Z}[\omega]$ are the six values: $\{\pm 1, \pm\omega, \pm(1 + \omega)\}$. Since $\omega^2 = -1 - \omega$, this is more simply stated as the six roots of unity: $\{\pm\omega^j \mid j = 0, 1, 2\}$.

4.2.3 Primes in $\mathbb{Z}[\omega]$

For clarity, we refer to primes $p \in \mathbb{Z}$ as the *rational primes*.

Theorem 4.17. *An Eisenstein integer $\zeta \in \mathbb{Z}[\omega]$ whose norm is a rational prime is a prime in $\mathbb{Z}[\omega]$.*

Proof. Assume otherwise; i.e. $\zeta = \alpha\beta$ for some $\alpha, \beta \in \mathbb{Z}[\omega]$ where α, β are not units. Then $N(\zeta) = N(\alpha)N(\beta) = p$ for some prime $p \in \mathbb{Z}$. So either $N(\beta) = 1$ or $N(\alpha) = 1$, a contradiction. \square

Theorem 4.18. *If $\pi \in \mathbb{Z}[\omega]$ is prime, then either $N(\pi) = p$ or $N(\pi) = p^2$ where p is a rational prime.*

Proof. Let $\pi \in \mathbb{Z}[\omega]$ be prime. Since π is not a unit, $N(\pi) = m > 1$. If m is prime, we are done, so let m be composite. Write $\pi\bar{\pi} = m = p_1p_2 \cdots p_n$ and so we have $\pi \mid pp_1p_2 \cdots p_n$. From the definition of primality (Definition 4.6), π divides exactly one of p, p_1, \dots, p_n ; without loss of generality, say $\pi \mid p$. Thus, $\exists \eta \in \mathbb{Z}[\omega]$ such that $\pi\eta = p$ and hence, $N(\pi\eta) = N(\pi)N(\eta) = N(p) = p^2$. So either $N(\pi) = N(\eta) = p$, or $N(\pi) = p^2$ and $N(\eta) = 1$, as desired

\square

Theorem 4.19. *For $\pi \in \mathbb{Z}[\omega]$, prime, $N(\pi) = p^2 \iff \pi$ is associate to a rational prime.*

Proof. The forward implication is clear from the proof of Theorem 4.18. For the converse, write $\pi = ep$ for $e \in \mathbb{Z}[\omega]$ a unit and $p \in \mathbb{Z}$ a rational prime. Then $N(ep) = N(p) = p^2$. \square

We are now in a position to classify the primes in $\mathbb{Z}[\omega]$.

Theorem 4.20. *Let p be a rational prime. If $p \equiv 2 \pmod{3}$ then p is a prime in $\mathbb{Z}[\omega]$.*

Proof. Assume otherwise; i.e. $p \equiv 2 \pmod{3}$ but p is composite in $\mathbb{Z}[\omega]$. Then we may write $p = \pi\eta$ where both π, η are non-units. Then $N(p) = p^2 = N(\pi)N(\eta)$ and $N(\pi) = p$. Write $\pi = a + b\omega$. Now $p = a^2 - ab + b^2 \equiv 4p = (2a - b)^2 + 3b^2$ and so $p \equiv (2a - b)^2 \pmod{3}$. But then $p \pmod{3} \in \{0, 1\}$ (the squares modulo 3), a contradiction. So p is prime in $\mathbb{Z}[\omega]$. \square

Theorem 4.21. *Let p be a rational prime. If $p \equiv 1 \pmod{3}$ then there exists a $\pi \in \mathbb{Z}[\omega]$ such that $p = \pi\bar{\pi}$ and π is prime in $\mathbb{Z}[\omega]$.*

Proof. Let p be a rational prime such that $p \equiv 1 \pmod{3}$. By quadratic reciprocity:

$$\left(\frac{-3}{p}\right) = \left(\frac{-1}{p}\right) \left(\frac{3}{p}\right) = (-1)^{(p-1)/2} \left(\frac{p}{3}\right) (-1)^{(p-1)/2 \cdot (3-1)/2} = \left(\frac{p}{3}\right) = \left(\frac{1}{3}\right) = 1.$$

So $\exists a \in \mathbb{Z}$ such that $a^2 \equiv -3 \pmod{p}$, and hence, $\exists k \in \mathbb{Z}$ such that $kp = a^2 + 3$. But then $p \mid (a^2 + 3)$. When we consider p as an element in $\mathbb{Z}[\omega]$, we see that $p \mid (a + 1 + 2\omega)(a - 1 - 2\omega)$. If p is prime in $\mathbb{Z}[\omega]$, it must divide exactly one of these factors; however $p \equiv 1 \pmod{3}$ (specifically, $p \neq 2$) and $\frac{2}{p} \notin \mathbb{Z}$, so this is nonsense, and p is composite in $\mathbb{Z}[\omega]$.

Thus we may write $p = \pi\eta$, with neither π, η units. Since $N(p) = p^2 = N(\pi)N(\eta)$, we have that $N(\pi) = \pi\bar{\pi} = p$. Hence, $\exists \pi \in \mathbb{Z}[\omega]$ such that $\pi\bar{\pi} = p$, as desired.

Uniqueness (up to associates) follows from the fact that $\mathbb{Z}[\omega]$ is a UFD; *i.e.* if $\pi\bar{\pi} = p = \gamma\bar{\gamma}$ (π, γ prime), then $\pi \mid \gamma$ and $\gamma \mid \pi$.

□

Theorem 4.22. $3 = -\omega^2(1 - \omega)^2$, and $(1 - \omega)$ is prime in $\mathbb{Z}[\omega]$.

Proof. $x^2 + x + 1 = (x - \omega)(x - \omega^2)$. Setting $x = 1$ we obtain:

$$3 = (1 - \omega)(1 - \omega^2) = (1 - \omega)(1 + \omega)(1 - \omega) = (1 + \omega)(1 - \omega)^2 = -\omega^2(1 - \omega)^2.$$

Since $-\omega^2$ is a unit, and $N(1 - \omega) = 3$, it follows that $(1 - \omega)$ is prime by Theorem 4.18.

□

To summarize, primes in $\mathbb{Z}[\omega]$ may be classified into three types (up to associates):

- the rational primes $p \equiv 2 \pmod{3}$ of norm p^2 ,
- the primes π of prime norm $\pi\bar{\pi} = p \equiv 1 \pmod{3}$,
- the prime $(1 - \omega)$ which lies over 3; *i.e.* $3 = -\omega^2(1 - \omega)^2$.

4.2.4 Primary elements in $\mathbb{Z}[\omega]$

Definition 4.23. A primary element in $\mathbb{Z}[\omega]$ is an element of the form $-1 + 3\beta$ for some $\beta \in \mathbb{Z}[\omega]$; *i.e.* the element $\alpha = a + b\omega \in \mathbb{Z}[\omega]$ is primary if and only if $a \equiv 2 \pmod{3}$ and $b \equiv 0 \pmod{3}$.

Note that any element $\alpha \in \mathbb{Z}[\omega]$ that is prime to $1 - \omega$ is associate to a product

of primary elements; *i.e.*

$$\alpha = \omega^j \prod_{i=0}^t \pi_i^{e_i}.$$

This is obvious when considering rational prime factors as any rational prime $q \equiv 2 \pmod{3}$ is already primary. We will now examine the remaining case.

Theorem 4.24. *If $N(\pi) = p$ a prime in \mathbb{Z} , and $p \equiv 1 \pmod{3}$ then π has exactly one primary associate.*

Proof. Since $N(\pi) = p = a^2 - ab + b^2 \equiv 1 \pmod{3}$, we may restrict the congruence possibilities for a and b to six: $a \equiv 0 \pmod{3}$, $b \pmod{3} \in \{1, 2\}$ $a \equiv 1 \pmod{3}$, $b \pmod{3} \in \{0, 1\}$ $a \equiv 2 \pmod{3}$, $b \pmod{3} \in \{0, 2\}$.

The six associates of π may be written:

$$\begin{aligned} \pi &= a + b\omega \\ -\pi &= -a - b\omega \\ \omega\pi &= a\omega + b\omega^2 = -b + (a - b)\omega \\ -\omega\pi &= -a\omega - b\omega^2 = b + (b - a)\omega \\ \omega^2\pi &= a\omega^2 + b = (b - a) - a\omega \\ -\omega^2\pi &= -a\omega^2 - b = (a - b) + a\omega. \end{aligned}$$

The theorem may now be proved by noticing the following:

$$\begin{aligned}
a \equiv 0, b \equiv 1 \pmod{3} &\implies -\omega^2\pi \text{ is primary,} \\
a \equiv 0, b \equiv 2 \pmod{3} &\implies \omega^2\pi \text{ is primary,} \\
a \equiv 1, b \equiv 0 \pmod{3} &\implies -\pi \text{ is primary,} \\
a \equiv 1, b \equiv 1 \pmod{3} &\implies \omega\pi \text{ is primary,} \\
a \equiv 2, b \equiv 0 \pmod{3} &\implies \pi \text{ is primary,} \\
a \equiv 2, b \equiv 2 \pmod{3} &\implies -\omega\pi \text{ is primary,}
\end{aligned}$$

and these are all the possibilities when $p = a^2 - ab + b^2 \equiv 1 \pmod{3}$. □

Corollary 4.25. *Every prime $\pi \in \mathbb{Z}[\omega]$, $\pi \neq (1 - \omega)$ has exactly one primary associate.*

Given a rational prime $p \in \mathbb{Z}$, it is easy, using an algorithm of Williams [Wil86] to obtain the primary $\pi \in \mathbb{Z}[\omega]$ such that $p = \pi\bar{\pi}$. This algorithm will be discussed in detail in Chapter 5.

4.3 Cubic Residue Character

First, note that if $\pi \in \mathbb{Z}[\omega]$ is prime, then $(\mathbb{Z}[\omega]/\pi\mathbb{Z}[\omega])^*$ has order $N(\pi) - 1$. Thus:

$$\zeta^{N(\pi)-1} \equiv 1 \pmod{\pi} \text{ for all } \pi \nmid \zeta.$$

Theorem 4.26. *If $\pi \in \mathbb{Z}[\omega]$ is prime, $N(\pi) \neq 3$, and $\pi \nmid \zeta$, then there exists a unique integer $k \in \{0, 1, 2\}$ such that*

$$\zeta^{(N(\pi)-1)/3} \equiv \omega^k \pmod{\pi}$$

Proof. [IR90, Prop. 9.3.2] □

Definition 4.27. *If $N(\pi) \neq 3$, we define the cubic residue character of ζ modulo π as follows:*

1. $\left(\frac{\zeta}{\pi}\right)_3 = 0$ if $\pi \mid \zeta$,
2. $\left(\frac{\zeta}{\pi}\right)_3 \equiv \zeta^{(N(\pi)-1)/3} \pmod{\pi}$ otherwise, where $\left(\frac{\zeta}{\pi}\right)_3 \in \{1, \omega, \omega^2\}$.

Theorem 4.28. *(Properties of the cubic residue character)*

Let $\pi \in \mathbb{Z}[\omega]$ be prime. Then

1. $\exists \alpha \in \mathbb{Z}[\omega]$ such that $\alpha^3 \equiv \zeta \pmod{\pi}$ is solvable $\iff \left(\frac{\zeta}{\pi}\right)_3 = 1$,
2. $\left(\frac{\alpha\beta}{\pi}\right)_3 = \left(\frac{\alpha}{\pi}\right)_3 \left(\frac{\beta}{\pi}\right)_3$,
3. If $\alpha \equiv \beta \pmod{\pi}$ then $\left(\frac{\alpha}{\pi}\right)_3 = \left(\frac{\beta}{\pi}\right)_3$,
4. $\overline{\left(\frac{\alpha}{\pi}\right)_3} = \left(\frac{\alpha}{\pi}\right)_3^2 = \left(\frac{\alpha}{\pi}\right)_3^{-1}$,
5. $\left(\frac{\alpha}{\pi}\right)_3 = \left(\frac{\bar{\alpha}}{\bar{\pi}}\right)_3$.

Proof. [IR90, §9.3] □

Corollary 4.29. *(Behavior of units)*

1. $\left(\frac{-1}{\pi}\right)_3 = 1$,
2. $\left(\frac{\omega}{\pi}\right)_3 = \omega^{\frac{N(\pi)-1}{3}}$.

Theorem 4.30. *(Cubic Reciprocity) Let π_1, π_2 be prime and primary in $\mathbb{Z}[\omega]$. If $N(\pi_1), N(\pi_2) \neq 3$ and $N(\pi_1) \neq N(\pi_2)$ then $\left(\frac{\pi_2}{\pi_1}\right)_3 = \left(\frac{\pi_1}{\pi_2}\right)_3$.*

Proof. [IR90, §9.3, 9.4] □

Theorem 4.31. *If $N(\pi) \neq 3$, with π prime and primary, write $\pi = r + s\omega$ with $r = 3m - 1, s = 3n$. Then*

1. $\left(\frac{1-\omega}{\pi}\right)_3 = \omega^{2m} = \omega^{2\frac{r-1}{3}}$
2. $\left(\frac{\omega}{\pi}\right)_3 = \omega^{m+n} = \omega^{\frac{r+s+1}{3}}$

Proof. 1. First, assume $\pi = q$, a rational prime $\equiv -1 \pmod{3}$. Observe that

$$(1 - \omega)^2 = 1 - 2\omega - \omega^2 = 1 - 2\omega - 1 - \omega = -3\omega \text{ and thus:}$$

$$\left(\frac{1 - \omega}{q}\right)_3^2 = \left(\frac{-3}{q}\right)_3 \left(\frac{\omega}{q}\right)_3.$$

From Property 5 of Theorem 4.28 it is clear that $\left(\frac{-3}{q}\right)_3 = 1$. From Definition 4.27, $\left(\frac{\omega}{q}\right)_3 = \omega^{\frac{q^2-1}{3}}$. Hence $\left(\frac{1-\omega}{q}\right)_3^2 = \omega^{\frac{q^2-1}{3}}$ and

$$\left(\frac{1 - \omega}{q}\right)_3^4 = \left(\frac{1 - \omega}{q}\right)_3 = \omega^{\frac{2}{3}(q^2-1)}.$$

Finally, $q^2 - 1 = (3m - 1)^2 - 1 = 9m^2 - 6m$, and hence $\frac{2}{3}(q^2 - 1) \equiv 2m \pmod{3}$, giving

$$\left(\frac{1 - \omega}{q}\right)_3 = \omega^{2m}$$

as desired.

For the case where π is not rational, see [Wil77].

2. From Corollary 4.29, we have $\left(\frac{\omega}{\pi}\right)_3 = \omega^{\frac{N(\pi)-1}{3}}$. Furthermore, $\omega^3 \equiv 1$, so we need simply consider the exponent modulo 3. It is clear that $N(\pi) = r^2 + s^2 - rs = 9m^2 - 6m + 1 + 9n^2 - 9mn + 3n$, and hence, $\frac{N(\pi)-1}{3} \equiv -2m + n \equiv m + n \pmod{3}$. Thus $\left(\frac{\omega}{\pi}\right)_3 = \omega^{m+n} \equiv \omega^{m+n} \pmod{\pi}$ as desired.

□

4.3.1 The Cubic Jacobi Symbol

We can extend the notion of cubic residue to include non-primes in the following manner:²

Definition 4.32. (*Cubic Jacobi Symbol*)

If $\alpha, \tau \in \mathbb{Z}[\omega]$ with $3 \nmid N(\tau)$, we define

$$\left(\frac{\alpha}{\tau}\right)_3 = \begin{cases} 1 & \text{if } \tau \text{ is a unit of } \mathbb{Z}[\omega], \\ \prod_{i=1}^k \left(\frac{\alpha}{\pi_i}\right)_3 & \text{otherwise.} \end{cases} \quad (4.4)$$

where $\tau = \prod_{i=1}^k \pi_i$ and all $\pi_i \in \mathbb{Z}[\omega]$ are prime.

The properties of the Cubic Jacobi Symbol are the same as those given in Theorem 4.28, with the following addition.

Theorem 4.33. (*Bimultiplicity of the Cubic Jacobi Symbol*)

$$1. \left(\frac{\alpha\beta}{\tau}\right)_3 = \left(\frac{\alpha}{\tau}\right)_3 \left(\frac{\beta}{\tau}\right)_3 \quad \text{and} \quad \left(\frac{\alpha}{\tau\rho}\right)_3 = \left(\frac{\alpha}{\tau}\right)_3 \left(\frac{\alpha}{\rho}\right)_3.$$

Proof. The former follows from Theorem 4.28. The latter from Definition 4.32 \square

The behaviour of the units mirrors Corollary 4.29.

Corollary 4.34. (*Behaviour of units*)

$$1. \left(\frac{-1}{\tau}\right)_3 = 1, \\ 2. \left(\frac{\omega}{\tau}\right)_3 = \omega^{\frac{N(\tau)-1}{3}}.$$

Proof. 1. Follows directly from Definition 4.32.

²In much the same way as the Legendre symbol was extended to the Jacobi symbol for dealing with quadratic characters.

2. We can show $\frac{\mathbb{N}(\pi_1)-1}{3} + \frac{\mathbb{N}(\pi_2)-1}{3} \equiv \frac{\mathbb{N}(\pi_1\pi_2)}{3} \pmod{3}$. The corollary follows shortly thereafter. □

Lemma 4.35. *Let $\pi_i \in \mathbb{Z}[\omega]$ be prime and primary for $i = 1, \dots, k$; i.e. $\pi_i = r_i + s_i\omega$ such that $r_i = 3m_i - 1, s_i = 3n_i$. Then $\prod_{i=1}^k \pi_i \equiv (-1)^{k-1} \sum_{i=1}^k m_i + n_i\omega \pmod{9}$.*

Proof. Observe that $\pi_i\pi_j = (r_i + s_i\omega)(r_j + s_j\omega) = r_i r_j - s_i s_j + (r_i s_j + r_j s_i - s_i s_j)\omega = A + B\omega$.

Through substitution, it is clear that $A + B\omega \equiv (-1)[(m_i + m_j) + (n_i + n_j)\omega] \pmod{9}$.

Continuing this argument, we obtain our lemma; namely

$$\prod_{i=1}^k \pi_i \equiv (-1)^{k-1} \sum_{i=1}^k m_i + n_i\omega \pmod{9}.$$

□

Proposition 4.36. *Let $\alpha = r + s\omega \in \mathbb{Z}[\omega]$ be primary; i.e. $r = 3m - 1, s = 3n$ for some $m, n \in \mathbb{Z}$. Then*

$$\left(\frac{\omega}{\alpha}\right)_3 = \omega^{m+n}.$$

Proof. Write α as a product of primary primes; i.e. $\alpha = (-1)^{k-1} \prod_{i=1}^k \pi_i$. Furthermore, write $\pi_i = r_i + s_i\omega$ with $r_i = 3m_i - 1$ and $s_i = 3n_i$.

From the properties of the cubic Jacobi symbol

$$\left(\frac{\omega}{\alpha}\right)_3 = \prod_{i=1}^k \omega^{\frac{\mathbb{N}(\pi_i)-1}{3}} = \omega^{\sum_{i=1}^k \frac{\mathbb{N}(\pi_i)-1}{3}}.$$

Since exponents of ω are cyclic modulo 3, we need only show that

$$\sum_{i=1}^k \frac{N(\pi_i) - 1}{3} \equiv m + n \pmod{3}.$$

By Lemma 4.35 and Theorem 4.31, this is equivalent to showing that

$$\sum_{i=1}^k N(\pi_i) - 1 \equiv 3 \sum_{i=1}^k m_i + n_i \pmod{9}.$$

This can be seen from the following:

$$\begin{aligned} \sum_{i=1}^k N(\pi_i) - 1 &= \sum_{i=1}^k r_i^2 + s_i^2 - r_i s_i - 1 \\ &= \sum_{i=1}^k (-1 + 3m_i)^2 + 9n_i^2 + 3n_i - 9m_i n_i - 1 \\ &= \sum_{i=1}^k 1 - 6m_i + 9m_i^2 + 9n_i^2 + 3n_i - 9m_i n_i - 1 \\ &\equiv \sum_{i=1}^k 3m_i + 3n_i \pmod{9}. \end{aligned}$$

□

Finally, we will prove a reciprocity law as it applies to the cubic Jacobi symbol:

Theorem 4.37. (*Cubic Reciprocity*) *Let α, β be primary in $\mathbb{Z}[\omega]$ and of coprime norm not equal to 3. Then $\left(\frac{\alpha}{\beta}\right)_3 = \left(\frac{\beta}{\alpha}\right)_3$.*

Proof. Let $\alpha = (-1)^{a-1} \prod_{i=1}^a \pi_i$ and $\beta = (-1)^{b-1} \prod_{j=1}^b \rho_j$ be the decomposition of α and β into primary primes. Since $\left(\frac{\pm 1}{\tau}\right)_3 = \left(\frac{\tau}{\pm 1}\right)_3 = 1$ for $\tau \in \mathbb{Z}[\omega]$, we may ignore

the signs and observe that:

$$\left(\frac{\alpha}{\beta}\right)_3 = \prod_{i=1}^a \left(\frac{\pi_i}{\beta}\right)_3 = \prod_{i=1}^a \prod_{j=1}^b \left(\frac{\pi_i}{\rho_j}\right)_3 = \prod_{i=1}^a \prod_{j=1}^b \left(\frac{\rho_j}{\pi_i}\right)_3 = \left(\frac{\beta}{\alpha}\right)_3.$$

□

4.4 Eisenstein Pseudocubes

Definition 4.38. Let p be a fixed rational prime. Define $\mu_p = a + b\omega \in \mathbb{Z}[\omega]$ $a, b \in \mathbb{Z}$ to be an element of $\mathbb{Z}[\omega]$ of minimal norm such that:

1. μ_p is primary,
2. $\gcd(a, b) = 1$,
3. $\left(\frac{q}{\mu_p}\right)_3 = 1$ for all rational primes $q \leq p$,
4. μ_p is not a cube in $\mathbb{Z}[\omega]$.

We will call μ_p a minimal Eisenstein pseudocube (or simply an Eisenstein pseudocube) for the prime p .

Eisenstein pseudocubes always occur in conjugate pairs, as shown in the following proposition:³

Proposition 4.39. μ_p is an Eisenstein pseudocube for the prime p if and only if $\overline{\mu_p}$ is also an Eisenstein pseudocube.

Proof. Clearly, $N(\mu_p) = N(\overline{\mu_p})$, so if one solution is of minimal norm, the other is also. For $\overline{\mu_p} = a + b\overline{\omega} = (a - b) - b\omega$ to be an Eisenstein pseudocube, it must satisfy the 4 criteria of Definition 4.38.

³The case where $\mu_p = \overline{\mu_p}$ can be ignored, as $\gcd(a, 0) \neq 1$ unless $\mu_p = 1$ (a perfect cube).

1. $\mu_p = a + b\omega$ is primary; *i.e.* $a \equiv 2 \pmod{3}, b \equiv 0 \pmod{3}$. Thus $\overline{\mu_p} = (a - b) - b\omega$ is primary by the same criteria.
2. $\gcd(a - b, -b) = \gcd(a, b) = 1$
3. $1 = \left(\frac{q}{\mu_p}\right)_3 = \left(\frac{q}{\mu_p}\right)_3^{-1} = \overline{\left(\frac{q}{\mu_p}\right)_3} = \left(\frac{q}{\overline{\mu_p}}\right)_3$, so $\left(\frac{q}{\overline{\mu_p}}\right)_3 = 1$ for all rational primes $q \leq p$.
4. Assume otherwise. If $\overline{\mu_p}$ is a cube then $\exists(x + y\omega)$ such that $(x + y\omega)^3 = \overline{\mu_p}$, but then $(x + y\overline{\omega})^3 = \mu_p$, a contradiction.

Thus, μ_p is an Eisenstein pseudocube if and only if $\overline{\mu_p}$ is an Eisenstein pseudocube. □

4.5 Eisenstein Pseudocubes and Primality Testing

Eisenstein pseudocubes may be employed to prove primality via the following theorem.

Theorem 4.40. (*Berrizbeitia, 2003, private communication*)

Let $\nu = a + b\omega$ be a primary element of $\mathbb{Z}[\omega]$, where $\gcd(a, b) = 1$, ν is not a unit, prime, or perfect power in $\mathbb{Z}[\omega]$, and $N(\nu) < N(\mu_p)$. Then there must exist a rational prime $q \leq p$ such that $\left(\frac{q}{\nu}\right)_3 \not\equiv q^{(N(\nu)-1)/3} \pmod{\nu}$.

Proof. We will assume that $\left(\frac{q}{\nu}\right)_3 \equiv q^{(N(\nu)-1)/3} \pmod{\nu}$ for all $q \leq p$ and show that this leads to a contradiction.

Recall Definition 4.38. Since $N(\nu) < N(\mu_p)$, there must exist some $q \leq p$ such that $\left(\frac{q}{\nu}\right)_3 \neq 1$. Fixing q , this implies $q^{(N(\nu)-1)/3} \pmod{\nu} \in \{\omega, \omega^2\}$.

Let π be any primary prime divisor of ν and define $k, s \in \mathbb{Z}$ by $3^k \parallel N(\nu) - 1$, $3^s \parallel N(\pi) - 1$. If σ_1 is the multiplicative order of $q \pmod{\pi}$ then $\sigma_1 \mid N(\pi) - 1$ and

since $\pi \mid \nu$, we must have $\sigma_1 \mid N(\nu) - 1$. But $\sigma_1 \nmid (N(\nu) - 1)/3$ implies that $3^k \parallel \sigma_1$ and hence $s \geq k$. Also, since $\pi \mid \nu$, we have $N(\pi) \leq N(\nu)$. Hence there exists a prime $r \leq p$ such that $\left(\frac{r}{\pi}\right)_3 \neq 1$. If σ_2 is the multiplicative order of $r \pmod{\pi}$ then $\sigma_2 \mid (N(\pi) - 1)$ and $\sigma_2 \nmid (N(\pi) - 1)/3$, giving $3^s \parallel \sigma_2$. Since $\sigma_2 \mid (N(\nu) - 1)$, we see that $k \geq s$, and it follows that $s = k$.

Let $N(\nu) = 1 + 3^k t_\nu$, $N(\pi) = 1 + 3^k t_\pi$, $3 \nmid t_\pi t_\nu$. We have for any $q \leq p$,

$$\left(\frac{q}{\nu}\right)_3 \equiv q^{3^{k-1}t_\nu} \pmod{\nu}, \quad \left(\frac{q}{\pi}\right)_3 \equiv q^{3^{k-1}t_\pi} \pmod{\pi}.$$

Hence $\left(\frac{q}{\nu}\right)_3 \equiv q^{3^{k-1}t_\nu} \pmod{\pi}$, and $\left(\frac{q}{\pi}\right)_3 \equiv q^{3^{k-1}t_\pi} \pmod{\pi}$. Since π is primary and $\left(\frac{q}{\nu}\right)_3, \left(\frac{q}{\pi}\right)_3 \in \{1, \omega, \omega^2\}$, we must have

$$\left(\frac{q}{\nu}\right)_3^{t_\pi} = \left(\frac{q}{\pi}\right)_3^{t_\nu}. \quad (4.5)$$

Furthermore, since $3 \nmid t_\pi t_\nu$, we see that

$$\left(\frac{q}{\nu}\right)_3 = 1 \iff \left(\frac{q}{\pi}\right)_3 = 1 \quad \text{for all } q \leq p. \quad (4.6)$$

The same must also hold for any other primary prime divisor ρ of ν . Since we assume that ν is not a prime or prime power, we may assume that $\rho \neq \pi$. Since $\rho\pi \mid \nu$ we have $N(-\rho\pi) = N(-\bar{\rho}\pi) \leq N(\nu) < N(\mu_p)$. Since $-\rho\pi$ is primary, there must be some prime $s \leq p$ such that

$$\left(\frac{s}{\rho\pi}\right)_3 \neq 1. \quad (4.7)$$

Since by Equation 4.6, if either $\left(\frac{s}{\rho}\right)_3$ or $\left(\frac{s}{\pi}\right)_3$ is 1, so must be the other. Hence, neither is 1. By Equation 4.7 this means that

$$\left(\frac{s}{\rho}\right)_3 = \left(\frac{s}{\pi}\right)_3,$$

and hence by Equation 4.5, $t_\pi \equiv t_\rho \pmod{3}$. Thus $\left(\frac{q}{\rho}\right)_3 = \left(\frac{\rho}{\pi}\right)_3$ for all $q \leq p$. Since $\overline{\left(\frac{q}{\rho}\right)_3} = \left(\frac{q}{\rho}\right)_3^{-1}$ and $\overline{\left(\frac{q}{\rho}\right)_3} = \left(\frac{q}{\rho}\right)_3$, we get $1 = \left(\frac{q}{\pi}\right)_3 \left(\frac{q}{\rho}\right)_3 = \left(\frac{q}{\pi\rho}\right)_3$ for all $q \leq p$. As $N(-\pi\bar{\rho}) < N(\mu_p)$ with $-\pi\bar{\rho}$ primary, we have a contradiction to the definition of μ_p . \square

4.6 Primality Proving in the Integers

Recall that if $m \equiv 1 \pmod{3}$ and m is a prime in \mathbb{Z} , then $m = \nu\bar{\nu}$, where ν is a primary prime in $\mathbb{Z}[\omega]$. Furthermore, if q is any rational prime, then

$$\left(\frac{q}{\nu}\right)_3 \equiv q^{\frac{m-1}{3}} \pmod{\nu}.$$

If we have a table of Eisenstein pseudocubes available to us, Berrizbeitia's result gives us a means to certify the primality of $m \equiv 1 \pmod{3}$; *i.e.*

1. Test that m is not a perfect power; *e.g.* via [Ber98].
2. Find a primary $\nu \in \mathbb{Z}[\omega]$ such that $N(\nu) = m$.
3. From a precomputed table of Eisenstein pseudocubes, choose $\mu_p \in \mathbb{Z}[\omega]$ of minimal norm such that $m < N(\mu_p)$.
4. For each prime $q \leq p$, test $\left(\frac{q}{\nu}\right)_3 \equiv q^{\frac{m-1}{3}} \pmod{\nu}$. If the test succeeds for all q , then m is prime.

At first glance, it is not obvious how to find a primary $\nu \in \mathbb{Z}[\omega]$ such that $N(\nu) = m$. In the subsequent chapter, we will examine this problem in some detail.

Chapter 5

How to Solve a Certain Diophantine Problem

Engineers like to solve problems. If there are no problems handily available, they will create their own...

—Scott Adams. *The Dilbert Principle*.

In the preceding chapter, we developed a primality proving algorithm for an integer $m \equiv 1 \pmod{3}$, that required us to find an Eisenstein integer $\pi = a + b\omega \in \mathbb{Z}[\omega]$ such that $N(\pi) = a^2 - ab + b^2 = m$. In [Wil86], Williams observed that we can find such a π as follows.

Algorithm 5.1: Williams' Algorithm

Input: $m \in \mathbb{Z}^{>0}$, x such that $x^2 \equiv -3 \pmod{m}$.

Output: $s, t \in \mathbb{Z}$ such that $s^2 + 3t^2 = m$.

- 1: Evaluate the Euclidean algorithm for (x, m) , obtaining a sequence of remainders R_i stopping when $R_n^2 < m < R_{n-1}^2$.
- 2: $s \leftarrow \pm R_n$
- 3: **if** $3 \mid R_{n-1}$ and $R_{n-1}^2 < 9m$ **then**
- 4: $t \leftarrow \pm R_{n-1}/3$
- 5: **else**
- 6: $t \leftarrow \pm(R_n - k)$ where

$$k \equiv (3R_n\epsilon_{n-1} - \epsilon_n\epsilon_{n-1}R_n - 2R_{n-1})/6 \pmod{R_n}.$$

$$0 < k < R_n, R_i \equiv \epsilon_i \pmod{3}, |\epsilon_i| \leq 1.$$

7: **end if**

8: **return** (s, t)

To use this algorithm, first obtain an x such that

$$x^2 \equiv -3 \pmod{m}.$$

Using Algorithm 5.1, find s, t such that $m = s^2 + 3t^2$. Then

$$m = s^2 + 3t^2 = (s + t)^2 - (2t)(s + t) + (2t)^2. \quad (5.1)$$

Thus, choose a, b from amongst $\pm\{2t, (s + t)\}$ such that $a \equiv -1 \pmod{3}$ and $3 \mid b$; *i.e.* such that $\pi = a + b\omega$ is primary.

