Challenges in Solving Large Sparse Linear Systems over Finite Fields[*]

Richard P. Brent

23 September 2005

[*]Mike Osborne "70th" Symposium, ANU, Sept 2005

 osbornet

Abstract

This talk outlines how very large, sparse linear systems arise in the solution of problems of interest in computational number theory and public-key cryptography, such as the integer factorization and discrete logarithm problems. The linear systems are over finite fields, often the field $\mathbf{GF}(2)$ of two elements. We describe some algorithms for solving large sparse linear systems over $\mathbf{GF}(2)$, and compare them with algorithms for the real field. In particular, some "iterative" algorithms which are well-known to numerical analysts, such as the conjugate gradient and Lanczos algorithms, can be adapted to work over $\mathbf{GF}(2)$, but there are significant differences between algorithms for the real field and for $\mathbf{GF}(2)$.

Outline

- Finite Fields
- Integer Factorization
  - The RSA Public-Key Cryptosystem
  - Record Integer Factorizations
  - Example of QS Factorization
- Discrete Logarithms
  - Diffie-Hellman Key Agreement
  - Elliptic Curve Cryptography
  - Example of Discrete Log Computation
- Linear Algebra over Finite Fields
- Use of the Lanczos Algorithm
  - Avoid Fill-in
  - Lanczos over Finite Fields
  - Finding Dependencies
  - Block and Parallel Implementations
- Some Statistics for NFS Factorizations

Finite Fields

In computational number theory and cryptographic applications, we often have to work over *finite* fields. A finite field F is a finite set with operations "+" and "×" which satisfy the usual associative, commutative and distributive laws:

$$(x+y)+z = x+(y+z), \quad (xy)z = x(yz),$$

$$x+y = y+x, \quad xy = yx, \quad x(y+z) = xy+xz\ .$$

Here, as usual, we have written $x \times y$ as xy.

$(F, +)$ is an abelian group with zero element 0. $(F\backslash\{0\}, \times)$ is a (nonempty) abelian group with identity 1.

We write $x + x$ as $2x$, $x + x + x$ as $3x$, $x \times x$ as x^2, $x \times x \times x$ as x^3, etc.

Characterisation of Finite Fields

If F is finite field, then $\#F$ is a prime power.

Conversely, for any prime power $q = p^k$, there is a finite field with q elements, often written $\mathbf{GF}(q)$, and this is unique up to isomorphism. ("G" is for Galois.)

p is called the *characteristic* of the finite field, and

$$px = 0$$

holds for all $x \in F$.

The familiar infinite fields Q, R and C are said to have characteristic zero.

Fields of Prime Order

The most important finite fields in applications are those whose order is a prime ($\mathbf{GF}(p)$) or a power of two ($\mathbf{GF}(2^k)$). For simplicity, we only consider the case $F = \mathbf{GF}(p)$, where p is a prime. A representation of F is the integers modulo p. In other words, $\mathbf{GF}(p) \equiv \mathbf{Z}/p\mathbf{Z}$.

Note: The ring $\mathbf{Z}/q\mathbf{Z}$ is *not* a field if $q = p^k$, $k > 1$, because it contains "zero-divisors" ($p \times p^{k-1} = 0$).

p-th powers

For all $x \in F = \mathbf{GF}(p)$, we have

$$x^p = x\ .$$

This is just Fermat's little theorem translated into the language of finite fields.

$\mathbf{GF}(2)$

The simplest finite field is $\mathbf{GF}(2)$, the set of two elements $\{0, 1\}$ with addition and multiplication mod 2.

Equivalently, $\mathbf{GF}(2)$ can be regarded as the set $\{\mathbf{F}, \mathbf{T}\}$ of truth values $\mathbf{F}$ (false) and $\mathbf{T}$ (true) with operations $\oplus$ ("exclusive or") and $\wedge$ ("conjunction").

We can perform linear algebra efficiently on vectors over $\mathbf{GF}(2)$ by packing 32 or 64 elements per word and using word-length $\oplus$ and $\wedge$ operations.

Differences between $\mathbf{GF}(p)$ and R

We can perform linear algebra with vectors and matrices over F, and some familiar concepts such as *rank*, *null-space* are meaningful. If your intuition about fields comes from knowledge of the real field R, then it is important to note the following facts about finite fields $F = \mathbf{GF}(p)$.

- We can represent elements of F exactly and perform arithmetic operations exactly. There is no problem with rounding error.
- We can not define a norm which satisfies the triangle inequality (because $px = 0$).
- We can not define a useful total ordering ">". For example, we might try to define $x > 0$ if $x = y^2 \neq 0$ for some $y \in F$. However, in $\mathbf{GF}(3)$ this gives $1 > 0 > -1 = 1 + 1$. In $\mathbf{GF}(5)$ we have $-1 = 2^2$ so $-1 > 0$. Thus, this definition is not useful.

