Faktorisieren mit dem Quadratischen Sieb auf dem Hypercube

Diplomarbeit

von

Thomas Sosnowski

nach einem Thema von Prof. Dr. Buchmann,
am Fachbereich 14 – Informatik –
der Universität des Saarlandes

Dezember 1994

Hiermit versichere ich an Eides Statt, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen verwendet habe.

Saarbrücken,

Ich danke Herrn Prof. Dr. Buchmann für die Vergabe des interessanten Themas.

Mein Dank gilt auch Thomas Denny für die kompetente Betreuung der Arbeit und für seine Hilfsbereitschaft.

Des weiteren bedanke ich mich bei Thomas Denny und Eric Schank für das Korrekturlesen der Arbeit und die vielen nützlichen Ratschläge.

Für hilfreiche Hinweise zum Quadratischen Sieb danke ich Peter Montgomery, Henk Boender und Georg Wambach.

Meinen Zimmerkollegen Klaus Kiefer, Thomas Papanikolaou, Ralf Roth, Damian Weber und Jörg Zayer danke ich für die angenehme Arbeitsatmosphäre.

Außerdem gilt mein Dank Franz-Dieter Berger, Frank Lehmann, Markus Maurer, Andreas Müller, Stefan Neis, Susanne Wetzel und Kay Wiese für zahlreiche Tips bei der Programmierung und bei der Erstellung dieser Arbeit.

Mein besonderer Dank gilt meinen Eltern für ihre Unterstützung.

Inhaltsverzeichnis

Einleitung

Die Faktorisierung natürlicher Zahlen ist ein Thema, das Mathematiker seit Jahrhundert beschäftigt. Carl Friedrich Gauß schrieb in Artikel 329 seiner "Untersuchungen über höhere Arithmetik" (Disquisitiones Arithmeticae) (siehe ([Gau65])):

> Daß die Aufgabe, die Primzahlen von den zusammengesetzten zu unterscheiden und letztere in ihre Primfaktoren zu zerlegen, zu den wichtigsten und nützlichsten der gesamten Arithmetik gehört und die Bemühungen und den Scharfsinn sowohl der alten wie auch der neueren Geometer in Anspruch genommen hat, ist so bekannt, daß es überflüssig wäre, hierüber viele Worte zu verlieren ... außerdem dürfte es die Würde der Wissenschaft erheischen, alle Hilfsmittel zur Lösung jenes so eleganten und berühmten Problems fleißig zu vervollkommnen ...

Das Faktorisierungsproblem ist nicht Bestandteil der reinen Mathematik geblieben, sondern hat mittlerweile auch Anwendungen gefunden. So existieren kryptographische Systeme, die zur Übertragung geheimer Nachrichten verwendet werden und deren Sicherheit gerade auf der Schwierigkeit, eine große Zahl in ihre Primfaktoren zu zerlegen, beruht.

Der Computer hat sich als eines der wichtigsten "Hilfsmittel" herausgestellt, von denen Gauß sprach. Durch seinen Einsatz sind viele Faktorisierungsalgorithmen erst praktikabel geworden. Sowohl die Anzahl der Rechenoperationen, die bei der Faktorisierung großer Zahlen durchgeführt werden, als auch die Größe der Datenmengen, die dabei anfallen können, sind beeindruckend. Bei der Faktorisierung einer 90-stelligen Zahl mit dem in dieser Arbeit vorgestellten Algorithmus, dem Quadratischen Sieb, fallen mehr als 200 MB Daten an. Würde man das Programm auf einer einzelnen Workstation laufen lassen, würde es mehr als einen Monat benötigen. Für eine 100-stellige Zahl erhöht sich diese Zeit etwa um den Faktor 8. Damit ist klar, daß bei Faktorisierungen von Zahlen dieser Größenordnung die Berechnungen auf mehrere Rechner verteilt werden müssen. In der Arbeitsgruppe von Professor Buchmann geschieht dies seit längerem durch die sogenannte $LiPS$-Bibliothek, die verteiltes Rechnen in einem Netz von Workstations ermöglicht (vgl. [LIPS91]).
Der erste Teil der vorliegenden Arbeit beschäftigt sich mit der Frage, wie die von Thomas Denny (vgl. ([Den93])) geschriebene Netz-Implementierung des Quadratischen Siebes sinnvoll auf einen Intel iPSC/860 Hypercube übertragen werden kann

und welche Besonderheiten dabei zu beachten sind.
Alford und Pomerance haben in [Al-Pom93] eine neue Methode zur Berechnung eines rechenintensiven Teilschrittes des Quadratischen Siebes vorgestellt. Im zweiten Teil dieser Arbeit wird die Implementierung dieser sogenannten *Selbstinitialisierungs-Variante* auf dem Hypercube besprochen und mit dem bisher üblichen Verfahren verglichen, das auf Silverman zurückgeht (vgl. [Sil87]).

Genauer gesagt gliedert sich die Arbeit folgendermaßen:

Kapitel 1:
Hier wird der Intel iPSC/860 Hypercube vorgestellt, auf dem das Quadratische Sieb implementiert wurde.

Kapitel 2:
In diesem Kapitel werden benutzte Bezeichungen, Definitionen und Sätze vorgestellt.

Kapitel 3:
In Kapitel 3 wird zunächst die prinzipielle Vorgehensweise des Quadratischen Siebes anhand der singlepolynomiellen Version dargelegt und darauf aufbauend Modifikationen des Grundalgorithmus erklärt. Als letzte Variante wird die sogenannte Selbstinitialisierung vorgestellt, die eine effizientere Berechnung der Polynome im Quadratischen Sieb erlaubt.

Kapitel 4:
Im Anschluß daran werden wichtige Erweiterungen eingeführt, die das Quadratische Sieb entscheidend beschleunigen. Dabei werden nur die wesentlichen Ideen dargestellt, da diese Erweiterungen unverändert aus der Implementierung von Thomas Denny (siehe ([Den93])) übernommen wurden.

Kapitel 5:
In diesem Abschnitt der Arbeit wird die Implementierung der zuvor geschilderten Algorithmen auf dem Hypercube vorgestellt.

Kapitel 6:
Es schließen sich Laufzeitvergleiche zwischen der Silverman- und der Selbstinitialisierungs-Methode an.

Kapitel 7:
In diesem Kapitel wird abschließend zum einen die Hypercube-Version des Quadratischen Siebes mit der Netz-Version und zum anderen die Selbstinitialisierungs-Methode mit der bisher üblicherweise verwendeten Methode von Silverman verglichen.

Kapitel 8:
Hier wird geschildert, wie der Programmaufruf des Quadratischen Siebes auf dem Hypercube aussieht.

Kapitel 9:

Im letzten Kapitel wird die Teilnahme an dem internationalen Faktorisierungswettbewerb der Firma "RSA Data Security" beschrieben sowie Beiträge zu einer Liste zu faktorisierender Zahlen einer bestimmten Form (erweitertes Cunningham-Projekt, vgl. [Bre-Rie92]) angeführt.

Anhang:

Im Anhang werden mathematische Eigenschaften der in Kapitel 9 eingeführten Partitions- und Cunningham-Zahlen vorgestellt.

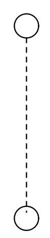
Kapitel 1

Der Hypercube

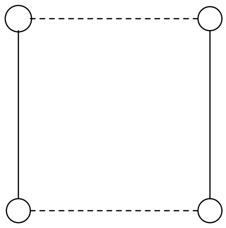
Bevor auf das Quadratische Sieb selbst eingegangen wird, wird in diesem Kapitel der Rechner vorgestellt, auf dem es implementiert wurde. Dazu erfolgt zunächst eine theoretische Beschreibung des n-dimensionalen Hypercubes. Daran schließen sich Anmerkungen zu dem konkreten, im Rechenzentrum der Universität des Saarlandes installierten Hypercube an.

Bei einem Hypercube handelt es sich um einen Parallelrechner, der dem *Multiple Instruction/ Multiple Data*-Konzept folgt. Das bedeutet, daß auf den einzelnen Prozessoren des Systems zur gleichen Zeit verschiedene Programme ablaufen können und daß jeder Prozessor einen eigenen Datenspeicher hat. Das System arbeitet nach dem *Client-Host*-Prinzip: es existiert ein Hostrechner, der Programme auf den Prozessoren der Clientrechner starten und abbrechen kann. Die Kommunikation zwischen dem Hostrechner und einem Clientrechner bzw. zwischen zwei Clientrechner erfolgt über Datenleitungen, die in einer spezifischen Weise gelegt sind. Diese hat dem gesamten System den Namen *Hypercube* (Hyperwürfel) gegeben. Die Lage der Datenleitungen macht man sich am besten über die induktive Konstruktion eines n-dimensionalen Würfels klar. Dabei entsprechen die Ecken des Würfels den Clientrechnern und die Kanten den Datenleitungen.

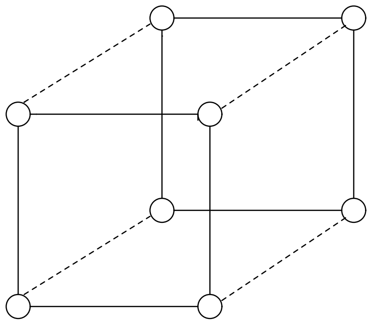
Ein Würfel der Dimension 0 besteht aus einem einzigen Knoten; es existieren keine Datenleitungen. Ein Würfel der Dimension 1 entsteht, indem man zwei Würfel der Dimension durch eine Kante (Leitung) verbindet.



Zwei Würfel der Dimension 1 bilden einen Würfel der Dimension 2, indem einander entsprechende Ecken der Würfel der Dimension 1 durch jeweils eine Kante verbunden werden. Somit ist ein Würfel der Dimension 2 ein Quadrat in der Ebene.



Verbindet man bei zwei solchen Würfeln der Dimension 2 einander entsprechende Ecken, so entsteht ein Würfel der Dimension 3, der der anschaulichen Vorstellung eines Würfels im $\mathbb{R}^3$ entspricht.



Für höherdimensionale Würfel fällt die geometrische Anschauung schwerer, die induktive Konstruktion läßt sich jedoch beliebig weit fortführen. Der n-dimensionale Würfel entsteht aus 2 Würfeln der Dimension $n-1$ durch das Verbinden einander entsprechender Ecken. Aus dieser induktiven Konstruktion folgt direkt, daß der n-dimensionale Hypercube 2^n Ecken hat und der längste Weg entlang der Kanten des Würfels n Schritte benötigt. Deshalb muß eine Nachricht im Hypercube im schlechtesten Falle n Datenleitungen durchlaufen.
Das Verschicken einer Nachricht ist, verglichen mit arithmetischen Operationen, sehr zeitaufwendig. Dies liegt daran, daß schon beim Verschicken lediglich eines Bits das Betriebssystem intern nicht nur einmal, sondern mehrmals eine Verbindung zwischen dem sendenden und dem empfangenden Knoten aufbauen muß. Aus diesem Grund besteht bei der Parallelisierung von Problemstellungen oft eine der wichtigsten Aufgaben darin, Anzahl und Umfang dieser Kommunikationsverbindungen möglichst gering zu halten. Wie man sehen wird, ist es bei der Parallelisierung des Quadratischen Siebes sogar möglich, nach einer Anlaufphase zu Beginn der Berechnungen vollkommen auf das Verschicken von Nachrichten zu verzichten, d.h. die Knoten asynchron arbeiten zu lassen. Das spart zum einen den Zeitaufwand für die Synchronisation (d.h. das zeitliche Aufeinanderabstimmen) der Prozessoren und

vermeidet zum anderen idle time (Leerlauf). Diese würde entstehen, wenn ein Prozessor auf das Ende der Teilberechnung eines anderen Prozessors warten müßte, weil er ein Datum von diesem benötigte.

Bei dem konkreten Parallelrechner, der im Rechenzentrum der Universität des Saarlandes installiert wurde, handelt es sich um einen iPSC/860 Hypercube der Firma Intel. Er besteht aus den folgenden Komponenten:

- Der *Host* oder *System Resource Manager (SRM)* ist eine Einprozessormaschine (Intel 80386) mit eigenem Plattenspeicher.
- Es gibt insgesamt 32 i860-Prozessoren, 4 davon mit 16 MB Hauptspeicher, die restlichen mit 8 MB. Sie werden im folgenden als *Knotenrechner* oder einfach *Knoten* bezeichnet.
- Es existiert ein gemeinsamer Plattenspeicher, auf den alle Knoten zugreifen können. Auf diesem *Concurrent File System (cfs)* sind insgesamt über 4 GByte verfügbar. Für den Benutzer erfolgt der Zugang über eine sogenannte *Node Shell.*

Der Benutzer kann Teilwürfel der Dimensionen 0 bis 5 mit $1, 2, 4, 8, 16$ oder 32 Knoten allokieren. Ein solcher Teilwürfel wird im folgenden mit "Cube" bezeichnet.

Auf die Benutzung des Hypercubes wird in Kapitel 8 näher eingegangen. Für weitere Details sei auf die umfangreichen Handbücher (siehe [iPSC91]) verwiesen.

Kapitel 2

Grundlagen

In diesem Kapitel werden benutzte Bezeichnungen, Definitionen und Sätze vorgestellt. Der später benötigte Gray-Code wird ausführlich dargestellt.

2.1 Bezeichnungen

Für die in dieser Arbeit auftretenden Zahlbereiche werden die folgenden Bezeichnungen verwendet:

$\mathbb{N}$ Menge der natürlichen Zahlen (ohne Null),

$\mathbb{Z}$ Ring der ganzen Zahlen,

$\mathbb{Q}$ Körper der rationalen Zahlen,

$\mathbb{R}$ Körper der reellen Zahlen,

$\mathbb{F}_p$ Körper mit p Elementen, p Primzahl (s.u.).

Neben diesen mathematischen Begriffen werden die folgenden Bezeichnungen für Datentypen verwendet: Mit "lange Zahl" wird eine ganze Zahl bezeichnet, die nicht mehr im normalen Datentyp Integer (32 Bit) abgespeichert werden kann. Für diese Zahlen existiert eine spezielle Datenstruktur in der verwendeten langen Arithmetik (vgl. [Dentz91]).
Kann eine ganze Zahl im Datentyp Integer abgespeichert werden, wird sie als "kurze Zahl" oder "int" bezeichnet.

2.2 Benutzte Definitionen und Sätze

Definition 2.1
Für $x \in \mathbb{R}$ bezeichnet man mit $\lfloor x \rfloor$ die größte ganze Zahl a mit $a \leq x$.

Definition 2.2 (Teilbarkeit, Primzahl)
Seien $a, n \in \mathbb{Z}$. Man definiert

$$a|n \overset{def.}{\iff} \exists b \in \mathbb{Z} : a \cdot b = n.$$

Man sagt "a teilt *n", "a ist* Teiler *oder* Faktor *von n". Gilt nicht $a|n$, so schreibt man $a \not| n$.*
Ist $n \in \mathbb{N}$, $1 < a < n$ und gilt $a|n$, so heißt a ein echter *oder* nichttrivialer *Faktor von n.*
Eine Zahl $1 < p \in \mathbb{N}$ heißt Primzahl, *wenn p keine echten Teiler hat. Die Menge aller Primzahlen bezeichnet man mit $\mathbb{P}$.*
Für $p \in \mathbb{P}, n, \alpha \in \mathbb{N}$ definiert man das Symbol $\|$ durch

$$p^\alpha \| n \overset{def.}{\iff} p^\alpha | n \text{ und } p^{\alpha+1} \not| n.$$

Der größte gemeinsame Teiler *zweier Zahlen $a, b \in \mathbb{N}$ - in Zeichen $ggt(a, b)$ - ist die Zahl $c \in \mathbb{N}$ mit*

$$c|a \text{ , } c|b \text{ und aus } c'|a, c'|b \text{ für ein } c' \in \mathbb{N} \text{ folgt } c'|c.$$

Zwei Zahlen $a, b \in \mathbb{N}$ heißen teilerfremd, *wenn $ggt(a, b) = 1$ gilt.*

Satz 2.3 (Fundamentalsatz der Arithmetik)
Jede natürliche Zahl $n > 1$ läßt sich auf (bis auf die Reihenfolge der Faktoren) eindeutige Weise als Produkt von Primzahlen schreiben. Mit anderen Worten: in der Darstellung

$$n = \prod_{i=1}^{r} p_i^{e_i} \tag{2.1}$$

ist $r \in \mathbb{N}$ eindeutig und die $p_i \in \mathbb{P}$ und $e_i \in \mathbb{N}$ sind bis auf die Reihenfolge eindeutig bestimmt.

Definition 2.4 (Kongruenz)
Seien $n \in \mathbb{N}, x, y \in \mathbb{Z}$. Man definiert

$$x \equiv y \bmod n \overset{def.}{\iff} n|(x - y)$$

und sagt: x ist kongruent *zu y* modulo *n.*
Der kleinste positive Rest bei ganzzahliger Division von x durch n wird mit $x \bmod n$ bezeichnet.

Satz 2.5 (Chinesischer Restsatz)

Für ein System von k Kongruenzen

$$x \equiv a_i \bmod n_i \qquad\qquad (1 \leq i \leq k)$$

mit $n_i, k \in \mathbb{N}$, $a_i \in \mathbb{Z}$ gilt:
Sind die n_i paarweise teilerfremd, so ist das System lösbar, und die Lösung

$$x := \sum_{i=1}^{k} a_i \cdot \left(\left(\frac{n}{n_i}\right)^{-1} \bmod n_i\right) \cdot \frac{n}{n_i} \bmod n$$

ist modulo $n := \prod_{i=1}^{k} n_i$ eindeutig bestimmt.

Beweis: [Ries87, S.268] ∎

Definition 2.6 (Legendre-Symbol)
Sei p eine ungerade Primzahl. Für $a \in \mathbb{Z}$ definiert man

$$\left(\frac{a}{p}\right) = \begin{cases} -1, & \text{falls } p \not| a \text{ und } a \text{ ist nicht Quadrat modulo } p, \\ 0, & \text{falls } p \mid a, \\ 1, & \text{falls } p \not| a \text{ und } a \text{ ist Quadrat modulo } p. \end{cases}$$

$\left(\frac{a}{p}\right)$ *heißt* Legendre-Symbol *von a und p.*

Das Legendre-Symbol kann effizient unter Verwendung des folgenden Satzes berechnet werden.