To see why this algorithm works, it is necessary to delve into the study of a particular class of Diophantine equations—notably, the following: given $m, f, g \in \mathbb{N}$, find $x, y \in \mathbb{Z}$ (if they exist) such that

$$fx^2 + gy^2 = m. \quad (5.2)$$

We call a solution with $x \neq 0$ and $y > 0$ a *non-trivial* solution. Assume without loss of generality that $g \geq f$. If $f = g = 1$, we add the additional condition that $|x| > y$. Furthermore, (without loss of generality), we can restrict ourselves to the case where f, g , and m are pairwise relatively prime, as follows. First, we may assume $\gcd(f, g) = 1$. Otherwise, $d = \gcd(f, g)$ implies that $d \mid m$ and (5.2) reduces to solving

$$\frac{f}{d}x^2 + \frac{g}{d}y^2 = \frac{m}{d}, \quad (5.2a)$$

where $\gcd\left(\frac{f}{d}, \frac{g}{d}\right) = 1$.

Next, if $\gcd(g, m) = d_1d_2^2$ where d_1 is squarefree, then $d_1d_2^2 \mid fx^2$ gives that

$d_1 d_2^2 \mid x^2$, hence $d_1 d_2 \mid x$, and (5.2) becomes

$$f d_1 \left(\frac{x}{d_1 d_2} \right)^2 + \frac{g}{d_1 d_2^2} y^2 = \frac{m}{d_1 d_2^2}, \quad (5.2b)$$

so we assume $\gcd(g, m) = 1$. Similarly, $\gcd(f, m) = 1$.

Finally, we define a solution as *primitive* if $\gcd(x, y) = 1$. Note that if $\gcd(x, y) = c$, then $c^2 \mid m$ and we may instead solve

$$f(x/c)^2 + g(y/c)^2 = m/c^2, \quad (5.2c)$$

where $\gcd\left(\frac{x}{c}, \frac{y}{c}\right) = 1$.

Before proceeding, it will be useful to first review certain properties of simple continued fractions.

5.1 Simple Continued Fractions

Consider $K, L \in \mathbb{N}$. Using the Euclidean algorithm, we can produce a series of partial quotients q_i from $\frac{K}{L}$ as follows. Set $R_{-2} = K, R_{-1} = L$, then:

$$R_{-2} = K = q_0 R_{-1} + R_0 \quad (5.3)$$

$$R_{-1} = L = q_1 R_0 + R_1 \quad (5.4)$$

$$R_0 = q_2 R_1 + R_2$$

\vdots

$$R_{n-2} = q_n R_{n-1} + R_n. \quad (5.5)$$

This process can be used to obtain the *continued fraction expansion* of $\frac{K}{L}$ as follows:

$$\begin{aligned} \frac{K}{L} = \frac{R_{-2}}{R_{-1}} &= q_0 + \frac{1}{R_{-1}/R_0} \\ \frac{R_{-1}}{R_0} &= q_1 + \frac{1}{R_0/R_1} \\ \frac{R_0}{R_1} &= q_2 + \frac{1}{R_1/R_2} \\ &\vdots \\ \frac{R_{n-2}}{R_{n-1}} &= q_n + \frac{1}{R_{n-1}/R_n}. \end{aligned}$$

And thus,

$$\frac{K}{L} = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \cdots + \frac{1}{q_n}}}}$$

which we will more compactly write as $K/L = [q_0; q_1, q_2, \dots, q_n]$. If $q_i \in \mathbb{Z}$ and $q_i > 0$ for $i \geq 1$, the continued fraction is said to be *simple*. It is straightforward to show that every $K/L \in \mathbb{Q}$ has exactly two simple continued fraction representations; *i.e.*

$$\frac{K}{L} = [q_0; q_1, q_2, \dots, q_n] = [q_0; q_1, q_2, \dots, q_n - 1, 1].$$

Recall from the theory of continued fractions [HW79, §10.2] that we refer to a

truncated representation of this sequence; *i.e.*

$$[q_0; q_1, \dots, q_k] \quad (0 \leq k \leq n)$$

as the k^{th} convergent to $[q_0; q_1, q_2, \dots, q_n]$, denoted C_k . Furthermore, we can define the recursive sequences,

$$\begin{aligned} A_{-2} = 0, A_{-1} = 1 & & B_{-2} = 1, B_{-1} = 0 \\ A_k = q_k A_{k-1} + A_{k-2} & & B_k = q_k B_{k-1} + B_{k-2} \end{aligned} \quad (5.6)$$

such that the k^{th} convergent is given by $C_k = \frac{A_k}{B_k}$.

We observe the following property of the sequence B_i :

Proposition 5.1.

$$B_i \geq B_{i-1} \text{ for } i \geq -1, \quad (5.7)$$

and this inequality is strict when $i > -1$.

Proof. Observe that $B_0 = 1$, $B_1 = q_1 \geq 1$, $B_i = q_i B_{i-1} + B_{i-2} \geq B_{i-1} + 1$. □

Next, observe the following.

Proposition 5.2.

$$L = B_i R_{i-1} + B_{i-1} R_i \text{ for } i = -1, 0, 1, \dots, n. \quad (5.8)$$

Proof. If $i = -1$, $L = R_{-1}$, which is true by definition. For $i = 0$, $L = B_0 R_{-1} + B_{-1} R_0 = r_{-1} = L$. Now assume the proposition is true for $i = -1, 0, 1, \dots, n-1$. If

$i = n$, then

$$\begin{aligned}
L &= B_n R_{n-1} + B_{n-1} R_n \\
&= (q_n B_{n-1} + B_{n-2}) R_{n-1} + B_{n-1} R_n \\
&= B_{n-1} (q_n R_{n-1} + R_n) + B_{n-2} R_{n-1} \\
&= B_{n-1} R_{n-2} + B_{n-2} R_{n-1}
\end{aligned}$$

which is true by the induction hypothesis. \square

Proposition 5.3.

$$LA_i - KB_i = (-1)^{i+1} R_i \quad (5.9)$$

Proof. For $i = 0$, $-R_0 = LA_0 - KB_0 = q_0 R_{-1} - R_{-2}$, which is true by (5.3).

Consider $i = 1$. By (5.4),

$$\begin{aligned}
R_1 &= LA_1 - KB_1 \\
&= R_{-1} (q_0 q_1 + 1) - R_{-2} q_1 \\
&= R_{-1} + q_1 (q_0 R_{-1} - R_{-2}) = R_{-1} + q_1 R_0.
\end{aligned}$$

Assume the proposition is true for $i < n$. If $i = n$:

$$\begin{aligned}
(-1)^{n+1} R_n &= LA_n - KB_n \\
&= L(q_n A_{n-1} + A_{n-2}) - K(q_n B_{n-1} + B_{n-2}) \\
&= q_n (-1)^{n-1} (LA_{n-1} - KB_{n-1}) + (-1)^{n-2} (LA_{n-2} - KB_{n-2}) \\
&= (-1)^{n-1} (q_n R_{n-2} - R_{n-2}) = (-1)^{n+1} R_n.
\end{aligned}$$

□

Proposition 5.4. *Let A_i/B_i be the i^{th} convergent to $x \in \mathbb{Q}$. Then*

$$x = \frac{A_n}{B_n} + \frac{\epsilon}{B_n B_{n+1}} \quad \text{where } -1 \leq \epsilon \leq 1. \quad (5.10)$$

Proof. [HW79, Theorem 164]

□

Finally, we will state an important property about convergents; namely that they provide the best possible approximation to a given $x \in \mathbb{Q}$ among all fractions of no greater complexity. To see what we mean by this notion, consider the following.

Theorem 5.5. *Let $x \in \mathbb{Q}$. If*

$$\left| \frac{p}{q} - x \right| < \frac{1}{2q^2}$$

then $\frac{p}{q}$ is a convergent in the simple continued fraction expansion of x .

Proof. [HW79, Theorem 184]

□

Note that we can show a relationship between certain related continued fraction expansions.

Lemma 5.6. *Let $K, L \in \mathbb{N}$ with $0 < L < K/2$. By the Euclidean Algorithm*

$$K = q_0 L + R_0$$

$$L = q_1 R_0 + R_1$$

$$R_0 = q_2 R_1 + R_2$$

$$\vdots$$

$$R_{s-2} = q_s R_{s-1} + R_s,$$

i. e.

$$\frac{K}{L} = [q_0; q_1, q_2, \dots, q_s]. \quad (5.11)$$

Then

$$\frac{L}{K} = [0; q_0, q_1, \dots, q_s], \quad (5.12)$$

$$\frac{K}{(K-L)} = [1; q_0 - 1, q_1, \dots, q_s], \quad \text{and} \quad (5.13)$$

$$\frac{(K-L)}{K} = [0; 1, q_0 - 1, q_1, \dots, q_s]. \quad (5.14)$$

Proof. Observe that $L < K$, thus

$$L = 0K + L$$

$$K = q_0L + R_0$$

$$L = q_1R_0 + R_1$$

$$\vdots$$

$$R_{s-2} = q_sR_{s-1} + R_s,$$

and it is clear that

$$\frac{L}{K} = [0; q_0, q_1, \dots, q_s].$$

Furthermore, $(K - L) = (q_0 - 1)L + R_0$, and since $L < (K - L)$, we may write

$$\begin{aligned} K &= (K - L) + L \\ (K - L) &= (q_0 - 1)L + R_0 \\ L &= q_1 R_0 + R_1 \\ &\vdots \\ R_{s-2} &= q_s R_{s-1} + R_s; \end{aligned}$$

i.e.

$$\frac{K}{(K - L)} = [1; q_0 - 1, q_1, \dots, q_s].$$

Finally, from

$$\begin{aligned} (K - L) &= 0K + (K - L) \\ K &= (K - L) + L \\ (K - L) &= (q_1 - 1)R_0 + R_1 \\ &\vdots \\ R_{s-2} &= q_s R_{s-1} + R_s, \end{aligned}$$

it is clear that

$$\frac{(K - L)}{K} = [0; 1, q_0 - 1, q_1, \dots, q_s].$$

□

We are now ready to turn our attention to the problem at hand.

5.2 The Hermite-Serret Algorithm

In 1848, in back-to-back articles in the *Journal de Mathématiques Pures et Appliquées*, Hermite [Her48] and Serret [Ser48] published algorithms for solving (5.2) in the case where $f = g = 1$. Though similar in structure, Hermite's algorithm had the added advantage of an early stopping condition. His argument proceeds as follows.

Since $p \equiv 1 \pmod{4}$ and $\left(\frac{-1}{p}\right) = 1$, there exists $a \in \mathbb{Z}, a < p$ such that

$$a^2 + 1 \equiv 0 \pmod{p}. \quad (5.15)$$

Find a specific a satisfying (5.15) and convert $\frac{a}{p}$ to a simple continued fraction until we obtain two consecutive convergents $\frac{A_n}{B_n}, \frac{A_{n+1}}{B_{n+1}}$ such that $B_n < \sqrt{p} < B_{n+1}$. Clearly, $B_n^2 < p$ and $\frac{p}{B_{n+1}} < \sqrt{p}$.

From (5.10) we know that

$$\frac{a}{p} = \frac{A_n}{B_n} + \frac{\epsilon}{B_n B_{n+1}},$$

where $-1 \leq \epsilon \leq 1$. From this we obtain

$$B_n a - A_n p = \epsilon \cdot \frac{p}{B_{n+1}}$$

and hence

$$(B_n a - A_n p)^2 < p.$$

Furthermore

$$(B_n a - A_n p)^2 + B_n^2 < 2p \quad (5.16)$$

as $B_n^2 < p$. From condition (5.15), we note that the left hand side of this inequality is an integer multiple of p , and since $\frac{a}{p} \neq \frac{A_n}{B_n}$, the equality $(B_n a - A_n p)^2 + B_n^2 = p$ holds.

In [Bri72], Brillhart noted that calculation of convergents in the Hermite-Serret algorithm was unnecessary, and could be replaced by the remainders obtained from the Euclidean algorithm applied to a and p . Interestingly, the key observation required for this improvement—the palindromic nature of the continued fraction expansion for $\frac{p}{a}$ —was made by Serret himself in [Ser48].

5.2.1 Adapting Hermite-Serret

With a little work, the Hermite-Serret algorithm can be adapted to find $s, t \in \mathbb{Z}$ such that $p = s^2 + 3t^2$, $p \equiv 1 \pmod{3}$, prime.

Consider $p \in \mathbb{Z}$, an odd prime such that $p \equiv 1 \pmod{3}$. Via quadratic reciprocity,

$$\begin{aligned} \left(\frac{-3}{p}\right) &= \left(\frac{-1}{p}\right) \left(\frac{3}{p}\right) \\ &= (-1)^{\frac{p-1}{2}} \left(\frac{p}{3}\right) (-1)^{\frac{p-1}{2} \frac{3-1}{2}} \\ &= \left(\frac{p}{3}\right) = \left(\frac{1}{3}\right) = 1, \end{aligned}$$

and hence, $a^2 + 3 \equiv 0 \pmod{p}$ has a solution.

Follow Hermite's algorithm to the point where $(B_n a - A_n p)^2 < p$. Now replace the inequality (5.16) with

$$(B_n a - A_n p)^2 + 3B_n^2 < 4p.$$

Since $p \mid (a^2 + 3)$, $\exists k \in \mathbb{Z}$ such that $(B_n a - A_n p)^2 + 3B_n^2 = kp$ where $1 \leq k \leq 3$.

It is clear that $k \neq 2$, as $p \equiv 1 \pmod{3}$, and $(B_n a - A_n p)^2 \equiv 2 \pmod{3}$ has no solution. If $k = 1$ then

$$p = (B_n a - A_n p)^2 + 3B_n^2, \quad (5.17)$$

and hence, $s = \pm(B_n a - A_n p)$, $t = \pm B_n$.

If $k = 3$, then $(B_n a - A_n p)^2 + 3B_n^2 = 3p$, and thus $3 \mid (B_n a - A_n p)$. We are once again finished, as

$$p = B_n^2 + 3 \left(\frac{B_n a - A_n p}{3} \right)^2 \quad (5.18)$$

and thus $s = \pm B_n$, $t = \pm \left(\frac{B_n a - A_n p}{3} \right)$.

5.2.2 Further Generalizations

In [Wil80], Wilker gave an equivalent (and more general) result for $p = s^2 + 5t^2$ that did not rely on a palindromic continued fraction expansion, giving hope that the technique could be useful in a more general setting. His method, however, does not appear to generalize well to other cases, and so will not be discussed here. Instead, we will turn to an algorithm that is applicable to (5.2) in its full generality, and show that, as with Brillhart's improvement to Hermite-Serret, it too may be adapted to require only remainders from a partial evaluation of the Euclidean algorithm.

5.3 Cornacchia's Algorithm

In 1908 [Cor08], Giuseppe Cornacchia wrote of an algorithm which allows us to solve (5.2) for the case when $f = 1$; *i.e.*

$$x^2 + gy^2 = m.$$

Like Hermite-Serret, Cornacchia's algorithm makes use of continued fractions, solving the Diophantine equation by first obtaining solutions k to the congruence

$$k^2 \equiv -g \pmod{m}, \quad (5.19)$$

and then iterating through a partial evaluation of the Euclidean algorithm on k and m .

Algorithm 5.2: Cornacchia's Algorithm

Input: f, g, m pairwise relatively prime.

Output: Primitive solutions (x, y) for which $fx^2 + gy^2 = m$.

- 1: Solve $k^2 \equiv -gf^{-1} \pmod{m}$.
 - 2: **for** solutions k with $m/2 < k < m$ **do**
 - 3: Apply the Euclidean algorithm to k and m , finding the first remainder R such that $R < \sqrt{m/f}$.
 - 4: **if** $\sqrt{(m - fR^2)/g} \in \mathbb{Z}$ **then**
 - 5: $(R, \sqrt{(m - fR^2)/g})$ is a solution to (5.2).
 - 6: **end if**
 - 7: **end for**
-

In [HMW90], Hardy *et al.* further improved this algorithm to remove the restriction on f by solving instead for

$$k^2 \equiv -gf^{-1} \pmod{m}. \quad (5.20)$$

This algorithm is given as Algorithm 5.2.

Though simple in concept, the most commonly cited proofs for this technique, including [Sch95], [MN90, Chap. 2], and [HMW90], have been described with such terms as “a little painful” [Coh93, §1.5.2], and “unenlightening” [MN90, §1].

A much clearer and more succinct treatment, applying to the more general (5.2) was given by Nitaj [Nit95]. This proof, published in French, relies almost entirely on elementary properties of continued fractions. Because few, if any, English-language sources for this material are currently available,¹ this technique (and its associated proof) will be given here.

We will begin by relating solutions (x, y) of (5.2) to solutions k of the congruence (5.20).

Proposition 5.7. *Let (x, y) be a non-trivial, primitive solution to $fx^2 + gy^2 = m$. Then there exists a solution k such that $k^2 \equiv -gf^{-1} \pmod{m}$.*

Proof. Let (x, y) be solutions to (5.2) such that $|x|y > 1$ (i.e. non-trivial). Since $\gcd(y, m) = 1$, we can define

$$k \equiv xy^{-1} \pmod{m}. \quad (5.21)$$

By assumption, $\gcd(f, m) = 1$, and so if (5.2) has a non-trivial solution, then

$$k^2 \equiv -gf^{-1} \pmod{m}. \quad (5.22)$$

□

¹[Bas04] gives a similar proof, although from a lattice perspective.

Proposition 5.8. *If (x, y) and (x', y') are two nontrivial, primitive solutions to (5.2) satisfying $x \equiv ky \pmod{m}$ and $x' \equiv ky' \pmod{m}$ then $x = x'$ and $y = y'$.*

Proof. From $x \equiv ky \pmod{m}$ and $x' \equiv ky' \pmod{m}$ we obtain

$$fxx' + gyy' \equiv (fk^2 + g)yy' \equiv 0 \pmod{m}$$

and thus $fxx' + gyy' = mq_1$ where $q_1 \in \mathbb{Z}$. We also have that $xy' - yx' \equiv 0 \pmod{m}$, hence $xy' - yx' = mq_2$, $q_2 \in \mathbb{Z}$.

From $fx^2 + gy^2 = m$ and $fx'^2 + gy'^2 = m$, we obtain

$$\begin{aligned} m^2 &= (fx^2 + gy^2)(fx'^2 + gy'^2) \\ &= (fxx' + gyy')^2 + fg(xy' - yx')^2 \\ &= m^2q_1^2 + fgm^2q_2^2 \end{aligned}$$

which gives

$$q_1^2 + fgq_2^2 = 1. \tag{5.23}$$

First, suppose $f = g = 1$. If $q_1 = 0$ then $xx' + yy' = 0$. By assumption $0 < y < |x|$ and $0 < y' < |x'|$. Since $\gcd(x, y) = 1$ and $\gcd(x', y') = 1$ we have that $x = \pm y'$ and $y = \mp x'$. Thus, $0 < y < |x|$ implies $0 < |x'| < y$, a contradiction. Hence, $q_2 = 0$ in (5.23), so $\frac{x}{y} = \frac{x'}{y'}$ and finally $x = x'$, $y = y'$ as desired.

If $fg > 1$, then (5.23) implies that $q_2 = 0$, and as before, $x = x'$, $y = y'$. \square

Thus, every solution k of (5.20) gives rise to *at most* one primitive solution, (x, y) of (5.2). We call such an (x, y) the solution *corresponding to* k . We will now show

how to obtain such a corresponding solution.

Proposition 5.9. *Suppose (x, y) is a primitive solution to (5.2) corresponding to k . Then there is a convergent A_i/B_i in the continued fraction expansion of k/m such that $x = kB_i - mA_i$ and $y = B_i$.*

Proof. From (5.21), $x \equiv ky \pmod{m}$, so there must exist some $w \in \mathbb{Z}$ such that $x = ky - mw$. By assumption, $f, g \in \mathbb{N}$, so it is clear that $x^2 + y^2 \leq fx^2 + gy^2$. Furthermore, since $(x - y)^2 > 0$ and $|x|y > 1$, we have that

$$0 < 2|x|y < x^2 + y^2 \leq fx^2 + gy^2 = m,$$

and thus

$$\left| \frac{w}{y} - \frac{k}{m} \right| = \frac{|x|}{my} < \frac{1}{2y^2}.$$

By Theorem 5.5, w/y must be a convergent in the simple continued fraction expansion of k/m . Thus, $\frac{w}{y} = \frac{A_i}{B_i}$ for some i , where A_i, B_i are defined as per (5.6). If $\gcd(w, y) = 1$, then since $y > 0$ we must have $w = A_i, y = B_i$. But this is clear, as if $\gcd(w, y) = d$, then $d \mid x$, a contradiction to the primitivity of (x, y) . Thus

$$(x, y) = (kB_i - mA_i, B_i)$$

is a solution to (5.2). □

Recall from Section 5.1 how to find the simple continued fraction expansion of

k/m by using the Euclidean algorithm; *i.e.*

$$R_{-2} = k, R_{-1} = m \quad (5.24)$$

$$R_{i-2} = q_i R_{i-1} + R_i, \quad q_i = \lfloor R_{i-2}/R_{i-1} \rfloor \quad (5.25)$$

where R_i is the remainder on division of R_{i-2} by R_{i-1} .

Recall also that by the properties of simple continued fractions (5.9), we know

$$kB_i - mA_i = (-1)^i R_i.$$

Thus, if (5.2) has a solution (x, y) with $\gcd(x, y) = 1$ and $|x|y > 1$, then $|x| = R_i$ and $y = B_i$ for some i and k satisfying (5.20) and R_i, B_i defined as above.

Note that if k is a solution to (5.20), then $(m - k)$ is also a solution.

Lemma 5.10. *Choose $0 < k < m/2$ and let A_i/B_i and A'_j/B'_j be the convergents of the simple continued fraction expansions of k/m and $(m - k)/m$ respectively. Then for all $i \geq 1$,*

$$A'_i = B_{i-1} - A_{i-1}, \text{ and}$$

$$B'_i = B_{i-1}.$$

Proof. We will proceed by induction. Recall that the convergents of a continued fraction are given by the recurrence equations (5.6), hence if $k/m = [q_0; q_1, q_2, \dots, q_s]$,

then since $k < m/2$, $q_0 = 0$, the first two convergents of k/m are:

$$\begin{aligned} A_0 &= q_0 A_{-1} + A_{-2} = 0 & B_0 &= q_0 B_{-1} + B_{-2} = 1 \\ A_1 &= q_1 A_0 + A_{-1} = 1 & B_1 &= q_1 B_0 + B_{-1} = q_1. \end{aligned}$$

From (5.14), $\frac{m-k}{m} = [0; 1, (q_1 - 1), q_2, \dots, q_s]$ and so its first two convergents are:

$$\begin{aligned} A'_0 &= 0A'_{-1} + A'_{-2} = 0 & B'_0 &= 0B'_{-1} + B'_{-2} = 1 \\ A'_1 &= 1A'_0 + A'_{-1} = 1 & B'_1 &= 1B'_0 + B'_{-1} = 1. \end{aligned}$$

Observe $A'_1 = B_0 - A_0 = 1 - 0 = 1$ and $B'_1 = B_0 = 1$ as expected.

Now, assume $A'_i = B_{i-1} - A_{i-1}$ and $B'_i = B_{i-1}$ for $i \leq n$. Then

$$\begin{aligned} A'_{n+1} &= q'_{n+1} A'_n + A'_{n-1} \\ &= q_n (B_{n-1} - A_{n-1}) + B_{n-2} - A_{n-2} \\ &= (q_n B_{n-1} + B_{n-2}) - (q_n A_{n-1} + A_{n-2}) \\ &= B_n - A_n \end{aligned}$$

and

$$\begin{aligned}
B'_{n+1} &= q'_{n+1}B'_n + B'_{n-1} \\
&= q_n B_{n-1} + B_{n-2} \\
&= B_n
\end{aligned}$$

as desired. □

Note that if (x, y) is a solution to (5.2), then $(-x, y)$ is also a solution. This leads to the following proposition.

Proposition 5.11. *Let k be a solution to the congruence (5.20) with $0 < k < m/2$. If (x, y) is a solution to (5.2) corresponding to k then $(-x, y)$ is a solution to (5.2) corresponding to $(m - k)$.*

Proof. Let (x, y) be the solution corresponding to k . Then by Proposition 5.9 we can write $x = kB_i - mA_i$ and $y = B_i$ for some convergent A_i/B_i in the continued fraction expansion of k/m .

Let (x', y') be the solution corresponding to $(m - k)$. By Proposition 5.9 there is a convergent A'_j/B'_j in the continued fraction expansion of $(m - k)/m$ such that $x' = (m - k)B'_j - mA'_i$ and $y' = B'_j$. By Lemma 5.10,

$$\begin{aligned}
x' &= (m - k)B'_j - mA'_j \\
&= (m - k)B_{j-1} - m(B_{j-1} - A_{j-1}) \\
&= -(kB_{j-1} - mA_{j-1}),
\end{aligned}$$

and

$$y' = B'_j = B_{j-1}.$$

If we set $j = i + 1$, $(x', y') = (-x, y)$ as desired. \square

Thus, when iterating through the roots of (5.20) in Algorithm 5.2, we need only consider the solutions k in half the interval; *i.e.* $m/2 < k < m$.

5.3.1 Terminating the Algorithm

Though we have established that a solution (x, y) to (5.2) may be obtained from the convergents of k/m ; *i.e.* $(|x|, y) = (R_i, B_i)$ for some $i \geq 0$, it remains to identify the stopping condition that will determine i .

Lemma 5.12. *Let $(x_i, y_i) = (kB_i - mA_i, B_i)$ be defined from the i^{th} convergent in the continued fraction expansion of k/m as usual. Then $fR_i^2 + gB_i^2 \equiv 0 \pmod{m}$ for all $i \geq 0$.*

Proof.

$$\begin{aligned} fx_i^2 + gy_i^2 &= f(kB_i - mA_i)^2 + gB_i^2 \\ &\equiv fk^2B_i^2 + gB_i^2 \\ &\equiv B_i^2(fk^2 + g) \\ &\equiv 0 \pmod{m} \end{aligned}$$

as $k^2 \equiv -gf^{-1} \pmod{m}$ by definition. \square

Proposition 5.13. *Let k be a solution to (5.20) with $m/2 < k < m$. Evaluate the*

convergents $A_i/B_i, i \geq 0$, to k/m until $i = n$, where n is chosen such that $R_n^2 < m/f \leq R_{n-1}^2$, then $(x_n, y_n) = (R_n, B_n)$ is the unique solution to (5.2) corresponding to k if and only if $B_n^2 < m/g$.

Proof. Let (x_s, y_s) be the unique solution to (5.2) corresponding to k . Since $fx_s^2 + gy_s^2 = m$, it is clear that $|x_s| < \sqrt{m/f}$ and $y_s < \sqrt{m/g}$.

Recall that the sequence of remainders R_i obtained in the continued fraction expansion of k/m is a strictly decreasing sequence, and consider $|x_n| = R_n, y_n = B_n$. From its definition, n is the *least* non-negative integer for which

$$fR_i^2 < m. \quad (5.26)$$

Since we also know that $fR_s^2 < m$, we must have $s \geq n$. If we can show that $s = n$, we are done.

Suppose $s > n$. From Proposition 5.1 we know that B_i is a strictly increasing sequence, so $B_n < B_s < \sqrt{m/g}$. It follows that

$$0 < fR_n^2 + gB_n^2 < 2m.$$

By Lemma 5.12, $fR_i^2 + gB_i^2 \equiv 0 \pmod{m}$ for all i , hence $fR_n^2 + gB_n^2 = m$ and (x_n, y_n) is also a primitive solution to (5.2) corresponding to k . By Proposition 5.8, this is impossible; hence $s = n$ and the proposition is proved. \square

5.4 Analyzing and Optimizing Cornacchia's Algorithm

Let k be a solution to (5.20) with $m/2 < k < m$, and define $k' = (m - k)$. From (5.11) – (5.14), we can see that:

$$\begin{aligned}\frac{k}{m} &= [0; 1, (q_1 - 1), q_2, \dots, q_s] \\ \frac{k'}{m} &= [0; q_1, q_2, \dots, q_s] \\ \frac{m}{k} &= [1; (q_1 - 1), q_2, \dots, q_s] \\ \frac{m}{k'} &= [q_1; q_2, \dots, q_s].\end{aligned}$$

Notice that in every case, the continued fraction expansion is uniquely given by the quotients $\{q_1, \dots, q_s\}$. Unless a solution is given by $(|x|, y) = (k, 1)$, we can save two iterations from the Euclidean algorithm by evaluating m/k' instead of k/m .

We will now incorporate this fact into a modified version of Cornacchia's algorithm. Also, we observe that we can evaluate B_i as we proceed through the algorithm using its recurrence relation given in (5.6); *i.e.* $B_{-2} = 1, B_{-1} = 0, B_k = q_k B_{k-1} + B_{k-2}$.

The new algorithm is given as Algorithm 5.3.

Analyzing this algorithm is straightforward. So long as a factorization of m is known,² solving the square root in Step 1 requires at most $O((\log m)^3)$ operations

²In fact, in Algorithm 5.4 we explicitly do *not* know the factorization of m —we simply assume it is prime. This does not pose a problem, however, as should m be composite, we will obtain a nonsensical solution k . This condition is easy to test for and since the purpose of Algorithm 5.4 is to reveal the prime or composite nature of m , this assumption suffices.

using, for example, Cipolla's method [BS96, §7.2].³ Since all inputs are bounded by m , the Euclidean algorithm requires at most $O((\log m)^2)$ operations [Sor94]. The additional multiplication and addition in Step 7 do not change the analysis, and hence, Cornacchia's algorithm requires $O((\log m)^3)$ operations to solve our Diophantine equation.

Algorithm 5.3: Modified Cornacchia

Input: f, g, m pairwise relatively prime.

Output: List of Primitive solutions (x, y) for which $fx^2 + gy^2 = m$

- 1: Solve $k^2 \equiv -gf^{-1} \pmod{m}$. If this step fails, return \emptyset
 - 2: **for** solutions k with $0 < k < m/2$ **do**
 - 3: $M \leftarrow m, x \leftarrow k, y \leftarrow 1, B \leftarrow 0$
 - 4: **while** $x > \sqrt{m/f}$ **do**
 - 5: Compute q, r such that $M = qx + r$.
 - 6: $M \leftarrow x, x \leftarrow r, t \leftarrow y$
 - 7: $y \leftarrow qy + B, B \leftarrow t$
 - 8: **end while**
 - 9: **if** $y < \sqrt{m/g}$ **then**
 - 10: Append (x, y) to solution set.
 - 11: **end if**
 - 12: **end for**
-

Example 5.14. Consider $f = 1, g = 3, m = 141592653589$. We wish to find $x, y \in \mathbb{Z}$ such that $x^2 + 3y^2 = 141592653589$ using Algorithm 5.3.

First, we require a k such that $k^2 \equiv -3 \pmod{141592653589}$. We may obtain such a solution quickly via the Tonelli-Shanks method (Algorithm A.1). In this case, we obtain $k \equiv 47520102470 \pmod{141592653589}$. Since $47520102470^2 \equiv -3 \pmod{141592653589}$, as expected, we may proceed with the rest of the algorithm.

We run the main loop until $x < \sqrt{141592653589} \approx 376288$:

1. $M \leftarrow 141592653589, x \leftarrow 47520102470, y \leftarrow 1, B \leftarrow 0$

³In practice, we will almost always use the Tonelli-Shanks algorithm instead, which offers better average-case performance at the cost of worsening the worst-case bound to $O((\log m)^4)$. This algorithm is described and analyzed in Appendix A.

$$M = 2x + 46552448649$$

$$2. \quad M \leftarrow 47520102470, x \leftarrow 46552448649, y \leftarrow 2, B \leftarrow 1$$

$$M = 1x + 967653821$$

$$3. \quad M \leftarrow 46552448649, x \leftarrow 967653821, y \leftarrow 3, B \leftarrow 2$$

$$M = 48x + 105065241$$

$$4. \quad M \leftarrow 967653821, x \leftarrow 105065241, y \leftarrow 146, B \leftarrow 3$$

$$M = 9x + 22066652$$

$$5. \quad M \leftarrow 105065241, x \leftarrow 22066652, y \leftarrow 1317, B \leftarrow 146$$

$$M = 4x + 16798633$$

$$6. \quad M \leftarrow 22066652, x \leftarrow 16798633, y \leftarrow 5414, B \leftarrow 1317$$

$$M = 1x + 5268019$$

$$7. \quad M \leftarrow 16798633, x \leftarrow 5268019, y \leftarrow 6731, B \leftarrow 5414$$

$$M = 3x + 994576$$

$$8. \quad M \leftarrow 5268019, x \leftarrow 994576, y \leftarrow 25607, B \leftarrow 6731$$

$$M = 5x + 295139$$

$$9. \quad M \leftarrow 994576, x \leftarrow 295139, y \leftarrow 134766, B \leftarrow 25607$$

Now, since $y < \sqrt{141592653589/3} \approx 217250$, we have a candidate solution:

$$(x, y) = (295139, 134766).$$

Furthermore, we can verify that

$$295139^2 + 3 \cdot 134766^2 = 141592653589,$$

as desired.