More Differences

- It is hard to discuss questions of convergence (except in terms of subspaces, e.g. for Krylov subspace methods, or without considering extension fields, p-adic numbers, ...). We restrict our attention to algorithms which (usually) give the *exact* answer in a *finite* number of steps.
- Nonzero vectors can be *self-orthogonal*, i.e. $y^T y = 0$, $y \neq 0$. For example, if $F = \mathbf{GF}(2)$, consider any vector y with an even number of nonzero elements. Self-orthogonal vectors cause technical difficulties when we try to apply some well-known algorithms (Gram-Schmidt orthogonalisation, the Lanczos algorithm, etc).
- We can not define a *positive definite* matrix in the usual way.

Sparse Systems over Finite Fields

Very large, sparse linear systems over finite fields arise in cryptanalysis, when we are trying to "crack" encrypted messages or forge digital signatures. Even if you don't want to do this, you need to know how difficult it is in order to know how secure your encryption/digital signature scheme is.

Large, sparse linear systems over finite fields also arise in other applications, e.g. factorisation of polynomials.

Public Key Cryptography

The RSA (Rivest-Shamir-Adleman) algorithm for public-key cryptography depends for its security on the difficulty of the *integer factorisation problem* – given an integer N which is a product of (say) two large primes p and q, find p and q. (The inverse problem, finding N when p and q are given, is easy.)

At present, N needs to be larger than 512 bits to be considered secure, since 512-bit numbers can be factored. 1024 bits is probably OK for a decade or more, unless a new factoring algorithm is found or a practical quantum computer is built.

Integer Factorisation by the NFS

There are many algorithms for finding a nontrivial factor f of a composite integer N. In cryptographic applications N is usually chosen to be difficult to factor. For example, in the RSA algorithm, $N = pq$, where p and q are primes with about half as many digits as N. (Thus p and q are approximately $\sqrt{N}$, but not too close to $\sqrt{N}$ as this would make it easy to find them!)

In such circumstances, the best available integer factorisation algorithm is the (general) *number field sieve* (NFS). Under plausible assumptions it has expected run time

$$O(\exp(c(\ln N)^{1/3}(\ln \ln N)^{2/3}))\ ,$$

where $c \approx 1.923$ is a constant.
The number of digits in N is $O(\log N)$. A polynomial-time algorithm would run in time polynomial in $\log N$, i.e. $\exp(O(\log \log N))$. Thus, NFS is not a polynomial-time algorithm, though it is better than $N^\varepsilon = \exp(\varepsilon \log N)$ for any $\varepsilon > 0$.

SNFS

Because NFS is a rather complicated algorithm, we concentrate on two simpler algorithms which lead to similar large sparse linear systems – the *special number field sieve* (SNFS) and the (multiple polynomial) *quadratic sieve* (MP)QS. SNFS works on numbers N of a special form, e.g. $N = 2^{512} + 1$. The expected run time is as for NFS except that the constant c is smaller (about 1.53). Historically, SNFS came first, then NFS was developed from it.

QS

The quadratic sieve (QS) and its variants such as MPQS are earlier and simpler algorithms than SNFS/NFS. MPQS has expected run time

$$\exp\left(\sqrt{(1 + o(1)) \ln N \ln\ln N}\right) ,$$

which is worse than NFS because of the square root (instead of cube root) in the exponent. Nevertheless, MPQS is faster than NFS for numbers N up to about 110 decimal digits.

Record Factorisations

At the time of writing (September 2005) the published record factorisations are –

- **MPQS** The 129-decimal digit RSA129 challenge number by Atkins *et al* in 1994. (Since NFS is more efficient on such large numbers, it seems that no one has tried to break this MPQS record.)
- **NFS** A 200-decimal digit number (RSA-200) by Kleinjung *et al* in May 2005.
- **SNFS** A 248-decimal digit number (a factor of $2^{1642} + 1$) by Aoki *et al* in April 2004.

Quadratic Sieve Algorithms

Quadratic sieve algorithms belong to a large class of algorithms which try to find two integers x and y such that $x \neq \pm y \pmod N$ but

$$x^2 = y^2 \pmod N\ . \qquad (1)$$

Once such x and y are found, then GCD $(x - y, N)$ is a nontrivial factor of N. One way to find x and y satisfying (1) is to find a set of *relations* of the form

$$u_i^2 = v_i^2 w_i \pmod N, \qquad (2)$$

where the w_i have all their prime factors in a moderately small set of primes (called the *factor base*). Each relation (2) gives a column in a matrix A whose rows correspond to the primes in the factor base.

Linear Algebra mod 2

Once enough columns have been generated, we can use sparse Gaussian elimination in $\mathbf{GF}(2)$ to find a linear dependency (mod 2) between a set of columns of A. Multiplying the corresponding relations now gives an expression of the form (1).