Satz 2.7 (Euler-Kriterium)
Sei p eine ungerade Primzahl. Für $a \in \mathbb{Z}$ mit $p \not| a$ gilt

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \bmod p.$$

Beweis: [Har-Wri79, Theorem 83] ∎

Definition 2.8
Sei $n \in \mathbb{N}$. Eine Partition von n *ist eine Darstellung von n als Summe natürlicher Zahlen :*

$$n = a_1 + \cdots + a_k \qquad\qquad \text{mit } k \in \mathbb{N}, a_i \in \mathbb{N} \text{ für } 1 \leq i \leq k.$$

Die Partitionszahl $p(n)$ *ist die Anzahl der Partitionen von n, wobei Umordnungen der Summanden nicht mitgezählt werden.*

Beispiel 2.9 Es ist $p(5) = 7$, denn

$$\begin{aligned} 5 &= 1+1+1+1+1 \\ &= 1+1+1+2 \\ &= 1+2+2 \\ &= 1+1+3 \\ &= 1+4 \\ &= 2+3 \\ &= 5 \end{aligned}$$

und dies sind alle Möglichkeiten.

Definition 2.10 (O-Notation)
Seien $f, g : \mathbb{N} \to \mathbb{N}$ Funktionen. Man definiert

$$\begin{aligned} g = O(f) &\overset{def.}{\Longleftrightarrow} \exists c \in \mathbb{R}_{>0} \text{ mit } g(n) \leq c \cdot f(n) \text{ für alle } n \in \mathbb{N}, \\ g = \Theta(f) &\overset{def.}{\Longleftrightarrow} \lim_{n\to\infty} \frac{g(n)}{f(n)} = 1. \end{aligned}$$

Definition 2.11 (ρ-Funktion)
Sei $T \in \mathbb{N}$ fest. Für eine zufällig gewählte Zahl $n \in [1 \ldots T]$ mit Primfaktorzerlegung

$$n = \prod_{i=1}^{r} p_i^{e_i}, \qquad p_1 > p_2 > \ldots > p_r \tag{2.2}$$

und $\alpha \geq 1$ ist $\rho(\alpha)$-Funktion definiert als der Grenzwert der Wahrscheinlichkeit Prob, daß der größte Primfaktor p_1 von n kleiner als $T^{\frac{1}{\alpha}}$ ist:

$$\rho : \{\alpha | \alpha \in \mathbb{R}, \alpha \geq 1\} \longrightarrow [0,1]$$

$$\alpha \longmapsto \rho(\alpha) := \lim_{T\to\infty} Prob(p_1 < T^{\frac{1}{\alpha}})$$

Die ρ-Funktion erfüllt die Differentialgleichung

$$\alpha\rho'(\alpha) + \rho(\alpha - 1) = 0$$

und kann durch numerische Integration aus

$$\rho(\alpha) = \begin{cases} 1, & \text{für } 0 \leq \alpha \leq 1 \\ \dfrac{1}{\alpha} \displaystyle\int_{\alpha-1}^{\alpha} \rho(t)dt, & \text{für } \alpha > 1 \end{cases}$$

berechnet werden. (siehe [Knu-Par76, S.333])

Für $5 < \alpha < 11$ kann folgende Approximation benutzt werden ([Pom82, S.34]) :

$$\rho(\alpha) \approx \exp\left(-\alpha \cdot \left(\log(\alpha) + 0.56 - \frac{1}{\log(\alpha)}\right)\right). \tag{2.3}$$

2.3 Gray-Codes

Definition 2.12 (Hamming-Abstand)
Für $n \in \mathbb{N}$ seien $x = x_1 \dots x_n, y = y_1 \dots y_n \in \{0,1\}^n$. Dann bezeichnet

$$h(x,y) := \#\{i \in \{1, \dots n\} | x_i \neq y_i\}$$

den Hamming-Abstand *von x und y.*

Definition 2.13 (Gray-Code)
Sei $n \in \mathbb{N}$. Eine Folge $g := (g(0), \dots, g(2^n - 1))$ mit $g(i) \in \{0,1\}^n$ für $0 \leq i \leq 2^n - 1$ heißt Gray-Code der Länge n *genau dann, wenn*

1. *$h(g(i), g(j)) > 0$ für $0 \leq i, j \leq 2^n - 1$ mit $i \neq j$,*
2. *$h(g(i), g(i+1)) = 1$ für $0 \leq i \leq 2^n - 2$*

gilt.

Es ist nicht auf Anhieb klar, daß eine solche Folge überhaupt existiert. Man kann ihre Existenz jedoch sogar konstruktiv beweisen. Die folgende induktive Konstruktion orientiert sich im wesentlichen an der Beweisskizze in [Nij-Wi78, Kap.1]:

Behauptung 2.14

1. *$(0,1)$ ist Gray-Code der Länge $n = 1$.*
2. *Für $n \in \mathbb{N}, n > 1$, sei $(g(0), \dots, g(2^{n-1} - 1))$ ein Gray-Code der Länge $n - 1$. Das i-te Folgenglied $(0 \leq i \leq 2^{n-1} - 1)$ sei $g(i) = e_{n-1}^{(i)} \dots e_1^{(i)}$, wobei $e_j^{(i)} \in \{0,1\}$ für $1 \leq j \leq n-1$ gelte. (Man nennt $e_j^{(i)}$ das j-te Bit des i-ten Folgengliedes.) Setze*

$$g'(i) := \begin{cases} 0e_{n-1}^{(i)} \dots e_1^{(i)} & \text{für } 0 \leq i \leq 2^{n-1} - 1 \\ 1e_{n-1}^{(2^n-i-1)} \dots e_1^{(2^n-i-1)} & \text{für } 2^{n-1} \leq i \leq 2^n - 1 \end{cases} . \tag{2.4}$$

Dann ist $g' := (g'(0) \dots, g'(2^n - 1))$ ein Gray-Code der Länge n.

Beweis:

1. klar

2. (a) Zunächst wird die Gültigkeit der ersten Forderung aus Definition 2.13 indirekt bewiesen.
Annahme: Für $g'(i) = \tilde{e}_n^{(i)} \dots \tilde{e}_1^{(i)}, g'(j) = \tilde{e}_n^{(j)} \dots \tilde{e}_1^{(j)}$ gilt $h(g'(i), g'(j)) = 0$. Dann folgt aus der Definition des Hamming-Abstandes:
$$\tilde{e}_n^{(i)} \dots \tilde{e}_1^{(i)} = \tilde{e}_n^{(j)} \dots \tilde{e}_1^{(j)}. \qquad (*)$$
Insbesondere ist $\tilde{e}_n^{(i)} = \tilde{e}_n^{(j)}$. Die beiden möglichen Fälle werden nacheinander betrachtet.

Fall 1: $\tilde{e}_n^{(i)} = 0$.
Nach der Definition von g' gilt dann
$$\begin{array}{lll} 0 \leq i \leq 2^{n-1} - 1 & \text{und} & \tilde{e}_{n-1}^{(i)} \dots \tilde{e}_1^{(i)} = e_{n-1}^{(i)} \dots e_1^{(i)} = g(i), \\ 0 \leq j \leq 2^{n-1} - 1 & \text{und} & \tilde{e}_{n-1}^{(j)} \dots \tilde{e}_1^{(j)} = e_{n-1}^{(j)} \dots e_1^{(j)} = g(j). \end{array}$$
Wegen $(*)$ folgt $g(i) = g(j)$ und somit $h(g(i), g(j)) = 0$. Dies ist ein Widerspruch, da g nach Voraussetzung Gray-Code ist.

Fall 2: $\tilde{e}_n^{(i)} = 1$.
Dann gilt nach Definition von g'
$$\begin{array}{l} 2^{n-1} \leq i \leq 2^n - 1 \text{ und } \tilde{e}_{n-1}^{(i)} \dots e_1^{(i)} = e_{n-1}^{(2^n-i-1)} \dots e_1^{(2^n-i-1)} = g(2^n - i - 1), \\ 2^{n-1} \leq j \leq 2^n - 1 \text{ und } \tilde{e}_{n-1}^{(j)} \dots e_1^{(j)} = e_{n-1}^{(2^n-j-1)} \dots e_1^{(2^n-j-1)} = g(2^n - j - 1). \end{array}$$
Aus $(*)$ folgt $g(2^n-i-1) = g(2^n-j-1)$ und $h(g(2^n-i-1), g(2^n-j-1)) = 0$, wiederum im Widerspruch zu der Voraussetzung, daß g Gray-Code ist.

   (b) Nun zu der zweiten Forderung aus Definition 2.13.
Sei $0 \leq i \leq 2^{n-1} - 2$.
Dann gilt
$$\begin{aligned} h(g'(i), g'(i+1)) &= h(0e_{n-1}^{(i)} \dots e_1^{(i)}, 0e_{n-1}^{(i+1)} \dots e_1^{(i+1)}) \\ &= h(e_{n-1}^{(i)} \dots e_1^{(i)}, e_{n-1}^{(i+1)} \dots e_1^{(i+1)}) \\ &= h(g(i), g(i+1)) \\ &= 1, \end{aligned}$$
da g Gray-Code ist. Analog gilt für $2^{n-1} \leq i \leq 2^n - 2$:
$$h(g'(i), g'(i+1)) = h(g(2^n - i - 1), g(2^n - i)) = 1.$$
Es bleibt der Fall $i = 2^{n-1} - 1$ zu betrachten. Nach Konstruktion gilt
$$\begin{aligned} g'(i) &= 0e_{n-1}^{(2^{n-1}-1)} \dots e_1^{(2^{n-1}-1)} \\ g'(i+1) &= 1e_{n-1}^{(2^n-2^{n-1}-1)} \dots e_1^{(2^n-2^{n-1}-1)}. \end{aligned}$$
Wegen $2^n - 2^{n-1} - 1 = 2^{n-1} - 1$ folgt $h(g'(i), g'(i+1))$=1.

■

Beispiel 2.15
$g = (0, 1)$ ist Gray-Code der Länge 1 mit $g(0) = 0, g(1) = 1$.
Gemäß der Zuweisung (2.4) bildet man

$$\begin{aligned} g'(0) &:= 0g(0) = 00 \\ g'(1) &:= 0g(1) = 01 \\ g'(2) &:= 1g(1) = 11 \\ g'(3) &:= 1g(0) = 10. \end{aligned}$$

Dann ist $g' = (00, 01, 11, 10)$ Gray-Code der Länge 2. Setzt man das Verfahren fort, ergibt sich $(000, 001, 011, 010, 110, 111, 101, 100)$ als Gray-Code der Länge 3.

Wie erhält man nun algorithmisch einen Gray-Code vorgegebener Länge? Um diese Frage zu beantworten, untersucht man zunächst, welches Bit beim Übergang eines Folgengliedes zum nächsten geändert wird. Aufschluß darüber gibt der folgende Satz:

Satz 2.16
Sei $n \in \mathbb{N}$ und g' der durch obige Konstruktion gebildete Gray-Code der Länge n. Sei weiterhin $1 \le i \le 2^n - 1$ und $g'(i-1) = \tilde{e}_n^{(i-1)} \ldots \tilde{e}_1^{(i-1)}$ das $(i-1)$-te Folgenglied. Dann wird beim Übergang zum i-ten Folgenglied das ν-te Bit $\tilde{e}_\nu^{(i-1)}$ geändert, wobei

$$\nu := l + 1 \qquad (2^l \parallel i) \tag{2.5}$$

gilt.

Beweis: (durch Induktion über n)
$\underline{n = 1}$:
Somit ist $i = 1$. $(0, 1)$ ist Gray-Code der Länge 1, also wird das erste Bit geändert. In der Tat gilt nach Vorschrift (2.5) $l = 0$, also $\nu = 0 + 1 = 1$.

$\underline{n - 1 \rightarrow n}$:
Betrachte den konstruierten Code g' der Länge n. Für die Übergänge $g'(0) \rightarrow g'(1), g'(1) \rightarrow g'(2), \ldots g'(2^{n-1}-2) \rightarrow g'(2^{n-1}-1)$, sowie $g'(2^{n-1}) \rightarrow g'(2^{n-1}+1), \ldots g'(2^n-2) \rightarrow g'(2^n-1)$ folgt die Behauptung nach Induktionsvoraussetzung.

Es bleibt somit nur der Übergang $g'(2^{n-1}-1) \rightarrow g'(2^{n-1})$ zu untersuchen. Für $2^{n-1}-1$ gilt nach Gleichung (2.4):

$$\begin{aligned} g'(2^{n-1}-1) &= 0e_{n-1}^{(2^{n-1}-1)} \dots e_1^{(2^{n-1}-1)} \\ \text{und } g'(2^{n-1}) &= 1e_{n-1}^{(2^n-2^{n-1}-1)} \dots e_1^{(2^n-2^{n-1}-1)} \\ &= 1e_{n-1}^{(2^{n-1}-1)} \dots e_1^{(2^{n-1}-1)} \end{aligned}$$

Somit wird bei diesem Übergang das n-te Bit gekippt. Wegen $i = 2^{n-1}$ und $2^l \parallel i$ liefert die Formel (2.5) in der Tat $l = n - 1$, also $\nu = l + 1 = n$. ■

Nachdem nun geklärt wurde, welches Bit beim Übergang von $g'(i-1)$ zu $g'(i)$ zu ändern ist, muß nur noch festgestellt werden, ob dieses Bit von 0 nach 1 oder von 1 nach 0 geändert wird. Dazu untersucht man, wie oft das Bit ν vor dem Schritt i geändert wurde. Aus der Gleichung (2.5) folgt, daß das ν-te Bit genau dann gekippt wird, wenn

$$2^l = 2^{\nu-1} \parallel k$$

galt für eine Zahl $1 \leq k < i$.

Für diese Zahl k gilt dann

$$k = 2^{\nu-1} \cdot (2 \cdot k' - 1)$$

(mit $k' \in \mathbb{N}$), und jedes k von dieser Form erfüllt die Bedingung $2^{\nu-1} \parallel k$. Somit ist die Gesamtanzahl der bisherigen Änderungen des ν-ten Bits gleich der Anzahl der Zahlen $k' \in \mathbb{N}$ mit

$$(2 \cdot k' - 1) \cdot 2^{\nu-1} < i \qquad\qquad (k' \geq 1)$$

also

$$t := \frac{1}{2} \cdot (\frac{i}{2^{\nu-1}} + 1) - 1 \ .$$

$g'(0)$ wird zu Beginn der Konstruktion immer auf eine Folge von n Nullen gesetzt. Deshalb ist zu Beginn des Zählens das ν-te Bit 0 und hat vor der Änderung in Schritt i den Wert

$$t \bmod 2 \ ,$$

muß also nach der Änderung in Schritt i den Wert

$$t + 1 \equiv \frac{1}{2} \cdot (\frac{i}{2^{\nu-1}} + 1) \bmod 2$$

annehmen.

Kapitel 3

Übersicht über das Quadratische Sieb

In diesem Kapitel wird zunächst die prinzipielle Vorgehensweise des Quadratischen Siebes (QS) vorgestellt. Darauf aufbauend werden die *Single Polynomial-*, die *Multi Polynomial-* und die *Selbstinitialisierungs-*Variante vorgestellt.

Im allgemeinen unterteilt man Faktorisierungsalgorithmen in zwei Klassen:

- Spezielle Faktorierungsalgorithmen
  Ihre Laufzeit hängt von besonderen Eigenschaften der zu faktorisierenden Zahl, z.B. der Form ihrer Primfaktoren, ab. Beispiele sind die Elliptische Kurven-Methode (ECM), das spezielle Number Field Sieve (SNFS) und Pollards p-1-Algorithmus.

- Allgemeine Faktorisierungsalgorithmen
  Ihre Laufzeit hängt im wesentlichen von der Länge der zu faktorisierenden Zahl ab. Das Quadratische Sieb (QS), das Thema dieser Arbeit ist, gehört ebenso zu dieser Klasse wie das allgemeine Number Field Sieve (GNFS) und die Continued Fractions-Methode (CFRAC).

Die erwähnten allgemeinen Algorithmen basieren alle auf einer einfachen Grundidee, die nun vorgestellt wird.

3.1 Die Grundidee

Sei im folgenden $N \in \mathbb{N}$ die zu faktorisierende Zahl.

Bemerkung 3.1 Im folgenden wird immer davon ausgegangen, daß N keine Primzahl ist. Dies läßt sich mit dem probabilistischen Primzahltest von Miller-Rabin (siehe ([Coh91, Kap. 8.2])) effizient prüfen. Ein exakter Beweis der Primzahleigenschaft ist aufwendiger, kann aber für Zahlen im Bereich von 100 Stellen mit dem Algorithmus von Goldwasser-Kilian-Atkin ([Bu-Mue92]) in wenigen Minuten durchgeführt werden.

Grundidee

Findet man $X, Y \in \mathbb{Z}$ mit

$$\begin{aligned} X &\not\equiv \pm Y \bmod N \\ \text{und } X^2 &\equiv Y^2 \bmod N\ , \end{aligned}$$

so hat man einen nichttrivialen Teiler von N gefunden.

Warum gilt das?

$$X^2 \equiv Y^2 \bmod N \tag{3.1}$$

ist äquivalent zu

$$N|(X+Y)\cdot(X-Y). \tag{3.2}$$

Da nun wegen $X \not\equiv \pm Y \bmod N$ weder $N|(X+Y)$ noch $N|(X-Y)$ gilt, verteilen sich die Primfaktoren von N auf beide Faktoren in Formel (3.2), und der größte gemeinsame Teiler von N und (X-Y) ist ein echter Teiler von N.

Um eine Kongruenz der Form (3.1) zu finden, geht man folgendermaßen vor:

Man sucht Kongruenzen r_j der Form

$$r_j: \qquad X_j^2 \equiv Y_j \bmod N. \tag{3.3}$$

Eine solche Kongruenz wird *Relation* genannt, wenn die Primfaktorzerlegung der Y_j bekannt ist:

$$Y_j = \prod_{p_i \in \mathbb{P}} p_i^{e_{i,j}}. \tag{3.4}$$

Man bestimmt zunächst eine große Menge R von Relationen. Dann löst man das Gleichungssystem

$$\sum_{j=1}^{|R|} e_{i,j} \cdot \lambda_j \equiv 0 \bmod 2 \tag{3.5}$$

über $\mathbb{F}_2$ und berechnet die Zahl

$$Y' := \prod_{j=1}^{|R|} Y_j^{\lambda_j} = \prod_{p_i \in \mathbb{P}} p_i^{\sum_{j=1}^{|R|} e_{i,j} \cdot \lambda_j} \tag{3.6}$$

Die $\lambda_i \in \{0, 1\}$ geben somit an, welche der Relationen aus R in Y' auftreten.