5.4.1 Further Improvements

Note that, as with Brillhart's improvement to Hermite's algorithm, complete solutions to (5.2) can be derived entirely from the remainders produced from the Euclidean algorithm. We will now examine this notion in more detail.

Consider the simple continued fraction expansion of k/m . From Lemma 5.12, we know that

$$fR_j^2 + gB_j^2 \equiv 0 \pmod{m}.$$

Write $fR_j^2 + gB_j^2 = a_j m$ for some $a_j \in \mathbb{Z}$. Similarly

$$(-1)^{j+2}R_{j+1} \equiv -kB_{j+1} \pmod{m}$$

and so from (5.22)

$$fR_j R_{j+1} - gB_j B_{j+1} \equiv 0 \pmod{m}.$$

Write $fR_j R_{j+1} - gB_j B_{j+1} = b_j m$ for some $b_j \in \mathbb{Z}$.

From the properties of continued fractions (5.8), we know that $m = B_j R_{j-1} + B_{j-1} R_j$, ($j = -1, 0, \dots$), hence

$$\begin{aligned} fR_{j+1}m &= fR_{j+1}(B_{j+1}R_j + B_j R_{j+1}) \\ &= fR_{j+1}R_j B_{j+1} + fB_j R_{j+1}^2 \\ &= fR_{j+1}R_j B_{j+1} + B_j(a_{j+1}m - gB_{j+1}^2) \\ &= B_{j+1}(fR_{j+1}R_j - gB_{j+1}B_j) + B_j a_{j+1}m. \end{aligned}$$

Thus, $fR_{j+1} = b_j B_{j+1} + a_{j+1} B_j$. If (5.20) has a solution for k then $a_{n+1} = 1$, and

thus

$$fR_{n+1} = b_n B_{n+1} + B_n. \quad (5.27)$$

Since

$$fR_{n+1}^2 + gB_{n+1}^2 = m = B_{n+1}R_n + B_nR_{n+1},$$

then from (5.27) we obtain $fR_{n+1}^2 + gB_{n+1}^2 = B_{n+1}R_n + R_{n+1}(fR_{n+1} - b_n B_{n+1})$, hence $gB_{n+1}^2 = B_{n+1}R_n - b_n R_{n+1}B_{n+1}$, and thus

$$gB_{n+1} = R_n - b_n R_{n+1}. \quad (5.28)$$

Rewrite (5.28) by introducing a variable $t_n \in \mathbb{Z}_{\geq 0}$:

$$R_n = (b_n + t_n)R_{n+1} + (gB_{n+1} - t_n R_{n+1}).$$

If we select t_n such that

$$0 \leq gB_{n+1} - t_n R_{n+1} < R_{n+1},$$

then $gB_{n+1} - t_n R_{n+1}$ is the remainder on dividing R_n by R_{n+1} ; *i.e.*

$$R_{n+2} = gB_{n+1} - t_n R_{n+1} \quad (5.29)$$

$$q_{n+2} = b_n + t_n.$$

By (5.27) we have

$$fR_{n+1} > b_n B_{n+1}.$$

Combining this fact with (5.29), we obtain

$$t_n \leq \frac{gB_{n+1}}{R_{n+1}} < \frac{gf}{b_n} \leq g \quad (b_n \geq f).$$

Furthermore, from (5.29),

$$t_n \equiv -R_{n+1}^{-1}R_{n+2} \pmod{g}, \quad (5.30)$$

so if $b_n \geq f$, we know that t_n is the least non-negative solution of (5.30), and

$$B_{n+1} = \frac{R_{n+2} + t_n R_{n+1}}{g}. \quad (5.31)$$

If $b_n < f$ then, returning to (5.28),

$$b_n \equiv R_{n+1}^{-1}R_n \pmod{g}. \quad (5.32)$$

Since $f < g$, b_n must be the least positive solution of (5.32). Thus

$$B_{n+1} = \frac{R_n - b_n R_{n+1}}{g}. \quad (5.33)$$

We may conclude, that in all cases, the solution (x, y) to (5.2) depends only on the remainders produced when applying the Euclidean algorithm to k and m , where k is a solution to the congruence (5.20). To demonstrate this improvement, we will revisit the problem solved in Example 5.14.

Example 5.15. As before, we wish to find $x, y \in \mathbb{Z}$ such that

$$x^2 + 3y^2 = 141592653589,$$

thus, $f = 1, g = 3$, and $m = 141592653589$. Using the Tonelli-Shanks method (Algorithm A.1), we obtain solutions $k \equiv \pm 47520102470 \pmod{141592653589}$ to $k^2 \equiv -3 \pmod{141592653589}$.

Evaluate the Euclidean algorithm on k and m , obtaining a sequence of remainders R_i , proceeding one iteration past the point where $R_n < \sqrt{m/f} < R_{n+1}$; i.e. $n = 7$ and $i = 0, \dots, 9$. This sequence of quotients and remainders is given below:

i	R_{i-2}	q_i	R_{i-1}	R_i
0	47520102470	0	141592653589	47520102470
1	141592653589	2	47520102470	46552448649
2	47520102470	1	46552448649	967653821
3	46552448649	48	967653821	105065241
4	967653821	9	105065241	22066652
5	105065241	4	22066652	16798633
6	22066652	1	16798633	5268019
7	16798633	3	5268019	994576
8	5268019	5	994576	295139
9	994576	3	295139	109159

From (5.30), we obtain that:

$$t_n \equiv -R_8^{-1}R_9 \equiv -2 \cdot 1 \equiv 1 \pmod{3}.$$

Algorithm 5.4: Eisenstein Pseudocube Primality Proving (EPPP)

Input: $m \equiv 1 \pmod{3}$, table of Eisenstein pseudocubes, \mathcal{T}

Output: TRUE or FALSE, indicating whether m is prime in \mathbb{Z} .

- 1: Test that m is not a perfect power.
 - 2: $(s, t) \leftarrow \text{CORNACCHIA}(1, 3, m)$. If this step fails, return FALSE.
 - 3: Choose a, b from $\pm\{2t, (s+t)\}$ such that $a + b\omega$ is primary. Set $\nu \leftarrow a + b\omega$.
 - 4: Choose the smallest $\mu_p \in \mathcal{T}$ such that $N(\nu) < \mu_p$.
 - 5: **for** each prime $q \leq p$ **do**
 - 6: **if** $\left(\frac{q}{\nu}\right)_3 \not\equiv q^{\frac{m-1}{3}} \pmod{\nu}$ **then**
 - 7: **return** FALSE
 - 8: **end if**
 - 9: **end for**
 - 10: **return** TRUE
-

Hence, from (5.31),

$$B_8 = \frac{R_{n+2} + t_n R_{n+1}}{g} = \frac{109159 + 295139}{3} = 134766.$$

And thus, $(x, y) = (295139, 134766)$ is a solution to our Diophantine equation; i.e.

$$295139^2 + 3 \cdot 134766^2 = 141592653589.$$

5.5 Primality Proving using Eisenstein Pseudocubes

At this point, we have an efficient algorithm that, given $m \equiv 1 \pmod{3}$, allows us to find ν such that $N(\nu) = m$. We are now in a position to implement the algorithm described in 4.6.

Note that the evaluation of the CORNACCHIA algorithm in Step 2 requires a factor-

ization of m in order to compute modular square roots using either of the algorithms described in Section A.2. This does not pose a problem as, since we are attempting to prove primality, we can simply assume m is prime, and should the algorithm fail, we conclude that our assumption was in error.

Example 5.16. *Consider $m = 141592653589$, as before. We can establish relatively quickly that this is not a perfect power. Evaluating Cornacchia’s algorithm, we obtain $s = 295139$, $t = 134766$. Note that $\nu = 295139 + 134766\omega$ is primary, and $N(\nu) = 141592653589$, as desired. From Table 10.3, we determine that $141592653589 < N(\mu_{73}) = 824621013649$. Thus, we must evaluate whether $\left(\frac{q}{\nu}\right)_3 \equiv q^{\frac{m-1}{3}} \pmod{\nu}$ is true for the 21 primes $q \in \{2, 3, \dots, 73\}$. If all these tests succeed, m is prime.*

5.5.1 Computational Complexity of the EPPP Algorithm

Perfect power detection (Step 1) can be done in $(\log m)^{1+o(1)}$ operations [Ber98]. Cornacchia’s algorithm—Step 2, analyzed in Section 5.4—requires $O((\log m)^3)$ operations. Evaluating a cubic Jacobi symbol and performing a single modular exponentiation (Step 6) both require $O((\log m)^2)$ operations. Thus, if it can be shown that the loop in Step 5 requires at most n iterations, where $n \sim \log m$, then primality proving via the EPPP method requires at most $(\log m)^{3+o(1)}$ operations—a complexity equivalent to that of the pseudosquare primality proving method.

The growth rate of Eisenstein pseudocubes, and hence, the complexity of the EPPP algorithm will be revisited in Section 10.6. To give us some insight into this growth rate, we will first develop a table of Eisenstein pseudocubes. We will now turn our attention to this problem.

Chapter 6

Computing Eisenstein Pseudocubes

The purpose of computing is insight, not numbers.

—Richard Hamming

In order to make use of the Eisenstein pseudocube primality test, we require a table of Eisenstein pseudocubes. We will now turn our attention to the problem of generating such a table. Recall the definition of an Eisenstein pseudocube:

Let p be a fixed rational prime. Define $\mu_p = a + b\omega \in \mathbb{Z}[\omega]$ $a, b \in \mathbb{Z}$ to be an element of $\mathbb{Z}[\omega]$ of *minimal norm* such that:

1. μ_p is primary,
2. $\gcd(a, b) = 1$,
3. $\left(\frac{q}{\mu_p}\right)_3 = 1$ for all rational primes $q \leq p$,
4. μ_p not a cube in $\mathbb{Z}[\omega]$.

We will now develop a method to generate these pseudocubes.

6.1 Rejecting Perfect Powers in $\mathbb{Z}[\omega]$

One important property of an Eisenstein pseudocube is that it is not actually a perfect cube; thus it is necessary to develop a test for this property. To simplify this discussion, we will consider only primary elements of $\mathbb{Z}[\omega]$.

First, observe that if α is a perfect k^{th} power in $\mathbb{Z}[\omega]$; *i.e.* $\alpha = \sigma^k$ where $\sigma \in \mathbb{Z}[\omega]$,

then $N(\alpha)$ is a perfect k^{th} power in \mathbb{Z} .

It is possible, however, for $N(\alpha)$ to be a perfect power, even though α is not. To see this, consider $\alpha \in \mathbb{Z}[\omega]$ as a decomposition of rational and Eisenstein primes; *i.e.* $\alpha = P\Gamma$ where

$$P = \prod_{i=1}^n p_i^{a_i}, \quad \Gamma = \prod_{j=1}^m \pi_j^{\lambda_j}$$

such that p_i and π_j are distinct primes $\in \mathbb{Z}[\omega]$, $p_i \equiv 2 \pmod{3}$ and $N(\pi_j) = \pi_j \overline{\pi_j} \equiv 1 \pmod{3}$ for all i, j .

Clearly, $N(P)$ is a perfect k^{th} power if and only if P is. However, this is not necessarily the case for Γ , as $N(\pi_j) = N(\overline{\pi_j})$.¹

Observe, however, that if the prime power decomposition of Γ contains both π_j and $\overline{\pi_j}$, then $\pi_j \overline{\pi_j} \mid \alpha$ if and only if $N(\pi_j) \mid (a + b\omega)$, hence $N(\pi_j) \mid \gcd(a, b)$. For Eisenstein pseudocubes, we insist that $\gcd(a, b) = 1$, and therefore $\alpha = a + b\omega$ is a perfect k^{th} power if and only if $N(\alpha)$ is a perfect k^{th} power. Perfect power testing may therefore be accomplished via traditional integer methods; *e.g.* [Ber98] or—in the specific case of cubes—[Luk95, Appendix B].

6.2 Sieve Criteria for Eisenstein Pseudocubes

We will now establish that finding Eisenstein pseudocubes can be considered an instance of a two-dimensional sieve problem; *i.e.* we will develop congruence criteria on x_p and y_p such that $\mu_p = x_p + y_p\omega$ satisfies:

- $\mu_p = x_p + y_p\omega$ is primary with $\gcd(x_p, y_p) = 1$,

¹Thus we may obtain counterexamples such as $N(\pi^2\overline{\pi}) = N(\pi)^3$.

- $\left(\frac{q}{\mu_p}\right)_3 = 1$ for all rational primes $q \leq p$,
- μ_p is not a cube in $\mathbb{Z}[\omega]$.

To develop these sieve criteria, there are 3 cases to consider.

Case 1: $q \equiv -1 \pmod{3}$

In this case, q is primary, μ_p is (by definition) primary, and hence $1 = \left(\frac{q}{\mu_p}\right)_3 = \left(\frac{\mu_p}{q}\right)_3$.

We can therefore obtain all possible residue classes by computing $\mu_p \equiv (m + n\omega)^3 \pmod{q}$ for all choices of m and n ; *i.e.* compute the residue classes given by

$$\begin{aligned} x_p &\equiv m^3 - 3mn^2 + n^3 \pmod{q} \\ y_p &\equiv 3mn(m - n) \pmod{q} \end{aligned}$$

for $0 \leq m, n < q$. There are

$$\frac{q^2 - 1}{3} \tag{6.1}$$

such solutions.

Example 6.1. $\mathcal{S}_2 = (1, 0)$, as $1^3 = \omega^3 = (1 + \omega)^3 = 1$.

Example 6.2. *By cubing the 24 candidate values $\{a + b\omega \mid 0 \leq a, b < 5, a, b \in \mathbb{Z}, \text{ not both zero}\}$ we obtain 8 acceptable residues:*

$$\begin{aligned} \mathcal{S}_5 &= \{(1, 0), (2, 0), (3, 0), (4, 0), \\ &\quad (3, 1), (1, 2), (4, 3), (2, 4)\}. \end{aligned}$$

Case 2: $q = 3$

Recall that $-3\omega = (1 - \omega)^2$. This gives

$$\left(\frac{3}{\mu_p}\right)_3 = \left(\frac{\omega(1-\omega)}{\mu_p}\right)_3^2.$$

Write $\mu_p = x_p + y_p\omega = (-1)^{k-1} \prod_{i=1}^k \alpha_i$ where $\alpha_i = r_i + s_i\omega$ are primary primes; *i.e.* $3 \mid s_i$ and $r_i \equiv -1 \pmod{3}$. From Theorem 4.31, we know that $\left(\frac{1-\omega}{\alpha_i}\right)_3 = \omega^{\frac{2(r_i+1)}{3}}$, and $\left(\frac{\omega}{\alpha_i}\right)_3 = \omega^{\frac{r_i+1+s_i}{3}}$ giving

$$\begin{aligned} \left(\frac{1-\omega}{\mu_p}\right)_3 &= \prod_{i=1}^k \omega^{\frac{2(r_i+1)}{3}} = \omega^{2\sum_{i=1}^k (r_i+1)/3} \\ \left(\frac{\omega}{\mu_p}\right)_3 &= \prod_{i=1}^k \omega^{\frac{r_i+1+s_i}{3}} = \omega^{\sum_{i=1}^k (r_i+1)/3 + \sum_{i=1}^k s_i/3}, \end{aligned}$$

and hence $\left(\frac{\omega(1-\omega)}{\mu_p}\right)_3 = \omega^{\sum_{i=1}^k s_i/3}$. Thus

$$\left(\frac{3}{\mu_p}\right)_3 = \omega^{\frac{2}{3}\sum_{i=1}^k s_i}. \quad (6.2)$$

Lemma 6.3. *Let $\mu_p = x_p + y_p\omega = (-1)^{n-1} \prod_{i=1}^n \alpha_i$ where $\alpha_i = r_i + s_i\omega$ are primary primes. Then $x_p \equiv (-1)^{n-1} \prod_{i=1}^n r_i \pmod{9}$ and $y_p \equiv \sum_{i=1}^n s_i \pmod{9}$.*

Proof. If $n = 1$, the statement is trivially true.

Let $\alpha_j = r_j + s_j\omega, \alpha_k = r_k + s_k\omega$ be primary; *i.e.* $r_j \equiv r_k \equiv -1 \pmod{3}$ and $s_j \equiv s_k \equiv 0 \pmod{3}$. Writing $s_i = 3S_i, r_i = -1 + 3R_i$ for some $S_i, R_i \in \mathbb{Z}$, observe

that

$$\begin{aligned}
-(r_k + s_k\omega)(r_j + s_j\omega) &= -(r_k + 3S_k\omega)(r_j + 3S_j\omega) \\
&\equiv -r_k r_j - 3(S_j r_k + S_k r_j)\omega \\
&\equiv -r_k r_j - 3(-S_k + 3R_j S_k - S_j + 3R_k S_j)\omega \\
&\equiv -r_k r_j + (s_k + s_j)\omega \pmod{9}
\end{aligned}$$

which is again primary. Thus, by induction,

$$(-1)^{n-1} \prod_{i=1}^n (r_i + s_i\omega) \equiv (-1)^{n-1} \prod_{i=1}^n r_i + \sum_{i=1}^n s_i\omega \pmod{9},$$

so writing $\mu_p = x_p + y_p\omega = (-1)^{n-1} \prod_{i=1}^n \alpha_i$ where $\alpha_i = r_i + s_i\omega$ are primary primes

$$\begin{aligned}
x_p &\equiv (-1)^{n-1} \prod_{i=1}^n r_i \pmod{9}, \\
y_p &\equiv \sum_{i=1}^n s_i \pmod{9}
\end{aligned}$$

as desired. □

From Lemma 6.3, $y_p \equiv \sum_{i=1}^k s_i \pmod{9}$, so $y_p/3 \equiv \sum_{i=1}^k s_i/3 \pmod{3}$. Combining these facts with Equation 6.2, we obtain $\left(\frac{3}{\mu_p}\right)_3 = \omega^{\frac{2}{3} \sum_{i=1}^k s_i} = \omega^{2y_p/3}$. Clearly, $\left(\frac{3}{\mu_p}\right)_3 = 1$ if and only if $3 \mid \frac{y_p}{3}$, giving the requisite congruence conditions:

$$\left(\frac{3}{\mu_p}\right)_3 = 1 \iff y_p \equiv 0 \pmod{9}, \text{ and } x_p \equiv -1 \pmod{3}.$$

Example 6.4.

$$\mathcal{S}_9 = \{(2, 0), (5, 0), (8, 0)\}.$$

From the preceding case, however, we know that there is only one acceptable residue modulo 2: $\mathcal{S}_2 = (1, 0)$. It is often convenient to combine these facts; i.e.

$$\mathcal{S}_{18} = \{(5, 0), (11, 0), (17, 0)\}.$$

Case 3: $q \equiv 1 \pmod{3}$

We can write $q = \pi_q \overline{\pi_q}$ where $\pi_q = a + b\omega$ and π_q is primary. Of course, $\overline{\pi_q}$ is also primary.

Lemma 6.5. *Let q be a rational prime, $\left(\frac{q}{\mu_p}\right)_3 = 1$ and $q = \pi_q \overline{\pi_q}$ with $\pi_q \in \mathbb{Z}[\omega]$ prime and primary. Then,*

$$\left(\frac{q}{\mu_p}\right)_3 = 1 \iff \left(\frac{\mu_p}{\pi_q}\right)_3 = \left(\frac{\overline{\mu_p}}{\pi_q}\right)_3.$$

Proof. Recall $q = \pi_q \overline{\pi_q}$, and that $\pi_q, \overline{\pi_q}$, and μ_p are all primary. From cubic reciprocity and the properties of the cubic Jacobi symbol [IR90, §9.3] we have that

$$\left(\frac{q}{\mu_p}\right)_3 = \left(\frac{\pi_q}{\mu_p}\right)_3 \left(\frac{\overline{\pi_q}}{\mu_p}\right)_3 = \left(\frac{\mu_p}{\pi_q}\right)_3 \left(\frac{\mu_p}{\overline{\pi_q}}\right)_3 = \left(\frac{\mu_p}{\pi_q}\right)_3 \overline{\left(\frac{\mu_p}{\pi_q}\right)_3} = \left(\frac{\mu_p}{\pi_q}\right)_3 \left(\frac{\overline{\mu_p}}{\pi_q}\right)_3^{-1}$$

And thus it is clear that $\left(\frac{q}{\mu_p}\right)_3 = 1$ if and only if $\left(\frac{\mu_p}{\pi_q}\right)_3 = \left(\frac{\overline{\mu_p}}{\pi_q}\right)_3$. □

If $\left(\frac{q}{\mu_p}\right)_3 = 1$, then from Lemma 6.5 and the properties of the cubic reciprocity symbol, $\mu_p^{\frac{q-1}{3}} \equiv \overline{\mu_p}^{\frac{q-1}{3}} \pmod{\pi_q}$. By complex conjugation, we have also that $\left(\frac{\overline{\mu_p}}{\pi_q}\right)_3 =$

$\left(\frac{\mu_p}{\overline{\pi}_q}\right)_3$, and hence $\mu_p^{\frac{q-1}{3}} \equiv \overline{\mu_p}^{\frac{q-1}{3}} \pmod{\overline{\pi}_q}$. Combining these facts, we obtain

$$\left(\frac{q}{\mu_p}\right)_3 = 1 \iff \mu_p^{\frac{q-1}{3}} \equiv \overline{\mu_p}^{\frac{q-1}{3}} \pmod{q}. \quad (6.3)$$

Writing $\mu_p = x_p + y_p\omega$, we will now endeavour to reduce (6.3) to a set of congruence conditions on x_p and y_p . Note that when q is small, these congruence conditions can be computed by exhaustion. A more elegant algorithm, however, can be obtained from the theory of Lucas sequences.

First, observe that if $q \mid y_p$ then (6.3) reduces to the trivial $x_p \equiv x_p \pmod{q}$; *i.e.* $x + 0\omega \in \mathcal{S}_q$ for $x = 1, \dots, q-1$. For the remaining case, consider the recurring sequences $S_n(x, y), T_n(x, y) \in \mathbb{Z}[x, y]$ given by:²

$$\begin{aligned} S_1(x, y) &= x \\ T_1(x, y) &= y \\ S_n + T_n\omega &= (S_1 + T_1\omega)^n \end{aligned}$$

with $S_n, T_n \in \mathbb{Z}$. Clearly, we have also that $S_n + T_n\omega^2 = (S_1 + T_1\omega^2)^n$. By subtraction, $(\omega - \omega^2)T_n = (S_1 + T_1\omega)^n - (S_1 + T_1\omega^2)^n$, and thus writing $\alpha = \mu_p = x_p + y_p\omega$, $\beta = \overline{\mu_p} = x_p + y_p\omega^2$, we have

$$T_n = \frac{\alpha^n - \beta^n}{\omega - \omega^2}, \quad (6.4)$$

a recurrent sequence whose properties are described in [Wil78b]. We may parameterize this recurrence by writing $G = \alpha + \beta$, $H = \alpha\beta$, and observing that $T_n(G, H)$ is given by the second-order recurrence: $T_{n+2} = GT_{n+1} - HT_n$. From (6.3), $\left(\frac{q}{\mu_p}\right)_3 = 1$

²For simplicity, we will usually write S_n and T_n for $S_n(x, y)$ and $T_n(x, y)$, respectively.

if and only if $q \mid (\alpha^{\frac{q-1}{3}} - \beta^{\frac{q-1}{3}})$ and hence from (6.4),

$$\left(\frac{q}{\mu_p}\right)_3 = 1 \iff q \mid T_{\frac{q-1}{3}}(G, H). \quad (6.5)$$

Since only the case $q \nmid y_p$ remains, we can rewrite (6.5) in terms of a single variable by defining $z_p \equiv x_p y_p^{-1} \pmod{q}$. Now $(x_p + y_p \omega)^{(q-1)/3} \equiv (x_p + y_p \omega^2)^{(q-1)/3} \pmod{q}$ if and only if $(z_p + \omega)^{(q-1)/3} \equiv (z_p + \omega^2)^{(q-1)/3} \pmod{q}$. Setting $\alpha = z_p + \omega$, $\beta = z_p + \omega^2$ in (6.4), we obtain

$$\left(\frac{q}{\mu_p}\right)_3 = 1 \iff q \mid T_{\frac{q-1}{3}}(G', H') \quad (6.6)$$

where $G' = 2z_p - 1$, $H' = z_p^2 - z_p + 1$. Since this relationship involves only one variable, we are in effect considering polynomials $T_n(x)$ where

$$\begin{aligned} T_0(x) &= 0, & T_1(x) &= 1 \\ T_{n+1}(x) &= (2x - 1)T_n x - (x^2 - x + 1)T_{n-1}(x) \end{aligned}$$

for a fixed $x \in \mathbb{Z}$. By induction, we see that $T_n(x)$ is a polynomial over \mathbb{Z} with coefficients of degree $n - 1$ and leading coefficient n .

In fact, $T_n(x) = U_n(G', H')$ where U_n is the Lucas function, $U_n = \frac{\alpha^n - \beta^n}{\alpha - \beta}$, $G' = \alpha + \beta = 2x - 1$, $H' = \alpha\beta = x^2 - x + 1$, and hence $\alpha = (x + \omega)$, $\beta = (x + \omega^2)$. By drawing on the rich theory of Lucas functions, we can obtain both an efficient algorithm for computing the acceptable congruence conditions on $x_p, y_p \pmod{q}$, and the number of acceptable residues for the prime q .

To obtain the candidate solutions z_p satisfying (6.6), compute $T_{\frac{q-1}{3}}(x)$ for all $0 \leq x < q$ by the method described in [Wil98, §4.4], retaining solutions for which

$T_{\frac{q-1}{3}}(x) \equiv 0 \pmod{q}$. Each z_p obtained in this fashion can then be used to produce $(q-1)$ acceptable values of μ_p by evaluating $x_p = 1, 2, \dots, q-1$ and computing the corresponding $y_p = x_p z_p \pmod{q}$ —a procedure illustrated in Example 6.6.

To obtain a count of these solutions, observe that in (6.6), we can write $\Delta = (\alpha - \beta)^2 = (2x - 1)^2 - 4(x^2 - x + 1) = -3$. If q is a prime $\equiv 1 \pmod{3}$ then $\epsilon = \left(\frac{\Delta}{q}\right) = 1$. Thus if $x \in \mathbb{Z}$ and $q \nmid x^2 - x + 1$ then $q \mid T_{q-\epsilon}(x)$ [Wil98, (4.3.3)]. It follows that the polynomial $T_{q-1}(x)$ of degree $q-2$ has precisely $q-2$ distinct zeros modulo q . Now $T_{\frac{q-1}{3}}(x) \in \mathbb{Z}[x]$, and so it divides $T_{q-1}(x)$ as, from the theory of Lucas functions [Wil98, (4.2.45)], we have

$$T_{3n}(2x-1, x^2-x+1) = 3T_n((x^2-x+1)^n - T_n^2).$$

It follows that $T_{\frac{q-1}{3}}(x)$ has exactly $\frac{q-1}{3} - 1$ distinct zeros modulo q .

By combining the cases when $q \nmid y_p$ and $q \mid y_p$, we see that there are

$$\left(\frac{q-1}{3} - 1\right)(q-1) + (q-1) = \frac{(q-1)^2}{3} \tag{6.7}$$

acceptable residues for a prime $q \equiv 1 \pmod{3}$.

Example 6.6. Consider the case $q = 7$. We can derive the acceptable residue conditions on μ_p as follows.

If $q \mid y_p$, then $(x + 0\omega)$ is acceptable for $x = 1, \dots, (q-1)$.

If $q \nmid y_p$ then from (6.6), we have that $\left(\frac{7}{\mu_p}\right)_3 = 1 \iff 7 \mid T_{\frac{7-1}{3}}(G', H') = T_2(G', H')$. Further, $T_2(G', H') = G'T_1(G', H') - HT_0(G', H') = G' - 0 = 2z_p - 1$ and

hence,

$$\left(\frac{7}{\mu_p}\right)_3 = 1 \iff 7 \mid 2z_p - 1.$$

Thus, $z_p \equiv 4 \pmod{7}$. Since we defined $z_p = x_p y_p^{-1} \pmod{q}$, $x_p \equiv 4y_p \pmod{7}$, and we can obtain all solutions by running x_p through all nonzero residue classes (modulo 7) and computing $y_p \equiv 4^{-1}x_p \equiv 2x_p \pmod{7}$; *i.e.*

x_p	1	2	3	4	5	6
$y_p \equiv 2x_p \pmod{7}$	2	4	6	1	3	5

Combining these solutions with the trivial case ($q \mid y_p$), we obtain a complete set of solutions (modulo 7):

$$\mathcal{S}_7 = \{(1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), \\ (4, 1), (1, 2), (5, 3), (2, 4), (6, 5), (3, 6)\}.$$

6.3 Summary

In this chapter, we showed how the problem of finding Eisenstein pseudocubes can be reduced to the problem of solving a system of simultaneous congruences for a given prime; *i.e.* a sieve problem. Unlike the sieve problems described in Chapter 2, however, acceptable residues are described as pairs $(x, y) \pmod{m}$. We call a sieve problem of this form a *two-dimensional sieve problem*. Modifying existing sieve devices to solve a problem of this type poses an interesting set of challenges, and is the subject of the next chapter.

Chapter 7

Extending CASSIE to Two-Dimensions

The good news about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do.

—Ted Nelson

To assist in the hunt for Eisenstein pseudocubes, the CASSIE toolkit, originally developed to implement the one-dimensional sieve problem—see [Woo04]—was extended to incorporate two-dimensional operation. In this chapter, we will describe the main enhancements to the CASSIE framework, notably:

- two-dimensional problem representation,
- sieving in a norm-bounded region,
- extending normalization to 2 dimensions, and
- multistage filtering and output processing.

7.1 Representing a Two-Dimensional Sieve Problem

In a two-dimensional sieve problem, we are concerned with obtaining solutions $(a, b) \pmod{m_i} \in \mathcal{R}_i$ in a bounded interval.¹ In the specific case of the Eisenstein integers, $\alpha = a + b\omega \in \mathbb{Z}[\omega]$ and $N(\alpha) = a^2 + b^2 - ab \leq H$.

¹Often, this bound is stated in terms of a norm. More precisely, in our case we are interested in solutions of *minimal* norm. In order to ensure all solution candidates have been examined, however, all solutions within a norm-bounded region must be considered.

One method for solving sieve problems of this type is to iterate over one parameter (say b), and treat the two-dimensional case as a series of one-dimensional sieve problems; *i.e.* by computing all $i = 1, \dots, k$ integer solutions for b such that $N(\alpha) \leq H$ and evaluating a one-dimensional sieve problem over a for each b where a solution is possible. Since we are used to using the term of “width” when referring to components of a one-dimensional sieve problem—see *e.g.* Section 2.1—we refer to b as the *height* index.

In the one-dimensional sieve problem, a sieve ring for the modulus m was represented as an array of m bits. To extend our sieve’s capabilities to two-dimensions, an $m \times m$ array of bits is used instead, with height index b used to indicate which set of residues is currently in use. Thus, most one-dimensional sieve functions need little more than an additional parameter—the current height index—in order to operate on a two-dimensional problem.

7.2 Sieving in a Norm-Bounded region

Though our discussion to this point has been about the two-dimensional problem in general, to discuss sieving within a norm-bounded region, it is convenient to focus our attention on sieving for Eisenstein integers.

A norm-bounded Eisenstein solution interval is given by $N(\alpha) = a^2 + b^2 - ab \leq H$. By fixing b and rearranging, we obtain $a^2 - ab + (b^2 - H) \leq 0$. This is a positive-increasing parabola, and hence all solutions $a \in \mathbb{Z}$ to this inequality lie between the

zeros of this polynomial:

$$\left\lceil \frac{b - \sqrt{4H - 3b^2}}{2} \right\rceil \leq a \leq \left\lfloor \frac{b + \sqrt{4H - 3b^2}}{2} \right\rfloor. \quad (7.1)$$

We denote the lower and upper (integer) bounds a_ℓ and a_h respectively. Furthermore, it is clear that for integer solutions to exist, we must have $4H \geq 3b^2$, and hence,²

$$\left\lceil -\sqrt{\frac{4}{3}H} \right\rceil \leq b \leq \left\lfloor \sqrt{\frac{4}{3}H} \right\rfloor. \quad (7.2)$$

We denote the integer upper bound of this inequality b_h .³

With this in mind, we can describe an algorithm—Algorithm 7.1—for sieving for Eisenstein integers within a given norm-bounded region. Similar algorithms may be developed for a variety of sieve problems.

7.2.1 Complex Conjugate and Sieve Bounds

In some cases, it is possible to reduce the search space of a two-dimensional sieve problem. As an illustration, consider the specific case of the Eisenstein pseudocubes.