With probability at least 1/2, we have $x \neq \pm y \bmod N$, and a nontrivial factor of N will be found. If $x = \pm y$, we need to obtain a different linear dependency and try again.

Example of the Quadratic Sieve

Here is a small example[1] of the quadratic sieve used to factor the integer $N = 1098413$.

Let $f(x) = x^2 - N$. We take a *factor base*

$$S = \{2, 7, 13, 17, 19, 23\} .$$

Generally, S will consist of small primes, but we can exclude primes (e.g. 3, 5, 11 here) for which N is not a quadratic residue, since we know that they will not contribute to the solution. We could include -1 to handle $x < \sqrt{N}$.

Since $\sqrt{N} \approx 1048$, we consider values of x close to 1048, and try to factor $f(x)$ over S. For example, trying $x > 1048$ we find:

[1] Similar to an example given by Montgomery [11, §7.6].

QS Example continued

$$\begin{aligned}
f(1051) &= 2^2 \cdot 7 \cdot 13 \cdot 17 \\
f(1063) &= 2^2 \cdot 7^3 \cdot 23 \\
f(1077) &= 2^2 \cdot 7 \cdot 13^3 \\
f(1119) &= 2^2 \cdot 7 \cdot 17^2 \cdot 19 \\
f(1142) &= 7^2 \cdot 13 \cdot 17 \cdot 19 \\
f(1237) &= 2^2 \cdot 13 \cdot 19^2 \cdot 23
\end{aligned}$$

Thus, we can write down a 6×6 matrix A whose rows correspond to the primes in S, and whose entries are 1 if the relevant exponent is odd:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We see that the second, third and sixth columns of A are linearly dependent, in fact $Ax = 0$ if

$$x^T = [011001] .$$

QS Example continued

Generally we would not expect a dependency until the number of columns exceeded the number of rows. Here the first row is zero and the nullspace has dimension 2.

Multiplying the second, third and sixth relations we obtain

$$(1063 \cdot 1077 \cdot 1237)^2 = (2^3 \cdot 7^2 \cdot 13^2 \cdot 19 \cdot 23)^2 \bmod N$$

which simplifies to

$$326330^2 = 391638^2 \bmod N .$$

Since

$$\mathrm{GCD}(391638 - 326330, N) = 563$$

we find the factorisation

$$N = 563 \cdot 1951 .$$

If we multiply the first, fourth and fifth relations we obtain only the trivial

$$810112^2 = 810112^2 \bmod N$$

and this does not yield a factorisation of N.

Sieving

In quadratic sieve algorithms the numbers w_i are the values of one (or more) polynomials with integer coefficients. This makes it easy to find relations by *sieving*. The sieving process is time-consuming but easily parallelised.

Similar comments apply to sieving for the number field sieve, although the details of the sieving process are more complicated than for the quadratic sieve.

The Discrete Logarithm Problem

Several cryptographic algorithms depend on the *discrete logarithm problem*. The simplest form of this problem is – given a prime p, a primitive root g, and an integer $y \in (1, p)$, find an integer $x \in [0, p-2]$ such that

$$g^x = y \text{ in } \mathbf{GF}(p)\ .$$

x is called the *discrete logarithm* of y and we could write

$$x = \log_g y\ .$$

If $p-1$ has a large prime factor q, then finding x seems difficult – as difficult as factoring a number of about the same size as q. The inverse problem, finding y when x, p and g are given, is easy.

The Diffie-Hellman key agreement protocol, the El Gamal algorithms for public key cryptography, and the digital signature algorithm (DSA) all depend for their security on the difficulty of the discrete logarithm problem.

Diffie-Hellman

To illustrate how the discrete logarithm can be used, we sketch the Diffie-Hellman key agreement protocol.

Alice and Bob have agreed on a large prime p and primitive root g (these are public).

Alice chooses a random integer $x \in [2, p-2]$ and sends

$$X = g^x \bmod p$$

to Bob. Meanwhile, Bob chooses an independent random integer $y \in [2, p-2]$ and sends

$$Y = g^y \bmod p$$

to Alice. Both Alice and Bob can then compute a session key

$$K = g^{xy} \bmod p = X^y \bmod p = Y^x \bmod p$$

but it seems difficult for an eavesdropper to compute K without solving a discrete logarithm problem to compute x or y.

Elliptic Curve Cryptography

For cryptographic applications, an advantage of the discrete logarithm problem over the integer factorisation problem is that the former has a natural generalisation to any a group G (more precisely, to the cyclic subgroup $<g>$ generated by an element $g \in G$).

In particular, we might take G to be the group defined by an elliptic curve over a finite field, and g to be some point of large order in G. This is the basis of *elliptic curve cryptography* (ECC).