Nach Konstruktion (siehe Gleichung (3.5)) kommt in Y' jede Primzahl in gerader Potenz vor (die $\lambda_i \in \{0, 1\}$ geben an, welche der Relationen aus R in Y' auftreten). Somit ist Y' ein Quadrat: $Y' = Y^2$ für ein $Y \in \mathbb{N}$. Bildet man

$$X^2 := \prod_{j=1}^{|R|} (X_j^{\lambda_j})^2,$$

so gilt wegen Gleichung (3.3)

$$X^2 \equiv Y^2 \bmod N$$

und eine Kongruenz der Form (3.1) ist gefunden. Gilt $X \not\equiv \pm Y$, so ist $\mathrm{ggt}(N, X - Y)$ ein echter Teiler von N. Ansonsten bestimmt man eine weitere Lösung des Gleichungssystems (3.5). (Dazu muß R eventuell vergrößert werden.)

Damit in der Formel (3.4) nicht beliebig große Primzahlen auftreten können und das Gleichungssystem endlich ist, läßt man nur Primzahlen bis zu einer Schranke B_N zu. Diese Menge wird als *Faktorbasis* bezeichnet. Man setzt

$$FB := \{p \in \mathbb{P} |\, p \leq B_N\}. \tag{3.7}$$

Zusammenfassend kann man sagen: jeder der oben erwähnten allgemeinen Faktorisierungsalgorithmen Faktorisierungsalgorithmen besteht aus zwei Phasen:

1. Wahl einer Faktorbasis FB;
   Finden von mehr als $|FB|$ vielen Relationen

   $$r_j : \qquad X_j^2 \equiv Y_j \bmod N \tag{3.8}$$

   mit bekannter Primfaktorzerlegung

   $$Y_j = \prod_{p_i \in FB} p_i^{e_{i,j}}. \tag{3.9}$$

2. Aufstellen und Lösen eines linearen Gleichungssystems über $\mathbb{F}_2$.

Auf das Lösen des linearen Gleichungssystem wird hier nicht weiter eingegangen. Der Rest der vorliegenden Arbeit beschäftigt sich mit der Frage, wie die Relationen (3.8) in Phase 1 möglichst effizient gefunden werden können. In der Art und Weise, wie dies geschieht, liegt auch der einzige Unterschied zwischen den drei Algorithmen QS, GNFS und CFRAC. Im Quadratischen Sieb werden die Relationen durch die Faktorisierung von Funktionswerten quadratischer Polynome berechnet. Im folgenden Abschnitt wird zunächst die sogenannte *Single Polynomial*-Version des Quadratischen Siebes vorgestellt. Obwohl sie in der Praxis an Bedeutung verloren hat, bildet sie die Grundlage für alle Erweiterungen, und die wesentlichen Ideen lassen sich an ihr gut aufzeigen.

3.2 Die *Single Polynomial*-Version des QS

Zur Generierung von Relationen (3.8) schlug Pomerance (vgl. ([Pom82])) vor, das Polynom

$$f(z) = (z + \lfloor\sqrt{N}\rfloor)^2 - N$$

zu betrachten.
Nach Konstruktion gilt für alle $z_j \in \mathbb{Z}$:

$$f(z_j) \equiv X_j^2 \bmod N$$

für ein $X_j \in \mathbb{Z}$.

Da die Funktionswerte $f(z_j)$ die Y_j in der Kongruenz (3.3) sein sollen, benötigt man ihre Faktorisierung. Deshalb sollten die Funktionswerte gegenüber der ursprünglich zu faktorisierenden Zahl N klein sein. Dies ist tatsächlich der Fall, denn wegen

$$|f(z)| < z^2 + 2 \cdot |z| \cdot \lfloor\sqrt{N}\rfloor$$

gilt für $|z| \ll \sqrt{N}$:

$$|f(z)| = O(\sqrt{N}).$$

Man erhält Funktionswerte, indem man sich ein Intervall $I = [-M, M]$ wählt und nacheinander für $z_j \in I$ den Wert $f(z_j)$ berechnet. Um diese $f(z_j)$ zu faktorisieren, verwendet man nicht die zeitaufwendige Probedivision, sondern eine effizientere Methode, die dem ganzen Algorithmus den Namen gegeben hat: Man benutzt ein Siebverfahren, das auf der folgenden Beobachtung beruht.

Satz 3.2
Sei $f \in \mathbb{Z}[z]$, $p \in \mathbb{P}$. Dann gilt

$$f(z + k \cdot p) \equiv f(z) \bmod p$$

für alle $k \in \mathbb{Z}$.

Beweis:
Sei $f(z) = a_0 + \ldots + a_n \cdot z^n,\ a_n \neq 0,\ n \geq 1$ (für $n = 0$ ist die Behauptung klar). Also:

$$\begin{aligned}
f(z + k \cdot p) &\equiv \sum_{i=0}^{n} a_i \cdot (z + k \cdot p)^i \bmod p \\
&= \sum_{i=0}^{n} a_i \cdot \left(\sum_{j=0}^{i} \binom{i}{j} \cdot z^j \cdot (k \cdot p)^{i-j} \right) \\
&\overset{(\star)}{\equiv} \sum_{i=0}^{n} a_i \cdot \left(\underbrace{\binom{i}{i}}_{=1} \cdot z^i \right) \bmod p \\
&\equiv f(z) \bmod p
\end{aligned}$$

wobei $(\star)$ gilt, weil $(k \cdot p)^{i-j} \equiv 0 \bmod p \Longleftrightarrow j \neq i$ gilt. ■

Insbesondere gilt für jedes Polynom $f \in \mathbb{Z}[z]$:

Sei $p \in FB$ und $z_0 \in \mathbb{N}$ mit $f(z_0) \equiv 0 \bmod p$

$$\begin{aligned}
&\Rightarrow\ f(z_0 + k \cdot p) \equiv 0 \bmod p \qquad \text{für alle } k \in \mathbb{Z} \\
&\Leftrightarrow\ p | f(z_0 + k \cdot p) \qquad (3.10)
\end{aligned}$$

Hat man für eine Primzahl p aus der Faktorbasis die (i.a.) zwei Nullstellen des Polynoms modulo p bestimmt, weiß man bereits, an welchen Stellen des Intervalls I die Funktionswerte durch p teilbar sind. Damit spart man bei allen anderen Funktionswerten die Probedivision durch p.
Somit geht man folgendermaßen vor: Für jedes $p \in FB$ werden die Nullstellen z_1, z_2 des Polynoms bestimmt. Für alle $z \in I$ mit $z = z_1 + k \cdot p$ oder $z = z_2 + k \cdot p$ für ein $k \in \mathbb{Z}$ merkt man sich, daß der Funktionswert $f(z)$ von p geteilt wird. Diesen Vorgang nennt man "Sieben mit der Primzahl p". Ist mit allen Faktorbasiselementen gesiebt worden, schließt sich eine Testphase an. In ihr wird anhand der während der Siebphase gesammelten Informationen überprüft, welche Funktionswerte $f(z)$ vollständig über der Faktorbasis zerfallen. Diese Funktionswerte und ihre Primfaktorzerlegungen werden zur Bildung von Relationen (3.8) genutzt.

3.3 Die *Multi Polynomial*-Version des QS

Die *Single Polynomial*-Version des QS hat einen großen Nachteil. Sind nämlich nicht genügend Faktorisierungen von Funktionswerten über der Faktorbasis gefunden worden, hat man in der Praxis keine andere Wahl, als das Siebintervall I zu vergrößern.

Das hat zur Folge, daß die Funktionswerte bei wachsendem z größer werden und die Wahrscheinlichkeit, daß sie über der fest gewählten Faktorbasis zerfallen, sinkt.

Die im folgenden beschriebene Idee, anstelle des Siebintervalls das Polynom zu wechseln, stammt von Peter Montgomery und wurde von Robert D. Silverman (vgl. ([Sil87])) weiter ausgeführt. Der resultierende Algorithmus wird als *Multi Polynomial Quadratic Sieve (MPQS)* bezeichnet.

Man verwendet dabei den allgemeinsten Ansatz für Polynome 2. Grades:

$$f_{A,B,C}(z) = Az^2 + Bz + C.$$

Einige Umformungen ergeben hinreichende Bedingungen dafür, daß alle Funktionswerte Quadrate modulo N sind:

$$\begin{aligned} f_{A,B,C}(z) &= Az^2 + B \cdot z + C \\ &= A\left(z^2 + \frac{B}{A}z\right) + C \\ &= A\left(z + \frac{B}{2A}\right)^2 - \frac{B^2}{4A} + C \\ &= A\left(z + \frac{B}{2A}\right)^2 - \frac{B^2 - 4AC}{4A} \end{aligned} \tag{3.11}$$

Falls also

$$\begin{aligned} A &\equiv D^2 \bmod N && \text{für ein D} \in \mathbb{Z} \text{ und} \\ B^2 - 4AC &\equiv 0 \mod N \end{aligned}$$

gilt, ist jeder Funktionswert $f_{A,B,C}(z)$ ein Quadrat modulo N. Setzt man

$$\begin{aligned} Y &:= f_{A,B,C}(z) \bmod N, \\ X &:= D\left(z + \frac{B}{2A}\right) \bmod N, \end{aligned}$$

so hat man eine Relation (3.3) gefunden, sobald die Faktorisierung des Funktionswertes $f_{A,B,C}(z)$ ermittelt worden ist.

Eine Bemerkung zur Notation: Sind A und B vorgegeben, so ist C modulo N eindeutig bestimmt. Deshalb wird das Polynom $f_{A,B,C}$ im folgenden auch mit $f_{A,B}$ bezeichnet.

Beim Berechnen der Koeffizienten ist die zusätzliche Nebenbedingung zu beachten, daß die Funktionswerte über dem gewählten Siebintervall $I = [-M, M]$ möglichst klein bleiben sollten. Dazu gibt es verschiedene Ansätze:

Minimiere

- $\max_{z \in I} |f(z)|$
- $\int_{-M}^{M} |f(z)|\, dz$
- $\int_{-M}^{M} |f(z)^2|\, dz$

Alle Ansätze laufen im wesentlichen auf die folgende Wahl der Koeffizienten im Reellen hinaus (vgl. [Sil87, S. 331]):

$$A \approx \frac{\sqrt{N}}{M},\ B = 0,\ C \approx -\sqrt{N} \cdot M.$$

Bei dieser Wahl sind die Beträge der Funktionswerte an der Stelle 0 und an den Rändern des Intervalls gleich groß und maximal, nämlich etwa $\sqrt{\frac{N}{2}} \cdot M$.

Bis zu diesem Punkt stellt sich die *Multi Polynomial*-Variante des QS folgendermaßen dar:

- Aufstellen der Faktorbasis FB;
- Schleife:
  solange die Anzahl der gefundenen Relationen kleiner als die Größe der Faktorbasis ist:

  Phase 1 Berechnen der Koeffizienten A, B, C von $f_{A,B,C}(z)$;
  Berechnen der Nullstellen modulo p für jedes $p \in FB$;

  Phase 2 Sieben mit diesen Nullstellen;

  Phase 3 Testen, welche Funktionswerte vollständig über der Faktorbasis zerfallen.

Die Phasen 2 und 3 laufen in der *Multi Polynomial*-Version genauso ab wie in der *Single Polynomial*-Variante (siehe dort). Die Phase 1 nimmt in QS-Implementierungen bis zu 30 Prozent der Gesamtlaufzeit in Anspruch. Das ist Anlaß genug, in dieser Phase nach Verbesserungen zu suchen. In den restlichen Abschnitten dieses Kapitels werden zwei verschiedene Methoden für die in Phase 1 notwendigen Berechnungen vorgestellt.

3.3.1 Die Wahl der Koeffizienten nach Silverman

Die erste Variante stammt von Robert D. Silverman (vgl. ([Sil87])):

1. Wähle $D \in \mathbb{N}$,$D \approx \sqrt{\frac{\sqrt{N}}{M}}$, D (Pseudo-)Primzahl, $D \equiv 3 \bmod 4$

2. Setze $A := D^2 \bmod N$

3. Löse $B^2 \equiv N \bmod A$

   Die effiziente Lösung dieser Kongruenz geschieht durch einen geschickten Algorithmus von Silverman (siehe [Sil87, S. 332]). (An dieser Stelle wird benötigt, daß D Primzahl und kongruent 3 modulo 4 ist.) Die Kongruenz $B^2 \equiv N \bmod A$ hat zwei i.a. verschiedene Lösungen B und $-B$. Da man ein symmetrisches Intervall I verwendet und $f_{A,B,C}(z) = f_{A,-B,C}(-z)$ für jedes $z \in I$ gilt, beschränkt man sich auf eine Lösung der Kongruenz.

4. Berechne für jedes Faktorbasiselement $p \in FB$ die Nullstellen $z_{1,2}$ des zugehörigen Polynoms modulo p

   Für $p \in FB$:

$$\begin{aligned} f_{A,B,C}(z) &= Az^2 + Bz + C \equiv 0 \mod p \\ \Leftrightarrow z^2 + \frac{B}{A}z + \frac{C}{A} &\equiv 0 \bmod p \\ \Rightarrow z &\equiv -\frac{B}{2A} \pm \sqrt{\frac{B^2}{4A^2} - \frac{4AC}{4A^2}} \bmod p \\ &\equiv \frac{1}{2A}(-B \pm \sqrt{N}) \bmod p \qquad (3.12) \end{aligned}$$

   Dabei ist $\sqrt{N} \bmod p$ Lösung der Kongruenz $X^2 \equiv N \bmod p$. Sie kann mit dem RESSOL-Algorithmus von Shanks ([Coh91, Kap. 1.5.1.]) berechnet werden. Es fällt auf, daß der Wert $\sqrt{N}$ nur von der Primzahl p, aber nicht mehr von den Koeffizienten A, B, C und somit nicht mehr von dem Polynom abhängt. Deshalb können die $\sqrt{N}$ für jede Primzahl $p \in FB$ bereits beim Aufstellen der Faktorbasis vorberechnet werden. Die Idee, durch Vorberechnungen Rechenzeit zu sparen, wird auch bei der folgenden Methode der Selbstinitialisierung eine große Rolle spielen.

3.3.2 Die Methode der Selbstinitialisierung

Ein Vorschlag von Peter Montgomery führte zur sogenannten Methode der *Selbstinitialisierung*. Sie wurde von W.R. Alford und Carl Pomerance ([Al-Pom93]) konkretisiert und ausgeführt.

Anstatt A als Quadrat einer Primzahl zu wählen, setzt man in dieser Variante der Berechnung A aus mehreren Primzahlen, sagen wir *bound* vielen, zusammen:

$$A = \prod_{i=1}^{bound} q_i, \qquad q_i \in \mathbb{P}. \tag{3.13}$$

Im Gegensatz zu der Silverman-Methode ist A nun nicht mehr notwendigerweise Quadrat modulo N. Betrachtet man Gleichung (3.11), so folgt daraus, daß (unter Beibehaltung der Bedingung $B^2 - 4AC \equiv 0 \bmod N$) der Funktionswert $f_{A,B,C}(z)$ i.a. kein Quadrat modulo N sein wird. Multipliziert man allerdings die Gleichung mit $4A$ und setzt man

$$\begin{aligned} Y &:= 4A \cdot f_{A,B,C}(z) \bmod N, \\ X &:= 2A\left(z + \frac{B}{2A}\right) \bmod N, \end{aligned}$$

so erhält man auch hier nach Faktorisierung des Funktionswertes $f_{A,B,C}(z)$ eine Relation (3.3), da die Primfaktorzerlegung von $4A$ bereits bekannt ist.

Der wesentliche Unterschied zu der zuvor vorgestellten Silverman-Variante besteht darin, daß die Kongruenz

$$B^2 \equiv N \bmod A \tag{3.14}$$

nicht mehr nur 2, sondern 2^{bound} viele Lösungen hat. Dies folgt aus dem chinesischen Restsatz (vgl. Definition 2.5):

Sei

$$H_i := w_{q_i} \cdot \frac{A}{q_i} \cdot \left(\left(\frac{A}{q_i}\right)^{-1} \bmod q_i\right),$$

wobei w_{q_i} eine Lösung der Kongruenz

$$w_{q_i}^2 \equiv N \bmod q_i$$

sei.

Dann gilt für diese H_i

$$H_i \equiv \begin{cases} w_{q_i} \bmod q_i & \text{für } i = j \\ 0 \quad \bmod q_i & \text{für } i \neq j \end{cases} \tag{3.15}$$

Setzt man

$$B := \sum_{i=1}^{bound} \pm H_i, \tag{3.16}$$

so ist B eine Lösung der Kongruenz (3.14), weil die Vorzeichen wegen der Quadrierung belanglos sind. Durch ihre Einführung hat man aber nicht mehr nur ein B zur Verfügung, sondern 2^{bound} viele. In der Praxis sind es $2^{bound-1}$ viele, da man auch hier nicht gleichzeitig B und $-B$ verwenden möchte.

Wie berechnet man nun die Folge der verschiedenen zu einem A gehörenden B's?

Dazu nimmt man an, man habe B_j schon berechnet:

$$B_j = \sum_{i=1}^{bound} (-1)^{e_i} H_i \qquad (e_i \in \{0,1\} \text{ für } 1 \leq i \leq bound) \tag{3.17}$$

Nun identifiziert man B_j mit dem Vektor $\hat{e}_j := (e_{bound}, \ldots, e_1)$, faßt B_j also als Binärzahl der Länge *bound* auf. Das Problem der systematischen Berechnung der B_j ist somit darauf zurückgeführt, die Binärdarstellungen der Zahlen von 0 bis $2^{bound}-1$ aufzuzählen.

Wie erhält man aus einem B_j den folgenden Koeffizienten B_{j+1}?

Die erste Möglichkeit besteht darin, die zu B_j gehörende Binärzahl $\hat{e}_j$ um 1 zu erhöhen, also binär zu zählen. Dabei würden aber pro Inkrementierung bis zu *bound* viele Bits gekippt werden, nämlich beim Übergang 01...1 $\rightarrow$ 10...0. Nun entspricht aber jedes Kippen eines Bits in einem der $\hat{e}_j$ der Addition oder Subtraktion einer Zahl (nämlich $2 \cdot H_i$) in Formel (3.17). Deshalb ist es günstig, die Zahl der zu kippenden Bits klein zu halten. Die gerade vorgestellte Methode des binären Zählens kippt insgesamt $\sum_{i=0}^{bound} \lfloor \frac{2^{bound}}{2^i} \rfloor < 2^{bound} \sum_{i=0}^{\infty} \frac{1}{2^i} = 2 \cdot 2^{bound}$ Bits (siehe [Co-Le-Ri90, Kapitel 18.1]) und stellt sich nicht als die in diesem Sinne optimale heraus. Es existiert ein Verfahren, das pro Schritt $B_j \rightarrow B_{j+1}$ nur die (minimal erforderliche) Anzahl von *einem* Bit in $\hat{e}_j$ kippt und somit insgesamt nur 2^{bound} Bits kippt. Es beruht auf dem sogenannten Gray-Code (vgl. Def. 2.13).