Let $\alpha = a + b\omega$ be an Eisenstein pseudocube candidate for the prime p ; *i.e.* α is primary, $\gcd(a, b) = 1$, $\left(\frac{q}{\alpha}\right)_3 = 1$ for all rational primes $q \leq p$, and α is not a cube in $\mathbb{Z}[\omega]$. By Proposition 4.39, α 's complex conjugate, $\bar{\alpha}$ is also an Eisenstein pseudocube

²This latter fact would also be clear from observing that $4a^2 - 4ab + 4b^2 < 4H$ and hence $(2a - b)^2 + 3b^2 = 4H$. Clearly $|b|$ is maximal when $2a - b = 0$, and the inequality of (7.2) is again obtained.

³We need only name the upper bound here, as $-[x] = \lceil -x \rceil$, and hence, the lower bound is given by $-b_h$. Further properties of the floor and ceiling functions are discussed in Section 7.3.2.

Algorithm 7.1: NBSIEVE: Sieve Eisenstein integers in a norm-bounded region

Input: $H, \mathcal{S} := \{\mathcal{R}, M\}$.

Output: All $\alpha \in \mathcal{S}$, such that $N(\alpha) \leq H$.

```

1:  $b_i \leftarrow \left\lceil -\sqrt{\frac{4}{3}H} \right\rceil$ 
2: while  $b_i \leq \sqrt{\frac{4}{3}H}$  do
3:    $A_\ell \leftarrow \left\lceil \frac{b_i - \sqrt{4H - 3b_i^2}}{2} \right\rceil$ ,  $A_h \leftarrow \left\lceil \frac{b_i + \sqrt{4H - 3b_i^2}}{2} \right\rceil$ 
4:   if all sieve rings  $\mathcal{R}_i$  contain at least one nonzero entry then
5:     SIEVE( $\mathcal{S}, A_\ell, A_h, b_i$ )
6:   end if
7:    $b_i \leftarrow b_i + 1$ 
8: end while

```

candidate with the same norm. Furthermore, since

$$\overline{a + b\omega} = a + b\bar{\omega} = (a - b) - b\omega,$$

exactly one of $\{\alpha, \bar{\alpha}\}$ will be in the upper half plane ($b > 0$).⁴ Thus, we may replace the assignment in line 1 of Algorithm 7.1 with $b_i \leftarrow 1$, and eliminate over half of the problem search space.

7.2.2 Number of Solution Candidates

Consider the graph of Figure 7.1, with the axes representing the components of the Eisenstein integer $\alpha = a + b\omega$. The shaded ellipse indicates a norm-bounded region satisfying $a^2 + b^2 - ab \leq H$. We may now establish that the number of integer lattice

⁴We may ignore the case where $b = 0$, as Definition 4.38 insists that for an Eisenstein pseudocube, $\gcd(a, b) = 1$. The only solution of this form when $b = 0$ is $\alpha = 1$, a perfect cube, which is rejected by the same definition.

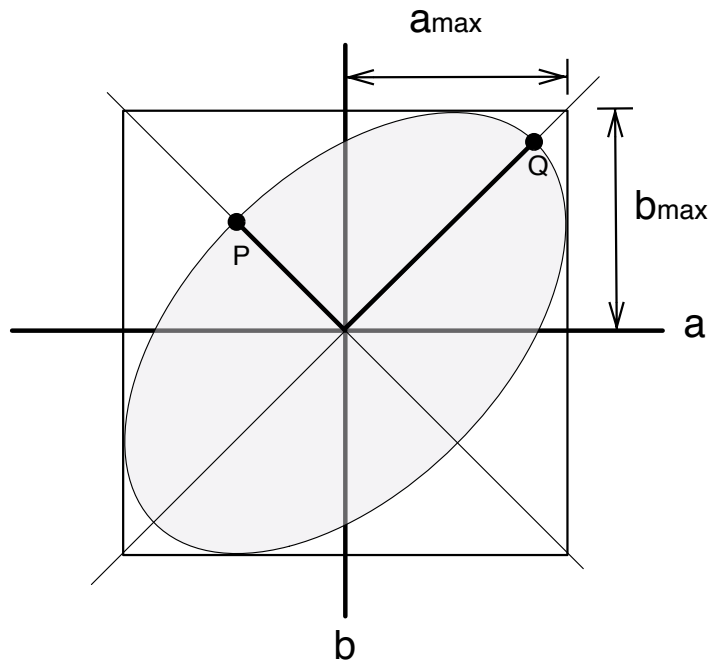


Figure 7.1: Region of Bounded Norm: $N(\alpha) = a^2 + b^2 - ab \leq H$

points in this region grows (roughly) linearly with H .⁵

From the preceding argument, we know that $a_{max} = b_{max} = \sqrt{\frac{4}{3}H}$. Hence there are $\frac{16H}{3}$ points contained within the bounding box of the ellipse. To find the number of integer lattice points contained within the ellipse, we first compute the intersections of the ellipse $a^2 + b^2 - ab = H$ and the lines $a = b$ and $a = -b$, given by points Q and P respectively in Figure 7.2. Thus, $P = \left(\sqrt{\frac{H}{3}}, -\sqrt{\frac{H}{3}}\right)$, $Q = \left(\sqrt{H}, \sqrt{H}\right)$, and the area of the ellipse—roughly the number of integer lattice points it contains—is given by $\frac{2\pi}{\sqrt{3}}H$. This is linear in H .

Finally, if we can compute the number of acceptable residues, s_i , for each of the k

⁵Though we are dealing with integer quantities, we do not need exact solutions for this argument, hence we will forego the floor and ceiling notation.

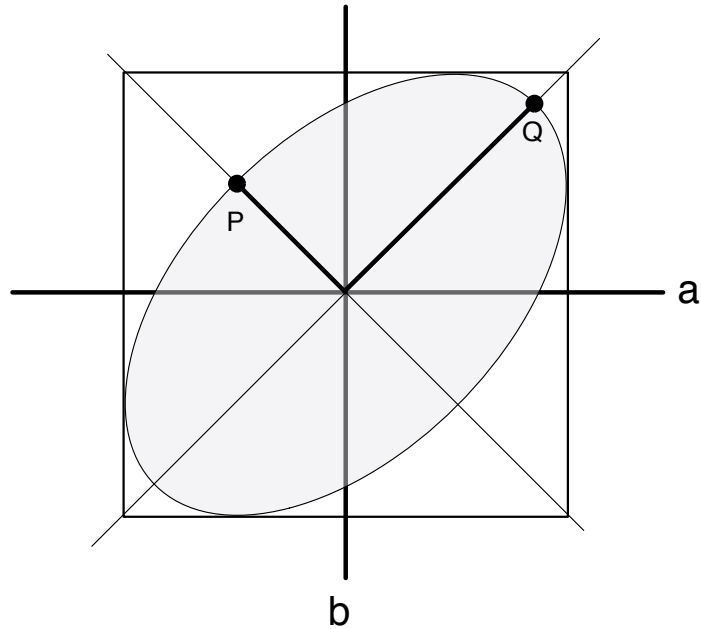


Figure 7.2: Estimating Integer Lattice Points

moduli m_i in our sieve problem, then by CRT and the preceding argument, the number of solution candidates in the norm-bounded region $a^2 + b^2 - ab \leq H$ is roughly:⁶

$$\frac{2\pi}{\sqrt{3}} H \prod_{i=1}^k \frac{s_i}{m_i}. \quad (7.3)$$

7.3 Normalization in Two-Dimensions

As in the one-dimensional case, consider the set \mathcal{R}_j of acceptable residues $(\text{mod } m_j)$. If we insist that any solution $a + b\omega \in \mathcal{R}_j$ also satisfy the congruence $a + b\omega \equiv x + y\omega$

⁶Prior to performing any filtering operations.

(mod m), then we can translate the problem as follows:

$$\begin{aligned} a + b\omega \equiv x + y\omega \pmod{m} &\implies m \mid (a + b\omega) - (x + y\omega) \\ &\implies m \mid (a - x) + (b - y)\omega. \end{aligned}$$

Since $m \in \mathbb{Z}$, we must have $m \mid (a - x)$ and $m \mid (b - y)$. So we can write $(a - x) = Am$, $(b - y) = Bm$ for integers $A, B \in \mathbb{Z}$. We now have the correspondence

$$a = mA + x \tag{7.4}$$

$$b = mB + y. \tag{7.5}$$

So long as $\gcd(m, m_j) = 1$, there exists a bijection between solutions $a + b\omega \in \mathcal{R}_j$ and solutions $A + B\omega = (a - x)m^{-1} + (b - y)m^{-1}\omega \pmod{m_j}$. This result is analogous to normalization in one dimension, as described in Section 2.2.2. We formalize this map as follows:

Definition 7.1. *Let \mathcal{R}_j be a set of acceptable residues modulo m_j such that $\alpha \equiv x + y\omega \pmod{m}$ for all $\alpha \in \mathcal{R}_j$. The normalization map $\eta : \mathbb{Z}[\omega] \rightarrow \mathbb{Z}[\omega]$ is given by*

$$\eta : a + b\omega \mapsto (a - x)m^{-1} + (b - y)m^{-1}\omega = A + B\omega \pmod{m_j}, \tag{7.6}$$

where A and B are defined by (7.4) and (7.5). For notational convenience, we will write

$$\eta(\mathcal{R}_j) = \{\eta(\alpha) \mid \alpha \in \mathcal{R}_j\}.$$

Note that if we are only interested in solutions $\alpha \in \mathcal{R}_j$ such that $N(\alpha) \leq H$, we

can reduce our search space by a factor of approximately m^2 by considering $A + B\omega$, and adjusting our problem bounds, (7.1) and (7.2), as follows.

7.3.1 Computing Normalized Bounds

Let a_ℓ, a_h, b_h be the non-normalized sieve bounds obtained from equations (7.1) and (7.2). Consider $0 \leq Bm + y \leq \sqrt{\frac{4}{3}H}$. The bounds on the normalized problem are given by

$$\frac{-y}{m} \leq B \leq \frac{\sqrt{\frac{4}{3}H} - y}{m}.$$

Since $0 \leq y < m$, $B \in \mathbb{Z}$, we obtain the bounds

$$B_h = \left\lfloor \frac{\sqrt{\frac{4}{3}H} - y}{m} \right\rfloor$$

$$B_\ell = \begin{cases} 0 & \text{if } y > 0 \\ 1 & \text{otherwise} \end{cases}.$$

A similar process can be used to obtain the upper and lower bounds on A , where $a = Am + x$; *i.e.*

$$A_\ell = \left\lfloor \frac{(b - 2x) - \sqrt{4H - 3b^2}}{2m} \right\rfloor$$

$$A_h = \left\lfloor \frac{(b - 2x) + \sqrt{4H - 3b^2}}{2m} \right\rfloor.$$

It should be noted that, unlike the traditional one-dimensional sieve problem, negative values for sieve bounds are frequently encountered. One potential headache

in evaluating the one-dimensional sieve bounds has to do with the interaction of the square root, division, floor/ceiling functions, and negative numbers, especially when dealing with integer arithmetic. Before we proceed, therefore, some basic properties about integer functions should be reviewed.

7.3.2 Integer Functions

In 1962, Kenneth Iverson [Ive62, §1.4] introduced the modern nomenclature for the *floor* and *ceiling* functions; *i.e.* given $x \in \mathbb{R}$, $n \in \mathbb{Z}$,

$$\lfloor x \rfloor = \min\{n \in \mathbb{Z} \mid n \leq x\}, \quad (7.7)$$

$$\lceil x \rceil = \max\{n \in \mathbb{Z} \mid n \geq x\}. \quad (7.8)$$

It should be noted from this definition that both floor and ceiling functions are *non-decreasing*. Furthermore, $\lceil -x \rceil = -\lfloor x \rfloor$, and

$$\lceil x \rceil = \begin{cases} \lfloor x \rfloor & \text{if } x \in \mathbb{Z} \\ \lfloor x \rfloor + 1 & \text{otherwise.} \end{cases} \quad (7.9)$$

In addition to the standard notation, we will use the notation $[x]$ to denote the truncated integer representation of x ; *i.e.*

$$[x] = \begin{cases} \lfloor x \rfloor & \text{if } x \geq 0 \\ \lceil x \rceil & \text{if } x < 0. \end{cases} \quad (7.10)$$

Thus, $[x]$ has the nice property that $[-x] = -[x]$.

The following observation is extremely useful for proving statements involving

ceilings and floors.

Proposition 7.2. (*R.J. McElice*) *Consider any continuous, monotonically increasing function*

$$f : \mathbb{R} \rightarrow \mathbb{R}.$$

If $f(x) \in \mathbb{Z}$ implies that $x \in \mathbb{Z}$, then

$$\lfloor f(x) \rfloor = \lfloor f(\lfloor x \rfloor) \rfloor, \text{ and} \tag{7.11}$$

$$\lceil f(x) \rceil = \lceil f(\lceil x \rceil) \rceil. \tag{7.12}$$

Proof. To prove (7.11), observe that if $x = \lfloor x \rfloor$ we are done, so assume $x > \lfloor x \rfloor$. Since f is nondecreasing, $f(x) > f(\lfloor x \rfloor)$. Furthermore, since the floor operator is nondecreasing, $\lfloor f(x) \rfloor \geq \lfloor f(\lfloor x \rfloor) \rfloor$. If $\lfloor f(x) \rfloor > \lfloor f(\lfloor x \rfloor) \rfloor$, then since f is continuous, there exists $y \in \mathbb{R}$ such that $\lfloor x \rfloor \leq y < x$ and $f(y) = \lfloor f(x) \rfloor$. By the proposition, this means that $y \in \mathbb{Z}$. By the definition of the floor function, however, there cannot be a distinct integer lying between x and $\lfloor x \rfloor$, a contradiction. Thus, $\lfloor f(x) \rfloor = \lfloor f(\lfloor x \rfloor) \rfloor$, as desired. Equation (7.12) can be proved analogously; e.g. [GKP89, §3.2]. \square

Corollary 7.3. *If $b > 0$*

$$\left\lfloor \frac{\lfloor x \rfloor}{b} \right\rfloor = \left\lfloor \frac{x}{b} \right\rfloor.$$

Corollary 7.4.

$$\left\lfloor \sqrt{\lfloor x \rfloor} \right\rfloor = \left\lfloor \sqrt{x} \right\rfloor.$$

Corollary 7.5. *If $n > 0$*

$$\left\lfloor \frac{m + \lfloor x \rfloor}{n} \right\rfloor = \left\lfloor \frac{m + x}{n} \right\rfloor.$$

When using integer functions to approximate non-integer quantities—especially those that produce truncation or rounding—care must be taken to ensure that the functions are compatible. One common area of confusion surrounds the integer division operator.⁷ To avoid this confusion, we will borrow an explicit definition of integer division from the Ada programming language [TDB⁺06, §4.5.5].

Definition 7.6. (*div*, *rem*, *divr*)

Recall the notion of a truncated integer representation (7.10). Define integer division (*div*) as truncating division; i.e.

$$\mathit{div}(A, B) \mapsto \left\lfloor \frac{A}{B} \right\rfloor.$$

Integer remainder (*rem*) is then defined to satisfy the relation:

$$A = \mathit{div}(A, B) \cdot B + \mathit{rem}(A, B),$$

where $\mathit{rem}(A, B)$ has the sign of A and $|\mathit{rem}(A, B)| < |B|$. Define *divr* to be the combined operation of integer division and remainder; i.e.

$$\mathit{divr}(A, B) \mapsto (q, r)$$

where $q = \mathit{div}(A, B)$, $r = \mathit{rem}(A, B)$.

Note that this definition has the nice property that integer division satisfies the

⁷As a notable example, the rounding direction for the integer division operator (`/`) in the C programming language was not standardized until C99 [IJ02, §7.6.1, p. 228], and hence may vary depending on the compiler in use.

following relation:

$$\operatorname{div}(-A, B) = -\operatorname{div}(A, B) = \operatorname{div}(A, -B).$$

In addition to the division operators, we will also make use of the following.

Definition 7.7. (*isqrt*)

Let $X \in \mathbb{R}$, $X \geq 0$, then the integer square root function (*isqrt*) is given by

$$\mathit{isqrt}(X) \mapsto (s, \delta), \tag{7.13}$$

where $s \in \mathbb{Z}$ such that $s^2 \leq X < (s+1)^2$, $\delta = \lceil X - s^2 \rceil$.

7.3.3 Computing Sieve Bounds Using Integer Operations

First, we wish to compute $b_h \in \mathbb{Z}$ such that $b_h = \left\lfloor \sqrt{\frac{4H}{3}} \right\rfloor$ using only integer operations.

As $H > 0$, Corollary 7.4 and (7.10) give that

$$b_h = \left\lfloor \sqrt{\left\lceil \frac{4H}{3} \right\rceil} \right\rfloor = \left\lfloor \sqrt{\left\lceil \frac{4H}{3} \right\rceil} \right\rfloor,$$

which can be computed using the integer functions from Definitions 7.6 and 7.7.

For normalized bounds—in other words, where $b = mB + y$ —we obtain:

$$B \leq \left\lfloor \frac{\sqrt{4H/3} - y}{m} \right\rfloor = \left\lfloor \frac{\left\lfloor \sqrt{\lceil 4H/3 \rceil} \right\rfloor - y}{m} \right\rfloor = \left\lfloor \frac{b_h - y}{m} \right\rfloor, \tag{7.14}$$

whenever $b_h \geq y$. Next we wish to compute the integer bounds a_ℓ, a_h such that

$$\left\lceil \frac{b - \sqrt{4H - 3b^2}}{2} \right\rceil \leq a \leq \left\lfloor \frac{b + \sqrt{4H - 3b^2}}{2} \right\rfloor.$$

Since the normalized bounds on A are given by

$$\left\lceil \frac{(b - 2x) - \sqrt{4H - 3b^2}}{2m} \right\rceil \leq A \leq \left\lfloor \frac{(b - 2x) + \sqrt{4H - 3b^2}}{2m} \right\rfloor,$$

we can consider both situations simultaneously if we consider an upper bound $a_h = \left\lfloor \frac{n + \sqrt{S}}{d} \right\rfloor$, and a lower bound $a_\ell = \left\lceil \frac{n - \sqrt{S}}{d} \right\rceil$ with $n \in \mathbb{Z}$, $S \in \mathbb{Z}_{\geq 0}$, $d \in \mathbb{N}$.

To compute the upper bounds, let

$$\alpha = \left\lceil \frac{n + \lfloor \sqrt{S} \rfloor}{d} \right\rceil.$$

If $(n + \lfloor \sqrt{S} \rfloor) \geq 0$, Corollary 7.5 gives that

$$\begin{aligned} \alpha &= \left\lceil \frac{n + \lfloor \sqrt{S} \rfloor}{d} \right\rceil = \left\lfloor \frac{n + \sqrt{S}}{d} \right\rfloor \\ &= a_h. \end{aligned}$$

Conversely, if $(n + \lfloor \sqrt{S} \rfloor) < 0$,

$$\begin{aligned} \alpha &= \left\lceil \frac{n + \lfloor \sqrt{S} \rfloor}{d} \right\rceil = \left\lfloor \frac{n + \lfloor \sqrt{S} \rfloor}{d} \right\rfloor + \epsilon = \left\lfloor \frac{n + \sqrt{S}}{d} \right\rfloor + \epsilon \\ &= a_h + \epsilon \\ \text{where } \epsilon &= \begin{cases} 0 & \text{if } \frac{n + \sqrt{S}}{d} \in \mathbb{Z} \\ 1 & \text{otherwise.} \end{cases} \end{aligned}$$

For the negative bound, consider

$$\beta = \left\lfloor \frac{n - \lfloor \sqrt{S} \rfloor}{d} \right\rfloor. \quad (7.15)$$

If $(n - \lfloor \sqrt{S} \rfloor) < 0$, then

$$\begin{aligned} \beta &= \left\lfloor \frac{n - \lfloor \sqrt{S} \rfloor}{d} \right\rfloor = - \left\lfloor \frac{-n + \lfloor \sqrt{S} \rfloor}{d} \right\rfloor = - \left\lfloor \frac{-n + \sqrt{S}}{d} \right\rfloor = \left\lceil \frac{n - \sqrt{S}}{d} \right\rceil \\ &= a_\ell. \end{aligned}$$

Algorithm 7.2: BOUNDB

Input: Normalization $(x, y) \pmod{m}$, upper bound H .

Output: b , the integer upper height bound of a 2D sieve problem.

```

1:  $(B, \delta) \leftarrow \text{isqrt}(\text{div}(4H, 3))$ 
2: if  $B < y$  then
3:   /*Upper bound must be positive, as  $b$  is symmetric around  $0^*$ */
4:   return 0
5: end if
6:  $b \leftarrow \text{div}(B - y, m)$ 
7: return  $b$ 

```

Conversely, if $(n - \lfloor \sqrt{S} \rfloor) \geq 0$,

$$\begin{aligned}
 \beta &= \left\lfloor \frac{n - \lfloor \sqrt{S} \rfloor}{d} \right\rfloor = - \left\lceil \frac{-n + \lfloor \sqrt{S} \rfloor}{d} \right\rceil \\
 &= - \left(\left\lfloor \frac{-n + \lfloor \sqrt{S} \rfloor}{d} \right\rfloor + \epsilon \right) = - \left\lfloor \frac{-n + \sqrt{S}}{d} \right\rfloor - \epsilon \\
 &= \left\lfloor \frac{n - \sqrt{S}}{d} \right\rfloor - \epsilon = a_\ell - \epsilon,
 \end{aligned}$$

where

$$\epsilon = \begin{cases} 0 & \text{if } \frac{n - \sqrt{S}}{d} \in \mathbb{Z} \\ 1 & \text{otherwise.} \end{cases}$$

These results are summarized in Algorithms 7.2 and 7.3. A fully optimized Eisenstein sieve algorithm making use of normalization is given in Algorithm 7.4.

Algorithm 7.3: BOUNDA

Input: Sieve bound H , normalization $(x, y) \pmod{m}$, sieve height, b .

Output: (a_ℓ, a_h) , the integer width bounds corresponding to height b .

```

1:  $(S, \delta) \leftarrow \text{isqrt}(4H - 3b^2)$ 
2:  $n \leftarrow (b - 2x), d \leftarrow 2m$ 
3:  $(q, r) \leftarrow \text{divr}(n - S, d)$ 
4: if  $(n - S \geq 0)$  and  $(\delta = 1$  or  $r > 0)$  then
5:    $a_\ell \leftarrow q + 1$ 
6: else
7:    $a_\ell \leftarrow q$ 
8: end if
9:  $(q, r) \leftarrow \text{divr}(n + S, d)$ 
10: if  $(n + S < 0)$  and  $(\delta = 1$  or  $r > 0)$  then
11:    $a_h \leftarrow q - 1$ 
12: else
13:    $a_h \leftarrow q$ 
14: end if
15: return  $(a_\ell, a_h)$ 

```

Algorithm 7.4: ESIEVE: Eisenstein sieve, with normalization

Input: $H, \mathcal{S} := \{\mathcal{R}, M\}$, Normalization $(x, y) \pmod{m}$.

Output: All $\alpha \in \mathcal{S}$, such that $N(\alpha) \leq H$.

```

1: if  $y = 0$  then
2:    $b_i \leftarrow 1$ 
3: else
4:    $b_i \leftarrow 0$ 
5: end if
6: while  $b_i \leq \text{BOUNDB}(H, x, y, m)$  do
7:   if all sieve rings  $\mathcal{R}_i$  contain at least one nonzero entry then
8:      $(A_\ell, A_h) \leftarrow \text{BOUNDA}(H, x, y, m, b_i)$ 
9:     Output SIEVE( $\mathcal{S}, A_\ell, A_h, b_i$ )
10:   end if
11:    $b_i \leftarrow b_i + 1$ 
12: end while

```

7.4 Filtering and Output Processing

With the incorporation of two-dimensional sieving into the CASSIE framework, a significantly wider variety of filters and output formats needed to be supported. In its one-dimensional form, CASSIE supported a single, optional filter routine to be applied to the sieve, and a hard-coded output file format. To add flexibility to the sieve, and to help address potential filter bottlenecks, a multistage filter mechanism was introduced. This multistage subsystem incorporated two distinct filter types: *exclusion filters*, used to exclude solution candidates, and *output filters*, used to log valid solutions.

Exclusion filters are used to selectively reject candidate solutions that emerge from the sieve process. For example, in the case of the Eisenstein pseudocubes two commonly used filters are `relatively_prime`, which ensures that the x and y components of a solution candidate (x, y) have no factors in common, and `perfect_cube`, which rejects perfect cubes using the ideas of Section 6.1.

Output filters transform the candidate solutions into a form convenient for post-processing, and manage the task of writing these solutions to an output file. Though multiple output filters are possible, in general there is only one, and it is the last filter applied to candidate solutions.

7.5 Summary

In this chapter, we described four main enhancements to the CASSIE framework in order to represent, optimize, and execute two-dimensional sieve problems. In Chapter 10, we will use this new framework to build a table of Eisenstein pseudocubes.

Chapter 8

Open Hardware for Mathematical Research (OHMR)

If a machine can be made so that an idiot can use it, then only an idiot will use it.

—Tadao Ichikawa

8.1 Overview

One of the frustrations in designing special-purpose hardware components, even on FPGA platforms is that without a great deal of support infrastructure, such components are virtually useless. Only once embedded into complete systems do these devices become capable of solving real problems, and often, designing and building the rest of the system constitutes as much work as building the custom device.

This chapter describes our attempt to reduce this extraneous effort by building a reusable toolkit capable of handling such mundane—yet vital—operations as moving data to and from the device, managing control signals, monitoring device state, and communicating over a high-speed serial port to a host machine. This toolkit, called the OHMR framework, is pictured in Figure 8.1.

OHMR is designed around an 8-bit PicoBlaze soft processor, is implemented in a combination of Verilog and PicoBlaze assembly, and consists of the following components:

- a high-speed Universal Asynchronous Receiver Transmitter (UART) for serial

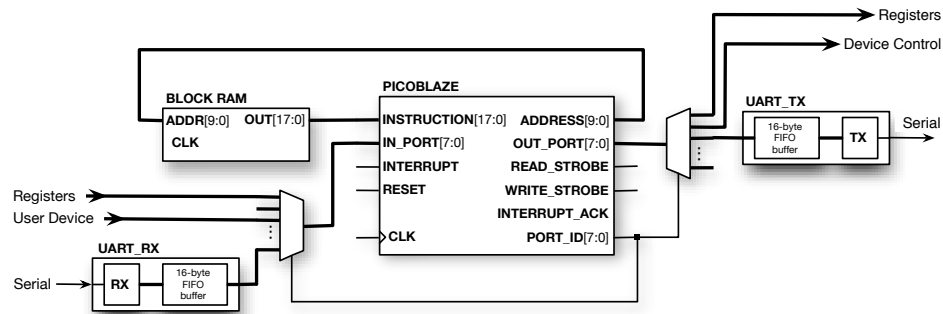


Figure 8.1: OHMR Architecture

connection to host machine,

- lightweight OHMR Monitor (OHMon) command language, implemented using a PicoBlaze 8-bit soft-processor,
- 2 multibyte (128-bit) data registers: **L**, for general-purpose device Input/Output (I/O), and **W**, for device output,
- 4 8-bit data registers: 3 general purpose—**AX**, **OX**, **MX**—and one wired to an 8-bit Light Emitting Diode (LED) output, **DX**,
- an onboard Read Only Memory (ROM) for compile-time device parameter storage, and
- 2 8-bit control banks (**CB1**, **CB2**), implementing one-shot signals for device control.

8.1.1 Input/Output (I/O) Architecture

OHMR was designed to be a general framework for implementing numerical coprocessors on FPGA silicon. Though it was intended to interact closely with software running on a host machine, the OHMR hardware platform was designed to be as independent as possible from the host architecture. For this reason, a serial connec-

tion was chosen as the main interface to the framework, allowing OHMR to be truly independent of the underlying host machine architecture.

Unfortunately, a serial connection is not as tight a coupling as, say, a Peripheral Component Interconnect (PCI) or SBus-connected device. As such, it is clear this architecture choice can result in an I/O bottleneck. To this end, OHMon was designed with a low communication overhead, employing one- or two-letter commands for most operations.

In Future Work (Chapter 11), we will consider the problem of integrating a higher-speed device interface. Universal Serial Bus (USB) is still an option, as a kernel-layer device driver would permit communication speeds up to USB2.0 speeds, 400-fold improvement over the current design, while reusing most of the existing I/O infrastructure. Such a driver, of course, would be architecture-specific, mitigating some of the benefit of an architecture-independent approach.

8.2 The PicoBlaze Embedded Soft Processor

Rather than incorporate a hardwired CPU device, the OHMR design makes use of an embedded 8-bit soft processor, the Xilinx PicoBlaze, as shown in Figure 8.2. This processor occupies a small amount of area on the FPGA device, and runs the OHMon command language that is responsible for all interaction with the coprocessor device. OHMon is responsible for four main activities:

- serial communication with the host system,
- interrupt Management,
- general I/O with the coprocessor unit, and

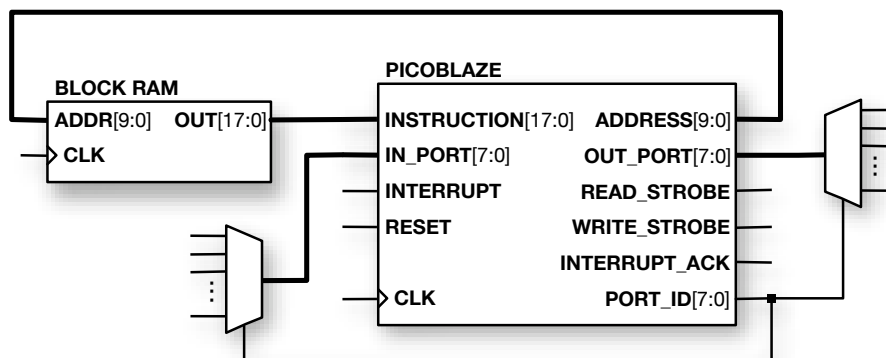


Figure 8.2: PicoBlaze in OHMR

- parameter storage, for information about the coprocessor device.

OHMon will be described starting in Section 8.3. Before proceeding with that description, however, we should first describe some features of the PicoBlaze architecture that figure prominently in the design of OHMon. Though we will not describe the PicoBlaze architecture in full—an excellent reference is available in [UG108]—there are two main facilities we will describe here: interrupt handling, and the PicoBlaze I/O mechanism.

8.2.1 PicoBlaze Input / Output

The PicoBlaze soft processor includes a general-purpose 8-bit I/O infrastructure which is a critical component for device communication in the OHMR architecture. PicoBlaze Assembly provides two instructions, `INPUT` and `OUTPUT`, to retrieve data into and send data from the PicoBlaze, respectively. Both instructions take a data register name, and either an immediate mode constant, or another register name indicating the 8-bit port identifier to be used. Like all other PicoBlaze instructions, `INPUT` and

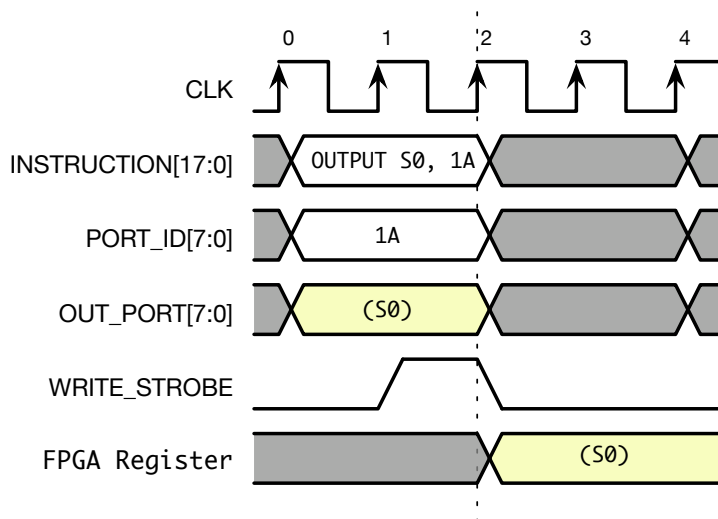


Figure 8.3: Port Timing for OUTPUT instruction

OUTPUT operate over two clock cycles; however, both operations also make use of a one-cycle read/write strobe to facilitate interfacing with external devices.

The OUTPUT cycle is described in Figure 8.3. Notably, when an OUTPUT instruction is executed, user data is made available for two cycles on OUT_PORT[7:0], the numerical port identifier is presented on PORT_ID[7:0], and WRITE_STROBE is asserted for the second cycle of valid data. This construction makes it convenient to pipeline decoding of the port data, as described in [UG108, pp. 58–60]. This feature is also an integral component of the control bank mechanism described in Section 8.5.

INPUT cycle timing is similar to the OUTPUT instruction, and is depicted in Figure 8.4. Again, a numerical port identifier is presented on PORT_ID[7:0] for two-cycles, facilitating pipelining of PORT_ID decoding. A one-cycle READ_STROBE is asserted to indicate when data is latched from IN_PORT[7:0]. The timing of this READ_STROBE is convenient for acknowledging the receipt of data, and is used explicitly for this

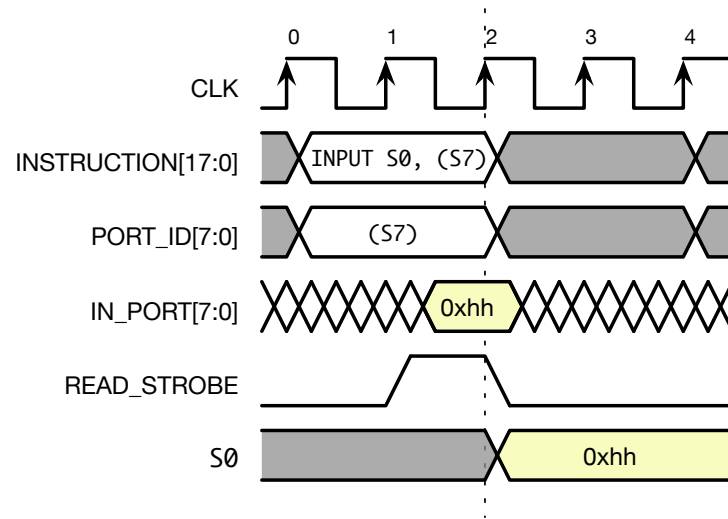


Figure 8.4: Port Timing for INPUT instruction

purpose in Section 8.7.1.

8.2.2 Interrupt Facility

The PicoBlaze processor employs a simple interrupt mechanism, illustrated in Figure 8.5, consisting of a single INTERRUPT line, and an associated INTERRUPT_ACK [UG108, §4]. An interrupt is signalled by asserting INTERRUPT for at least two clock cycles, which generates an interrupt event. At this time, the processor preempts execution of the next instruction, saves the current contents of the flags register and instruction pointer, disables further interrupts, and jumps to the end of the PicoBlaze address space (0x3FF). A user application wishing to make use of the interrupt mechanism will store a JUMP instruction in location 0x3FF, passing execution off to a custom interrupt service routine (ISR). When this routine completes, normal execution flow is resumed by issuing a RETURNI instruction—restoring program counter and flags, and re-enabling interrupt handling.

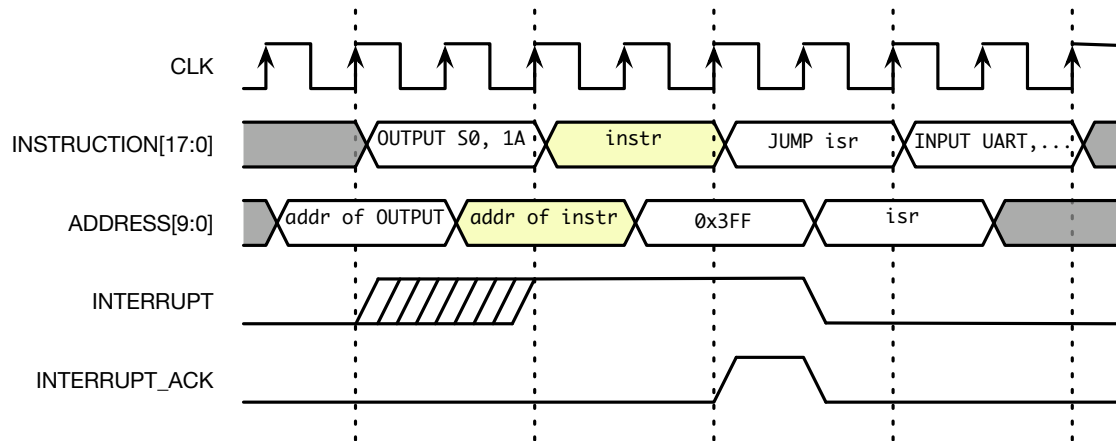


Figure 8.5: Interrupt Timing

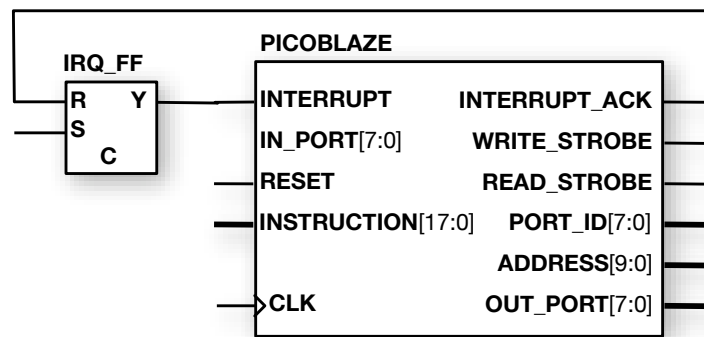


Figure 8.6: Handling Interrupts using a Dedicated Flip-Flop

As indicated in Figure 8.5, the PicoBlaze processor asserts an INTERRUPT_ACK signal during the second cycle of the interrupt event. This signal greatly simplifies hardware interfacing for this interrupt mechanism; the most simple of which consists of a single Set-Reset (SR) flip-flop, as depicted in Figure 8.6.

In OHMR, the interrupt facility is used both for serial communication with the host machine, which will be discussed in Section 8.7, and for indicating when data is available from the coprocessor device. Since this latter operation is specific to the

coprocessor device being implemented, it will not be discussed here.

8.3 OHMon and the OHMR Architecture

Interaction between the host system and coprocessor unit occurs via OHMon, a text-based shell language implemented in PicoBlaze assembly. OHMon has facilities for managing communication between the host machine and all components of the OHMR architecture. Software running on the host machine requires only the ability to communicate via a serial port in order to use the OHMR framework.

There are 3 main command types in OHMon:

- Load commands (**L...**): get data into the coprocessor device,
- Display commands (**D...**): read data back from the coprocessor device,
- User commands (**Uhh** and **Vhh**): send one-shot signals to device control ports.

Though most designs will be functional using these three command types, some designs will require the addition of special-purpose instructions to OHMon. Examples of this type of extension can be found in Sections 9.4 and 9.5.1.

8.4 OHMR Register Architecture

Data is passed to and from the coprocessor device using a mixture of 8-bit and 128-bit (multi-byte) registers. The OHMR register architecture is shown in Figure 8.7.

8.4.1 8-bit registers

There are seven single-byte registers in the OHMR framework—four general purpose registers (**AX**, **DX**, **MX**, **OX**) and three special-purpose registers: **SS**, **LX**, and **WX**:

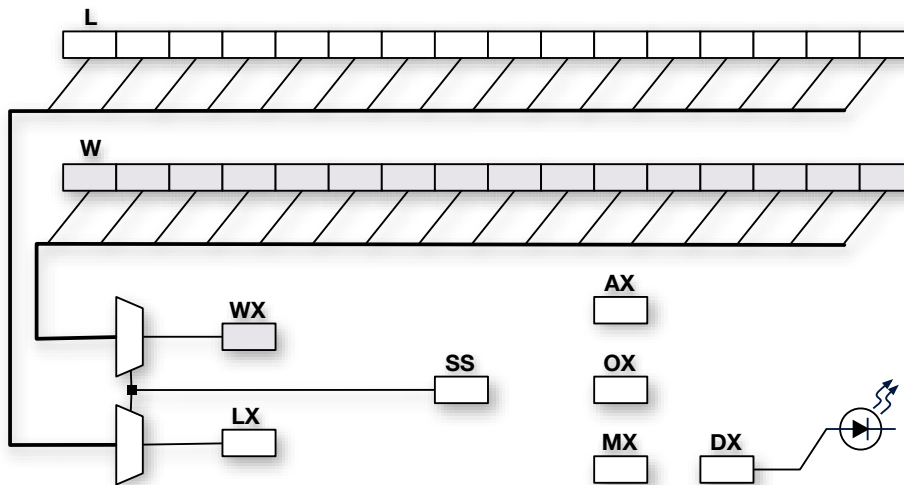


Figure 8.7: Registers in the OHMR architecture

- AX** Address or general purpose register,
- DX** General purpose LED register, typically wired to output LEDs,
- MX** General purpose (mode) register,
- OX** General purpose (output) register,
- SS** Special-purpose (select) register,
- LX** Special purpose—windowed view of **L** register,
- WX** Special purpose—windowed view of **W** register.

The **AX**, **OX**, and **MX** registers are general-purpose registers that may be read to and written from the monitor. The **DX** register is also general-purpose, but as it is wired to the onboard 8-bit LED display, it is typically used for display purposes. The special-purpose **SS**, **LX**, and **WX** are associated with multibyte registers. Their use is described in Section 8.4.2.

The OHMon commands used to manipulate 8-bit registers are as follows:

- DX?** Display contents of **?X** register, where **?** is one of **A,D,L,M,O,S,W**;

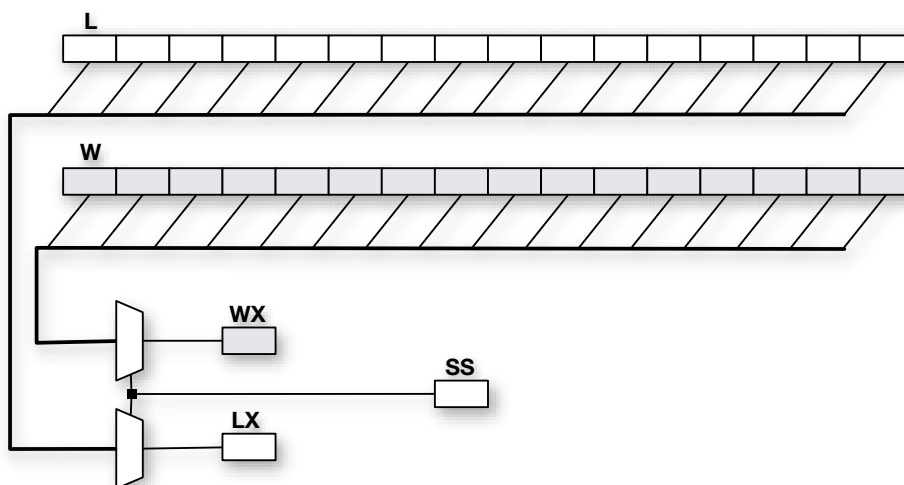


Figure 8.8: OHMR Multibyte Registers

$LX?hh$ Load hh into $?X$ register, where $?$ is one of A, D, L, M, O, S .

8.4.2 Multi-byte Registers

To facilitate the handling of large-precision coprocessor operations, 128-bit (multi-byte) registers provide the main data path between the coprocessor unit and OHMon. There are two multi-byte registers in the OHMR framework: **L** and **W**. Due to the 8-bit nature of the PicoBlaze soft processor, however, multi-byte registers must be windowed into 8-bit slices. To handle the windowing of the 128-bit data, a dedicated data select register, **SS** is used as a Multiplexer (MUX) select. The 8-bit windowed views of the 128-bit registers **L** and **W** are named **LX** and **WX** respectively. This arrangement is depicted in Figure 8.8.

The following OHMon commands are used to manipulate multibyte OHMon registers:

DL Display contents of **L** register,

- Dw Display contents of **W** register,
- DW Latch output of device into **W**, then display register contents,
- LLhh...hh Load **L** register with hex data *hh...hh*,
- LP Latch device parameters (using address in **AX**) into **W**,
- LW Latch data from device into **W**.

In general, data to be loaded in the **L** register originates from the host machine, and data to be loaded into the **W** register originates from the coprocessor device. For this reason, both the **W** and **WX** registers appear to be read-only from the OHMon framework; *i.e.* there is no *LWhh..hh* or *LXW* instruction. The contents of **W** are affected only by latch commands; *i.e.* *LW* or *LP*. Note that the *DW* command is equivalent to *LW* followed by *Dw*.

8.5 Control Banks

Control lines represent the primary means of controlling a hardware coprocessor unit in the OHMR framework. Unlike more general-purpose I/O facilities, control lines are typically active for only one clock cycle at a time.¹ Hence, unlike most other I/O paths with the hardware coprocessor unit, control signals do not make use of a dedicated hardware register. Instead, control signals are arranged into *control banks* and are implemented directly via the PicoBlaze I/O port mechanism.

Recall from Section 8.2.1 the PicoBlaze microcontroller supports up to 2⁸ output ports [UG108, Chapter 6]. When user data is made available on OUT_PORT[7:0], a numerical port identifier, PORT_ID[7:0] and a one-cycle WRITE_STROBE signal the

¹For simplicity of discussion, control signals are assumed to be active-high. Judicious use of inverters can be used to adjust for this assumption in implementation.

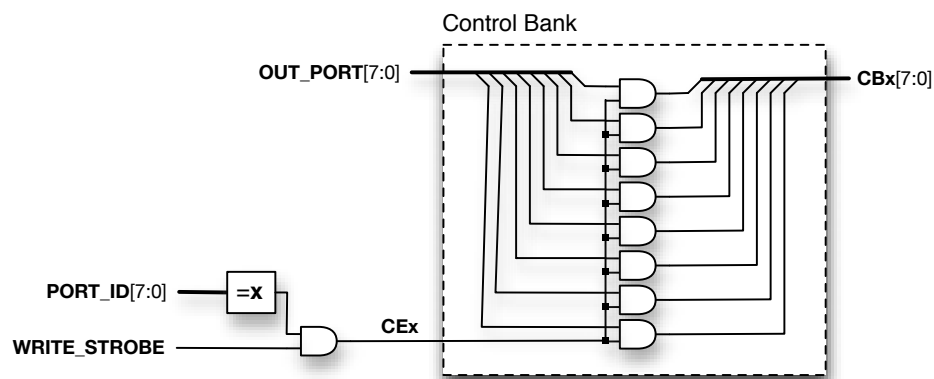


Figure 8.9: Control Bank Detail

presence of valid data. With this in mind, it is straightforward to synthesize a control signal bank as in Figure 8.9. Here, the $=x$ operation refers to the combinational circuit that is active when the contents of `PORT_ID[7:0]` match a hardcoded port identifier (x) for the `CB x` output bank. Only one control line is currently used by the OHMR toolkit—UART reset (`URST`). This command is mapped to control bank 2, address `0x01` (OHMon: `v01`).

8.6 Parameter Storage

It is often convenient to store device-specific parameters associated with a hardware device in a manner that can be retrieved by the host machine. One common example occurs in parametrized designs, where compile-time constants determine synthesis parameters such as counter widths, input buffer sizes, and so on.

One particularly useful constant is the *hardware version* or `BUILD_NO`. This is an integer value that is automatically incremented whenever the hardware design is recompiled, allowing a running instance of OHMon to be associated with a particular

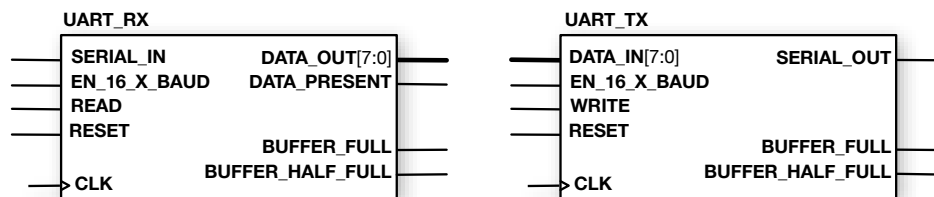


Figure 8.10: UART Macros

revision of the hardware in a source code repository. Because this parameter is useful for almost any design, location 0x00 in the ROM is set aside for this purpose, and a special command (*DV*) is made available for displaying this version number. Other device parameters are hardware specific. As such, no special ordering is set aside for them.

To access a device parameter, the address of the parameter must be present in the address register (*AX*) when the display command is issued. OHMon commands for printing device parameters are as follows:

- DP Display Device Parameter using address indicated in *AX* register;
- DV Display Hardware Version. Equivalent to 'LXA0' 'DP'.

8.7 Serial Communication in OHMon

For communication with the host machine via a serial port, a pair of highly optimized serial UARTs are incorporated into the OHMR design. These macros, pictured in Figure 8.10, occupy a minimal amount of area in the completed design—only 15 CLBs on a typical Xilinx FPGA [Cha08].

8.7.1 Receive Path

The receive UART (**UART_RX**) is used to receive data serially, 8 bits at a time, Least Significant Bit (LSB) first, using 1 start bit (active low) and 1 stop bit (active high). Serial data is received on the `DATA_IN` line. When enough data has been received to form a complete byte, this byte is pushed into an internal 16-byte First-In First-Out (FIFO) and the `DATA_PRESENT` signal is asserted. FIFO data is made available at the `DATA_OUT[7:0]` port. Once a reader successfully obtains the data at `DATA_OUT`, the `READ` signal may be applied, acknowledging the read and causing the next data byte to be made available at the FIFO. The `DATA_PRESENT` signal is deasserted when all data has been removed from the FIFO. In addition to the `DATA_PRESENT` signal, the **UART_RX** also has both `BUFFER_FULL` and `BUFFER_HALF_FULL`, though these signals are unused in the OHMR design.

To receive data from the serial line, the `DATA_PRESENT` signal from **UART_RX** is used to set an interrupt flip-flop, **IRQ_FF**, as described in Section 8.2.2. An ISR handles this interrupt by reading a byte from **UART_RX**, storing it in an internal register before proceeding. The received byte is retransmitted as part of the OHMon idle loop.

Data is read from the receive UART via the PicoBlaze `INPUT` instruction. To indicate port data is read correctly, the PicoBlaze asserts a `READ_STROBE` signal after port data is successfully latched. This signal, combined with a `PORT_ID[0]` of 0x82, is used as a read-acknowledge signal back to the receive UART. The complete input path for serial communication is shown in Figure 8.11.

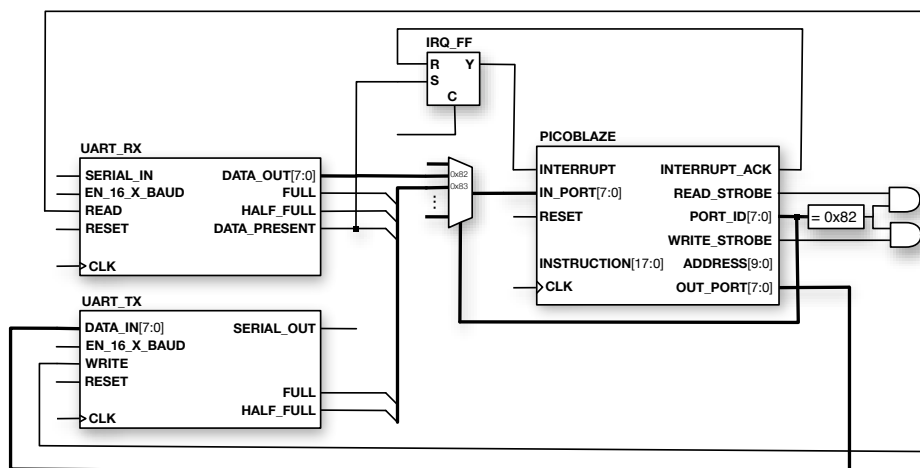


Figure 8.11: I/O Path for Serial Communications

Transmit path

The transmit path, also pictured in Figure 8.11 is similar. The transmit UART (**UART_TX**) receives data 8-bits at a time from the PicoBlaze, emitting the data to the serial line. Data is sent to the transmit UART via the **OUTPUT** instruction, asserting the UART's write signal when a **PORT_ID[O]**f 0x82 and **WRITE_STROBE** are decoded. Like in the receive path, **BUFFER_FULL** and **BUFFER_HALF_FULL** signals may be read from the UART by issuing an **INPUT** with **PORT_ID[O]**f 0x83. Unlike the receive case, however, these signals are used by OHMR, which will wait until the buffer is at least half empty before transmitting data.

Serial Port Clocking

The send (**UART_TX**) and receive (**UART_RX**) UARTs both make use of a baud clock running at $16\times$ the desired data rate. A common technique for supplying this baud clock is to make use of a divide-by- n counter associated to the system clock. This results in a single pulse at a fixed integer divisor (n) of the system clock rate. At high

Serial Speed	Clock Speed (MHz)	Divider Ratio	% Error
115200	50	27	0.47
115200	62.5	34	-0.27
230400	50	14	-3.12
230400	62.5	17	-0.27
460800	50	7	-3.12
460800	62.5	8	5.96
921600	50	3	13.03
921600	62.5	4	5.96

Table 8.1: Error Rates in Baud Generation

serial port speeds, however—230400 bps and above—it is difficult to generate these pulses without accumulating excessive error. In Table 8.1, a number of standard baud rates are shown, along with the “best” clock division ratio, and associated error.

To reduce the error to tolerable levels at high baud rates, a simple asymmetric baud clock generator is used. Though this introduces some jitter into the baud clock, the result is well within the tolerances of the UART—unlike the more commonly used clock-division method. The design of this asymmetric clock divider is simple. First, choose an accumulator size; *e.g.* B bits (more bits will result in a closer approximation to the desired frequency). Next, compute the register increment (Δ) closest to the desired clock frequency; *i.e.*

$$\Delta = B \cdot \frac{\text{Baud Clock Frequency}}{\text{Master Clock Frequency}}.$$

At each master clock edge, add the increment Δ to the B -bit register. When it overflows, emit a pulse, and continue adding to the truncated result.

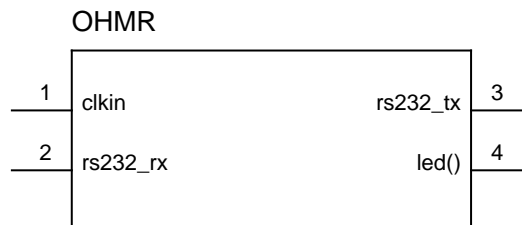


Figure 8.12: Schematic View of OHMR device

8.8 The Completed OHMR Design

A schematic view of OHMR is given in Figure 8.12. Note that OHMR requires very little in terms of external circuitry. Virtually all interaction with the device occurs via the serial port. As such, OHMR may be implemented on a wide variety of Xilinx FPGA devices and development boards.

Currently, our implementation of OHMR is realized onto the Xilinx Spartan-3A DSP XtremeDSP Starter Platform. This development board incorporates the following features:

- Xilinx Spartan-3A DSP 1800 FPGA,
- 125 MHz master clock,²
- 64 Mbit Serial Peripheral Interface (SPI) configuration/storage Flash RAM,
- RS-232 serial port, and
- 8 LEDs.

²This clock is run through an on-board Digital Clock Manager (DCM). The actual clock signal presented to the OHMR design operates at 62.5 MHz.

Chapter 9

Sieve Computing for the OHMR Toolkit (SCOT)

Beware of programmers carrying screwdrivers.

—Chip Salzenberg

The prototype sieve device developed for use within the OHMR framework—dubbed Sieve Computing for the OHMR Toolkit (SCOT)—is similar in design to many previous sieve devices; *e.g.* UMSU [Pat83], MSSU [Luk95], Bronson and Buell’s Splash-based sieve [BB94], and the proposed Wake/Buell sieve device [WB03]. SCOT is capable of achieving a raw canvass rate of 8×10^9 trials/sec, and consists of the following:

- 31 sieve rings, representing the moduli 8, 9, and the primes from 5 to 127;
- 128-bit solution tap;
- 64-bit residue counter; and
- solution-counting mode, with associated 64-bit solution counter.

In this chapter, we will discuss the SCOT design, including the particular way that it integrates into the OHMR framework. The SCOT prototype is pictured in Figure 9.1.

9.1 Device Parameters

Like the OHMR framework itself, SCOT was designed and implemented in the Verilog Hardware Description Language (HDL). To facilitate the retargeting of designs to a

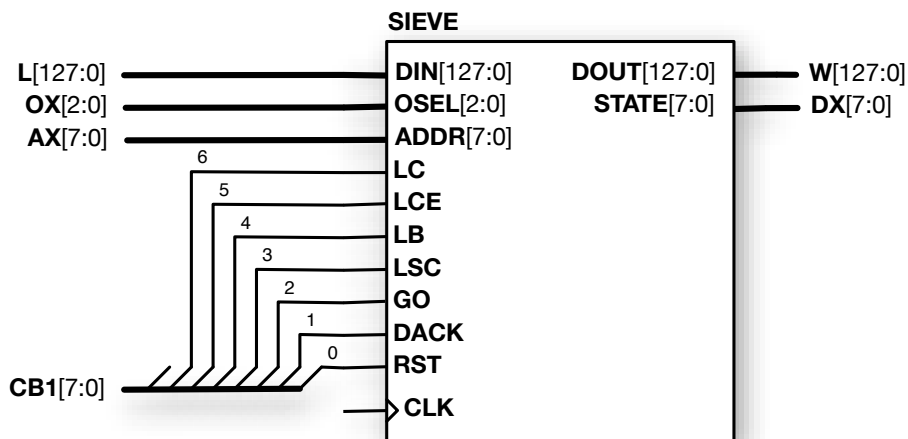


Figure 9.1: OHMR Implementation of the SCOT Design

wide variety of target platforms, Verilog incorporates a powerful *parameter* mechanism, allowing many facets of a hardware device design to be specified or overridden at compile-time. In order to accommodate a wide variety of potential operating requirements, the SCOT design makes heavy use of parameters. Most of the key compile-time parameters are stored within the OHMR framework itself, allowing them to be examined within OHMon via the DP command, as described in Section 8.6. These parameters (and their defaults) are given in Table 9.1.

9.2 Overview of SCOT

As is evidenced by Figure 9.1, the heart of the SCOT design is a one-dimensional sieve device. The sieve device interacts with the OHMR framework via a combination of five registers, seven control lines. Debug and checkpoint capabilities are built into the sieve device, allowing for complete examination of internal registers and state via an output select mechanism.

AX	Parameter	Default	Meaning
0x00	BUILD_NO	(auto)	Auto-incremented Build Number
0x01	BAUDRATE	460800	Baud rate of UART
0x02	CLKMHZ	62.5	Clock Rate after clock division
0x03	PERIODNS	8	Duration of High clock (in ns)
0x04	MASTERMHZ	125	Master clock Rate, before clock division
0x08	LCB	16	Bytes in L
0x09	LXW	4	Width of L Select = $\log_2(LCB)$
0x0A	LW	128	Width of L (bits)
0x0B	CW	64	Counter Width (bits)
0x0C	SCW	64	Solution Counter width (bits)
0x0D	WXW	4	Width of w Select = $\log_2(SW)$
0x0E	WWB	16	Bytes in w
0x0F	WW	128	Width of w (bits)
0x10	MXW	2	Mode register width (bits)
0x11	OXW	3	Output Select width (bits)
0x12	CXW	1	Comb Select width (bits)
0x13	RXW	4	Ring Select width (bits)
0x14	BXW	4	BSEL Bits
0x15	AXW	8	Address Bus Width (Bits)
0x16	SDW	7	Solution detection (offset) width = $\log_2(SD)$
0x17	SD	128	Width of solution detector (bits)

Table 9.1: SCOT Device Parameters

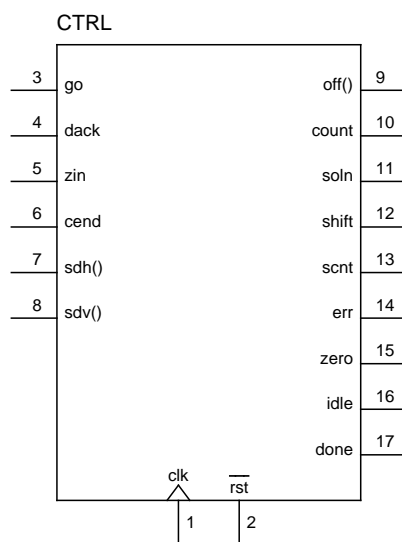


Figure 9.2: Sieve State Machine Controller

Data is loaded into the device using a combination of the 128-bit data register (**L**), and an 8-bit address register (**AX**). Control lines are divided between a global reset, two user-driven control signals, and four data load signals. Output is made available via the 128-bit output register, **W**, and 8-bit state indicator, **DX**. Choice of outputs is controlled by an output select register, **OX**. Finally, a mode register, **MX**, allows the device to be switched between solution count and sieve modes.

9.3 SCOT State Machine

Because all sieve activity is ultimately managed by the Finite State Machine (FSM), depicted in Figure 9.3, we will begin our exposition there. There are eight possible states in the SCOT device, described in Table 9.2. Transitions occur as follows.

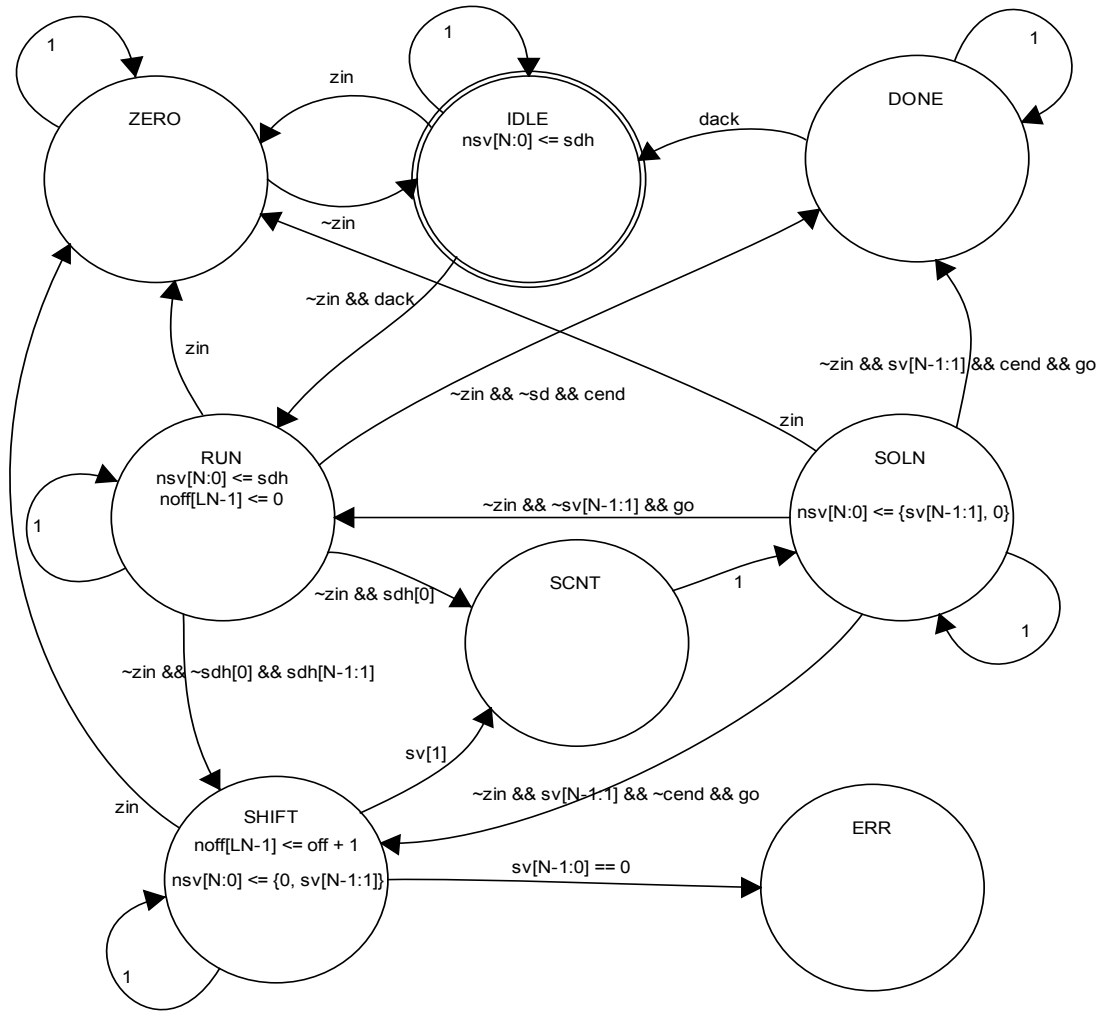


Figure 9.3: Sieve State Machine

State	Meaning
SOLN	Solution found.
RUN	Sieve running.
SHIFT	Solution found, computing offset.
ERR	Error state: illegal transition.
ZERO	No solutions possible.
IDLE	Sieve is idle.
DONE	Counter reached end value.
SCUP	Increment solution counter.

Table 9.2: States in the Sieve FSM

Upon reset or power-up, the device is placed into the initial state: IDLE.

The ZERO state is entered whenever the ZIN signal is asserted, indicating a sieve ring contains all zeros. Since no solutions are possible in this configuration, it is important that the device not enter the RUN state, as it will run forever. Removal of the ZIN signal will cause the device to return the device to the IDLE state.

The device may be moved from IDLE to RUN by sending a DACK signal. This signal should be generated on the host machine.

While in the RUN state, the device examines the solution vector input, SDH[N-1:0]. If any bits are nonzero, the contents of SDH are latched into a save register (SV[N-1:0]), and the device moves either to SOLN (via SCNT), or SHIFT to decode the solution. If no solution is pending, the CEND will send the device into a DONE state; otherwise, execution in the RUN state will continue.