Man beginnt mit

$$B_0 := \sum_{i=1}^{bound} H_i \tag{3.18}$$

und berechnet rekursiv:

$$B_{j+1} := B_j + 2 \cdot (-1)^{(\frac{2j}{2^\nu}+1)/2} \cdot H_\nu \qquad \text{mit } 2^\nu \parallel 2 \cdot j. \tag{3.19}$$

Der Beweis, daß dieses Verfahren wirklich eine Aufzählung aller B_j in (3.17) liefert, ist etwas technisch und befindet sich in am Ende von Kapitel 2. Man sieht nun sofort aus Formel (3.19), daß man in der Tat beim Übergang von B_j zu B_{j+1} im wesentlichen nur eine Addition oder Subtraktion einer langen Zahl $(2 \cdot H_\nu)$ durchführen muß.

Ein weiterer großer Vorteil dieser Methode der Berechnung der B_j besteht darin, daß sich nun auch eine rekursive Abhängigkeit zwischen den Nullstellen nacheinander berechneter Polynome ergibt:

Sei A im folgenden fest. Die Nullstellen des Polynoms $f_{A,B_j} := Az^2 + B_j z + C$ (modulo einer Primzahl p) seien $z_{1,2}^{(j)}$. Für die Nullstellen des darauffolgenden Polynoms $f_{A,B_{j+1}}$ modulo p gilt nach den Formeln (3.12) und (3.19):

$$\begin{aligned}
z_{1,2}^{(j+1)} &\equiv \frac{1}{2A} \cdot (-B_{j+1} \pm \sqrt{N}) \bmod p \\
&\equiv \frac{1}{2A} \cdot (-(B_j + 2 \cdot (-1)^{(\frac{2j}{2^\nu}+1)/2} \cdot H_\nu) \pm \sqrt{N}) \bmod p \\
&\equiv \frac{1}{2A} \cdot \left(-B_j \pm \sqrt{N}\right) - \frac{1}{2A} \cdot 2 \cdot (-1)^{(\frac{2j}{2^\nu}+1)/2} \cdot H_\nu \bmod p \\
&\equiv z_{1,2}^{(j)} - \frac{1}{A} \cdot (-1)^{(\frac{2j}{2^\nu}+1)/2} \cdot H_\nu \bmod p
\end{aligned}$$

Auf einen Vergleich der Kosten dieser Nullstellenberechnung mit der Berechnung gemäß Formel (3.12) und auf die Möglichkeit von Vorberechnungen wird in Abschnitt 5.2 eingegangen.

3.3.3 Anmerkungen zur Praxis

Nach dieser eher abstrakten Übersicht werden in diesem Abschnitt einige Besonderheiten vorgestellt, die bei einer Implementierung des Quadratischen Siebes zu beachten sind. Alle Anmerkungen treffen sowohl auf die Silverman- als auch auf die Selbstinitialisierungs-Methode zu.

Da für jedes $z \in I$ der Funktionswert $f(z)$ in der Größenordnung $O(\sqrt{N})$ liegt, würde die explizite Berechnung und das Abspeichern der Funktionswerte lange dauern und viel Speicherplatz beanspruchen. Stattdessen werden zu Beginn der Faktorisierung ganzzahlige Näherungen für die Logarithmen $log(p)$ der Primzahlen in der Faktorbasis berechnet und ein Feld *sieb*, das die Länge des Siebintervalls hat, mit Nullen initialisiert. Sieben mit den beiden zu einer Primzahl $p \in FB$ gehörenden Nullstellen bedeutet dann Addition der Näherung an $log(p)$ an den durch Gleichung (3.10) angegebenen Stellen im Feld *sieb*.

Eine vollständige Faktorisierung über der Faktorbasis liegt dann vor, wenn ein Eintrag in *sieb* in der Größenordnung der Logarithmen der Funktionswerte liegt. Dieser Wert beträgt etwa $log\left(\sqrt{\frac{N}{2}} \cdot M\right)$. Aus mehreren Gründen verwendet man für den oben erwähnten Testwert einen kleineren Wert:

1. $\sqrt{\frac{N}{2}} \cdot M$ ist das Maximum der Beträge der Funktionswerte; fast alle Funktionswerte sind kleiner als dieser Wert.

2. Man arbeitet nur mit ganzzahligen Näherungen an $log(p)$.

3. Da kleine Primzahlen in der Faktorbasis nur sehr kleine $log(p)$-Werte haben, tragen sie in der Siebphase kaum etwas zu den Werten in *sieb* bei. Gleichzeitig erfordern sie aber viele Additionen während des Siebens. Deshalb setzt man zu Beginn der Faktorisierung eine Schranke *kleinstart* fest und siebt nur mit Faktorbasiselementen ab dieser Schranke. Um in der Auswertphase die Funktionswerte, die von Primzahlen unter dieser Grenze geteilt werden, nicht zu verlieren, sollte der Testwert unter der angegebenen Schranke liegen.

4. Aus ähnlichen Effizienzgründen siebt man auch nicht mit Primzahlpotenzen.

Weil es außerdem in den im nächsten Kapitel beschriebenen Erweiterungen des QS wichtig ist, den Testwert und damit die Anzahl der zu untersuchenden Stellen im Array *sieb* zu steuern, definiert man den Testwert zusätzlich in Abhängigkeit eines Toleranzwertes T:

$$Testwert(T) := log\left(\sqrt{\frac{N}{2}} \cdot M\right) - T \cdot log(FB_{max}) \ , \tag{3.20}$$

wobei FB_{max} die größte in der Faktorbasis enthaltene Primzahl ist.

Eine weitere Anmerkung betrifft das Aufstellen der Faktorbasis. Um für eine Primzahl $p \in FB$ den Siebvorgang starten zu können, muß die Kongruenz $f_{A,B,C}(z) \equiv 0 \bmod p$ lösbar sein. Dies ist nicht für jedes $p \in \mathbb{P}$ der Fall, denn aus Gleichung (3.11) folgt: $f_{A,B,C}(z) \equiv 0 \bmod p \Rightarrow$

$$\begin{aligned} A\left(z + \frac{B}{2A}\right)^2 &\equiv \frac{B^2 - 4AC}{4A} \bmod p \\ \Longleftrightarrow 4A^2\left(z + \frac{B}{2A}^2\right) &\equiv N \bmod p. \end{aligned}$$

Die letzte Umformung gilt, da man $B^2 - 4AC = N$ setzt. Somit ist gezeigt: Eine Primzahl p kann nur dann als Teiler eines Funktionswertes auftreten, wenn N ein Quadrat modulo p ist. Die Primzahlen, für die diese Bedingung zutrifft, können durch die Berechnung des Legendre-Symbols (siehe Definition 2.6) bestimmt werden. Da man außerdem auch negative Funktionswerte erlauben möchte und deshalb -1 in die Faktorbasis aufnimmt, erhält man die gegenüber Gleichung (3.7) modifizierte Definition:

$$FB := \left\{p \in \mathbb{P} | p \leq B_N \text{ und } \left(\frac{N}{p}\right) = 1\right\} \cup \{-1\}.$$

Der Wert B_N ergibt sich abhängig von der Größenordnung von N aus einer komplexitätstheoretischen Analyse (siehe [Den93, Kapitel 3.3]).

Kapitel 4

Erweiterungen des Grundalgorithmus

In diesem Kapitel werden wichtige Erweiterungen des Grundalgorithmus vorgestellt, die sowohl in der Silverman- als auch in der Selbstinitialisierungs-Methode verwendet werden können. Sie ändern nichts an der komplexitätstheoretischen Laufzeit des QS, beschleunigen aber den Algorithmus in der Praxis und ermöglichen dadurch erst den Einsatz des Quadratischen Siebes für Zahlen im Bereich von über 100 Dezimalstellen. Für beide Erweiterungen werden nur die Grundideen geschildert. Eine sehr detaillierte Beschreibung findet sich in [Den93].

4.1 Multiplier

Das Ziel bei der Verwendung eines Multipliers ist die Erhöhung der Wahrscheinlichkeit, daß ein Funktionswert über der Faktorbasis zerfällt. Diese Wahrscheinlichkeit wird größer, wenn die Faktorbasis (bei gleicher Größe) viele kleine Primzahlen enthält. Das liegt daran, daß bei festem Siebintervall eine Primzahl $p \in FB$ mehr Funktionswerte als eine Primzahl $q \in FB$ teilt, wenn $p < q$ gilt. Deshalb kann man versuchen, anstelle der Zahl N eine Zahl $k \cdot N$ zu faktorisieren, wobei k eine kleine Zahl mit der Eigenschaft ist, daß die Faktorbasis für $k \cdot N$ mehr kleine Primzahlen enthält als für N. Die Zahl k wird *Multiplier* genannt. Der Nachteil bei diesem Vorgehen besteht darin, daß die zu faktorisierenden Funktionswerte um den Faktor $\sqrt{k}$ größer werden. In der Praxis hat sich die Wahl von Multipliern kleiner 30 bewährt.

4.2 *Large Prime/Double Large Prime*-Varianten

Die Wahrscheinlichkeit, daß ein Funktionswert $f(z)$ vollständig über der Faktorbasis zerfällt, ist sehr gering. Sie läßt sich mit Hilfe der in Definition 2.11 eingeführten ρ–Funktion berechnen. Im nächsten Kapitel wird noch näher auf diese Berechnung eingegangen. Hier sei nur erwähnt, daß bei typischen Faktorbasisgrößen von 15 000 bzw. 30 000 diese Wahrscheinlichkeit für eine 80-stellige Zahl etwa $2 \cdot 10^{-8}$, für eine 90-stellige Zahl sogar nur $4 \cdot 10^{-9}$ beträgt. Das bedeutet im letzteren Fall, daß im Durchschnitt 250 Millionen Funktionswerte untersucht werden müssen, um eine Relation zu finden. In gewissem Sinne kann man deshalb den größten Teil der beim Quadratischen Sieb geleisteten Arbeit als wertlos bezeichnen. Folgender Ansatz versucht, auch "fast vollständige" Zerlegungen über der Faktorbasis zu nutzen:

Ist q eine Primzahl, die nicht in der Faktorbasis enthalten ist, und sind Zerlegungen

$$\begin{aligned} f(z_1) &= q \cdot \prod_{p_i \in FB} p_i^{e_{i,1}}, \qquad (4.1) \\ f(z_2) &= q \cdot \prod_{p_i \in FB} p_i^{e_{i,2}} \end{aligned}$$

$(z_1, z_2 \in \mathbb{Z})$ bekannt, so liefert

$$\frac{f(z_1) \cdot f(z_2)}{q^2} = \prod_{p_i \in FB} p_i^{e_{i,1}+e_{i,2}}$$

eine Relation (3.3).

Eine Zerlegung wie in (4.1) nennt man *Large Prime-Relation.* Zur Unterscheidung wird eine Relation der Form (3.9) im weiteren als *full-Relation* bezeichnet.

Das Sammeln von *Large Prime*-Relationen würde wenig Sinn machen, wenn die Wahrscheinlichkeit, daß das selbe $q \in \mathbb{P}$ in zwei verschiedenen dieser Relationen vorkommt, sehr gering wäre. Daß dies nicht der Fall ist, zeigt das sogenannte *Birthday-Paradoxon.* Dieses lautet:

Ist eine Menge mit m verschiedenen Elementen gegeben, und ist die Wahrscheinlichkeit, ein Element aus dieser Menge auszuwählen, für alle Elemente gleich groß, so ist die Wahrscheinlichkeit, ein Element zweimal ausgewählt zu haben, nach etwa $(\sqrt{2 \cdot ln\, 2} \cdot \sqrt{m})$-maligem Ziehen mit Zurücklegen größer als $\frac{1}{2}$ (siehe [Ries87, S. 177]).

Angewandt auf die Situation im Quadratischen Sieb heißt das: Sei FB_{max} die größte in der Faktorbasis enthaltene Primzahl und $LARGE > FB_{max}$ eine Grenze, die man sich vorgibt. Sammelt man alle Funktionswerte, die bis auf eine Primzahl q mit

$$FB_{max} < q < LARGE$$

über der Faktorbasis zerfallen, so kann man laut Birthday-Paradoxon bereits nach $O(\sqrt{LARGE})$ gesammelten Werten erwarten, daß ein q zweimal aufgetreten ist.

Die *Large Prime*-Variante für das QS wurde von Pomerance in [Pom82] eingeführt. Von A.K. Lenstra und M.S. Manasse (siehe [Len-Man91]) stammt die Verallgemeinerung auf zwei Primzahlen: Zerfällt der Funktionswert bis auf zwei Primzahlen q_1 und q_2 über der Faktorbasis, so merkt man sich die Zerlegung

$$f(z) = q_1 \cdot q_2 \cdot \prod_{p_i \in FB} p_i^{e_i} , \tag{4.2}$$

falls q_1 und q_2 kleiner als $LARGE$ sind, und nennt die Zerlegung *Double Large Prime-Relation.*

Wie diese Relationen zu einer *full*-Relation kombiniert werden können, zeigt das folgende

Beispiel 4.1 Seien $q_1, q_2, q_3 \in \mathbb{P}$ mit $FB_{max} < q_i < LARGE$ für $i = 1, 2, 3$. Dann lassen sich die *Large*- und Double Large-Relationen

$$\begin{aligned}
f(z_1) &= q_1 \cdot \prod_{p_i \in FB} p_i^{e_{i,1}}, \\
f(z_2) &= q_1 \cdot q_2 \cdot \prod_{p_i \in FB} p_i^{e_{i,2}}, \\
f(z_3) &= q_2 \cdot q_3 \cdot \prod_{p_i \in FB} p_i^{e_{i,3}}, \\
f(z_4) &= q_3 \cdot \prod_{p_i \in FB} p_i^{e_{i,4}}
\end{aligned}$$

($z_1, z_2, z_3, z_4 \in \mathbb{Z}$) zu einer full-Relation

$$\frac{f(z_1) \cdot f(z_2) \cdot f(z_3) \cdot f(z_4)}{q_1^2 \cdot q_2^2 \cdot q_3^2} = \prod_{p_i \in FB} p_i^{e_{i,1}+e_{i,2}+e_{i,3}+e_{i,4}}$$

kombinieren.

Das Problem, festzustellen, ob sich in einer großen Menge gesammelter *Large*- und *Double Large Prime*-Relationen solche finden lassen, die sich zu einer *full*-Relation aufmultiplizieren lassen, ist wesentlich schwieriger als bei der reinen *Large Prime*-Variante und läßt sich auf die Suche nach elementaren Zykeln in einem Graph zurückführen. Das Vorgehen ist in der Arbeit von Lenstra und Manasse (vgl. [Len-Man91]) beschrieben.

Bemerkung 4.2

1. Um den Anteil des Funktionswertes $f(z)$ in Gleichung (4.2), der nicht über der Faktorbasis zerfällt, in die Primzahlen q_1 und q_2 zu zerlegen, verwendet man die Faktorisierungsalgorithmen SQUFOF und Pollard-ρ (siehe ([Coh91])).

2. Mit dem in Kapitel 3, Gleichung (3.20) definierten Wert T läßt sich die Anzahl der zu untersuchenden Stellen im Siebintervall und somit der Einfluß der *Large Prime*- und *Double Large Prime*-Variante steuern. Siehe dazu [Den93, Kapitel 4].

Kapitel 5

Die Implementierung auf dem Hypercube

Nachdem in den beiden vorherigen Kapiteln das Quadratische Sieb vorgestellt wurde, wird nun die Frage behandelt, wie eine Parallelisierung des QS auf dem Hypercube aussehen kann. Im ersten Abschnitt wird erklärt, wie Polynome auf verschiedene Knoten verteilt werden können. Auf die Implementierung der in Kapitel 3 vorgestellten Selbstinitialisierungs-Variante und ihre Laufzeitauswirkungen wird im zweiten Abschnitt eingegangen.

5.1 Das Quadratische Sieb auf mehreren Rechnern

Die leichte Verteilbarkeit bzw. Parallelisierbarkeit ist einer der größten Vorteile der *Multi Polynomial*-Version des Quadratischen Siebes. Müßte man die drei Phasen in der Schleife - Berechnung der Polynome und Nullstellen/Sieben/Testen - auf verschiedene Knoten verteilen, würden sich sofort Datenabhängigkeiten ergeben, die eine teuere Kommunikation erfordern würden. Stattdessen läßt man nach einer Vorberechnungsphase alle Knotenrechner vollkommen asynchron auf verschiedenen Polynomen rechnen. Dabei möchte man sicherstellen, daß Polynome nicht mehrfach berechnet und zum Sieben verwendet werden. Die Frage, wie den einzelnen Knoten eineindeutig Polynome zugeordnet werden können, wird in den folgenden beiden Abschnitten geklärt.

5.1.1 Die Zuordnung bei der Methode von Silverman

Wie gesehen, beginnt die Berechnung der Koeffizienten eines Polynomes in der Silverman-Methode mit der Berechnung eines $D \in \mathbb{N}$ mit folgenden Eigenschaften:

1. $D \approx \sqrt{\sqrt{N}/M}$,
2. $D \equiv 3 \bmod 4$,
3. D Pseudo-Primzahl.

Dabei ist ein Polynom (bei Vorgabe der zu faktorisierenden Zahl N) durch die Angabe von D bereits eindeutig bestimmt.