The sieve device examines solutions w bits (usually 128) at a time. The purpose of the SHIFT state is to translate a bit pattern into its corresponding offset value OFF. Because the sieve solution detector width, w , is guaranteed to be a power of two, this offset value simply forms the lowest LN bits of the solution. Once a solution is present in the lowest bit of SV[0], the device moves to the SOLN state, via SCNT.

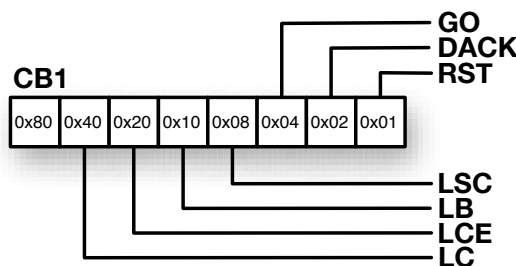


Figure 9.4: Control Bank 1 Mappings in SCOT

Note that since the shift state schedules the contents of SV to be right-shifted on the next clock edge, it is actually the second-least bit, SV[1] that is tested to determine whether to shift to SOLN.

SCNT is used to increment the solution counter. The device passes immediately to the SOLN state afterwards. The device waits in SOLN until a GO signal is received—either automatically (when in *solution-count* mode), or via acknowledgement from the host machine once the pending solution has been recorded.

9.4 Control Signals and Inputs

SCOT makes use of seven control signals, described in Table 9.3. These control signals can be categorized into three distinct groups: *load lines*, which instruct the sieve unit to load user data into the sieve device, *user signals*, which control sieve operation, and a global asynchronous reset, for returning the device to a known configuration. These control signals are mapped to OHMR’s control bank 1, as per Figure 9.4.

To facilitate interactive use and reduce operational bandwidth, OHMon was extended with a series mnemonic equivalents to the standard command bank commands

Control Line	Command	Mnemonic	Action
RST	U01	Z	Reset sieve device.
DACK	U02	A	Acknowledge done / move from idle.
GO	U04	G	Acknowledge solution. Continue sieving.
LSC	U08	Ls	Load Solution Counter from L
LB	U10	Lb ¹	Load Bank number AX from L
LCE	U20	Le	Load Counter End value from L
LC	U40	Lc	Load Counter value from L

¹ This command also increments the value stored in **AX**. See Section 9.6.2 for a more detailed description of this command.

Table 9.3: Control Lines in SCOT

(Uxx); *e.g.* G for GO, Lc for *load counter* (LC), and so on. These mnemonic shortcuts are detailed in Table 9.3.

In addition to these control signals, two registers are dedicated to getting data into the sieve device:

- **DIN**: data load register—used primarily for data load operations; and
- **ADDR**: address register—used for data load and debug/checkpoint operations.

Before describing the details of the data load mechanism, it is necessary to delve into the architectural details of the sieve implementation itself, notably, shift registers, combs, and counters. Since counters are the simplest component to describe, we will begin our exposition there.

9.5 Counter Design

There are two different counter types used within the SCOT design: the *residue counter* which counts in step with the cyclic shift registers, indicating the value of the current solution candidate, and the *solution counter* which is incremented whenever

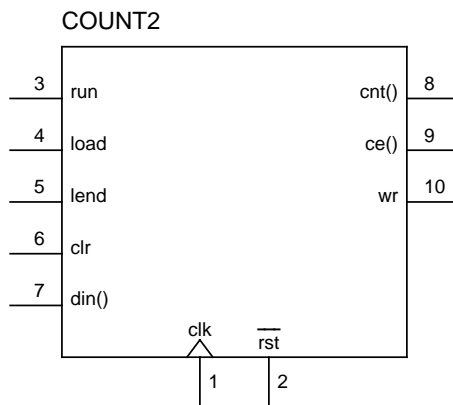


Figure 9.5: Counter Design in SCOT

a solution candidate is encountered. Like all components of SCOT, counter design is parametrized, taking a parameter W to indicate the desired counter width. By default, $W = 64$.

As shown in Figure 9.5, residue and solution counters make use of the following input and output lines:

- $DIN[W-1:0]$: data-in bus, for data loads;
- $LOAD$: when asserted, sets current counter contents to value at DIN ;
- RUN : count up when asserted—hold value otherwise;
- CLR : reset counter to zero;
- $CNT[W-1:0]$: (*output*) current counter contents.

In addition to pure counting functionality, the residue counter makes use of one additional feature not used by the solution counter—a loadable wrap register. When the counter contents reach the wrap value, WR is asserted. This output allows the sieve to stop when the desired end value has been reached. Implementing this feature requires an additional input and two additional outputs not used by the solution

counter:

- LEND: when asserted, sets wrap register contents to value at DIN;
- CE[W-1:0]: (*output*) wrap register contents;
- WR: (*output*) output wrap—asserted when counter is at end value.

9.5.1 Loading Counter Data using OHMon

As with most of the SCOT design, the data load register, **L**, is used for all data loading operations, including setting the residue counter, solution counter, and wrap register contents. Once data is present in **L**, the appropriate control line must be asserted, as shown in Figure 9.4. Though the regular OHMon facilities—*i.e.* LL and Uxx—can be used to load counter data, the OHMon command language was extended to include several additional commands that combine these operations, as follows:

- LChh...hh: load counter immediately—equivalent to LLhh...hh followed by U40;
- LEhh...hh: load counter end (wrap register) immediately—equivalent to LLhh...hh followed by U20;
- LShh...hh: load solution counter immediately—equivalent to LLhh...hh followed by U08.

Here *hh...hh* indicates the hexadecimal value to be loaded into the **L** register.

9.6 Shift Register Architecture

At the heart of any hardware-based sieve lies a means of cyclically shifting residues. In the case of our sieve design, acceptable residues are stored and shifted via cyclic shift-by-*K* registers—**CSRKs**—shown in Figure 9.6. Like all components in the SCOT architecture, shift register design was parametrized, in this case, by modulus (*M*),

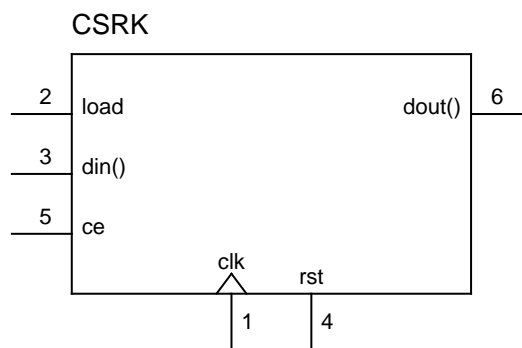


Figure 9.6: CSRK: Cyclic Shift-by- K Register (Right)

and solution detector width (W). By default, $W = 128$. Since virtually all shift register moduli are smaller than the solution detector width, we do not normally construct a shift-by- W register. Instead, we define the quantity K such that $K \equiv W \pmod{M}$, and shift by this value instead.

Like most of the components of the hardware sieve, cyclic shift registers make use of an asynchronous reset (RST). Unlike most other components, however, this reset signal replaces the contents of the shift register with all ones, not zeros. This is because the shift register outputs are ANDed together by the solution detection mechanism. Filling with ones, therefore, effectively disables the device.

The remaining I/O lines are as follows:

- CE: Chip enable. If asserted, the contents of the shift register are cyclically-shifted-right by K bits each clock cycle;
- DIN[M-1:0]: Data to be loaded. When the LOAD signal is asserted, data present on this bus is copied into the shift register;
- LOAD: This signal causes the data at DIN to be copied into the shift register.

This signal has priority over CE;

- DOUT[M-1:0]: (*output*) Shift register contents. Note that the entire contents of the sieve register are made available, ensuring that no shifting is required to recover ring contents during the checkpoint/verification process.

9.6.1 Additional Sieve Register Requirements

Though the **CSRK** design is functional, to make efficient use of it in our sieve design, several additional operational requirements were identified.

- Output emerging from the shift register should be exactly W bits, where W is the width of the solution detection mechanism. Smaller shift registers should have their outputs expanded to this length. Larger **CSRKs** should have their output truncated to the low-order W bits.
- **CSRKs** should be bank-loadable; *i.e.* several **CSRKs** with small modulus should be loadable from the contents of a single 128-bit register. This helps to minimize I/O bandwidth when loading multiple sieve rings from the data register **L**.
- **CSRKs** should have the ability to be disabled. In this configuration, their actual contents should not affect the search for solutions by the remaining **CSRKs**.
- **CSRKs** should indicate when no solutions are possible (all bits zero), or when a solution is present. These signals will serve as two of the primary inputs to the sieve state machine described in Section 9.3.

Of these features, bank loading requires additional explanation.

Modulus	Bank ID
59, 37	0
61, 67	1
71, 41, 7	2
73, 53	3
79, 47	4
83, 43	5
89, 31, 8	6
97, 29	7
101, 23	8
103, 5, 9, 11	9
107, 17	10
109, 19	11
113, 13	12
127	13

Table 9.4: Arranging Sieve Moduli in Banks

9.6.2 Loading Shift Register Data via Banks

The bank-loadable design of the **OCSR** allows for efficient use of registers in OHMR. By packing the sieve moduli into banks of 128-bits (or less)—as indicated in Table 9.4—and using the 8-bit ADDR (**AX**) input to indicate which bank ID is to be loaded, only two registers and minimal logic are required to implement data loads.

Since multiple rings are typically loaded in succession, OHMon was extended to include two explicit bank loading commands.

- **Lb**: Assert load signal (**U10**), loading ring data from the **L** register. Bank ID is taken from **AX**. Increment **AX** by 1 when done.
- **LBhh...hh**: Immediate data load. Equivalent to **LLhh...hh** followed by **Lb**.

These new instructions permit the loading of successive banks without added overhead of reloading the address register each time.

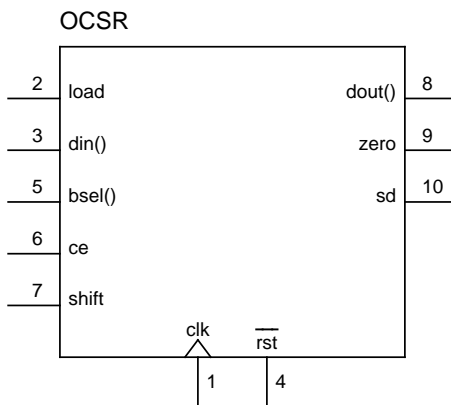


Figure 9.7: OCSR: Operational Cyclic Shift Register

9.6.3 Operational Cyclic Shift Register

Rather than implement additional features directly in the **CSRK** device, an Operational Cyclic Shift Register—**OCSR**—module was designed which makes use of the **CSRK** device internally. This allows the Verilog compiler the opportunity to optimize the **CSRK** implementation. This optimization is likely, since cyclic shift registers are a common library component in many hardware design environments.

The **OCSR** module, pictured in Figure 9.7, employs the following I/O signals.

- **RST**: Asynchronous reset. When asserted, contents of shift register are replaced with all 1s. This signal has the highest priority.
- **CE**: Chip Enable. If asserted, then **LOAD** and **SHIFT** signals operate as expected. If deasserted, its outputs are locked to values that have no effect on the remaining **CSRKs** in the sieve; *i.e.* **DOUT** becomes all 1s, **SD** remains a 1, and **ZERO** remains at zero, regardless of the current contents of the shift register.
- **LOAD**: This signal causes the data at **DIN** to be copied (and replicated) through-

out the shift register if and only if the Bank data at BSEL matches the BANK parameter specified at compile time. Priority of LOAD and SHIFT is established by the underlying **CSRK**.

- SHIFT: Shift the register contents by W each clock cycle. Priority of LOAD and SHIFT is established by the underlying **CSRK**.
- DIN[M-1:0]: Data to be loaded.
- BSEL[BW-1:0]: Bank ID. If this pattern matches the BANK parameter specified at compile time, then the LOAD signal will function as expected. Otherwise, data load is disabled.
- DOUT[W-1:0]: (*output*) Low-order W bits of the replicated shift register.
- ZERO: (*output*) Register contents are all zero; *i.e.* no solutions are possible.
- SD: (*output*) Solution Detected. Low order W -bits of register contain at least one nonzero bit.

9.7 Address Bus Decoding

In addition to storing a Bank ID, SCOT also makes use of the ADDR input—mapped to OHMR’s **AX**—for parameter and checkpoint data selection. Address bus decoding is summarized as follows:

- Parameter retrieval: **AX** must be loaded with the parameter ID when retrieving device parameters via **DP**—see Section 9.1 for a list of these IDs;
- Loading ring data: when issuing a bank load command (**Lb** or **LB**), **AX** holds the bank ID value;
- Comb/Ring Select for checkpoint data: This topic is the subject of the next section.

OSEL	ADDR	Output to DOUT
0x0	-	Residue Counter
0x1	Ring/Comb Address	Comb Output
0x2	Ring/Comb Address	Ring Contents
0x3	-	Solution Array (SDH)
0x4	-	Counter End
0x5	-	Solution Counter
0x6	-	State Vector
0x7	Ring/Comb Address	Ring Parameters

Table 9.5: Sieve Output Select

9.8 Sieve Output

A basic sieve device, such as the one described in Section 2.4, provides only a single (bus) output: the residue counter; however, to build a truly reliable and flexible sieve device we require a better view of sieve internals—specifically, the ability to read and verify (checkpoint) the current sieve state.

To achieve this goal SCOT provides the following I/O lines.

- **OSEL**: Output select register. Contents indicate the device information that will appear at DOUT. Used in conjunction with ADDR for certain outputs.
- **DOUT**: Sieve device output: Controlled by contents of **OSEL**.
- **STATE**: State Register: Current device state, for output to LEDs.

Outputs presented on DOUT are summarized in Table 9.5. They fall into three main categories:

- counter outputs: residue counter, solution counter, wrap register;
- state vector;
- checkpoint/debug information: comb output, ring contents, ring parameters, solution array.

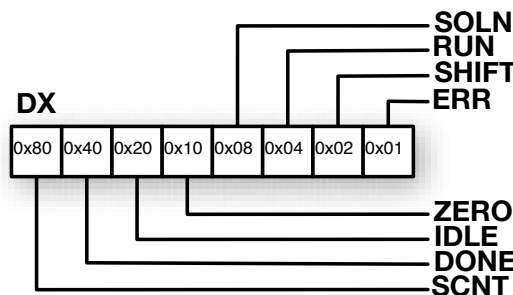


Figure 9.8: State Decoding for **DX** register

9.8.1 State Vector

The STATE output, mapped in SCOT to OHMR's **DX** register, is used to display the current sieve state on SCOT's onboard LEDs. Though this mapping is largely superfluous,¹ LED output is useful for debugging purposes, and hence the device state was exposed as a dedicated output.

The contents of the state register are decoded as per Figure 9.8. As with other SCOT features, the OHMon command was extended in order to facilitate the use of the state facility:

DS Display contents of **DX** and enable automatic state updates;

LXD Manually load **DX** register, disabling automatic state updates.

Normally, when the sieve device is in operation, sieve state is automatically latched into **DX** (and hence, to the LEDs) as part of the idle loop operation. Manually loading the **DX** register (via the **LXD** instruction) disables automatic updates of the **DX** register. Issuing **DS** will output the current contents of the state register to the serial line, re-enabling automatic update of state information if necessary.

¹As shown in Table 9.5, the state vector is available as one of the outputs of DOUT.

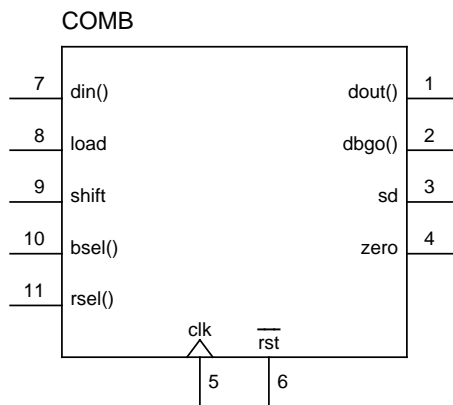


Figure 9.9: COMB: Comb Unit

9.9 Checkpoint/Debug Information

Verifying—or checkpointing—the sieve internals is an important component of a reliable sieve device. Before we can describe the checkpoint/verification process, it is necessary to reveal one additional detail of how cyclic shift registers are implemented within the sieve device: the comb.

9.9.1 Comb Units

The sieve design implemented in SCOT differs from previous shift-register based designs in that cyclic shift registers (rings) are arranged into computational units known as *combs*. A comb operates as a basic sieve device in its own right, and is shown in Figure 9.9. Combs consists of up to 16 **OCSRs**. The output of the comb—DOUT[W-1:0]—is the W -bit logical AND of each of these **OCSR** outputs. A comb also produces combined Solution Detect (SD) and Zero Detect (ZERO) outputs, indicating whether a solution is present on the comb’s DOUT, or whether no solution is possible due to

one of its **OCSRs** containing a zero ring.

The comb unit also presents a second output—DBGO—dedicated to the checkpoint/validation process. This debugging output presents the contents of a specified **OCSR**. Output appearing on DBGO is controlled by the ring select input, RSEL. The complete set of I/O lines for a comb is as follows.

- RST_N: Asynchronous Reset, active low.
- DIN[DIW-1:0]: Data In, for data loads
- LOAD: Load Data, using Bank ID indicated at BSEL
- SHIFT: Perform Shift operation on clock edge
- BSEL[BW-1:0]: Bank Select. Used with LOAD top
- RSEL[KW-1:0]: Ring Address. Determines output of appearing at DBGO
- DOUT[W-1:0]: Data Out (Solution Vector)
- DBGO[W-1:0]: Debug (checkpoint) Data out. Ring data, as determined by contents of RSEL.
- SD: (*output*) Solution Detected. Output is high if all rings contain a solution.
- ZERO: (*output*) No Solution Possible. Output is high if one ring indicates a zero present.

It should be observed that every cyclic shift register in a SCOT-based sieve is addressable in two different ways: via a comb and ring number—indicating its position within a specified comb—or via a modulus, bank number and bit offset. In fact, the sieve device has the capability to map between these representations, as will now be discussed.

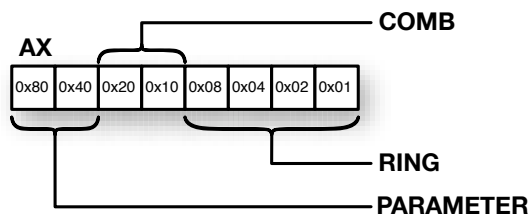


Figure 9.10: Address Register Decoding for ROM Use

9.10 Sieve ROM

In order for a host machine to load data into the sieve, the precise mapping of a ring modulus to a bank must be known; *i.e.* the bank in which it resides, and the bit offset within that bank. To make discovery of this information possible, the sieve device contains a small ROM, mapping ring numbers and comb IDs to modulus, bank, and offset IDs. Thus the host machine may enumerate the attached rings, learning enough information to construct the appropriate data load bit-strings.

Parameters stored in the sieve ROM are accessible by setting the sieve output select—OSEL—to 0x7, and using the ADDR register to indicate the desired ring/comb combination. The ADDR register decoding is illustrated in Figure 9.10, and is given by:

- bits 7–6: parameter type: 00 = ring modulus, 01 = offset, 10 = ring id;
- bits 5–4: comb number;
- bits 3–0: ring number.

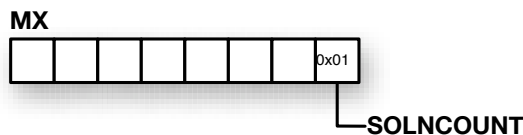


Figure 9.11: Mode Register Use

9.11 Solution Count Mode

We have now accounted for all but one of the OHMR registers used in the SCOT design. The final register—called the mode register—is mapped to **MX**, and is intended to control runtime configuration information. This register is shown in Figure 9.11.

At present, only the least significant bit of the mode register is used. When this mode bit is set, the sieve enters *solution count mode*. In this mode, solutions emerging from the sieve are not sent to the serial port. Instead they are simply accumulated by the solution counter. Solution count mode is useful primarily as a test environment, to benchmark sieve performance, and to ensure a sieve problem is correctly configured—producing the expected number of solutions for a given interval.

9.12 Summary

This chapter described the design of a reliable one-dimensional sieve prototype—SCOT—and its embedding into the OHMR framework described in Chapter 8. In the next chapter, we will analyze the performance of this device. In Chapter 11 we will indicate several ways the prototype can be enhanced in order to improve performance for the two-dimensional sieve problem, and eventual integration in to the CASSIE framework.

Chapter 10

Results and Analysis

Sit down before fact as a little child, be prepared to give up every preconceived notion, follow humbly wherever and to whatever abysses nature leads, or you shall learn nothing.

—Thomas Huxley

In this chapter, we have three main goals:

- Evaluate the performance of the two-dimensional sieve, in both its software and prototype hardware implementations.
- Use the CASSIE framework to produce a table of Eisenstein pseudocubes.
- Analyze the efficiency the Eisenstein Pseudocube Primality Proving (EPPP) method. In particular, we evaluate its effectiveness versus the pseudosquare and pseudocube methods described in Chapter 3.

10.1 Performance Metrics

In previous analyses of sieve devices—*e.g.* [Woo04], [WB03],[Ste89], [Luk95]—the primary metric used to gauge sieve performance was the canvass rate of the sieve device; *i.e.* the number of solution candidates evaluated per second after all optimizations had been applied. This was an effective choice, as these sieve devices were essentially unaffected by choice of problem parameters such as the number of acceptable

residues, or the bounds of the sieve problem.¹

With more recent optimization techniques—notably, Bernstein’s doubly-focusing technique—canvass rates are much more dependent on a particular choice of problem parameters. In this thesis, we propose a variant of this metric in order to permit a more rational comparison between sieve devices.

Define *raw canvass rate* as the rate of output of a one-dimensional sieve device (or wheel) unit before any optimizations are applied; *e.g.* normalization or doubly-focusing. It should be clear that any gains achieved in raw canvass rate will translate into improved performance once all additional optimizations are applied.

In this thesis, we also propose an additional metric: cost. Until recently, sieve devices have generally been constructed out of custom hardware components or Application Specific Integrated Circuits (ASICs). Adding additional computing nodes to these devices has generally been limited by the up-front production cost of custom Integrated Circuits (ICs), or the difficulty inherent in assembling such a device from diverse components. As such, adding additional nodes one at a time was generally not a cost-effective option.

Recent approaches, however, have involved either software running on general-purpose computers—*e.g.* [Woo04] [Ber04b] [Sor06]—or off-the-shelf reprogrammable hardware—*e.g.* [WB03] and SCOT. Adding additional computing nodes to these devices is relatively straightforward, with costs that scale almost linearly as nodes are added. When comparing different technologies for solving the sieve problem, it is worth considering the relative cost of constructing the device.

¹With virtually any sieve problem it is possible for the sieve performance to be constrained by some aspect of the problem parameters—filtering is a good example. For the discussion of this section we assume that this is not the case; *i.e.* we assume the problem is *sieve-bound*.

10.2 Evaluating Software Sieve Performance

In a dedicated device, such as SCOT, raw canvass rates of a sieve device are tied closely to the physical parameters of the underlying hardware; *e.g.* clock rate, and width of the solution detection mechanism. In software, however, performance can be affected by a number of factors that can vary wildly from run-to-run, including interrupt processing, virtual memory layout, cache misses, network activity, and filesystem latency. Even repeatedly running the same sieve problem on the same physical computing node leads to a surprising spectrum of canvass rates.²

Despite these difficulties, two key parameters *are* under the users' control, and these parameter choices have a dramatic effect on the speed of the sieve. The most significant performance effect in a software sieve is controlled by:

1. the number of moduli to be considered, and
2. the number, type, and order of filters applied to the output emerging from the sieve.

We will now examine these parameters in more detail.

10.2.1 Number of Moduli

The main drawback of implementing an exclusion sieve on a general purpose computer is that the main operation of the sieve—forming the logical AND of t inputs, where t is the number of moduli in the sieve—must be processed serially. Each additional modulus that must be compared adds a potential iteration to the main sieve loop, and hence, slows down sieve operation.

²We deliberately ignore a discussion of background tasks running on the compute node. For hopefully obvious reasons, background tasks should be kept to a minimum when measuring sieve performance.

There are several ways to speed up this multi-way AND operation. Of these, a key architecture-independent technique is *early-abort*. Sieve performance is greatly improved if the repeated-AND operation is aborted as soon as the solution register contains all zeros, skipping redundant comparison operations.

Other techniques for improving sieve speed tend to be more tightly-coupled with the underlying machine architecture. For instance, most modern processors have SIMD assembly-language extensions available that permit multi-way, or extra-wide AND operations. Of course, making use of these instructions requires coding the main sieve loop in assembly language, or similar non-portable constructs. This, and other SIMD-based techniques are discussed in Future Work, §11.3.

10.2.2 Number, Type, and Order of Filters

In most sieve problems of interest, additional criteria are applied to solutions emerging from the exclusion sieve process. In the case of the Eisenstein pseudocubes, solution candidates $a + b\omega$ must be filtered to eliminate perfect cubes, and solutions for which $\gcd(a, b) > 1$. Operations of this type are called *filter* operations, and make use of specialized filter functions—exclusion filters—within the sieve framework.

The ordering of exclusion filters is extremely important. When too many solutions are emerging from the sieve subsystem, these filtering operations can become the dominant computational component of a sieve problem. When the maximum sieve rate of a problem is largely determined by the speed of the filter routines in use, the sieve problem is said to be *filter-bound*. Conversely, when the canvass rate is limited by the sieving step itself, the problem is said to be *sieve-bound*.

By ordering filters from least to most computationally intensive, early rejection of

solution candidates is possible. Just as in Section 10.2.1, an early-abort philosophy saves computation time by terminating sequential operations as early in the execution chain as possible.

10.3 CASSIE Performance

With these considerations in mind, we set out to evaluate the two-dimensional sieve performance of CASSIE. Our goals were fourfold: first, to establish solution counts for the Eisenstein pseudocube problem using a variety of problem parameters, so that correctness of optimized or hardware-based sieve results could be verified using the solution counting method; second, to verify our predictions on how various parameter choices, such as norm bound, number of moduli, and solution filtering affect the sieve outputs; third, to establish the ideal number of sieve moduli for a longer run, maintaining a high canvass rate while ensuring the sieve problem is not filter-bound; finally, to establish software canvass rates for the two-dimensional sieve problem.

10.3.1 Solution Counts and Parameter Choices

In Section 7.2.2, we predicted that the number of solution candidates, and hence the runtime of the sieve, would increase linearly with the sieve bound. In Table 10.1, we summarize the number of solutions candidates canvassed, and number of solutions obtained for a particular instance of the Eisenstein pseudocube problem: μ_{59} . Using the congruence criteria developed in Chapter 6, we searched for all Eisenstein pseudocubes $\alpha = a + b\omega$ with $N(\alpha) < H$, counting both the total solutions canvassed, and the number of solutions found. From these results, it seems clear that the argument

H	No. Solutions	No. Canvassed	$\pi H/\sqrt{3}$
2^{43}	3826	1595 43449 65255	1595 43479 31426
2^{44}	7918	3190 86916 67228	3190 86958 62852
2^{45}	16362	6381 73857 94456	6381 73917 25704
2^{46}	33582	12763 47750 63277	12763 47834 51407
2^{47}	68028	25526 95550 40673	25526 95669 02815
2^{48}	1 38012	51053 91170 30930	51053 91338 05629
2^{50}	5 67582	2 04215 65016 61904	2 04215 65352 22517

Table 10.1: Candidate and Solution Counts by Sieve Bound

of Section 7.2.2 holds; *i.e.* the problem grows linearly in the bound H .

10.3.2 Adding Exclusion Moduli

Next, we fixed the bound of the problem to search for solutions $N(\alpha) \leq 2^{45}$, varying the number of moduli used from 11 to 30. Two exclusion filters were then applied to the sieve output: `relatively_prime` and `perfect_cube`, the former rejecting solution candidates with $\gcd(a, b) > 1$, and the latter eliminating perfect cubes by the method of Section 6.1. In this problem instance, no normalization was applied, so the output filter was responsible only for logging the solution candidates. Table 10.2 indicates the number of solutions emerging from the sieve for varying moduli. The resulting raw canvass rates are shown in Figure 10.1.

From this figure, it is clear that when searching for Eisenstein pseudocubes with less than 16 exclusion moduli, our problem is filter-bound. As additional moduli are added, however, the problem instance becomes sieve-bound, and the computational overhead of adding additional moduli is seen. Thus, peak (raw) sieve rates are obtained by running the sieve problem with 16–18 rings for a variety of problem sizes.

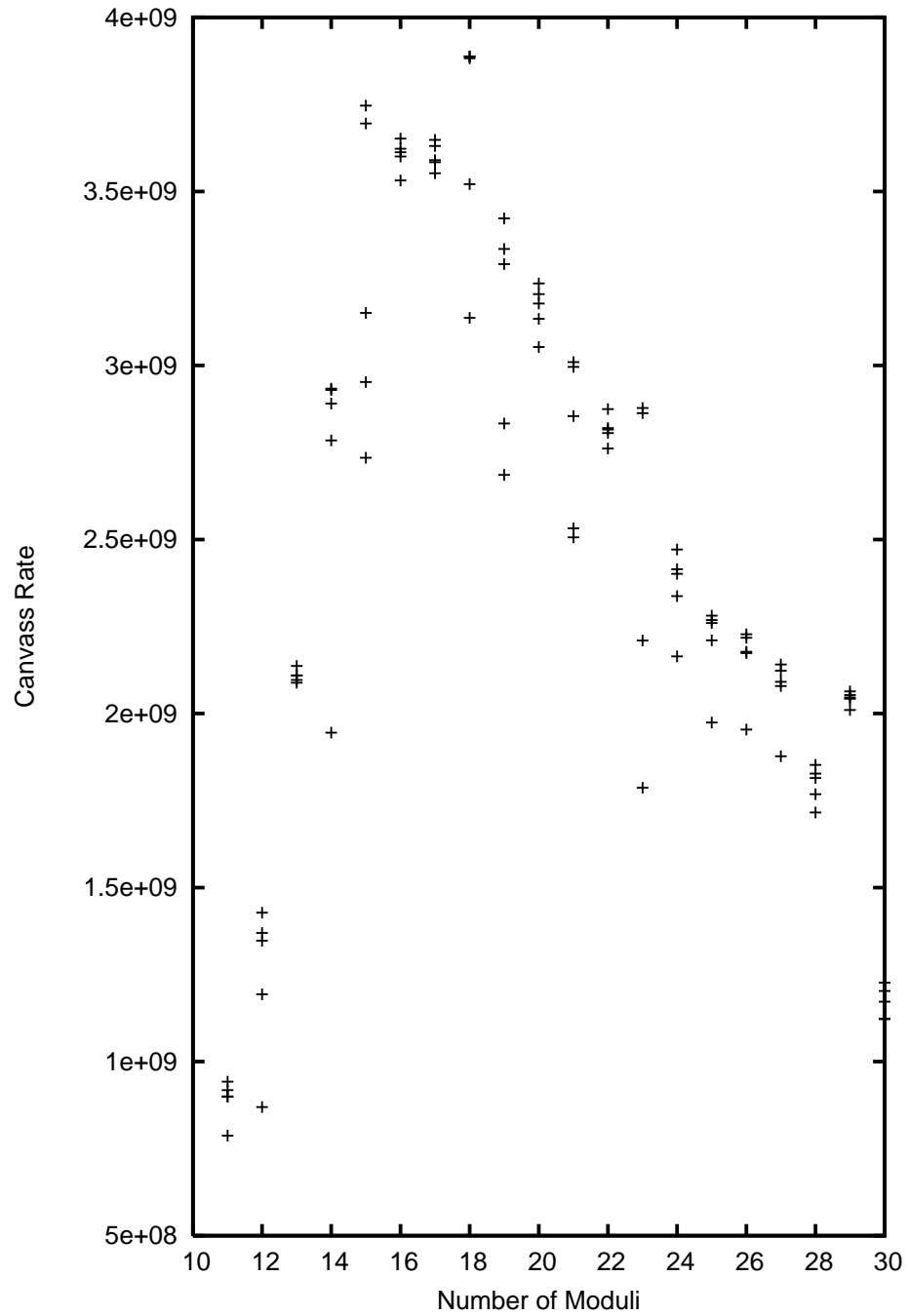


Figure 10.1: Raw Canvass Rates vs. Number of Moduli

Primes to	k	Solutions
37	11	46 26635
41	12	15 28896
43	13	4 79015
47	14	1 56362
53	15	50639
59	16	16362
61	17	4960
67	18	1499
71	19	486
73	20	131
79	21	37
83	22	10
89	23	3
97	24	1
101+	25+	0

Table 10.2: Number of Solutions with Varying Moduli

10.3.3 Raw Canvass Rate

Though useful for establishing a known-good solution set, this completely unnormalized sieve problem produces artificially high raw canvass rates. This is because the sieve problem contains 18 as an exclusion modulus, for which the acceptable residues are $(5, 0)$, $(11, 0)$, and $(17, 0)$. Thus, only every 18th choice for b will lead to a one-dimensional sieve problem for which the sieve ring for $m = 18$ is nonzero. Recall from Algorithms 7.1 and 7.4 that sieve instances containing a ring of all zeros will be skipped by the sieve, as no solution is possible. This should result in an 18-fold increase in raw canvass rate over a problem where the modulus 18 is omitted.

Figure 10.2 shows the result of removing certain small moduli—those that would normally be used as normalization moduli—on raw canvass rates; *i.e.* 18, 5, and 7. No filtering was performed on the outputs. The resulting canvass rates were, as expected,

approximately 18 times slower than those of the previous run. Furthermore, with no filtering to contend with, the overhead of adding moduli to a software sieve is quite clear.

10.4 SCOT Performance

Compared to the software case, computing raw canvass rates for hardware devices such as SCOT is almost trivial. In hardware, sieve rates tend to depend only on the clock rate of the sieve device, and the width of the solution detection mechanism.

SCOT, for instance, has a 128-bit solution detection mechanism, and runs at a clock rate of 62.5 MHz. This means the hardware is capable of examining up to 8×10^9 solution candidates per second. Furthermore, so long as chip area is available, this sieve rate is effectively independent of the number of moduli used. For instance, the current one-dimensional sieve configuration, uses two combs connected serially to incorporate up to 32 exclusion moduli.

10.4.1 Deviation from Maximum Sieve Rate

It should be noted that, due to the current design of the SCOT state machine (described in Section 9.3), sieving stops when one (or more) solutions are encountered in the 128-bit solution window. It may take up to 128 clock cycles to compute the resulting solution offset. If a sieve problem has an unusually high density of solutions, this will lower the sieve rate from the theoretical rate of 8×10^9 trials/second. In practice, our sieve problems are nowhere near dense enough for this effect to slow us down; however a solution for mitigating this theoretical bottleneck is discussed in

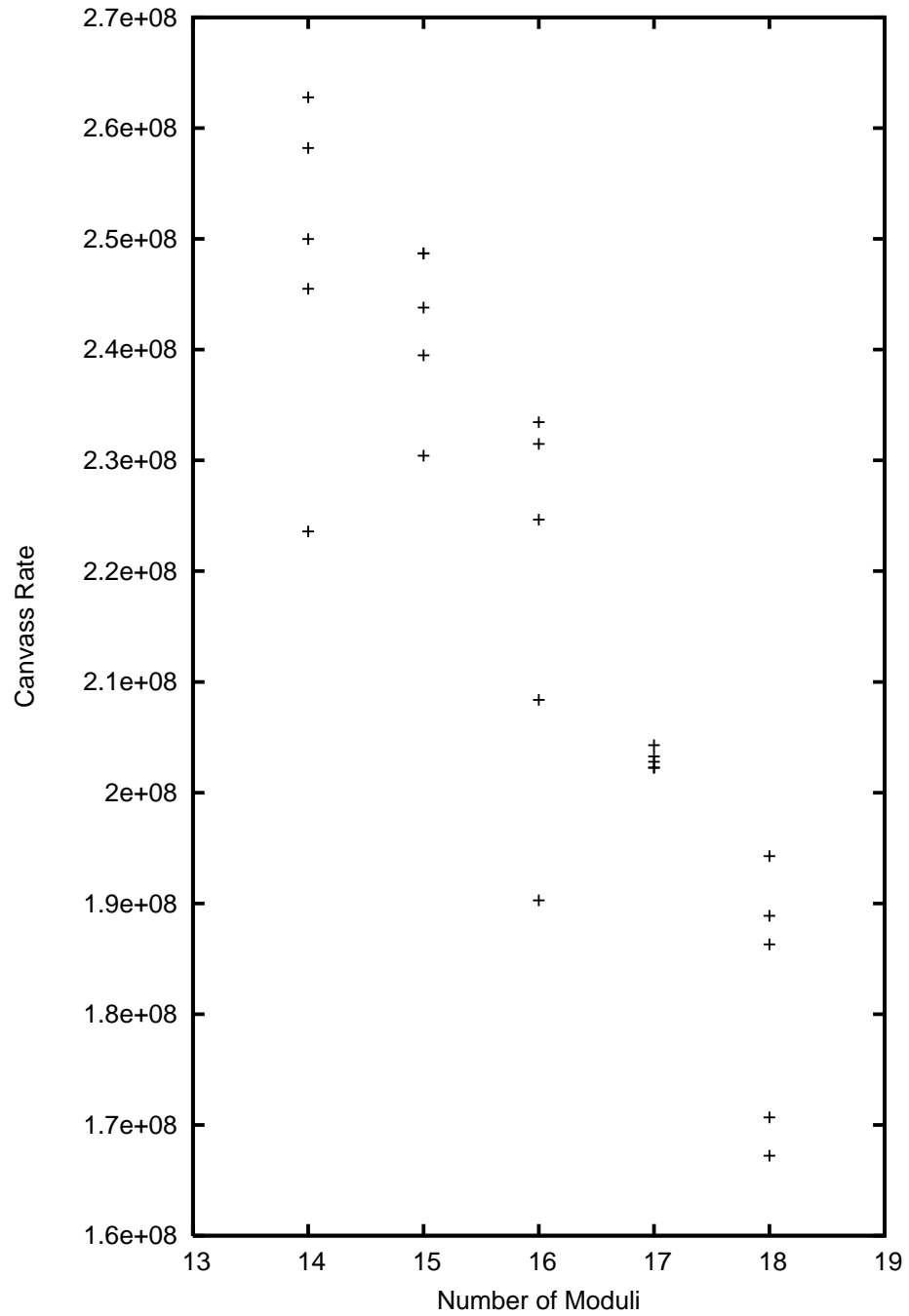


Figure 10.2: Raw Canvass Rates, Normalized Problem

Chapter 11.

Of course, as with any F+V sieve design, device throughput may still be limited by bottlenecks in the host machine. As in the case with the software sieve, it is still quite possible for the sieve to become filter-bound if solutions are emerging too quickly from SCOT. Remedies for this are identical to those discussed in Section 10.2.2; *i.e.* careful ordering of filters.

In addition to the filtering bottleneck, however, the OHMR architecture introduces another potential bottleneck in the sieve process: the communications path with the host machine.

10.4.2 Communication Bandwidth in SCOT

Though designed to operate at speeds up to 921600 bits per second, the serial interface provided by the OHMR framework is currently limited to 460800 bits per second. This is due to a limitation in the serial processing capabilities of Tcl—the scripting language currently used on the host machine to interact with the OHMR framework.

In a traditional one-dimensional sieve problem, this communication path is used primarily for obtaining output from the sieve device. Overhead associated with problem loading is tiny, and occurs only once in the problem run. Thus a communication path of 460800 bps can handle at least 3388 64-bit outputs/sec.³

In the sieve design in Chapter 7, a two-dimensional sieve problem is partitioned into a series of one-dimensional sieve jobs. Thus the communication to a dedicated sieve device must accommodate both solutions emerging from the sieve, and the

³Solutions emerge from the sieve device as hexadecimal strings, separated by carriage return characters. Thus, a single 64-bit output requires up to 17 octets to transmit. Of course, leading zeros are suppressed, so this figure is a lower bound.

overhead required to load each of the one-dimensional sieve problems. In practice, this latter requirement represents a bottleneck in the current SCOT design.

There are several potential remedies for this issue. One trivial modification would be to introduce a second serial connection, dedicated to sieve output. This (effectively) doubles the communication bandwidth, allowing data loads to proceed independently of solution generation. A more complete solution would make use of the various high-speed serial communications options available for the Xilinx architecture; *e.g.* the 3.125 Gbps serializer/deserializer daughterboard described in [AVN]. These and other potential bandwidth remedies are described in Chapter 11.

10.4.3 Cost of Hardware Sieving

In Section 10.1 we pointed out that cost may serve as an enlightening metric for comparing sieve implementations. Consider, for example, the cost of constructing the University of Calgary’s ACL cluster. This cluster, consisting of approximately 280 compute nodes, cost on the order of \$750,000 to build. With each node capable of sieving at a rate of 2.25×10^8 trials/second, this works out to approximately \$11900 per billion trials/sec.

A much more recent implementation, *e.g.* Sorenson’s recent computation [Sor10], used Butler’s *Big Dawg* supercomputer. This machine consists of 384 compute nodes, and was constructed for a reported \$350,000.⁴ This works out to approximately \$285 per billion trials/sec—41 times more cost effective than the ACL.

Contrast these devices with the hardware sieve. The Spartan 3A DSP evaluation board used to implement SCOT cost \$300, and can run 3 sieve instances. This works

⁴Personal communication.

out to a meagre \$12 per billion trials/sec. Clearly, FPGA-based sieve devices are vastly more cost effective than general-purpose computers for solving the congruential sieve problem.

10.5 Eisenstein Pseudocube Results

With the prototype hardware and software sieves operational, we now turn to the problem that motivated this work: computing Eisenstein pseudocubes.

Using the congruence criteria developed in Chapter 6, we searched for all Eisenstein pseudocubes $\alpha = a + b\omega$ with $N(\alpha) < H$ for various small choices of H , starting with the exclusion modulus 18. As Eisenstein pseudocubes were found, additional moduli were added to the sieve, and the bound slowly increased to compute larger and larger Eisenstein pseudocubes. No normalization was used for this phase of sieving. In all, pseudocubes to μ_{109} were found using this method, the final run requiring a bound of $H = 2^{50}$. These pseudocube results are summarized in Table 10.3.

The next run involved a bound of $H = 2^{64}$, and produced solutions to μ_{157} . This was normalized over the acceptable residues for 18, 5, 7, 11—11520 residue classes in all. Though SCOT is by far the faster device, the vast amount of parallelism available on the ACL cluster—260 nodes vs. SCOT’s 3—made it more practical to run the entire job in software on the ACL. We expect, however, that future runs will be partitioned to use both the ACL nodes, and SCOT—especially once the bottleneck issues described in Section 10.4.2 are addressed.

Once partitioned into normalized jobs, each of the 11520 sieve problems required approximately 8000 seconds to complete. In all, approximately 25,600 CPU-hours on

p	$N(\mu_p)$	μ_p
18	247	$11 + 18\omega$
5	643	$29 + 18\omega$
7	5113	$71 + 72\omega$
11	13507	$23 + 126\omega$
13	39199	$227 + 90\omega$
17	1 07803	$-181 + 198\omega$
19	3 60007	$653 + 126\omega$
23	39 04969	$443 + 2160\omega$
29	61 07191	$-1669 + 1170\omega$
31	103 18249	$3617 + 2520\omega$
37	273 33067	$6023 + 3366\omega$
41	991 79467	$4973 + 11466\omega$
43	5329 97833	$-15451 + 11088\omega$
47	22785 22747	$54017 + 17514\omega$
53	27417 02809	$47477 + 56160\omega$
59	1 85007 66499	$66887 + 156510\omega$
61, 67	4 15475 53813	$235061 + 107172\omega$
71	11 94233 48797	$-139813 + 253764\omega$
73	82 46210 13649	$-267733 + 744120\omega$
79, 83	115 18103 60731	$1227419 + 761670\omega$
89	2507 90827 69801	$5052689 + 4961880\omega$
97	3393 26375 28481	$-2127709 + 4462200\omega$
101	9175 67688 29893	$10322861 + 8601732\omega$
103, 107	21408 90619 32079	$3056387 + 15918570\omega$
109	81221 66151 53761	$-27791551 + 1366560\omega$

Table 10.3: Small Eisenstein Pseudocubes

p	$N(\mu_p)$	μ_p
113	10 70670 04348 13749	$109364777 + 13014540\omega$
127	15 84695 56547 47279	$-114717193 + 19952010\omega$
131	21 44850 97583 41459	$160585853 + 126202050\omega$
137	596 03669 06441 31739	$845355437 + 667764090\omega$
139	2127 62708 04110 19739	$-724036477 + 954969030\omega$
149	5736 34194 93471 77659	$696254903 + 2666049750\omega$
151	9708 82344 17235 68077	$2979509543 + 3236384556\omega$
157	14102 28178 31706 25921	$3671532959 + 3833807040\omega$

Table 10.4: Eisenstein Pseudocubes

the ACL were required. These results are summarized in Table 10.4.

10.6 Eisenstein Pseudocube Growth

The search for Eisenstein pseudocubes was motivated by a method of proving primality, described in Section 4.6. The computational complexity of this algorithm depends on the growth rate of the Eisenstein pseudocubes.

In Sections 3.1 and 3.2.1, we described theoretical growth rate predictions for the classical pseudosquares and pseudocubes using some elementary assumptions. In this section, we will endeavour to do the same for the Eisenstein pseudocubes.

Let p_i denote the i^{th} prime ($p_1 = 2$), and let \mathcal{S}_p denote the set of acceptable residues modulo p for the Eisenstein pseudocubes as developed in Section 6.2. Writing $p = p_n$,

we have that

$$\mathcal{S}_2 = \{(1, 0)\},$$

$$\mathcal{S}_9 = \{(2, 0), (5, 0), (8, 0)\}, \text{ and}$$

$$\mathcal{S}_p = \left\{ (a, b) \in \mathbb{Z} \times \mathbb{Z} \left| \left(\frac{p}{a + b\omega} \right)_3 = 1, -\left(\frac{p-1}{2} \right) \leq a, b \leq \left(\frac{p-1}{2} \right) \right. \right\} \text{ for } p > 3$$

Recall from Equations (6.1) and (6.7) that there are

$$|\mathcal{S}_p| = \begin{cases} \frac{(p-1)^2}{3} & \text{if } p \equiv 1 \pmod{3} \\ \frac{(p^2-1)}{3} & \text{if } p \equiv 2 \pmod{3} \end{cases}$$

acceptable residues modulo p for the Eisenstein pseudocube problem. Writing

$$\begin{aligned} S_1 &= \prod_{p \equiv 1 \pmod{3}} \frac{(p-1)^2}{3} & H_1 &= \prod_{p \equiv 1 \pmod{3}} p \\ S_2 &= \prod_{p \equiv 2 \pmod{3}} \frac{(p^2-1)}{3} & H_2 &= \prod_{p \equiv 2 \pmod{3}} p \end{aligned}$$

for primes $p \leq p_n$, and invoking the CRT we see that there are $S = 3S_1S_2$ solutions satisfying the congruence criteria of the Eisenstein pseudocubes in the region $-H/2 \leq a, b < H/2$, where $H = 9H_1H_2$.

Assume the S solutions $\mu = a + b\omega$ are equidistributed in the region $-H/2 \leq a, b < H/2$. By a similar argument to that of Lukes *et al.* [LPW96], we might expect the solution of *minimal norm* in this region, denoted by μ_p , to be given by $a \approx b \approx \frac{H}{\sqrt{3}}$; *i.e.*

$$N(\mu_p) \approx \frac{H^2}{S}. \tag{10.1}$$

Consider the primes $p = p_n$ as $n \rightarrow \infty$. Making an assumption that the primes are distributed equally between $p \equiv 1 \pmod{3}$ and $p \equiv 2 \pmod{3}$, we can approximate H^2/S as follows. Write

$$\frac{H_1^2}{S_1} = \prod_{\substack{p \equiv 1 \\ (\text{mod } 3) \\ p \leq x}} \frac{3p^2}{(p-1)^2}, \text{ and} \quad (10.2)$$

$$\frac{H_2^2}{S_2} = \prod_{\substack{p \equiv 2 \\ (\text{mod } 3) \\ p \leq x}} \frac{3p^2}{(p^2-1)}. \quad (10.3)$$

From Mertens's Theorem [HW79, p. 351], $\prod_{p \leq x} \left(1 - \frac{1}{p}\right) \sim \frac{e^{-\gamma}}{\ln x}$ as $x \rightarrow \infty$, so (10.2) becomes

$$\begin{aligned} \frac{H_1^2}{S_1} &\approx 3^{\pi(x)/2} \prod_{\substack{p \equiv 1 \\ (\text{mod } 3) \\ p \leq x}} \left(\frac{p}{p-1}\right)^2 \\ &\approx 3^{\pi(x)/2} \prod_{p \leq x} \frac{p}{p-1} \\ &\sim e^{\gamma} 3^{\pi(x)/2} \ln x \quad (\text{as } x \rightarrow \infty). \end{aligned}$$

For (10.3), recall that $\prod_{p \leq x} \left(1 - \frac{1}{p^2}\right) = \zeta(2) = \frac{\pi^2}{6}$ as $x \rightarrow \infty$.⁵ Hence

$$\frac{H_2^2}{S_2} \sim 3^{\pi(x)/2} \sqrt{\frac{6}{\pi^2}}$$

Putting these together and writing $n = \pi(x)$, $c = \frac{27e^{\gamma}\sqrt{6}}{\pi}$, we obtain

$$N(\mu_{p_n}) \approx \frac{(9H_1H_2)^2}{3S_1S_2} \sim c3^n \ln p_n \quad (10.4)$$

⁵See, for example, [CP05, Theorem 1.4.1].

as $n \rightarrow \infty$. Thus $\log(N(\mu_{p^n})) \approx \log 3 \cdot n(1 + o(1))$.

From the discussion of Section 5.5.1, we can therefore predict that the Eisenstein Pseudocube Primality Proving (EPPP) method has (randomized⁶) computational complexity $(\log m)^{3+o(1)}$. In Figure 10.3, we compare the growth rate of the Eisenstein pseudocubes obtained in Section 10.5 to the growth rate of classical pseudosquares and pseudocubes.

First, the slope of $\log N(\mu_p)/\log p \approx 1.05557$. This is very close to the slope that would be predicted by (10.4); *i.e.* $\log 3$. This result would seem to validate our growth rate prediction.

Second, though asymptotically similar, this growth rate vastly outpaces that of both the classical pseudosquares and pseudocubes. This gives good credence to our prediction that, for integers $m \equiv 1 \pmod{3}$, the EPPP algorithm will eventually give a more efficient means of proving these integers prime than either the classical pseudosquare or pseudocube methods described in Chapter 3.

10.6.1 A Caveat

It should be noted, however, that the pseudosquare primality test presented in Theorem 1.5 is slightly simplified from the version actually proposed by Lukes *et al.* in [LPW96]. The full version of that test incorporates a factor bound B , and is given as follows. If:

1. All prime divisors $q \mid N$ exceed the bound $B \in \mathbb{Z}^+$,
2. $\frac{N}{B} < M_{2,p}$ for some prime, p ,

⁶The randomized nature of the algorithm stems solely from the requirement for a quadratic nonresidue in Cornacchia's algorithm. Finding this quadratic nonresidue (the random step) requires on average, two evaluations of a Jacobi symbol.

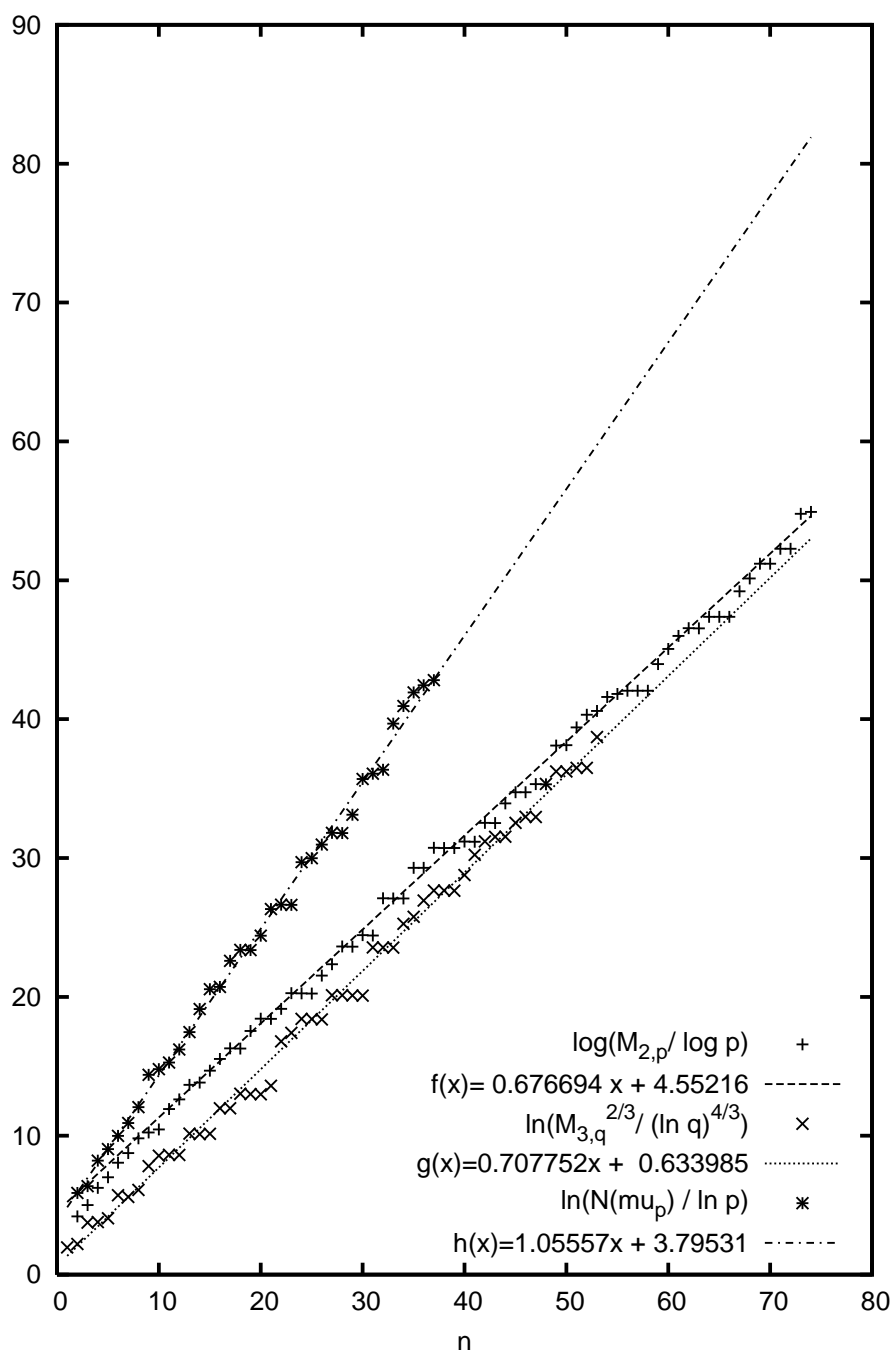


Figure 10.3: Growth Rate of Eisenstein Pseudocubes vs. Classical

3. $p_i^{\frac{N-1}{2}} \equiv \pm 1 \pmod{N}$ for all primes p_i , $2 \leq p_i \leq p$,
4. $p_j^{\frac{N-1}{2}} \equiv -1 \pmod{N}$ for some odd $p_j \leq p$ when $N \equiv 1 \pmod{8}$, or
 $2^{\frac{N-1}{2}} \equiv -1 \pmod{N}$ when $N \equiv 5 \pmod{8}$

then N is a prime or a power of a prime.

If we set $B = 1$ in the above test, this version of the algorithm is identical to what was presented in Chapter 3. If combined with techniques for eliminating small factors of N , however, the pseudosquare primality test can be extended to handle larger primes than would be possible by using a table of pseudosquares alone. At present, we have no analogue of this factor bound in the EPPP method.

Chapter 11

Future work

Do not worry about tomorrow, for tomorrow will worry about itself. Each day has enough trouble of its own.

—Matthew 6:25

11.1 Deploying the Hardware Sieve

At present, the hardware sieve described in Chapter 9—SCOT—is but a prototype. Through the process of its development and testing, much was learned about the problems inherent in two-dimensional sieving. Though the design was successful in many key areas, several design decisions need to be revisited before the device is ready for large-scale deployment.

11.1.1 Decoupled Solution Decoding

Sieve rates for a shift register-based device are based essentially on two key aspects of the sieve design: clock rate, and solution detector width. Because solution decoding is performed by the sieve FSM directly (see Section 10.4.1), a particularly dense set of solutions can theoretically reduce the overall canvass rate of the sieve device.

This decoding overhead could be reduced by decoupling the solution decoding logic from the sieve FSM; *i.e.* partitioning the state machine into two separate control

devices. When a solution is encountered, the current contents of the counter and offset registers would be latched to a set of decoding registers. The sieve device could then continue sieving in parallel with the solution decoding process.

Another potential bottleneck with the current solution decoding logic is that solutions must be acknowledged by the host machine before the device can continue sieving. If solution rates are high, waiting for this acknowledgment begins to dominate the sieve process. By adding a FIFO device to the sieve output. Solutions emerging from the output decoding circuit are stored in the FIFO, and recovered by the host machine at will. Only if the FIFO becomes full would it be necessary to halt the sieve process.

This is especially convenient when working with the Xilinx platform, as numerous dedicated Block RAM devices are already present on the Xilinx FPGA silicon. The Xilinx Block Ram can be configured to act as a FIFO, with minimal impact to chip area [XAp05].

11.1.2 Improving Clock Rate

One of the most direct means of improving hardware sieve rates is to simply increase the master clock rate of the sieve devices. In practice, of course, this is a nontrivial task; however there are several design ideas that could be considered to achieve improvements in clock rate.

One of the most complex components of the SCOT design drives the OHMR framework: the embedded PicoBlaze soft-processor. According to a post by its designer [Cha07], the PicoBlaze can operate at up to 117 MHz on the Spartan FPGA used in the SCOT design. Connection to external circuitry will certainly reduce this clock

rate—possibly by as much as 20%—giving a practical maximum clock rate of around 100 MHz.

It should be noted, however, that the PicoBlaze is primarily responsible for managing I/O with the host machine, and hence, is not necessarily speed-critical. Thus, there is the possibility of splitting the clock domain, running the sieve device faster than the embedded PicoBlaze soft-processor. Decoupled solution decoding, described in Section 11.1.1 above, actually facilitates a split-clock design, as a FIFO is the primary means of safely moving data between clock domains.

The most speed-critical sieve device component is the shift register. As shift registers become larger, more CLBs are required for their construction, leading to increased propagation delay and limiting the maximum clock rate of the device. One interesting means of avoiding this problem is to avoid the use of CLBs altogether when constructing shift registers by using a Block RAM device instead.

Block RAM components are available on most Xilinx FPGA devices. The Spartan 3A device used by SCOT, for instance, has 84 Block RAMs available. These devices have the virtue of being among the fastest components available on the Xilinx FPGA—operating at up to 250 MHz on the aforementioned Spartan 3A. Using only an additional 8-bit counter, a single Block RAM can be converted into a shift register with 72-bit output, with up to 256 entries [Alf08]. This approach is somewhat reminiscent of the OASiS designs of [Ste89], which also used RAM in place of shift registers.

11.1.3 Multi-Chip Sieve Devices

At present, the SCOT design is confined to a single FPGA. While additional parallelism is possible by instantiating complete instances of SCOT, this requires a dedicated serial connection from each SCOT instance to the host machine, as each chip will be controlled by a different OHMon instance.

To eliminate these potentially redundant serial connections, the OHMR framework could be modified to permit a single OHMon instance to communicate with several FPGA devices. At present, OHMR uses a 128-bit data bus to communicate to and from the target designs. A bus this large could introduce a great deal of headache when communicating off-chip. For this reason, it would be prudent to modify the design to incorporate a smaller inter-chip bus. One idea would be to place a mirrored **L** (Data) register on each of the FPGA devices, and make use of a chip select register for data load operations. Again, a decoupled solution decoder would facilitate this mechanism, as the transfer of candidate solutions between FPGA devices could occur in parallel with the sieving operation.

11.2 Solving the Communications Bottleneck

In Section 10.4.2, we described how the serial communications in OHMR can pose a bottleneck to two-dimensional sieve operation. To alleviate this bottleneck, we require a faster connection to the OHMR framework; *e.g.* USB, or a serial/deserializer device like [AVN]. Though not technically difficult, implementing such a device would require a custom device driver on the host machine. This has an impact on portability—moving the OHMR framework to a new platform would require the redevelopment of

the custom hardware drivers.

Another means of relaxing two-dimensional sieve I/O requirements would be to modify the data load mechanism. Currently, the computation of one-dimensional ring contents is performed on the host machine. If we are willing to sacrifice chip area, this functionality could be moved on-chip. There are two main requirements in computing residue data in the two-dimensional sieve problem. First, the set of one-dimensional residues corresponding to b must be looked up. Ideally, b should be reduced modulo m (where m is the sieve ring modulus) on the host machine, as performing this operation in the hardware would be needlessly time-consuming. Next, the m acceptable residues for a particular choice of b must be rotated to a new start position, *i.e.* $a_\ell \pmod{m}$.

11.3 Improving Software Sieve Rates

In 1976, D. H. Lehmer wrote a software implementation of the GSP on the ILLIAC IV—the first machine to employ what we now call a SIMD architecture. Due to the parallel nature of the computing model, this implementation was highly efficient—Lehmer’s second fastest sieve, in fact. Though the ILLIAC project itself was eventually deemed a flop, the SIMD model has been gaining popularity in general-purpose computing circles, especially in the form of General Purpose Graphics Processing Units (GPGPUs).

GPGPU is the practice of using graphics processors for general purpose computing. NVidia’s Compute Unified Device Architecture (CUDA) and the Open Computing Language (OpenCL) [Gro09] are the two most common frameworks for GPGPU development.

As an inherently parallel problem, congruential sieving would seem to be ideally suited to the GPGPU metaphor. As such, the advent of widely available GPGPU hardware could be a great boon to software sieve development.

11.4 Doubly-focusing in Two Dimensions

In [Ber04a], Bernstein sketched a technique for extending doubly-focused enumeration to higher dimensions. Briefly, the idea is as follows.

Consider $\pi \in \mathbb{Z}[\omega]$. From the Chinese Remainder Theorem, this value may be written $\pi = \alpha m_1 - \beta m_2$, $\alpha, \beta, m_1, m_2 \in \mathbb{Z}[\omega]$ where $\gcd(m_1, m_2) = 1$. If we define $m = m_1 m_2$ as the set of primes over which our sieve problem is defined, then we see that a sieve problem on π may be transformed into two equivalent (and smaller) sieve problems involving α, β , as each of these new problems will involve a subset of our original sieve moduli.

In the case of the Eisenstein pseudocubes, we are interested in quantities $\mu_p \in \mathbb{Z}[\omega]$ satisfying certain sieve criteria, and of *minimal norm*. Observe that $N(\pi) = N(\alpha m_1 - \beta m_2) = |\alpha m_1 - \beta m_2|^2$ where $|\cdot|$ refers to the traditional concept of Euclidean distance. Write $A = a + b\omega$, $B = c + d\omega$, hence $N(A - B) = (a - c)^2 + (b - d)^2 - (a - c)(b - d)$, and observe that

$$\begin{aligned}
|A - B|^2 &= \left| (a - c) + (b - d) \frac{(-1 + \sqrt{3}i)}{2} \right|^2 \\
&= \left| (a - c) - \frac{(b - d)}{2} + (b - d) \frac{\sqrt{3}i}{2} \right|^2 \\
&= \left(\frac{2(a - c) - (b - d)}{2} \right)^2 + \left(\frac{(b - d)\sqrt{3}}{2} \right)^2 \\
&= \frac{4(a - c)^2 + (b - d)^2 - 4(a - c)(b - d) + 3(b - d)^2}{4} \\
&= (a - c)^2 + (b - d)^2 - (a - c)(b - d) \\
&= N(A - B).
\end{aligned}$$

If we refer to the solutions of the sieve problems involving α, β as \mathcal{A} and \mathcal{B} respectively, it should be clear that producing solutions to a sieve problem for π of minimal norm $N(\pi)$ is equivalent to solving the sieve problems involving $\alpha, \beta \in \mathbb{Z}[\omega]$, multiplying each solution by m_1 or m_2 respectively, and then computing nearest neighbours between the resulting sets $m_1\mathcal{A}$ and $m_2\mathcal{B}$.

Solving for each of these sets is an instance of a two-dimensional sieve problem, albeit over (approximately) half the primes of the original problem. By repeated application of this technique, the two-dimensional sieve problem may be reduced to one that is manageable to compute, either exhaustively, or via traditional (one-dimensional) sieve techniques.

Since double-focusing has been a highly successful technique for improving sieve rates—at least for the one-dimensional sieve problem—this idea is certainly worth investigating in more detail.

Chapter 12

Conclusions

You're still here? It's over. Go home.

—Ferris Bueller

In this thesis, we have described an efficient new primality proving method for integers $m \equiv 1 \pmod{3}$: the Eisenstein Pseudocube Primality Proving (EPPP) method. This method improves on a previous generalization of the pseudosquare primality proving method—the pseudocube-based method of Berrizbeitia *et al.* —to deliver a method for proving primality using $(\log m)^{3+o(1)}$ operations.