Die folgende Methode, verschiedene D's systematisch auf die Knoten zu verteilen, ist der Netzversion des QS entnommen. Den einzelnen Knoten werden Teilintervalle von möglichen D-Werten zugeordnet. Diese Intervalle sind so angeordnet, daß zu jedem Zeitpunkt der Faktorisierung die Großenordnung der gerade verwendeten D's auf allen Knoten die gleiche ist. Dies garantiert, daß keine suboptimalen Werte für D (und somit betragsmäßig unnötig große Funktionswerte) verwendet werden. Im Detail sieht das Vorgehen so aus: Die Intervalle haben eine Länge von $L = 10\,000$. Ist $nodes$ die Anzahl der Knoten des momentan verwendeten Cubes, führt man eine Größe $abstand = nodes \cdot L$ ein. Zu Beginn wird ein D berechnet, das Eigenschaft 1 erfüllt. Der i-te Knoten $(0 \leq i < nodes)$ entnimmt nun Werte für D aus dem Intervall

$$I_0^i := [D + i \cdot L, D + (i+1) \cdot L].$$

Zu dem Startwert des Intervalls wird solange 4 addiert (Eigenschaft 2), bis ein Pseudo-Primzahltest ein D liefert, das auch Bedingung 3 erfüllt. Aus diesem D wird das neue Polynom berechnet. Ist das Ende des Intervalls erreicht, geht man zum nächsten Intervall über. Der i-te Knoten bearbeitet nach j bereits abgearbeiteten Intervallen

$$I_{j+1}^i := [D + i \cdot L + j \cdot abstand, D + (i+1) \cdot L + j \cdot abstand].$$

Damit nach einem Programmabbruch ein Aufsetzpunkt vorhanden ist, wird der aktuelle Wert von D jeweils zu Beginn der Abarbeitung eines neuen Intervalls auf die Festplatte geschrieben. Um doppelte Berechnungen zu vermeiden, wird beim Wiederaufsetzen der um $abstand$ erhöhte Wert des von der Platte gelesenen D's auf die Eigenschaften 1-3 getestet und wie oben fortgefahren.

5.1.2 Die Zuordnung bei der Selbstinitialisierung

Hier wird die Zuordnung der Polynome auf die einzelnen Knoten über die Primzahlen im Koeffizienten A vorgenommen: verschiedene Primzahlen bilden verschiedene Produkte A, somit verschiedene Werte für B und dadurch verschiedene Polynome. Die vom Verwaltungsaufwand her einfachste Methode wäre es, auf jedem Knoten disjunkte Primzahlmengen zu verwenden. Da in der Implementierung ein A aus 10 verschiedenen Primzahlen besteht (d.h. in Gleichung (3.13) wurde $bound = 10$ gewählt), würde bei der Verwendung von 16 oder 32 Knoten die Gesamtzahl der benötigten Primzahlen in die Hunderte gehen. Bildet man jeweils Produkte aus 10 aufeinanderfolgenden Primzahlen, wären viele dieser Produkte weit vom optimalen Wert $\sqrt{N}/M$ entfernt. Wie stark sich solche Schwankungen in der Größe von A auf die Funktionswerte und damit auf die Zerfallswahrscheinlichkeit über der Faktorbasis auswirken, soll anhand des folgenden Beispiels aufgezeigt werden.

Untersucht wurde eine 90-stellige Zahl; die Länge des Siebintervalls war 3 000 000. Es wurden Produkte von jeweils 10 aufeinanderfolgenden Primzahlen untersucht, die erste betrachtete Primzahl ist die 400-ste der Faktorbasis, 5 869. Die erste Spalte gibt an, welche Primzahlen verwendet wurden, nämlich jeweils die $n-9, \dots, n$-ten Primzahlen aus der Faktorbasis. In der Spalte "T" sind die Durchschnitte der Beträge von $f_{A,B,C}(z)$ bei einer Faktorisierung mit dem Koeffizienten $A = \prod_{i=n-9}^{n} p_i$ angegeben. Wie man sieht, liegt der optimale Wert bei n=440. Entfernt man sich von diesem, so wachsen die Funktionswerte schnell an.

n	p_n	T
410	6043	$1.13 \cdot 10^{51}$
420	6217	$0.75 \cdot 10^{51}$
430	6361	$0.52 \cdot 10^{51}$
440	6553	$0.37 \cdot 10^{51}$
450	6733	$0.43 \cdot 10^{51}$
460	6857	$0.50 \cdot 10^{51}$
470	6997	$0.69 \cdot 10^{51}$
480	7177	$0.89 \cdot 10^{51}$
490	7393	$1.20 \cdot 10^{51}$
500	7537	$1.49 \cdot 10^{51}$
550	8363	$4.66 \cdot 10^{51}$
600	9209	$14.10 \cdot 10^{51}$
650	10133	$37.28 \cdot 10^{51}$
700	10987	$77.64 \cdot 10^{51}$
750	11953	$177.87 \cdot 10^{51}$
800	12703	$330.72 \cdot 10^{51}$
850	13.469	$732.49 \cdot 10^{51}$
900	14.549	$1257.54 \cdot 10^{51}$
1400	22.937	$233762.89 \cdot 10^{51}$

Um die Auswirkung dieses Anwachsens auf die Dauer der Faktorisierung zu untersuchen, betrachtet man anhand des Beispieles, wie sich die Wahrscheinlichkeit des vollständigen Zerfallens über der Faktorbasis mit wachsender Größe der Funktionswerte ändert.

Nach Definition 2.11 gilt:

Für eine Zahl kleiner T ist $\rho(\alpha)$ die Wahrscheinlichkeit, daß sie vollständig über einer Faktorbasis zerfällt, deren größtes Element $T^{\frac{1}{\alpha}}$ ist. Ist also FB_{max} die größte Zahl in der Faktorbasis, so gilt:

$$FB_{max} = T^{\frac{1}{\alpha}} \Longrightarrow \alpha = \frac{log(T)}{log(FB_{max})}$$

In unserem Fall ist bei einer Faktorbasis von 30 500 Elementen $FB_{max} = 757\,271$, also z.B. für $L = 0.37 \cdot 10^{51}$

$$\alpha = \frac{\log(0.37) \cdot 10^{51}}{\log(757271)} \approx 8.6011 \tag{5.1}$$

Zur Berechnung von $\rho(\alpha)$ verwende ich die in (2.11) angegebene Näherungsformel:

$$\rho(8.6011) \approx 4.0347 \cdot 10^{-9}$$

Das bedeutet, daß man etwa alle $\frac{1}{\rho(\alpha)} \approx 250\ 000\,000$ Funktionswerte eine komplette Faktorisierung erwarten kann.

Führt man die Rechnung für die verschiedenen Werte von n durch, so ergibt sich folgende Tabelle:

n	T	α	Wahrscheinlichkeit
410	$1.13 \cdot 10^{51}$	8.6836	$3.0311 \cdot 10^{-9}$
420	$0.75 \cdot 10^{51}$	8.6533	$3.3670 \cdot 10^{-9}$
430	$0.52 \cdot 10^{51}$	8.6263	$3.6981 \cdot 10^{-9}$
440	$0.37 \cdot 10^{51}$	8.6011	$4.0347 \cdot 10^{-9}$
450	$0.43 \cdot 10^{51}$	8.6122	$3.8825 \cdot 10^{-9}$
460	$0.50 \cdot 10^{51}$	8.6234	$3.7354 \cdot 10^{-9}$
470	$0.69 \cdot 10^{51}$	8.6472	$3.4397 \cdot 10^{-9}$
480	$0.89 \cdot 10^{51}$	8.6660	$3.2224 \cdot 10^{-9}$
490	$1.20 \cdot 10^{51}$	8.6880	$2.9847 \cdot 10^{-9}$
500	$1.49 \cdot 10^{51}$	8.7040	$2.8234 \cdot 10^{-9}$
550	$4.66 \cdot 10^{51}$	8.7883	$2.1061 \cdot 10^{-9}$
600	$14.10 \cdot 10^{51}$	8.8700	$1.5831 \cdot 10^{-9}$
650	$37.28 \cdot 10^{51}$	8.9419	$1.2314 \cdot 10^{-9}$
700	$77.64 \cdot 10^{51}$	8.9961	$1.0183 \cdot 10^{-9}$
750	$177.87 \cdot 10^{51}$	9.0573	$0.8213 \cdot 10^{-9}$
800	$330.72 \cdot 10^{51}$	9.1031	$0.6990 \cdot 10^{-9}$
850	$732.49 \cdot 10^{51}$	9.1618	$0.5683 \cdot 10^{-9}$
900	$1257.54 \cdot 10^{51}$	9.2018	$0.4936 \cdot 10^{-9}$
1400	$233762.89 \cdot 10^{51}$	9.5877	$0.1253 \cdot 10^{-9}$

Wie man sieht, ergeben sich bereits für kleine Änderungen in der Wahl von n bemerkenswerte Schwankungen. Entfernt man sich vom optimalen Startwert z.B. nur um 30 Primzahlen (von n=440 nach n=470), so sinkt die Erfolgswahrscheinlichkeit bereits um 15 Prozent. Beträgt der Abstand zum Optimum etwa 100 Primzahlen, ergibt sich eine Verschlechterung um fast 50 Prozent, und sind es sogar über 400 Primzahlen, ist die Wahrscheinlichkeit um den Faktor 8 kleiner.

Wegen dieser Empfindlichkeit der Funktionswerte gegenüber Schwankungen im Koeffizienten A möchte man die Anzahl der Primzahlen, die als Faktoren von A auftreten sollen, möglichst klein halten. Andererseits müssen genügend Polynome zur Verfügung stehen. Das im folgenden vorgestellte Verfahren kommt mit lediglich 30 Primzahlen aus. Das wiederholte Auftreten von gleichen A's wird vermieden, indem zu Beginn einer Faktorisierung jedem der 32 möglichen Knoten des Hypercubes ein Produkt von wenigen Primzahlen, den sogenannten Markern, zugewiesen wird, bezüglich dessen er sich von jedem der anderen Knoten unterscheidet. In der Implementierung sind die ersten 8 der 30 Primzahlen Marker, wobei jedem Knoten jeweils 3 der 8 Marker zugewiesen werden. Das ergibt $\binom{8}{3} = 56 > 32$ verschiedene Kombinationen. Wie bereits erwähnt, wurde für die Anzahl der Primzahlen in A der Wert 10 gewählt, also bleiben auf jedem Knoten jeweils $10 - 3 = 7$ frei zu besetzende Zahlen übrig, die nacheinander aus den $30 - 8 = 22$ restlichen Primzahlen gewählt werden. Dann hat jeder Knoten $\binom{22}{7} = 170544$ Möglichkeiten, ein A zu wählen, und pro A, wie gesehen, $2^{10-1} = 512$ verschiedene Polynome. Das sind insgesamt etwa 87 Millionen Polynome, was für praktische Zwecke mehr als genügt. Es ist kein Problem, diese Zahl notfalls durch eine andere Aufteilung der Primzahlen zu erhöhen. Auch beim Übergang auf einen Parallelrechner mit mehr oder weniger als 32 Prozessoren läßt sich dieses Verfahren des Aufteilens der Primzahlen verwenden.

5.2 Vorberechnungen bei der Selbstinitialisierung

In diesem Abschnitt, der sich thematisch an Abschnitt 3.3.2 anschließt, wird näher darauf eingegangen, wie bei der Selbstinitialisierungs-Methode die Berechnung der Nullstellen eines Polynoms in der Praxis aussieht und wie Vorberechnungen zu einer Laufzeitverbesserung führen können.

Der Koeffizient A sei fest gewählt und die Nullstellen $z_{1,2}^{(j)}$ des Polynoms f_{A,B_j} modulo einer Primzahl $p \in FB$ seien bekannt. Nach Formel (3.20) gilt für die Nullstellen $z_{1,2}^{(j)}$ des Polynoms $f_{A,B_{j+1}}$ modulo p:

$$z_{1,2}^{(j+1)} := z_{1,2}^{(j)} - \frac{1}{A} \cdot (-1)^{(\frac{2j}{2^\nu}+1)/2} \cdot H_\nu \bmod p \qquad (5.2)$$

mit $2^\nu \| 2j$.

Man vergleiche dies mit der entsprechenden Formel (3.12) der Silverman-Methode:

$$z_{1,2}^{(j+1)} := \frac{1}{2A_{j+1}}(-B_{j+1} \pm \sqrt{N}) \bmod p \tag{5.3}$$

Der Unterschied besteht darin, daß in Gleichung (5.2) A wesentlich schneller modulo p invertiert werden kann als in (5.3). Das liegt daran, daß bei der Selbstinitialisierung die in A auftretenden Primzahlen mehrmals vorkommen. Aus

$$A = \prod_{i=1}^{10} q_i \ ,$$

folgt

$$A^{-1} \equiv \prod_{i=1}^{10} q_i^{-1} \bmod p$$

für jedes $p \in FB$, wobei die q_i aus einer zu Beginn der Faktorisierung ausgewählten Menge von 30 Primzahlen stammen (siehe vorheriger Abschnitt). Berechnet man zu Beginn der Faktorisierung für jede der 30 Primzahlen die Inversen modulo der Faktorbasiselemente, so kann in Gleichung (5.2) A^{-1} durch das Aufmultiplizieren von 10 Zahlen und nicht durch einen teueren ggt-Aufruf berechnet werden. Da die Inversion die teuerste Operation bei der Berechnung gemäß Gleichung (5.2) ist, ergibt dieses Verfahren eine große Laufzeitbeschleunigung bei der Berechnung der Nullstellen.

Der Nachteil dieser Methode ist der hohe Speicherplatzbedarf. Um bei einer Faktorbasis von 40 000 Elementen die Inversen der 30 Primzahlen in Integers (4 Byte) zu speichern, sind 4.8 MB nötig. Da man sie bei jeder Koeffizientenberechnung benötigt, wäre ein Speichern der Inversen auf der Platte wegen der hohen Zugriffszeit zu teuer.

Aus diesem Grunde untersucht man nun, wieviel Hauptspeicher für Vorberechnungen auf einem Knoten zur Verfügung steht. Durch das Betriebssystem, die Faktorbasis, die Felder mit den Nullstellen und alle anderen Variablen sind bei einer 100-stelligen Zahl etwa 1.5 MB der 8 MB eines Knotens des Hypercubes belegt. Die verbleibenden 6.5 MB reichen i.a. nicht aus, um das gesamte Siebintervall $[-M, M]$ auf einmal im Hauptspeicher zu halten. Deshalb wird das Intervall in Teilintervalle der Länge *siebintern* unterteilt. Das Siebarray *sieb* hat dann *siebintern* viele Einträge. Der für die Geschwindigkeit des Siebens optimale *siebintern*-Wert ist stark maschinenabhängig, wie die folgende Messung zeigt:

Untersucht wurde das Verhalten bei einer 90-stelligen Zahl auf dem Hypercube. Der Wert von M war 3 000 000. Es wurde ab der fünfzigsten Primzahl der Faktorbasis gesiebt (siehe Bemerkung am Ende von Kapitel 3). Die Zeiten beziehen sich jeweils auf das Sieben mit 8 Polynome. Die letzte Spalte gibt die Laufzeit relativ zum besten Wert von *siebintern* an.

siebintern	Siebzeit [in s]	rel. Laufzeit
6000000	78.68	1.35
3000000	78.41	1.34
2000000	77.77	1.33
1000000	75.69	1.30
500000	70.25	1.20
250000	58.33	1.00
125000	64.02	1.10
62500	81.90	1.40
31250	118.33	2.03
15625	192.37	3.30

Zum Vergleich die Messung für eine 104-stellige Zahl auf einer Sparc Workstation aus ([Den93, Tabelle 3.4]) :

siebintern	Siebzeit [in s]	rel. Laufzeit
16000000	24.80	1.00
8000000	24.82	1.00
4000000	25.00	1.01
2000000	25.19	1.02
1000000	25.70	1.04
800000	25.95	1.05
500000	26.65	1.07
250000	28.96	1.16
100000	36.37	1.46
50000	52.03	2.09
5000	402.35	16.22

Wie man sieht, liegt der optimale Wert beim Hypercube aus Hardwaregründen um den Faktor 64 niedriger.

Wegen der sehr geringen Größe des optimalen *siebintern* wäre somit die durch Gleichung (5.2) gegebene Methode trotz ihres hohen Speicherplatzbedarfes auf dem Hypercube einsetzbar. Es gibt allerdings ein Verfahren, das weniger Speicher verbraucht und sogar noch etwas schneller ist. Es beruht auf der Beobachtung, daß in Gleichung (5.2) die verschiedenen ν's nicht gleich häufig auftreten. Wegen der Beziehung

$$2^{\nu} \| 2 \cdot j \qquad\qquad (j = 0, \ldots, 2^9 - 1) \tag{5.4}$$

ist ν in der Hälfte aller Fälle, nämlich für $j \not\equiv 0 \bmod 2$, gleich 1. Berechnet man nach der Wahl von A für jedes $p \in FB$ den Wert $H_1 \cdot A^{-1} \bmod p$, so beinhaltet die Berechnung (5.2) für die Hälfte aller Polynome nur noch eine Addition oder Subtraktion modulo p.

Das System läßt sich fortführen: ν ist genau dann gleich 2, wenn sowohl $j \equiv 0 \bmod 2$ als auch $j \not\equiv 0 \bmod 2^2$ gilt, was in einem Viertel aller Fälle zutrifft. Hat man noch genügend Speicherplatz, um $H_2 \cdot A^{-1} \bmod p$ zu speichern, kann man diese Werte vorberechnen und muß nur noch für ein Viertel aller j's in (5.2) eine echte Inversion durchführen.
Allgemein gilt:

$$\nu = k \Longleftrightarrow \{j \equiv 0 \bmod 2^{k-1} \text{ und } j \not\equiv 0 \bmod 2^k\}$$

Somit halbiert sich für jedes ν, für das Vorberechnungen möglich sind, die Anzahl der Inversionen in (5.2). Ist genügend Speicherplatz für alle $H_1, \ldots, H_{10}$ vorhanden, so kommt man in (5.2) vollkommen ohne Inversionen aus und muß nur eine Addition/Subtraktion ausführen. Im Vergleich zu der durch Gleichung (5.2) gegebenen Methode spart man zusätzlich die dort notwendigen Multiplikationen zur Berechnung von A^{-1}, die Multiplikation mit H_ν und die Reduktion modulo p.

Wie sich die gesparten Inversionen in der Praxis auszahlen und wie groß der Speicherbedarf dieser Methode ist, zeigt die folgende Messung:

Untersucht wurden die Koeffizienten der Polynome für eine 90-stelligen Zahl. Die kleinste in A verwendete Primzahl war das 400-ste Faktorbasiselement.
Die erste Spalte der Tabelle gibt an, wieviele Terme vorberechnet wurden (die erste Zeile gibt die Laufzeit für die Methode ohne Vorberechnungen an). Die zweite und dritte Spalte gibt den für die Vorberechnungen zusätzlich benötigten Speicherplatz bzw. Zeitaufwand in Sekunden an. Der Speicheraufwand berechnet sich folgendermaßen: für die Faktorbasis von 40 000 Elementen muß pro ν ein Feld von Integern (4 Byte) angelegt werden, also ergibt sich ein Bedarf von $\nu \cdot 4 \cdot 40000 \approx \nu \cdot 160$ kB. Die Durchläufe ohne Vorberechnung benötigten (für die Nullstellenbestimmung) 2.840 Sekunden pro Polynom, mit Vorberechnungen 0.180 Sekunden. Die Spalten 4 und 5 geben an, wie oft bei den insgesamt 512 Polynomen jeder der ungünstigen bzw. günstigen Fälle auftrat. Die letzte Spalte enthält die Gesamtlaufzeit für die Berechnung der 512 Polynomen.