To describe this method, we developed the notion of an Eisenstein pseudocube, and described how the search for these quantities can be viewed as an instance of a two-dimensional congruential sieve problem. We then turned our attention to the problem of sieving in two dimensions, developing tools—mathematical, software, and hardware—for solving this problem.

In order to test and deliver this sieve device, we developed a highly flexible toolkit—Open Hardware for Mathematical Research (OHMR)—for implementing special-purpose computing devices on Xilinx FPGA platforms. This toolkit makes it possible to quickly design, test, specialized hardware, and has applications far beyond the sieve problem.

Within this framework, we built a prototype sieve device—SCOT—that is capable

of raw canvass rates at least $30\times$ faster than the existing software sieve, and $2.5\times$ faster than the any other modern sieve device. Though not yet fully integrated into the CASSIE framework, we expect that when SCOT is scaled-up to a production-ready device, this F+V sieve design will offer unprecedented sieve capabilities for a variety of problems, new and old.

Finally, though we have examined Eisenstein pseudocubes in detail, we have but scratched the surface of the two-dimensional sieve problem. With our improved sieve devices—CASSIE and SCOT—a wide variety of previously unexplored problems in number theory are now accessible to us.

We have built the sieve. Let the solutions come.

Bibliography

- [AKS02] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, *PRIMES is in P*, (preprint) Internet Archive URL: http://web.archive.org/web/*/http://www.cse.iitk.ac.in/news/primalty.pdf, August 2002.
- [AKS04] ———, *PRIMES is in P*, *Annals of Mathematics* **160** (2004), no. 2, 781–793.
- [Alf08] Peter Alfke, *Creative Uses of Block RAM*, Tech. Report WP335, Xilinx, June 2008, Version 1.0.
- [AVN] AVNET Electronics Marketing, *EXP high-performance SerDes module with easy FPGA interface*, URL: <http://www.em.avnet.com/exp-serdes>.
- [Bac90] Eric Bach, *Explicit bounds for primality testing and related problems*, *Mathematics of Computation* **55** (1990), no. 191, 355–380. MR 91m:11096
- [Bas04] J. M. Basilla, *On the solution of $x^2 + dy^2 = m$* , *Proceedings of the Japan Academy. Ser. A Mathematical Sciences* **80** (2004), no. 5, 40–41.
- [BB94] Nathan D. Bronson and D. A. Buell, *Congruential sieves on FPGA computers*, *Mathematics of computation, 1943–1993: a half-century of computational mathematics: Mathematics of Computation 50th Anniversary Symposium, August 9–13, 1993, Vancouver, British Columbia (Providence, RI, USA) (Walter Gautschi, ed.)*, vol. 48, American Mathematical Society, 1994, pp. 547–551.
- [Bee39] N. G. W. H. Beeger, *Report on some calculations of prime numbers*, *Nieuw Archief voor Wiskunde* **20** (1939), no. 2, 48–50.

- [Bee46] ———, *Note sur la factorisation de quelques grands nombres*, Institut Grand-Ducal de Luxembourg, Section des sciences naturelles, physiques, et mathématiques archives **16** (1946), 93–95.
- [Ber98] D. J. Bernstein, *Detecting perfect powers in essentially linear time*, Mathematics of Computation **67** (1998), 1253–1283.
- [Ber01a] ———, *Faster algorithms to find non-squares modulo worst-case integers*, December 2001, URL: <http://cr.yp.to/papers.html#nonsquare>.
- [Ber01b] ———, *Multidigit multiplication for mathematicians*, <http://cr.yp.to/papers.html#m3>, August 2001.
- [Ber02] ———, *Proving primality after Agrawal-Kayal-Saxena*, <http://cr.yp.to/papers/aks.pdf>, 2002.
- [Ber04a] ———, *Doubly-focused enumeration in two dimensions*, <http://cr.yp.to/talks/2004.06.24/transcript.txt>, 2004.
- [Ber04b] ———, *Doubly focused enumeration of locally square polynomial values*, High Primes and Misdemeanors—Lectures in Honour of the 60th Birthday of Hugh Cowie Williams (Alf van der Poorten and Andreas Stein, eds.), Fields Institute Communications, vol. 41, AMS, 2004, pp. 67–76.
- [Ber05] P. Berrizbeitia, *Sharpening PRIMES is in P for a large family of numbers*, Mathematics of Computation **74** (2005), no. 252, 2043–2059.
- [Ber07] D. J. Bernstein, *Proving primality in essentially quartic random time*, Mathematics of Computation **76** (2007), no. 257, 389–403.
- [BMW04] P. Berrizbeitia, S. Müller, and H. C. Williams, *Pseudocubes and primality testing.*, Proceedings of the Sixth International Symposium, ANTS-VI, Lecture Notes in Computer Science, vol. 3076, Springer-Verlag, 2004,

- pp. 102–116.
- [Bri72] John Brillhart, *Note on representing a prime as a sum of two squares*, *Mathematics of Computation* **26** (1972), no. 120, 1011–1013. MR 93k:11089
- [BS67] John Brillhart and J. L. Selfridge, *Some factorizations of $2^n \pm 1$ and related results*, *Mathematics of Computation* **21** (1967), no. 97, 87–96.
- [BS96] Eric Bach and J. O. Shallit, *Algorithmic Number Theory: Efficient Algorithms*, The MIT Press, 1996.
- [CEFT62] D. G. Cantor, G. Estrin, A. S. Fraenkel, and R. Turn, *A very high-speed digital number sieve*, *Mathematics of Computation* **16** (1962), no. 78, 141–154.
- [Cha07] Ken Chapman, December 2007, URL: <http://forums.xilinx.com/xlnx/board/message?board.id=PicoBlaze&thread.id=87>.
- [Cha08] ———, *200 MHz UART with internal 16-byte buffer*, Xilinx, April 2008, Version 1.2.
- [Che03] Qi Cheng, *Primality proving via one round in ECPP and one iteration in AKS*, *Advances in Cryptology: Proceedings of CRYPTO 2003*, Lecture Notes in Computer Science, vol. 2729, Springer-Verlag, 2003, pp. 338–348.
- [Cip03] Michele Cipolla, *Un metodo per la risoluzione della congruenza di secondo grado*, *Rendiconto dell'Accademia delle Scienze Fisiche e Matematiche Napoli* **9** (1903), 154–163.
- [Cob66] Alan Cobham, *The recognition problem for the set of perfect squares*, Tech. report, IBM Research, April 1966.
- [Coh93] Henri Cohen, *A Course in Computational Algebraic Number Theory*, 4th

- ed., Springer, 1993.
- [Col03] F. N. Cole, *On the factoring of large numbers*, Bulletin of the American Mathematical Society (1903), 134–137.
- [Cor08] Giuseppe Cornacchia, *Su di un metodo per la risoluzione in numeri interi dell' equazione $c_h x^{n-h} y^h = p$* , Giornale di Matematiche di Battaglini **46** (1908), 33–90.
- [CP05] Richard Crandall and Carl Pomerance, *Prime numbers: A computational Perspective*, 2nd ed., Springer, New York, 2005. MR 2156291 (2006a:11005)
- [ES98] Shawna M. Eikenberry and J. P. Sorenson, *Efficient algorithms for computing the jacobi symbol*, Journal of Symbolic Computation **26** (1998), no. 4, 509–523.
- [Gal02] Joseph A. Gallian, *Contemporary Abstract Algebra*, 5th ed., Houghton Mifflin Company, 2002.
- [GKP89] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics*, Addison-Wesley, 1989.
- [Gro09] Khronos OpenCL Working Group, *OpenCL 1.0 Specification*, 2009, URL: <http://www.khronos.org/registry/cl/>.
- [Hal33] Marshall Hall, *Quadratic residues in factorization*, Bulletin of the American Mathematical Society **39** (1933), 758–763.
- [Her48] C. Hermite, *Note au sujet de l'article précédent*, Journal de Mathématiques Pures et Appliquées **13** (1848), 15.
- [HMW90] Kenneth Hardy, Joseph B. Muskat, and Kenneth S. Williams, *A deterministic algorithm for solving $n = fu^2 + gv^2$ in coprime integers u and*

- v*, *Mathematics of Computation* **55** (1990), no. 191, 327–343.
- [HW79] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, 5th ed., Oxford University Press, 1979.
- [HWLP09] Jr. H. W. Lenstra and C. Pomerance, *Primality testing with Gaussian periods*, October, 2009, URL: <http://math.dartmouth.edu/~carlp/aks102309.pdf>.
- [IJ02] S. P. Harbison III and G. L. Steele Jr., *C: A Reference Manual*, 5th ed., Prentice Hall, 2002.
- [IR90] Kenneth Ireland and Michael Rosen, *A Classical Introduction to Modern Number Theory*, 2nd ed., Graduate Texts in Mathematics, no. 84, Springer-Verlag, 1990.
- [Ive62] Kenneth E. Iverson, *A Programming Language*, John Wiley and Sons, Inc., 1962.
- [Kna99] Anthony W. Knapp, *Frank Nelson Cole*, *Notices of the American Mathematical Society* **46** (1999), no. 8, 860.
- [Kra24] Maurice Kraitchik, *Récherches sur la Théorie des Nombres. Tome I*, Gauthier-Villars, Paris, 1924.
- [Kra29] ———, *Récherches sur la Théorie des Nombres. Tome II*, Gauthier-Villars, Paris, 1929.
- [Leh28] D. H. Lehmer, *The mechanical combination of linear forms*, *The American Mathematical Monthly* **35** (1928), no. 4, 114–121.
- [Leh30] ———, *A fallacious principle in the theory of numbers*, *Bulletin of the American Mathematical Society* **36** (1930), 847–850.
- [Leh54] ———, *A sieve problem on pseudo-squares*, *Mathematical Tables and*

- Other Aids to Computation **8** (1954), no. 48, 241–242. MR 16,113e
- [Leh69] ———, *Computer technology applied to the theory of numbers*, Studies in Number Theory (William J. Leveque, ed.), MAA Studies in Mathematics, vol. 6, Prentice-Hall, Englewood Cliffs, New Jersey, 1969, pp. 117–151. MR 0246815
- [Leh76] ———, *Exploitation of parallelism in number theoretic and combinatorial calculation*, Proceedings of the Sixth Manitoba Conference on Numerical Mathematics (B. L. Hartnell and H. C. Williams, eds.), Utilitas Mathematica, 1976, pp. 95–111.
- [Lin99] Scott Lindhurst, *An analysis of Shanks's algorithm for computing square roots in finite fields*, Number Theory: Fifth Conference of the Canadian Number Theory Association. August 17-22, 1996 (Rajiv Gupta and Kenneth S. Williams, eds.), American Mathematical Society, 1999, URL: <http://scott.lindhurst.com/papers/shanks.ps.gz>, pp. 231–242.
- [LPW96] Richard F. Lukes, Cameron D. Patterson, and H. C. Williams, *Some results on pseudosquares*, Mathematics of Computation **65** (1996), no. 213, 361–372, S25–S27. MR 96e:11010
- [Luk95] Richard F. Lukes, *A very fast electronic number sieve*, Ph.D. thesis, The University of Manitoba, 1995.
- [Mö4] Siguna Müller, *On the computation of square roots in finite fields*, Designs, Codes and Cryptography **31** (2004), no. 3, 301–312.
- [MA] Preda Mihăilescu and Roberto M. Avanzi, *Efficient “quasi”-deterministic primality test improving AKS*, URL: <http://caccioppoli.mac.rub.de/website/papers/aks-mab.pdf>.

- [Mac02] Martin Macaj, *Some remarks and questions about the AKS algorithm and related conjecture*, 2002, URL: <http://thales.doa.fmph.uniba.sk/macaj/aksremarks.pdf>.
- [MN90] F. Morain and J. L. Nicolas, *Courbes elliptiques et tests de primalité*, Thèse, Université Claude Bernard–Lyon I, September 1990.
- [MvOV96] Alfred J. Menezes, P. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, 5th ed., CRC Press, 1996.
- [Nit95] Abderrahmane Nitaj, *l’algorithme de Cornacchia*, Exposition. Math. **13** (1995), no. 4, 358–365. MR 1358213 (97a:11044)
- [Ous94] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley, 1994.
- [Pat83] Cameron D. Patterson, *Design and use of an electronic sieve*, Master’s thesis, University of Manitoba, 1983.
- [Pat92] ———, *Derivation of a high speed sieve device*, Ph.D. thesis, The University of Calgary, 1992.
- [Pom87] C. Pomerance, *Very short primality proofs*, Mathematics of Computation **48** (1987), no. 177, 315–322.
- [PS09] C. Pomerance and I. E. Shparlinski, *On pseudosquares and pseudopowers*, Combinatorial Number Theory, Proceedings of Integers Conference 2007 (Berlin), de Gruyter, 2009, pp. 171–184.
- [Sch71] Arnold Schönhage, *Schnelle Berechnung von Kettenbruchentwicklungen*, Acta Informatica **1** (1971), no. 2, 139–144.
- [Sch95] R. Schoof, *Counting points on elliptic curves over finite fields*, Journal de théorie des nombres de Bordeaux **7** (1995), no. 1, 219–254.
- [Sch97] A. Schinzel, *On pseudosquares*, New Trends in Probability and Statistics **4**

- (1997), 213–220.
- [See86] P. Seelhoff, *Ein neues Kennzeichen für die Primzahlen*, Zeitschrift für Mathematik und Physik **31** (1886), 306–310.
- [Ser48] J. A. Serret, *Sur un théorème relatif aux nombres entiers*, Journal de Mathématiques Pures et Appliquées **13** (1848), 12–14.
- [Sha73] Daniel Shanks, *Five number-theoretic algorithms*, Proceedings of the Second Manitoba Conference on Numerical Mathematics. October 5–7, 1972 (R. S. D. Thomas and H. C. Williams, eds.), Congressus Numerantium, vol. VII, Utilitas Mathematica Publishing Inc., 1973, pp. 51–70.
- [Sor94] J. P. Sorenson, *Two fast GCD algorithms*, Journal of Algorithms **16** (1994), 110–144.
- [Sor06] ———, *The pseudosquares prime sieve*, Algorithmic Number Theory (F. Hess, S. Pauli, and M. Pohst, eds.), LNCS, vol. 4076, Springer-Verlag, 2006, pp. 193–207.
- [Sor09] Jonathan P. Sorenson, 2009, Private correspondence.
- [Sor10] ———, *Sieving for pseudosquares and pseudocubes in parallel using doubly-focused enumeration and wheel datastructures*, Algorithmic Number Theory, Lecture Notes in Comput. Sci., Springer-Verlag, 2010, 9th International Symposium, ANTS-IX, Nancy, France, July 19-23, 2010, Proceedings.
- [SS71] Arnold Schönhage and Volker Strassen, *Schnelle Multiplikation grosser Zahlen*, Computing **7** (1971), 281–292.
- [Ste89] Allan J. Stephens, *Oasis: An open architecture sieve system for problems in number theory*, Ph.D. thesis, The University of Manitoba, 1989.

- [SW90] Allan J. Stephens and H. C. Williams, *An open architecture number sieve*, Number Theory and Cryptography (Sydney, 1989), London Math. Soc. Lecture Note Ser., vol. 154, Cambridge Univ. Press, Cambridge, 1990, pp. 38–75. MR 1 055 399
- [TDB⁺06] S. Tucker Taft, Robert A. Duff, Randall Brukardt, Erhard Plöereder, and Pascal Leroy, *Ada 2005 Reference Manual. Language and Standard Libraries - International Standard ISO/IEC 8652/1995 (E) with Technical Corrigendum 1 and Amendment 1*, Lecture Notes in Computer Science, vol. 4348, Springer, 2006.
- [Ton91] Alberto Tonelli, *Bemerkung über die Auflösung quadratischer Congruenzen*, Nachrichten von der Königlichen Gesellschaft der Wissenschaften und der Georg-Augusts-Universität zu Göttingen (1891), 344–346, URL: <http://resolver.sub.uni-goettingen.de/purl?GDZPPN002525739>.
- [UG108] Xilinx, *PicoBlaze 8-bit embedded microcontroller user guide*, June 24 2008, Version 1.1.2.
- [WB03] H. A. Wake and D. A. Buell, *Congruential sieves on a reconfigurable computer*, 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (2003), 11–18.
- [Wil77] K. S. Williams, *On Eisenstein's supplement to the law of cubic reciprocity*, Bulletin of the Calcutta Mathematical Society **69** (1977), 311–314.
- [Wil78a] H. C. Williams, *Primality testing on a computer*, Ars Combinatoria **5** (1978), 127–185.
- [Wil78b] ———, *Some properties of a special set of recurring sequences*, Pacific Journal of Mathematics **77** (1978), no. 1, 273–285.

- [Wil80] Peter Wilker, *An efficient algorithmic solution of the Diophantine equation $u^2 + 5v^2 = m$* , *Mathematics of Computation* **35** (1980), no. 152, 1347–1352.
- [Wil86] H. C. Williams, *An m^3 public-key encryption scheme*, *Advances in Cryptology: Proceedings of CRYPTO '85* (H. C. Williams, ed.), *Lecture Notes in Computer Science*, Springer-Verlag, 1986, pp. 358–368.
- [Wil98] ———, *Édouard Lucas and Primality Testing*, *Canadian Mathematical Society Series of Monographs and Advanced Texts*, vol. 22, Wiley-Interscience, 1998.
- [Woo04] Kjell Wooding, *Development of a high-speed hybrid sieve architecture*, Master's thesis, The University of Calgary, Calgary, AB, January 2004.
- [WW06] Kjell Wooding and H. C. Williams, *Doubly-focused enumeration of pseudosquares and pseudocubes*, *Proceedings of the Seventh International Symposium, ANTS-VII, Lecture Notes in Computer Science*, vol. 4076, Springer-Verlag, 2006, pp. 208–221.
- [XAp05] Xilinx, *FIFOs using Virtex-II Block RAM*, January 2005, Version 1.4.
- [Zur94] D. Zuras, *More on squaring and multiplying large integers*, *IEEE Transactions on Computers* **43** (1994), no. 8, 899–908.

Appendix A

Algorithms

A.1 Basic Algorithms

For the following analyses, we will adopt the *naïve bit complexity* model of computation described in [BS96]. Thus we assume an integer n can be represented as a bit string of length $\lceil \log_2 n \rceil + 1 = O(\log n)$ bits. For our analyses, we will count bit operations¹ of the following basic types:

- Basic Operations (\mathcal{B}): Addition, Subtraction, Bit Shifts, and are essentially $\Theta(\log n)$.
- Multiplications (\mathcal{M}): For a relatively complete survey of multiplication methods, see [Ber01b]. Complexity ranges roughly from $O((\log n)^2)$ for the school-book method to $O(\log n \log \log n \log \log \log n)$ for [SS71].
- Squarings (\mathcal{Q}): Conceptually, squarings can be seen as more efficient than multiplication. Asymptotically, however, squaring and multiplication must be considered equivalent, as we have the identity: $ab = \frac{(a+b)^2 - (a-b)^2}{4}$.
- Modular Exponentiations (\mathcal{X}): Roughly $(\log n)\mathcal{M}$.

Also, we consider the following basic algorithms:

- GCD: Euclid's algorithm [BS96, §4.2]: $O(n^2)$, Sorenson's k -ary methods [Sor94]: $O(n^2/\log n)$. Schönhage: $O(n(\log n)^2 \log \log n)$.
- Jacobi Symbol $\left(\frac{a}{n}\right)$: Essentially, the same as GCD. Basic method (via Bach,

¹Here we use the terms bit operations, time, and cost interchangeably.

Shallit) [BS96, §5.9]: $O(n^2)$, Eikenberry and Sorenson [ES98]: $O(n^2/\log n)$, Schönhage [Sch71]: $O(n(\log n)^2 \log \log n)$. Note that speedups are possible when several Jacobi symbols must be evaluated for the same modulus [Ber01a].

- Quadratic Non-Residue (QNR): Randomized, with probability $1/2^k$, where k is the number of trials. Each trial requires one Jacobi symbol evaluation.

A.2 Computing Modular Square Roots

A.2.1 Cipolla-Lehmer

In [Cip03] Cipolla published an algorithm to compute square roots modulo a prime using $(\log n)^3$ operations [BS96, §7.2]. This algorithm makes use of field operations in a quadratic extension of \mathbb{F}_p . In [Leh69], Lehmer eliminates the requirement to work in a quadratic field, employing Lucas sequences instead. Müller [Mö4], further improved Lehmer's method in practice, though the asymptotic bound remains unchanged. Though this algorithm offers a better asymptotic bound than other methods, in practice, it is rarely used. Instead, we will turn to the algorithm of Tonelli-Shanks, which offers excellent average-case performance, with only a slight worst-case performance variation.

A.2.2 Tonelli-Shanks

Shanks [Sha73], published an algorithm closely related to (but independent of) of Tonelli [Ton91] to produce what is now known as the Tonelli-Shanks algorithm. Shanks referred to this algorithm as RESSOL.

Algorithm A.1: RESSOL: Find square root modulo a prime

Input: p , an odd prime, a a quadratic residue modulo p .

Output: r such that $r^2 \equiv a \pmod{p}$.

- 1: Write $(p - 1) = 2^s q$ where $q = 2k + 1$
 - 2: Set $r \leftarrow a^{\frac{q+1}{2}} \pmod{p}$, $n \leftarrow a^q \pmod{p}$
 - 3: $z \leftarrow \text{QNR}(a, p)$, $c \leftarrow z^q \pmod{p}$
 - 4: **while** $n \not\equiv 1$ **do**
 - 5: Compute the least i , $0 < i < s$ such that $n^{2^i} \equiv 1$, and set $b \leftarrow c^{2^{s-i-1}}$.
 - 6: Set $r \leftarrow br$, $n \leftarrow bn^2$, $s \leftarrow i$.
 - 7: **end while**
 - 8: **return** r
-

A.2.3 Analysis

All operations below are assumed to take place in the field $(\mathbb{Z}/p\mathbb{Z})^*$.

- In Line 1, we obtain the parameters q, s necessary to run the algorithm.
- In Line 2, parameters r, n are constructed to form our loop invariant condition: $r^2 \equiv an \pmod{p}$. Note this condition remains true for any choice of b if we replace $r \leftarrow rb \pmod{p}$ and $n \leftarrow nb^2 \pmod{p}$. Furthermore if $n \equiv 1$, we are done, as $r^2 \equiv a$.
- In Line 3 we use a quadratic nonresidue, z , to construct an element c of order 2^s in $(\mathbb{Z}/p\mathbb{Z})^*$. Note by repeatedly squaring this element, we can obtain an element of order 2^{s-e} for any $0 \leq e < s$.
- The main loop starts at Line 4. The algorithm terminates when $n \equiv 1$.
- Once we know the order of n , computed in Line 5, we compute a second element b such that n and b^2 both have order 2^i .
- In Line 6, our invariant variables are updated so as to preserve the relationship $r^2 \equiv an$; *i.e.* $r \leftarrow rb, n \leftarrow nb^2$. Furthermore, since n and b^2 are both of order 2^i , their multiplication will produce an element of order strictly less than i .

Since $s \leftarrow i$, it is clear that the algorithm will eventually terminate with $s = 0$; *i.e.* $n = 1$.

A.2.4 Efficiency

All operations are in $(\mathbb{Z}/p\mathbb{Z})^*$.

- The computation of q, s in Line 1 requires 1 subtraction, and s bit shifts.
- In Line 2, if we first compute k (one subtraction and one bit shift), we can compute $A \equiv a^k$, (1 exponentiation), and $r \equiv Aa \equiv a^{k+1}$ and $n \equiv Ar \equiv a^{2k+1}$ using 2 additional multiplications.
- In Line 3, we require a quadratic nonresidue, and one exponentiation.
- Shanks observed that the computations of the main loop can be efficiently computed by replacing Lines 4–7 with:

```

4: while  $n \neq 1$  do
5.a:    $L \leftarrow s, b \leftarrow n$ 
5.b:   for  $0 \leq i < L$  do
5.c:     if  $b \equiv 1 \pmod{p}$  then
5.d:        $b \leftarrow c, s \leftarrow i$ 
5.e:     else
5.f:        $b \leftarrow b^2 \pmod{p}$ 
5.g:     end if
5.h:   end for
6.a:    $c \leftarrow b^2 \pmod{p}, r \leftarrow br \pmod{p}, n \leftarrow cn \pmod{p}$ 
7: end while

```

If $p - 1 = 2^s q$, then the worst case performance of this loop requires $\frac{s(s-1)}{2}$ squarings, and $2s$ multiplications.

All told, the algorithm requires:

- Initialization: $(s + 3)\mathcal{B} + 2\mathcal{X} + 2\mathcal{M}$ operations, and the evaluation of one QNR.
- Loop: $(2s)\mathcal{S} + \frac{s(s-1)}{2}\mathcal{Q}$.

Algorithm A.2: NORMALIZE

Input: Ring $\{\mathcal{R}, M\}$, normalization $x_n \pmod{m_n}$.

Output: Normalized ring, $\{\mathcal{R}_n, M\}$.

```

1: if ( $m_n = 1$ ) and ( $x_n = 0$ ) then
2:   return  $\{\mathcal{R}, M\}$ 
3: end if
4:  $\bar{x} \leftarrow x_n \pmod{M}$ 
5: for  $i \leftarrow 0$  to  $(M - 1)$  do
6:    $\mathcal{R}_n[i] \leftarrow \mathcal{R}[\bar{x}]$ 
7:    $\bar{x} \leftarrow \bar{x} + m_n \pmod{M}$ 
8: end for
9: return  $\{\mathcal{R}_n, M\}$ 

```

In the worst case, we assume $s = \log_2 p$. Using conservative estimates, $\mathcal{B} = \log p$, $\mathcal{Q} = \mathcal{M} = (\log p)^2$, $\mathcal{X} = (\log p)^3$ giving a worst-case time complexity of: $(\log p)^4$ operations. This is randomized, due to the requirement for a QNR.

Actual performance is, in practice, much quicker. In [Coh93, pp. 33], Cohen indicates that we would expect the body of the loop to require $s^2/4$ multiplications. In [Lin99], Lindhurst supplies empirical evidence to support this claim. Furthermore, it should be noted that virtually all of these “multiplications” are in fact squarings— asymptotically equivalent to multiplications, but much faster in practice [Zur94].

A.3 Sieve Algorithms

In [Luk95, Algorithm 4.1], Lukes gives an explicit algorithm for transforming a sieve ring using the normalization map described in Section 2.2.2. His approach is given as Algorithm A.2.

An equivalent technique, dubbed *denormalization*, may be developed to reverse

Algorithm A.3: DENORMALIZE

Input: Ring $\{\mathcal{R}_n, M\}$, normalization $x_n \pmod{m_n}$.

Output: Denormalized ring, $\{\mathcal{R}, M\}$.

```

1: if ( $m_n = 1$ ) and ( $x_n = 0$ ) then
2:     return  $\{\mathcal{R}_n, M\}$ 
3: end if
4:  $\bar{x} \leftarrow x_n - m_n \pmod{M}$ 
5: for  $i \leftarrow (M - 1)$  downto 0 do
6:      $\mathcal{R}[\bar{x}] \leftarrow \mathcal{R}_n[i]$ 
7:      $\bar{x} \leftarrow \bar{x} - m_n \pmod{M}$ 
8: end for
9: return  $\{\mathcal{R}, M\}$ 

```

this operation. Here the inverse permutation is generated by first locating the bit position of the final $(M - 1)^{th}$ entry in the sieve, and then repeatedly subtracting the normalization modulus. An explicit version of this technique is given as Algorithm A.3.

As a practical consideration, it should be noted that in many common programming languages, use of the modular operator with negative inputs can result in unexpected—even undefined—behaviour.² For this reason, it may be desirable to modify the denormalization algorithm to use only positive quantities. This modification is described in Algorithm A.4.

Algorithm A.4: DENORMALIZE*

Input: Ring $\{\mathcal{R}_n, M\}$, normalization $x_n \pmod{m_n}$.

Output: Denormalized ring, $\{\mathcal{R}, M\}$.

```

1: if ( $m_n = 1$ ) and ( $x_n = 0$ ) then

```

²In the C programming language, for example, it is true that $(a/b) * b + a \% b == a$ whenever $b \neq 0$; however, prior to the ratification of the C99 standard, if either a or b was negative the rounding direction of the division—and hence the sign of the remainder—was machine-dependent [IJ02, §7.6.1]. For a more complete discussion of this issue, see Section 7.3.2.

```

2:   return  $\{\mathcal{R}_n, M\}$ 
3: end if
4:  $m^* \leftarrow M - (m_n \pmod{M})$ 
5:  $\bar{x} \leftarrow x_n + m^* \pmod{M}$ 
6: for  $i \leftarrow (M - 1)$  downto 0 do
7:    $\mathcal{R}[\bar{x}] \leftarrow \mathcal{R}_n[i]$ 
8:    $\bar{x} \leftarrow \bar{x} + m^* \pmod{M}$ 
9: end for
10: return  $\{\mathcal{R}, M\}$ 

```

Given a normalized sieve problem \mathcal{S} , define an accessor function $\text{next}(\mathcal{S})$ which returns the next sieve output. Now, fix M_n, M_p and consider the doubly-focused sieve problem given by $\delta[M_n, M_p] : (\mathcal{S})$. This consists of an upper bound, H , and positive and negative sieve problems $\mathcal{S}_p, \mathcal{S}_n$. Given the current output of positive and negative sieves (t_p, t_n respectively), Algorithm A.5 (HUNT) describes a method for obtaining an initial solution $x \in \mathcal{S}$ such that $x = t_p M_n - t_n M_p$, $0 \leq x < H$. Algorithm A.6 (DFSIEVE) describes a method for obtaining all such solutions $x \in \mathcal{S}$ in the given interval.

Algorithm A.5: HUNT: Find an initial solution

Input: $H, \mathcal{S}_p := \{\mathcal{R}_p, M_p\}, \mathcal{S}_n := \{\mathcal{R}_n, M_n\}, t_n, t_p$
Output: $t_p \in \mathcal{S}_p, t_n \in \mathcal{S}_n$ such that $x = t_p M_n - t_n M_p$, $0 \leq x < H$.

```

1:  $x \leftarrow t_p M_n - t_n M_p$ 
2: repeat
3:   while  $x < 0$  do
4:      $t_p \leftarrow \text{next}(\mathcal{S}_p)$ 
5:      $x \leftarrow (t_p M_n - t_n M_p)$ 
6:   end while
7:   while  $(x \geq H)$  do
8:      $t_n \leftarrow \text{next}(\mathcal{S}_n)$ 
9:      $x \leftarrow (t_p M_n - t_n M_p)$ 
10:  end while
11: until  $(x \geq 0)$ 
12: Return  $(t_p, t_n)$ 

```

Algorithm A.6: DFSIEVE: Doubly-focused enumeration of sieve solutions

Input: $H, \mathcal{S}_p := \{\mathcal{R}_p, M_p\}, \mathcal{S}_n := \{\mathcal{R}_n, M_n\}$
Output: All solutions x such that $x = t_p M_n - t_n M_p, t_p \in \mathcal{S}_p, t_n \in \mathcal{S}_n, 0 \leq x < H$.

```

1:  $t_p \leftarrow \text{next}(\mathcal{S}_p)$ 
2:  $t_n \leftarrow \text{next}(\mathcal{S}_n)$ 
3:  $(t_p, t_n) \leftarrow \text{HUNT}(H, \mathcal{S}_p, \mathcal{S}_n, t_p, t_n)$ 
4: while not DONE do
5:   Append  $x_{\text{last}} = t_p M_n - t_n M_p$ 
6:    $t_p \leftarrow \text{next}(\mathcal{S}_p); a_p \leftarrow t_p M_n$ 
7:   while  $(a_p - a_n < H)$  do
8:     Extend  $x$  array with  $x_{\text{last}} \leftarrow a_p - a_n$ 
9:   end while
10:  for All  $x_i$  in  $x_{\text{first}}, \dots, x_{\text{last}}$  do
11:    Filter and/or print  $x_i$ 
12:  end for
13:   $\delta_n \leftarrow \text{next}(\mathcal{S}_n) - t_n; \Delta_n \leftarrow \delta_n M_p; a_n \leftarrow a_n + \delta_n$ 
14:  if  $t_n \geq M_n$  then
15:    return (DONE)
16:  end if
17:  for All  $x_i$  in  $x_{\text{first}}, \dots, x_{\text{last}}$  do
18:     $x_i = x_i - \Delta_n M_p$ 
19:    if  $x_i < 0$  then
20:      Delete  $x_{\text{first}}$ 
21:      if  $x$  array empty then
22:         $t_n \leftarrow \text{next}(\mathcal{S}_n)$ 
23:         $(t_p, t_n) \leftarrow \text{HUNT}(H, \mathcal{S}_p, \mathcal{S}_n, t_p, t_n)$ 
24:      end if
25:    end if
26:  end for
27: end while

```