ν	zus. Sp.bedarf	Vorb.[s]	# ung. Fälle	# g. Fälle	insgesamt [s]
-	-	-	512	0	1 454.0
1	160 kB	2.685	256	256	775.8
2	320 kB	5.370	128	384	438.0
3	480 kB	8.055	64	448	258.4
4	640 kB	10.741	32	480	188.0
5	800 kB	13.427	16	496	148.1
6	960 kB	16.111	8	504	127.5
7	1120 kB	18.797	4	508	121.6
8	1280 kB	21.481	2	510	119.0
9	1440 kB	24.167	1	511	119.0
10	1600 kB	26.852	0	512	119.0

Wie man sieht, lassen sich die Zeiten für das Berechnen der Koeffizienten um den Faktor 12 verbessern.
Bei der vorliegenden Implementierung auf dem Hypercube ist wegen der relativ geringen Auslastung des Hauptspeichers (siehe oben) der zusätzliche Speicherplatzbedarf in dieser Variante kein Problem. Im Hinblick auf eine mögliche Implementierung auf anderen Rechnermodellen ist jedoch bemerkenswert, daß sich eine starke Verbesserung der Laufzeit bereits bei einem geringeren zusätzlichen Speicherplatzbedarf als den kompletten 1.6 MB einstellt. Schon der erste Schritt ($\nu = 1$) halbiert die Laufzeit. Bis $\nu = 5$ (0.8 MB) ist die Laufzeit bereits um den Faktor 9.8 gesunken. Ab $\nu = 8$ ist überhaupt keine Verbesserung mehr meßbar. Das liegt daran, daß die zusätzlich erforderlichen Vorberechnungen teuerer sind als die wenigen (2, 1 bzw. 0) noch verbleibenden ungünstigen Fälle.
Das bedeutet: verwendet man eine Maschine, die einen kleinen Hauptspeicher besitzt und für die der optimale Wert für *siebintern* nicht so extrem niedrig liegt wie auf dem Hypercube, so kann durch eine nur teilweise Benutzung der oben beschriebenen Vorberechnungen bereits eine merkliche Beschleunigung der Gesamtlaufzeit erreicht werden. Die Strategie beim Einsatz der Vorberechnungen hängt aber nicht nur von dem optimalen Wert für *siebwert* ab, sondern auch davon, wie sich die Laufzeit der Siebphase für nicht optimale Werte von *siebwert* verhält. Für die Sparc Workstation wirken sich große Schwankungen in *siebintern* nicht stark aus, solange *siebintern* nicht sehr klein ist. So beträgt der Unterschied zwischen den Zeiten für die Siebphase beim Optimum (16 000 000) und z.B. 1 000 000 nur 4 Prozent. Verwendet man den kleineren Wert und steckt auch nur einen Bruchteil des gesparten Speicherplatzes in die Vorberechnungen, ergibt sich für die Laufzeit der Phase des Berechnens der Koeffizienten eine Verbesserung, die die 4 Prozent aus der Siebphase bei weitem aufwiegt.

Bemerkung 5.1
Die 30 Primzahlen, die als Faktoren in Gleichung (5.2) auftreten können, werden zu Beginn der Faktorisierung der Faktorbasis entnommen. Für ein $p \in FB$ mit $p|A$ ist aber der Ausdruck A^{-1} in Gleichung (5.2) nicht mehr definiert. Man hat nun zwei Möglichkeiten:

1. Man berechnet die Nullstellen für ein solches p gesondert. Wegen $A \equiv 0 \bmod p$ gilt:

$$\begin{aligned} Az^2 + Bz + C &\equiv 0 \bmod p \\ \Longleftrightarrow z &\equiv \frac{-C}{B} \bmod p. \end{aligned}$$

   Somit gibt es für diese Primzahlen nur eine statt zwei Nullstellen. Beim Sieben muß man darauf achten, nicht zweimal mit der gleichen Nullstelle zu arbeiten.

2. Man verzichtet ganz auf das Sieben mit diesen Faktorbasiselementen. Um if-Abfragen in der sehr oft aufgerufenen Siebprozedur zu vermeiden, kann man folgendermaßen vorgehen: Man läßt für diese p die Startwerte der Siebphase

zu, die sich aus der (in diesem Fall unsinnigen) Berechnung der Nullstellen ergeben, und siebt wie gewöhnlich mit diesen Werten. Gleichzeitig setzt man aber die Näherungen an $log(p)$ für diese Primzahlen p auf 0, womit das Sieben keinen Effekt hat.

Die folgenden Tabellen zeigen Laufzeit und Anzahl der gefundenen Relationen für beide Möglichkeiten. Die zu faktorisierende Zahl hatte 89 Dezimalstellen, die Länge des Siebintervalles betrug 3 000 000, die Faktorbasis bestand aus 32 000 Elementen. $full$-Relationen werden mit *Full*, *Large Prime*-Relationen mit LP, *Double Large Prime*-Relationen mit DLP bezeichnet.

	# Full	#LP	#DLP
Version 1	16	364	755
Version 2	15	359	734

Die Laufzeiten (in Sekunden) betrugen

	Koeff./Nullst.	Sieben	Testen	insgesamt
Version 1	37	2030	1064	3131
Version 2	37	2065	1049	3151

Somit ist die erste Version sowohl von der Laufzeit als auch von den gefundenen Relationen her die günstigere der beiden Möglichkeiten. Allerdings sind die Unterschiede nicht gravierend.

Kapitel 6

Wahl der Parameter und Laufzeiten

In diesem Kapitel werden die Einflüsse verschiedener Parameter auf die Laufzeit des Quadratischen Siebes untersucht. Dabei soll ein Gefühl dafür vermittelt werden, nach welchen Kriterien die Parameterwahl erfolgen sollte. Parallel dazu werden die Laufzeiten der Silverman-Methode und der Selbstinitialisierungsmethode für Zahlen mit 70, 80, 90 und 95 Dezimalstellen einander gegenübergestellt.

Der Übersichtlichkeit halber wurden alle Daten auf ein Einheitsmaß zurückgerechnet. Dieses Maß gibt an, wieviele Relationen ein Cube mit 8 Knoten in 24 Stunden bei der vorgegebenen Parameterwahl finden würde. Tatsächlich wurden pro Messung jeweils nur etwa 500 Polynome abgearbeitet. Bei allen Messungen wurde LARGE=$5 \cdot 10^8$ und $kleinstart = 50$ gewählt (vgl. Abschnitt 4.2 bzw. 3.3.3).

6.1 70 Dezimalstellen

6.1.1 Silverman-Methode

Als Ausgangspunkt werden die Parameter untersucht, die in [Den93, Kap. 5.3] für die Netzimplementierung vorgeschlagen wurden (zur Definition von $Testwert$ siehe Gleichung (3.20)). Im folgenden seien $full$-Relationen mit $Full$, *Large Prime*-Relationen mit LP und *Double Large Prime*-Relationen mit DLP abgekürzt.

1.: $\#FB = 8\,000, M = 1\,000\,000, Testwert = 131$

#Full	#LP	#DLP
34 260	80 600	0

Ein wichtiges Indiz für die Güte der Parameterwahl ist die prozentuale Laufzeitaufteilung zwischen den drei Hauptphasen des Algorithmus: Berechnen der Koeffizienten und Nullstellen/Sieben/Testen. Sie ist für die angegebenen Parameter die folgende:

Koeff.	Sieben	Testen
38.7 %	58.1 %	3.2 %

Bei der Faktorisierung einer 70-stelligen Zahl sind typischerweise etwa 60 bis 65 Prozent der Relationen, die am Ende beim Aufstellen des Gleichungssystems verwendet werden, reine *full*-Relationen. 35 bis 40 Prozent sind aus LPs gebildet und nur 0 bis 2 Prozent aus LPs und DLPs kombiniert. Das bedeutet, daß die Anzahl der gefundenen LPs für eine 70-stellige Zahl nur eine sehr untergeordnete Rolle spielt. Die Optimierung sollte sich zunächst auf die Anzahl der Fulls konzentrieren, dann auf die der LPs. Bei der vorliegenden Parameterwahl ist die Anzahl der LPs relativ niedrig. Dies liegt daran, daß der Testwert zu hoch gewählt wurde und viele Funktionswerte, die vollständig oder fast vollständig über der Faktorbasis zerfallen, in der Testphase gar nicht untersucht werden. Das läßt sich auch daran erkennen, daß die Testphase mit nur 3.2 Prozent einen sehr geringen Anteil an der Gesamtlaufzeit hat. Deshalb wurde beim nächsten Testlauf ein niedrigerer Testwert gewählt.

2.: $\#FB = 8\,000, M = 1\,000\,000, Testwert = 119$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
43 030	446 110	2 700	38.0 %	56.9 %	5.1 %

Die Anzahl der LPs ist gegenüber der letzten Messung um mehr als den Faktor 5 gestiegen. Zu beachten ist, daß die Gesamtlaufzeit der Faktorisierung damit nicht notwendigerweise um den Faktor 5 gesunken ist. Das liegt daran, daß viele der *Large Prime*-Relationen in Messung 2 eine schlechtere Güte als die in Messung 1 haben: Da der Testwert gesenkt wurde, ist die *Large Prime* q aus Gleichung (4.1) im Durchschnitt größer als in Messung 1 und die Wahrscheinlichkeit, daß sich zwei LPs zu einer Full kombinieren lassen, somit etwas gesunken. Da andererseits die in Messung 1 gefundenen LPs auch in Messung 2 noch gefunden werden, wird im folgenden davon ausgegangen, daß eine Erhöhung der gefundenen LPs grundsätzlich wünschenswert ist (sinngemäß gilt dies auch für die *Double Large Prime*-Relationen).

Um zu untersuchen, ob die Anzahl der gefundenen LPs pro Zeit weiter steigt, wird nun der Testwert weiter gesenkt.

3.:: $\#FB = 8\,000, M = 1\,000\,000, Testwert = 107$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
39 800	921 600	436 690	2.3 %	48.2 %	19.5 %

Der Rückgang bei den Fulls wird durch den starken Anstieg bei den LPs mehr als wettgemacht. Ein weiteres Mal wird der Testwert gesenkt:

4.: $\#FB = 8\,000, M = 1\,000\,000, Testwert = 100$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
28 740	811 380	1 005 040	22.7 %	33.9 %	43.4 %

Dies ist eine Verschlechterung; es wird zuviel Zeit auf das Testen verwendet. Deshalb kehrt man zum Testwert 107 zurück.

In der vorletzten Messung war der Anteil für das Berechnen der Koeffizienten und Nullstellen recht hoch. Man kann ihn senken, indem man das Siebintervall vergrößert, weil dann die Polynome seltener gewechselt werden. Der Nachteil bei diesem Vorgehen ist, daß die Größe der Funktionswertbeträge ($O(\sqrt{N} \cdot M)$) dabei wächst. Die Laufzeitauswirkungen zeigt die folgende Messung.

5.:$\#FB = 8\,000, M = 2\,000\,000, Testwert = 107$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
38 250	1 019 150	490 630	18.3 %	60.5 %	21.2 %

Nach dieser Verbesserung betrachtet man eine weitere Erhöhung der Siebintervallgröße auf 3 000 000.

6.:$\#FB = 8\,000, M = 3\,000\,000, Testwert = 107$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
33 480	973 620	361 260	14.0 %	69.4 %	16.6

Die Erhöhung bringt nichts, also ist der Fall M = 2 000 000 (die Messung 5) von den vorgestellten Fällen der günstigste.

6.1.2 Selbstinitialisierungs-Methode

Um einen ersten Vergleich mit der Silverman-Methode zu haben, startet man mit den Parametern, die sich dort als gut herausgestellt haben:

1.: $\#FB = 8\,000, M = 2\,000\,000, Testwert = 107$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
46 690	1 072 070	523 220	0.7 %	71.9 %	27.4 %

Die Zeit für das Berechnen der Koeffizienten ist dramatisch gesunken. Da somit das Wechseln der Polynome billiger geworden ist, kann man nun kleinere Siebintervalle wählen und erhält kleinere Funktionswerte (man geht also, verglichen mit der Bemerkung vor der Messung 5 im letzten Abschnitt, genau umgekehrt vor).

2.: $\#FB = 8\,000, M = 500\,000, Testwert = 107$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
64790	1 308 320	362 990	2.2 %	67.5 %	30.3 %

3.: $\#FB = 8\,000, M = 125\,000, Testwert = 107$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
58 730	1 014 300	158 270	4.6 %	42.1 %	53.3 %

Hier ist bereits eine Verschlechterung gegenüber der letzten Messung festzustellen. Die Auswirkung einer weiteren Verkleinerung des Siebintervalles zeigen die folgenden beiden Messungen.

4.: $\#FB = 8\,000, M = 62\,500, Testwert = 107$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
37 740	656 420	57 960	5.7 %	24.5 %	69.8 %

5.: $\#FB = 8\,000, M = 31\,250, Testwert = 107$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
39 380	447 360	36 230	6.6 %	14.2 %	79.2 %

Die schlechten Resultate in den letzten drei Messungen lassen sich darauf zurückführen, daß der Laufzeitanteil der Testphase zu hoch war. Das liegt daran, daß trotz der Verkleinerung des Siebintervalls der Testwert nicht geändert wurde. Ausgehend von der noch relativ guten Messung 3 erhöht man deshalb den Testwert:

6.: $\#FB = 8\,000, M = 125\,000, Testwert = 113$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
65 280	815 430	39 370	7.0 %	54.5 %	38.5 %

Der Anteil der Testphase ist deutlich gesunken. Eine weitere Erhöhung des Testwertes hat folgenden Effekt:

7.: $\#FB = 8\,000, M = 125\,000, Testwert = 119$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
67 050	575 090	2 490	8.7 %	65.5 %	25.8 %

Obwohl in Messung 7 die Laufzeitverteilung annähernd die selbe ist wie in Messung 2, ist Messung 2 die deutlich bessere, da die Anzahl der gefundenen LPs in ihr wesentlich höher ist. Messung 2 bleibt somit von allen getesteten Parameterwahlen die beste.

6.1.3 Vergleich

	#Full	#LP	#DLP
Selbstin.	64 790	1 308 320	362 990
Silverman	38 250	1 019 150	490 630
Verhältn.	1.69	1.28	0.74

Geht man (wie oben erwähnt) davon aus, daß die Fulls 60 Prozent und die LPs 40 Prozent zu den Relationen beitragen, ergibt das grob geschätzt eine Beschleunigung um den Faktor $0.6 \cdot 1.69 + 0.4 \cdot 1.28 \approx 1.53$.

Daß diese Rechnung zuläßig ist, zeigt der Vergleich mit den tatsächlich benötigten Zeiten bei einer Faktorisierung mit den jeweils besten Parametern:

Methode	Zeit [h]
Silverman	1.96
Selbstin.	1.24
Verhältn.	1.58

Für die folgenden Zahlen werden nur noch die jeweils besten gefundenen Parameter für die beiden Methoden angegeben.

6.2 80 Dezimalstellen

Mit wachsender Länge der zu faktorisierenden Zahl wächst der Einfluß der *Large-* und *Double Large Prime*-Varianten. Für eine 80-stellige Zahl beträgt der Anteil der an Relationen beteiligten LPs bereits 35-40 Prozent, der der aus LPs und DLPs kombinierten 20 bis 25 Prozent. Der Anteil der Fulls ist auf 35 bis 45 Prozent gesunken. Das bedeutet, daß man bei der Optimierung sowohl die Fulls als auch die LPs im Auge behalten muß.

6.2.1 Silverman-Methode

$\#FB = 15\,000, M = 3\,000\,000, Testwert = 115$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
5 963	165 574	379 581	20.2 %	46.3 %	33.5 %

6.2.2 Selbstinitialisierungs-Methode

$\#FB = 15\,000, M = 500\,000, Testwert = 120$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
11 380	279 850	191 850	3.7 %	66.0 %	30.3 %

6.2.3 Vergleich

	#Full	#LP	#DLP
Selbstin.	11 380	279 850	191 850
Silverman	5 963	165 574	379 581
Verhältn.	1.91	1.69	0.51

Geht man, wie oben angegeben, von Anteilen der drei Relationstypen von 40, 37.5 und 22.5 Prozent aus, ergibt sich eine Beschleunigung von etwa $0.40 \cdot 1.91 + 0.375 \cdot 1.69 + 0.225 \cdot 0.51 \approx 1.51$.

Die wirkliche Faktorisierung mit diesen Parametern ergab folgende Zeiten:

Methode	Zeit [h]
Silverman	16.81
Selbstin.	10.71
Verhältn.	1.57

6.3 90 Dezimalstellen

Für Zahlen dieser Länge haben Fulls einen Anteil von 20-25 Prozent an der Gesamtzahl der Relationen, *Large Prime*-Relationen 20-25 Prozent und durch Kombinationen von *Large Prime*- und *Double Large Prime*-Relationen entstandene Relationen 50-60 Prozent.

6.3.1 Silverman-Methode

$\#FB = 30\,500, M = 4\,000\,000, Testwert = 130$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
1 220	35 865	130 774	28.4 %	47.7 %	23.9%

6.3.2 Selbstinitialisierungs-Methode

$\#FB = 30\,500, M = 500\,000, Testwert = 128$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
2190	52 550	181 020	6.4 %	54.6 %	39.0 %

6.3.3 Vergleich

	#Full	#LP	#DLP
Selbstin.	2 190	52 550	181 020
Silverman	1 220	35 865	130 774
Verhältn.	1.80	1.47	1.38

Damit ergibt sich rechnerisch eine Beschleunigung um den Faktor $0.225 \cdot 1.80 + 0.225 \cdot 1.47 + 0.55 \cdot 1.38 = 1.50$.

Tatsächlich ergaben sich die folgenden Zeiten:

Methode	Zeit [h]
Silverman	121.44
Selbstin.	83.28
Verhältn.	1.46

6.4 95 Dezimalstellen

6.4.1 Silverman-Methode

$\#FB = 41\,500, M = 7\,000\,000, Testwert = 140$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
524	14 624	54 412	29.4 %	57.1 %	13.5 %

6.4.2 Selbstinitialisierungs-Methode

$\#FB = 41\,500, M = 2\,000\,000, Testwert = 135$

Relationen			Laufzeit		
#Full	#LP	#DLP	Koeff.	Sieben	Testen
870	20 310	89 900	3.6 %	66.7 %	29.7 %

6.4.3 Vergleich

	#Full	#LP	#DLP
Selbstin.	870	20 310	89 900
Silverman	524	14 624	52 412
Verhältn.	1.66	1.39	1.72

Für 95-stellige Zahlen haben Fulls und LPs jeweils einen Anteil von ca. 20 Prozent an den Relationen, DLPs einen von etwa 60 Prozent. Damit ergibt sich eine Beschleunigung um etwa $1.66 \cdot 0.2 + 1.39 \cdot 0.2 + 0.6 \cdot 1.72 = 1.64$.

Mit diesen Parametern wurden aus Zeitgründen keine vollständigen Faktorisierungen durchgeführt. Der Vollständigkeit halber sei angegeben, daß die Faktorisierung einer 95-stelligen Zahl mit der Silverman-Variante etwa 280 Stunden benötigt.

6.5 Fazit

Abschließend läßt sich sagen, daß die Selbstinitialisierungs-Methode, unabhängig von der Länge der zu faktorisierenden Zahl, gegenüber der Silverman-Methode eine Beschleunigung um den Faktor 1.5 bringt.

Verwendet man die für die Silverman-Methode optimale Wahl der Parameter für die Selbstinitialisierungs-Methode, ergibt sich eine Senkung des Anteils der Phase des Berechnens der Koeffizienten und Nullstellen an der Gesamtlaufzeit von zwischen 20 und 30 Prozent auf weniger als 1 Prozent. Es zeigt sich, daß für die Selbstinitialisierungs-Methode eine Erhöhung dieses Anteils, d.h. das häufigere Wechseln von Polynomen, sinnvoll ist. Dies geschieht durch eine Senkung der Siebintervallgrößen. Wie die Messungen zeigen, sind gute Werte hierfür um den Faktor 4-8 kleiner als in der Silverman-Variante.

Kapitel 7

Abschließende Vergleiche

7.1 Netz ↔ Hypercube

In diesem Abschnitt wird die QS-Implementierung auf dem Hypercube der mit LiPS auf dem Netz verteilten gegenübergestellt. Die wichtigsten Unterschiede zwischen den beiden Implementierungen machen sich in den folgenden Punkten bemerkbar:

+ *Homogenität*
  In der Homogenität liegt die wohl größte Stärke des Hypercubes. Abgesehen von den 4 Knoten mit 16 statt 8 MB Hauptspeicher sind alle Rechner des Hypercubes identisch und haben deshalb die gleiche Rechenleistung. Gute Parameter nur einmal gefunden und nicht für verschiedene Maschinentypen jeweils von neuem angepaßt werden.

+ *Single user-Betrieb*
  Im engen Zusammenhang mit der Homogenität steht die Tatsache, daß der einzelne Knoten des Hypercubes im Single user-Modus betrieben wird. Da die Rechenzeit auf dem einzelnen Rechner nicht mit den Prozessen anderer Benutzer geteilt werde muß, lassen sich Zeitmessungen und Vorhersagen, die auf einem Knoten gewonnen wurden, problemlos auf mehrere Knoten hochrechnen.

+ *Größe von siebintern*
  Der durch die Hardware bedingte, im Vergleich zur Netzversion sehr niedrige optimale Wert von *siebintern* macht die Laufzeitverbesserung bei der Selbstinitialisierung erst möglich. Läge der optimale Wert nicht bei $250\,kB$, sondern bei 4 oder $5\,MB$, wären die in Abschnitt 5.2 vorgestellten zeitsparenden Vorberechnungen nicht komplett durchführbar gewesen.

– *nur Teillösung*
  Bei einem Vergleich der Implementierungen darf nicht vergessen werden, daß

auf dem Hypercube keine komplette Faktorisierung stattfindet, sondern nur das Sammeln von Relationen. Dies ist zwar der zeitaufwendigste Teil des Faktorisierungsproblems, aber eine effiziente Übertragung der restlichen Teilaufgaben (Konstruktion von *full*-Relationen aus *Large-* und *Double Large*-Relationen, Lösen des Gleichungssystems) auf den Hypercube wäre eine nichttriviale Programmieraufgabe.

7.2 Selbstinitialisierung ↔ Silvermans Methode

In diesem Abschnitt werden Vor- und Nachteile der Methode der Selbstinitialisierung gegenüber der Methode von Silverman angesprochen.

\+ *Geschwindigkeit*
Durch die Selbstinitialisierung kann die für das Berechnen der Koeffizienten und Nullstellen benötigte Zeit stark gesenkt werden. Der daraus resultierende Geschwindigkeitsgewinn ist der wesentliche Vorteil der Selbstinitialisierung.

– *Probleme bei kleinem Hauptspeicher*
Im Hinblick auf eine eventuelle Umstellung der mit LiPS verteilten Netzversion des QS auf die Selbstinitialisierung ist anzumerken, daß bei Rechnern mit kleinem Hauptspeicher entweder auf Teile der Vorberechnung verzichtet oder die Arbeit mit einer kleineren, suboptimalen internen Sieblänge (*siebintern*) in Kauf genommen werden muß.

7.3 Ausblick

Vergleicht man die jeweils als gut erkannten Parameter aus Kapitel 6, so stellt man fest, daß der Anteil der Testphase an der Gesamtlaufzeit in der Selbstinitialisierungs-Methode i.a. höher ist als in der Silverman-Methode. Deshalb ist es sinnvoll, nach Optimierungen in der Test-Phase zu suchen. Es existiert z.B. eine effizientere Implementierung des SQUFOF-Algorithmus, die noch nicht in das Programm eingebaut wurde. Die Wahl der zahlreichen Parameter (Größe der Faktorbasis, Siebintervallänge M, Toleranzwert T, die Schranken aus den Multiple Prime-Varianten) beeinflußt die Laufzeit des Algorithmus u.U. stark. Es wäre sehr interessant zu untersuchen, ob an dieser Stelle der Einsatz genetischer Algorithmen möglich ist. Ein genetischer Algorithmus ist eine Optimierungsstrategie, die in einem Suchraum von potentiellen Lösungen nach optimalen oder fast optimalen Lösungen sucht. Potentielle Lösungen - hier die zu untersuchenden Parameter - sind als Binärstrings ("Chromosomen") kodiert. Eine "Anfangsbevölkerung" von Chromosomen wird zufällig oder Heuristiken folgend gewählt. Eine Güte- oder Fitness-Funktion bewertet die

Chromosomen. Im Fall des QS wäre eine solche Funktion die Anzahl der gefundenen Relationen pro Zeit bei einem Lauf des QS mit den entsprechenden Parametern für einige wenige Polynome. Die neue Population entsteht, indem die in diesem Sinne besten Chromosomen ausgewählt, die anderen verworfen werden. Eine (zufällig bestimmte) Teilmenge der ausgewählten Chromosomen wird außerdem mittels zweier genetischer Operatoren verändert: Bei der Mutation werden zufällig gewählte Bits eines Chromosoms geändert, beim Crossover werden Teilsegmente zweier verschiedener Chromosomen gegeneinander ausgetauscht. Nach der Anwendung der genetischen Operatoren wird die Fitness-Funktion wieder für die ganze Population ausgewertet, und so fort. Nach einer Reihe von Durchläufen konvergiert das jeweils beste Chromosom einer Bevölkerung (hoffentlich) gegen die optimale Lösung.

Kapitel 8

Programmablauf und Benutzung

8.1 Programmablauf

In diesem Abschnitt wird geschildert, was nach dem Aufruf des Programmes auf Host- und Knotenrechner geschieht.
Nach dem Aufruf des ausführbaren Hostprogrammes f_N_host durch den Benutzer werden auf dem Host folgende Aktionen durchgeführt:

1. Lesen der zu faktorisierenden Zahl N aus der Eingabe.
2. Berechnen des Multipliers k.
3. Überprüfen, ob Aufsetzdatei existiert. Falls nein: Näherung an D bzw. $start\_fb$ berechnen. ($start\_fb$ ist in der Selbstinitialisierungs-Variante die kleinste Primzahl, die in A auftritt)
4. Laden des Programmes f_N_node auf alle Knoten des allokierten Cubes.
5. Folgende Daten werden an alle Knoten verschickt: Die Zahl N, der Multiplier k, die Größe der Faktorbasis, der Toleranzwert T, die Siebintervallänge M. In der Silverman-Variante erhält jeder Knoten verschiedene Grenzen für die Intervalle, aus denen D gewählt wird (siehe Kapitel 5).

Damit ist die Aufgabe des Hosts beendet. Nach dem Verschicken der angegebenen Daten führt er keine Berechnungen mehr aus. Im weiteren Ablauf wurde der Knoten 0 gegenüber den restlichen Knoten hervorgehoben, indem zuerst auf ihm Berechnungen durchgeführt werden, die danach von den anderen Knoten genutzt werden. Damit soll die Client-Server-Struktur eines verteilten Systems nachgebildet werden. Es ist genauso gut möglich, die zuerst auf Knoten 0 durchgeführten Berechnungen sofort auf allen Maschinen ablaufen zu lassen.

Knoten 0 hat zunächst folgende Aufgaben:

1. Empfangen der vom Host verschickten Daten.
2. Bestimmung der Faktorbasis, Ablegen in dem Verzeichnis, in dem der Aufruf von *f_N_host* erfolgte (Aufrufverzeichnis).
3. Bestimmung der $\sqrt{N_p}$ für alle $p \in FB$ und Ablegen im Aufrufverzeichnis.
4. Schicken einer Startnachricht an alle anderen Knoten.

Die anderen Knoten gehen so vor:

1. Empfangen der Daten vom Host.
2. Warten auf Startsignal von Knoten 0.
3. Lesen der Faktorbasis und der $\sqrt{N_p}$.

Danach gehen alle Knoten (nach einer Vorberechnung im Falle der Selbstinitialisierung) in die Endlosschleife

- Berechnen der Koeffizienten und Nullstellen.
- Sieben.
- Testen.

Nach dem Empfangen des Startsignals von Knoten 0 findet keine Kommunikation mehr zwischen Knotenrechnern und Host statt.

8.2 Aufruf

Zunächst wird erklärt, wie der Aufruf des Programms durch einen Benutzer aussehen sollte.
Wie in Kapitel 1 bereits erwähnt, besteht der System Resource Manager nur aus einer Einprozessormaschine. Wegen der geringen Rechenleistung dieser Maschine wurde im Rechenzentrum Saarbrücken das Hypercube-System so installiert, daß eine UNIX-Maschine mit mehreren Prozessoren (Hostname *sbusol*) einen Teil der Aufgaben des Hostrechners übernehmen kann: Sowohl der Programmentwurf als auch das Kompilieren der Knotenprogramme kann auf der *sbusol* erfolgen. Für letzteres steht ein sogenannter Cross-Compiler namens *icc* zur Verfügung. Hostprogramme können nur auf dem SRM mit dem dort installierten *cc*-Compiler kompiliert werden. Um sowohl von der *sbusol* als auch vom SRM aus auf die gleichen Daten zugreifen zu

können, sind die mit ipsc gekennzeichneten sbusol-Verzeichnisse auf dem SRM gemountet.

Die für einen Programmstart der MPQS-Implementierung notwendigen Schritte werden nun der Reihe nach durchgegangen.

1. Das Hostprogramm *f_N_host* ist sowohl für die Silverman-Methode als auch für die Selbstinitialisierung-Methode das selbe. Sowohl das ausführbare Programm als auch der Source-Code befinden sich im Verzeichnis

   `/usr5/in11jb/in11jbto/ipsc/faktor` .

2. Aus Gründen der Übersichtlichkeit befinden sich die Source-Codes des Knotenprogrammes und der vom Knotenprogramm benutzten Funktionen in einem anderen Verzeichnis, nämlich

   `/usr5/in11jb/in11jbto/faktor` .

   Dort liegen jeweils ein Makefile für die Silverman- und die Selbstinitialisierungs-Methode vor. Sie heißen *make4* bzw. *make4_self*. Der Aufruf

   ```
   make -f make4
   ```

   bzw.

   ```
   make -f make4_self
   ```

   erstellt das ausführbare Programm namens *f_N_node*.

3. Nun loggt man sich auf dem Hypercube ein:

   `rlogin ipsc860` .

   Das *sbusol*-Directory /usr5/in11jb/in11jbto/ipsc ist auf dem Hypercube gemountet und heißt dort

   `/usr/in11jb/sbusol/in11jbto/ipsc` .

4. Jetzt muß ein Cube allokiert werden. Mit dem Befehl

   ```
   cubeinfo
   ```

   erhält man eine aktuelle Übersicht über die Belegung des Hypercubes. Diese Übersicht hat folgendes Aussehen:

```
  CUBENAME        USER     SRM            HOST          TYPE
iocube          root     ipsc860        ipsc860       0
sokrates        ph11ahsm ipsc860        ipsc860       4m8rxn8
cube8           in11wptr ipsc860        ipsc860       8m8rxn16
623.ipsc860     ph11rsjb ipsc860        ipsc860       8m8rxn0
defaultname     in11jbto ipsc860        ipsc860       8m8rxn24
```

Die interessanteste Information ist die erste Zahl in der Spalte TYPE. Sie gibt in diesem Beispiel an, daß drei Cubes mit jeweils acht Knoten und ein Cube mit vier Knoten belegt sind.
Mit dem Befehl *getcube* kann man einen eigenen Cube allokieren. Die Option -t gibt an, aus wie vielen Knoten er bestehen soll (wird *getcube* ohne die -t-Option aufgerufen, wird automatisch die in diesem Moment maximal mögliche Zahl von Knoten allokiert). Aufgrund der besonderen Topologie des Hypercubes (siehe Kapitel 1) ist die Anzahl der Knoten in einem Cube immer eine Zweierpotenz. Z.B. fordert man mit

```
getcube -t8
```

einen Cube mit 8 Knoten an. Sind genügend Knoten frei, erscheint die Meldung

```
getcube successful ,
```

anderenfalls

```
cubetype not available .
```

5. Nach erfolgreichem *getcube* kann nun das Hostprogramm durch den Aufruf von

```
f_N_host < zahl
```

gestartet werden, wobei

```
zahl
```

der Name einer ASCII-Datei ist, die die Dezimaldarstellung der zu faktorisierenden Zahl enthält und die in diesem Verzeichnis steht.

6. Ebenfalls im Aufrufdirectory steht eine Datei namens CONFIG. In ihr können unter anderem die Namen der Verzeichnisse angegeben werden, in denen gefundene Relationen und statistische Daten abgelegt werden. In der momentanen Einstellung werden alle diese Dateien auf dem cfs abgelegt. Der Zugang zum cfs erfolgt über eine Shell, die sogenannte *Node Shell*, die durch den Befehl

```
nsh
```

aufgerufen werden kann. Nach einem Wechsel in das Directory

```
/cfs/usr/in11jb/in11jbto
```

können die Dateien angesehen werden.

Die Dateien auf dem cfs können zur Weiterverarbeitung (Suche nach Zykeln in der *DoubleLargePrime*-Variante, Aufstellen des Gleichungssystems, Auswerten der statistischen Daten) per ftp auf einen Rechner des Lehrstuhles mit genügend großem Plattenplatz und Hauptspeicher übertragen werden, indem dort der Aufruf

```
ftp 134.96.92.5
```

erfolgt.

Die zur Weiterverarbeitung nötigen Schritte sind in einem README-File auf dem Rechner crypt5 im Verzeichnis /TMP2/sosnow angegeben.

7. Die Berechnungen auf dem Hypercube können mit dem Befehl

   ```
   killcube
   ```

   abgebrochen werden. Der Befehl

   ```
   relcube
   ```

   bricht die Berechnungen ab und gibt zusätzlich den Cube wieder frei.

Zu beachten ist, daß zu Beginn der Faktorisierung einer neuen Zahl die Aufsetzdateien *last_d* (im Aufrufverzeichnis des Hosts) bzw. MERKE.i (auf dem cfs) sowie die Faktorbasis gelöscht werden müssen. Eine sehr wichtige Rolle spielt die Datei PARAMETER im Aufrufverzeichnis. In ihr können wichtige Parameter wie die Anzahl der Faktorbasiselemente, die Länge des Siebintervalls und die Größe des Toleranzwertes T (vgl. Formel (3.20)) festgelegt werden.
Die mathematischen Funktionen, die während des Programmablaufes benutzt werden (z.B. ggt-Berechnung, Primzahltest) sind zu einer Bibliothek *libmath.a* zusammengefaßt, die sich zusammen mit den Quelldateien in

```
/usr5/in11jb/in11jbto/faktor/mathe_lib
```

auf der *sbusol* befindet.

8.3 Reservierung

Um die Rechenzeiten auf dem Hypercube gerechter und übersichtlicher verteilen zu können, wurde vom Rechenzentrum der Universität vor einiger Zeit ein Scheduling-System eingeführt. Seitdem können durch eine Nachricht an den Systemverwalter bis zu 16 Knoten reserviert werden. Die anderen 16 Knoten (oder alle 32, falls keine Reservierungen vorliegen) stehen für den kurzfristigen Gebrauch zur Verfügung.

Die Reservierung erfolgt durch eine Mail an

`ipsc-macs@rz.uni-sb.de` .

Informationen über das erforderliche Format einer solchen Mail und über die aktuellen Reservierungen erhält man durch einen Aufruf des Befehles *qshow*. Abgesehen von der Reservierung von Rechenzeit gibt es eine andere Methode, die Ressourcen des Hypercubes zu nutzen. Als noch kein Scheduling zur Verfügung stand, habe ich abends die maximal verfügbare Zahl von Knoten allokiert und das Host-Programm gestartet. Dieses Vorgehen hat folgenden Nachteil: habe ich z.B. 8 Knoten erhalten und gibt ein anderer Benutzer einige Minuten nach dem Starten meines Programmes seinen Cube mit 16 Knoten frei, so werden diese 16 Knoten über Nacht nicht genutzt. Ein Ausweg wäre eine Folge von *at*-Jobs; diese sind aber vom Systemverwalter gesperrt. Stattdessen verwende ich ein Shell-Script, das in bestimmten Abständen das Programm abbricht, die Knoten freigibt, mit einem *getcube* die momentan maximal verfügbare Zahl von Knoten allokiert und das Programm neu startet. Der Abstand zwischen zwei Abbrüchen sollte nicht zu groß sein, da man ansonsten zu viele Veränderungen in der Auslastung des Hypercubes verpaßt. Ist andererseits der Abstand zu klein, so geht der größte Anteil der Rechenzeit in die Vorberechnungen und nicht in das Finden von Relationen. Als sinnvoll haben sich Abstände von etwa einer Stunde herausgestellt.

Kapitel 9

Praktische Erfolge

Die Zahlen, die von der QS-Implementierung auf dem Hypercube faktorisiert wurden, fallen in zwei Gruppen: Partitionszahlen aus der "RSA Factoring Challenge" und Zahlen aus dem erweiterten Cunningham-Projekt. Beide Gruppen werden in diesem Kapitel vorgestellt. Außerdem wird in diesem Kapitel auf die Zusammenarbeit zwischen dem QS und der *Elliptic Curve Method* (ECM) eingegangen.

9.1 RSA Factoring Challenge

Eine wichtige Anwendung von Faktorisierungsalgorithmen liegt in der Kryptographie. Eines der bedeutendsten und in der Praxis häufig verwendeten kryptographischen Verfahren ist das RSA-Kryptosystem, dessen Sicherheit auf der Schwierigkeit des Faktorisierungsproblems beruht (vgl. ([Bres89, Kap. 4])). Die Rechte des RSA-Systems für die USA liegen bei der Firma "RSA Data Security", die 1991 einen internationalen Wettbewerb initiiert hat, welcher den Fortschritt der Faktorisierungsalgorithmen dokumentieren und die Forschung auf diesem Gebiet anregen will.
Einer Anregung Hendrik Lenstras folgend, besteht ein Teil dieser "RSA Factoring Challenge" aus einer Liste von Partitionszahlen $p(n)$ (vgl. Definition 2.8), wobei als Argumente n nur Primzahlen auftreten. Partitionszahlen scheinen sich, was die Schwierigkeit ihrer Faktorisierung angeht, mehr oder weniger zufällig zu verhalten. Deshalb treten in der Liste sowohl Zahlen auf, die nur relativ kleine Faktoren haben, als auch solche, die aus zwei etwa gleich großen Faktoren bestehen.

Am Ende jedes Quartals wird das von "RSA Data Security" ausgesetzte Preisgeld von 1750 \$ unter den erfolgreichen Teilnehmern aufgeteilt. Dazu wird folgendes Punktesystem verwendet: Für jede Faktorisierung einer bis dahin nicht faktorisierten Partitionszahl erhält man Punkte. Die Anzahl der Punkte hängt von der Differenz der Dezimalstellen der Zahl und der kleinsten noch nicht faktorisierten Zahl ab.

Sei D die Anzahl der Dezimalstellen, der kleinsten noch nicht faktorisierten Partitionszahl. Für die Faktorisierung einer Zahl mit $D + d$ Dezimalstellen erhält man

$$\begin{array}{lllcccr} 8 \text{ Punkte,} & \text{falls} & 0 & \leq & d & \leq & 9, \\ 4 \text{ Punkte,} & \text{falls} & 10 & \leq & d & \leq & 19, \\ 2 \text{ Punkte,} & \text{falls} & 20 & \leq & d & \leq & 29, \\ 1 \text{ Punkt,} & \text{falls} & 30 & \leq & d & \leq & 49. \end{array}$$

Die Höhe des Gewinns am Ende des Quartals berechnet sich folgendermaßen:

$$\text{Gewinn} = 1750\$ \cdot \frac{\text{Anzahl von eigenen Punkte in diesem Quartal}}{\text{Gesamtanzahl aller Punkte in diesem Quartal}}.$$

Die folgende Liste zeigt die Erfolge unserer Arbeitsgruppe in diesem Wettbewerb:

```
Partition Challenge List:
------------------------

        (3rd Quarter, 1993)
                30 points, Arjen K. Lenstra, $395
                 4 points, Paul C. Leyland, $53
                73 points, Bruce Dodson, Matt Fante, $960
                11 points, Franz-Dieter Berger, Thomas Denny,
                           Johannes Buchmann, $145
                 5 points, Paul Leyland, Arjen K. Lenstra, $66
                 5 points, Bruce Dodson, Arjen K. Lenstra, $66
                 1 point,  Reynald Lercier, Francois Morain, $13
                 1 point,  Factoring Factory Buchmann
                           (Franz-Dieter Berger, Johannes Buchmann,
                            Thomas F. Denny, Andreas M"uller,
                            Thomas Sosnowski), Heinz-Bernd Eggenstein, $13
                 3 points, Factoring Factory Buchmann
                           (Franz-Dieter Berger, Johannes Buchmann,
                            Thomas F. Denny, Andreas M"uller,
                            Thomas Sosnowski), $39
                --------------
               133 points total

        (4th Quarter, 1993)
                 7 points, Factoring Factory Buchmann
                           (Franz-Dieter Berger, Johannes Buchmann,
                            Thomas F. Denny, Andreas M"uller,
                            Thomas Sosnowski), $227
                 3 points, Bruce Dodson, Matt Fante, $97
                26 points, Bruce Dodson, Arjen K. Lenstra, $843
                17 points, Arjen K. Lenstra, $551
```

```
        1 point,  Paul C. Leyland, $32
       --------------
       54 points total

(1st Quarter, 1994)
       48 points, Bruce Dodson, Arjen K. Lenstra, $1218
       19 points, Factoring Factory Buchmann
                  (Franz-Dieter Berger, Johannes Buchmann,
                   Thomas F. Denny, Andreas M"uller,
                   Thomas Sosnowski), $482
        1 point,  Paul C. Leyland, $25
        1 point,  Reynald Lercier, Francois Morain, $25
       --------------
       69 points total

(2nd Quarter, 1994)
       30 points, Factoring Factory Buchmann
                  (Franz-Dieter Berger, Johannes Buchmann,
                   Thomas F. Denny, Andreas M"uller,
                   Thomas Sosnowski), $665
       28 points, Bruce Dodson, Arjen K. Lenstra, $620
       16 points, Magnus Y. Alvestad, Arjen K. Lenstra, $354
        4 points, Paul C. Leyland, $89
        1 point,  Heinz-Bernd Eggenstein,
                  Factoring Factory Buchmann
                  (Franz-Dieter Berger, Johannes Buchmann,
                   Thomas F. Denny, Andreas M"uller,
                   Thomas Sosnowski), $22
       --------------
       79 points total
```

Bei den Partitionszahlen ist das Quadratische Sieb stark auf die Hilfe anderer Faktorisierungsalgorithmen angewiesen, da die zur Zeit (Dezember 1994) kleinste noch nicht faktorisierte Partitionszahl in der Liste bereits 119 Dezimalstellen hat. Geht man davon aus, daß sich beim QS die Laufzeiten bei der Vergrößerung der zu faktorisierenden Zahl um jeweils etwa drei Dezimalstellen verdoppeln, so würde die Faktorisierung dieser Zahl auf den 32 Knoten eines Hypercubes grob geschätzt eineinhalb Jahre dauern. Um in den vom QS auf dem Hypercube in vernünftiger Zeit zu bewältigenden Bereich von nicht mehr als 105 Dezimalstellen zu gelangen, müssen deshalb bereits Faktoren der Partitionszahlen bekannt sein. Besonders hilfreich ist dabei die Zusammenarbeit mit der ECM-Implementierung von Franz-Dieter Berger und Andreas Müller ([Ber93], [Mue95]).

Ein Beispiel für diese Zusammenarbeit liefert die Partitionszahl $p(15923)$:

$$
\begin{array}{rll}
\mathrm{p}(15923) = & 6961480999493689087990123139968445907171929901157011149431262362304\,8 & \\
& 196329781396078589137968583715723220746480934215572541838765128\,7\ (132) & \\
= & 19465999333830490275285946921528097\ (35) & (\mathrm{ECM}) \\
* & 78787867102141936431606615607675035323984250487\ (47) & (\mathrm{QS}) \\
* & 453905655854145899354711077878465162255885535349633\ (51) & (\mathrm{QS})
\end{array}
$$

Der 98-stellige Kofaktor konnte auf dem Hypercube faktorisiert werden. Eine Faktorisierung der ursprünglichen Zahl mit 132 Dezimalstellen wäre mit dem QS nicht möglich gewesen. Anderseits findet ECM (noch) keine 47-stelligen Faktoren. Der weltweite größte bisherige Faktor wurde von der ECM-Implementierung von Franz-Dieter Berger gefunden (vgl. [Ber93]). Er hat 43 Dezimalstellen.

Insgesamt wurden auf dem Hypercube bisher 9 Kofaktoren von Partitionszahlen faktorisiert. Diese Zahlen hatten 77, 83, 84, 87, 87, 88, 98, 101 und 103 Dezimalstellen.

9.2 Erweitertes Cunningham-Projekt

Die andere Gruppe von Zahlen, die mit dem QS auf dem Hypercube faktorisiert wurden, stammt aus dem erweiterten Cunningham-Projekt, das von Richard Brent, Peter Montgomery und Herman Te Riele koordiniert wird (siehe [Bre-Mon-Rie94]). Hierbei geht es um die Faktorisierung von Zahlen der Form

$$a^n \pm 1, \qquad a, n \in \mathbb{N}, 13 \leq a \leq 100.$$

Im Anhang wird darauf eingegangen, in welchen Gebieten der Mathematik diese Zahlen auftreten. Aus dem erweiterten Cunningham-Projekt wurden mit der vorliegenden QS-Implementierung bisher 24 Zahlen faktorisiert, die im Durchschnitt 92 Dezimalstellen lang waren. Die ersten sechs dieser Faktorisierungen sind bereits in dem neuesten Update zum erweiterten Cunningham-Projekt erschienen (siehe [Bre-Mon-Rie94]).

Anhang

In diesem Kapitel werden einige mathematische Eigenschaften der in Kapitel 9 erwähnten Partitions- und Cunninghamzahlen vorgestellt.

9.3 Partitionszahlen

Die Partitionszahlen $p(n)$ sind ein wichtiger Bestandteil der additiven Zahlentheorie. Bereits ihre Berechnung ist für wachsendes n ein nichttriviales Problem. Dies liegt zum Teil daran, daß die Folge der $p(n)$ sehr schnell anwächst, wie der folgende Satz zeigt:

Satz 9.1 (Größenordnung der Partitionsfunktion)

$$p(n) = \Theta\left(\frac{1}{4 \cdot n \cdot \sqrt{3}} \cdot e^{K \cdot \sqrt{n}}\right)$$

für eine Konstante $K \in \mathbb{R}_{>0}$.

Beweis: [Har40, S.116] ■

Die Berechnung von $p(200)$ durch MacMahon stellte im Jahre 1917 eine große Rechenleistung dar. Dabei wurde eine Rekursionsformel verwendet. Hardy und Ramanujan entwickelten mit analytischen Hilfsmitteln die berühmte Kreismethode, in der $p(n)$ zunächst durch eine unendliche Reihe dargestellt und dann durch eine endliche Summe angenähert wird. Mit dieser Methode berechnete D.H. Lehmer 1936 die Zahl $p(721)$, wobei Rademacher erst ein Jahr später der Beweis gelang, daß der genäherte Wert mit dem tatsächlichen übereinstimmt (siehe [Har40]).
Obwohl sich Partitionszahlen, was ihre Faktorisierungen angeht, wie zufällig gewählte Zahlen zu verhalten scheinen, lassen sich interessante Aussagen über ihr Kongruenzverhalten treffen. So fand Ramanujan Beziehungen der Art

$$n \equiv 4 \bmod 5 \Longrightarrow p(n) \equiv 0 \bmod 5.$$

(siehe [Har-Wri79, Theorem 359]).

In der Mathematik treten die $p(n)$ in folgendem Zusammenhang auf:

Satz 9.2
Die Anzahl nichtisomorpher Gruppen der Ordnung q^n ($q \in \mathbb{P}, n \in \mathbb{N}$) ist

$$p(n)$$

oder noch allgemeiner

Satz 9.3
Sei $n = \prod_{i=1}^{k} q_i^{e_i}, e_i \geq 1, q_i \in \mathbb{P}$. Dann ist die Anzahl der nichtisomorphen Gruppen der Ordnung n gleich

$$\prod_{i=1}^{k} p(e_i).$$

Beweis: [Mey80, S.78f] ■

9.4 Cunningham-Zahlen

Cunningham-Zahlen, also Zahlen der Form $a^n \pm 1, a, n \in \mathbb{N}$, treten an verschiedenen Stellen in der Zahlentheorie auf. Ist a eine Primzahl, so existiert für jedes $n \in \mathbb{N}$ ein Körper mit a^n Elementen, dessen multiplikative Gruppe $a^n - 1$ Elemente hat. Eine Faktorisierung dieser Gruppenordnung gibt Aufschluß darüber, welche Untergruppenordnungen überhaupt auftreten können.

Eine andere Anwendung ist die Suche nach *perfekten* Zahlen:

Definition 9.4 (perfekte Zahl)
Für $m \in \mathbb{N}$ ist $\sigma(m)$ die Summe der positiven Teiler von m:

$$\sigma(m) = \sum_{d|m, d>0} d.$$

Eine perfekte Zahl ist eine Zahl m mit

$$\sigma(m) = 2 \cdot m.$$

Da immer $m|m$ gilt, ist eine perfekte Zahl m somit gleich der Summe ihrer positiven Teiler kleiner m.

Beispiel 9.5

$$\sigma(6) = 1 + 2 + 3 + 6 = 2 \cdot 6$$

Den Zusammenhang zwischen der σ-Funktion und den Cunningham-Zahlen stellt der folgende Satz her:

Satz 9.6
Für $n \in \mathbb{N}$, $p \in \mathbb{P}$ ist

$$\sigma(p^n) = \frac{p^{n+1} - 1}{p - 1}.$$

Das bedeutet: bis auf den kleinen Faktor $\frac{1}{p-1}$ ist die Summe der Teiler einer Primzahlpotenz eine Cunningham-Zahl.

Alle bekannten perfekten Zahlen sind gerade. Die Frage nach der Existenz ungerader perfekter Zahlen wird seit Jahrhunderten untersucht, ist aber bis heute unbeantwortet geblieben. Allerdings wurden Methoden entwickelt, um untere Schranken für eine (potentielle) ungerade perfekte Zahl zu beweisen. Bei den Fallunterscheidungen in diesen Beweisen sind die Faktorisierungen von Zahlen $\sigma(p^n)$ für kleine Basen p hilfreich. Das letzte Ergebnis in dieser Richtung war:

Satz 9.7
Ist m eine ungerade perfekte Zahl, so ist $m > 10^{300}$.

Beweis: Siehe [Bre-Coh-Rie91]. ■

Der andere Typ von Cnunningham-Zahlen, nämlich solche der Form $a^n + 1$, tritt als Faktor von $a^{2 \cdot n} - 1$ auf.

Literaturverzeichnis

[Al-Pom93] W.R. Alford, Carl Pomerance: *Implementing the self initializing quadratic sieve on a distributed network*, Manuskript, 1993

[Ber93] Franz-Dieter Berger: *ECM-Faktorisieren mit elliptischen Kurven*, Diplomarbeit, Universität des Saarlandes, 1993

[Bre-Coh-Rie91] R.P. Brent, G.L. Cohen, H. J. J. te Riele: *Improved techniques for lower bounds for odd perfect numbers*, Mathematics of Computations 57, 1991

[Bre-Rie92] R.P. Brent, H.J.J. te Riele: *Factorizations of* $a^n \pm 1, 13 \leq a \leq 100$, CWI Report, Department of Numerical Mathematics NM-R9212 June, 1992

[Bre-Mon-Rie94] R.P.Brent, P.L. Montgomery, H.J.J. te Riele: *Update 1 to Factorizations of* $a^n \pm 1, 13 \leq a \leq 100$, CWI Report, Department of Numerical Mathematics NM-R9419 September, 1994

[Bres89] David M. Bressoud: *Factorization and Primality Testing*, Springer Verlag, 1989

[Bu-Mue92] Johannes Buchmann, Volker Müller: *Primality Testing*, Universität des Saarlandes, 1992

[Coh91] H. Cohen : *A Course in Computational Algebraic Number Theory*, Springer Verlag , 1991

[Co-Le-Ri90] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest :*Introduction to Algorithms*, MIT Press, 1990

[Den93] Thomas F. Denny: *Faktorisieren mit dem Quadratischen Sieb*, Diplomarbeit, Universität des Saarlandes, 1993

[Dentz91] Ralf Dentzer: *libI: eine lange ganzzahlige Arithmetik*, Heidelberg, 1991

[Gau65] C.F. Gauß: *Untersuchungen über höhere Arithmetik (Disquisitiones Arithmeticae)*, Chelsea Publishing, 1965

[Har40] G.H. Hardy: *Ramanujan - Twelve Lectures suggested by his Life and Work*, Chelsea Publishing, 1940

[Har-Wri79] G.H. Hardy, E.M. Wright: *An Introduction to the Theory of numbers*, Clarendon Press Oxford, 1979

[iPSC91] Intel Manuals: *iPSC/2 & iPSC/860. Concurrent Programming, Volume 1 and 2, 1991*

[Knu-Par76] Donald E. Knuth, Luis Trabb Pardo : *Analysis of a simple Factorization Algorithm*, Theoretical Computer Science 3 (1976), S. 321-348

[Len-Man91] A.K. Lenstra, M.S. Manasse: *Factoring with two large primes*, Advances in Cryptology, Eurocrypt '90, 1991

[LIPS91] Ralf Roth, Thomas Setz : *LiPS - A System For Distributed Processing on Workstations*, Dokumentation, 1991

[Mey80] Kurt Meyberg: *Algebra I* , Carl Hanser, 1980

[Mue95] Andreas Müller: *Eine FFT-Continuation für ECM*, Diplomarbeit, Universität des Saarlandes, 1995

[Nij-Wi78] A. Nijenhuis, H.S. Wilf: *Combinatorical Algorithms*, Academic Press, 1978

[Pom82] C. Pomerance: *Analysis and comparison of some integer factoring algorithms*, Computational Methods in Number Theory, Math. Centrum Tract 154, 1982

[Rie89] H. Te Riele, W. Lioen, D. Winter: *Factoring with the quadratic sieve on large vector computers*, Journal of Computational and Applied Mathematics 27, 1989

[Ries87] Hans Riesel: *Prime Numbers and Computer Methods for Factorization*, Birkhäuser, 1987

[Sil87] Robert D. Silverman: *The Multiple Polynomial Quadratic Sieve*, Mathematics of Computations 48, 1987