The elliptic curve discrete logarithm problem is thought to be more difficult than the ordinary discrete logarithm problem (for inputs of the same length). No sub-exponential algorithm for the general case of the elliptic curve discrete logarithm problem is known.

Discrete log example

Here is a small example[2] of a discrete logarithm problem and how we might reduce it to a problem of linear algebra.

Let $p = 229$, $g = 6$, $y = 13$. We want to compute $\log_g y$. We take a *factor base* $S = \{2, 3, 5, 7, 11\}$ of small primes. We now compute powers of g mod p and attempt to factor the results using elements of S. Discarding unsuccessful attempts, we find

$$\begin{aligned} g^1 &= 2 \cdot 3 \\ g^2 &= 2^2 \cdot 3^2 \ \text{(omit)} \\ g^3 &= 2^3 \cdot 3^3 \ \text{(omit)} \\ g^7 &= 2 \cdot 7^2 \\ g^{10} &= 2^2 \cdot 5^2 \\ g^{12} &= 3 \cdot 5 \cdot 11 \\ g^{18} &= 2^4 \cdot 11 \end{aligned}$$

[2] Similar to an example in Menezes *et al* [10, §3.69].

Discrete log example cont.

This gives a system of linear equations mod $p-1$ for $\log_g 2, \ldots, \log_g 11$:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 \\ 2 & 0 & 2 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \log_g 2 \\ \log_g 3 \\ \log_g 5 \\ \log_g 7 \\ \log_g 11 \end{bmatrix} = \begin{bmatrix} 1 \\ 7 \\ 10 \\ 12 \\ 18 \end{bmatrix} \bmod 228$$

Solving this linear system (how? – see below), we find $\log_g 2 = 21$, $\log_g 3 = 208$, $\log_g 5 = 98$, $\log_g 7 = 107$, and $\log_g 11 = 162$.

Finally, computing $13g^k \bmod p$ for successive k, we find

$$13g^2 = 2 \cdot 5 \bmod p$$

so $\log_g 13 + 2 = 21 + 98 \bmod p-1$, so $\log_g 13 = 117$. Note that everything except this last step is independent of y, so can be done once in a precomputation if we need several different discrete logs with the same (p, g).

Solution of the linear system

How do we solve a system of equations mod 228 ? Note that $228 = 2^2 \cdot 3 \cdot 19$ is not a prime. If we attempt to use Gaussian elimination, we may try to divide by a pivot which has no inverse (e.g. any multiple of 2, 3 or 19).

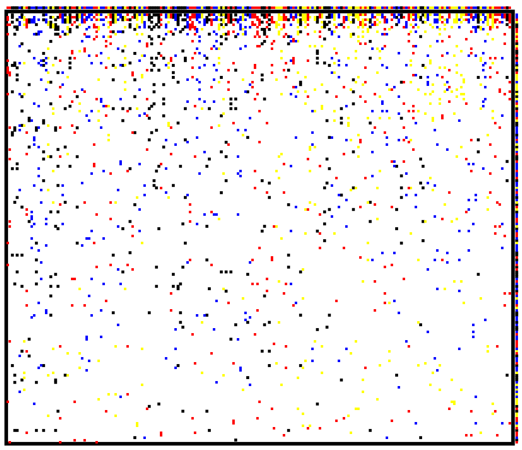
Better is to solve three separate systems mod 2, mod 3 and mod 19. The solution mod 2 gives a solution mod 2^2 by a Newton-like process (Hensel lifting). We can then put the pieces together to construct a solution mod 228 by the Chinese Remainder theorem.

Linear Algebra

In the MPQS and NFS/SNFS factorisation algorithms we have to solve very large, sparse linear systems *exactly* over the finite field $\mathbf{GF}(2)$. More precisely, we have to find dependencies amongst the columns of a large matrix A. If we generate a random x_0, set $b = Ax_0$, and solve $Ax = b$, then $A(x - x_0) = 0$, and there is a good chance that $x \neq x_0$ if the columns of A are dependent.

When solving discrete log problems we also get large sparse linear systems, although the finite fields are no longer $\mathbf{GF}(2)$. Instead, they are $\mathbf{GF}(p)$ where p is a large prime number.

A (small) Large Sparse Matrix



This 179×210 matrix arose in the factorisation of $N = (10^{14} + 31)(10^{15} + 37)$ by MPQS. The matrix has 1916 nonzeros (10.7 per row). The four colours correspond to a partitioning (found by Mondriaan) of the matrix and corresponding vectors (shown at the top and right) across four processors.

Use of Iterative Methods

To avoid problems with "fill in", variants of some familiar "iterative" methods can be used. These methods (based on conjugate gradients or Lanczos) only require matrix-vector multiplications and inner products. Some important points are:

- Nonzero vectors can be orthogonal to themselves ! Montgomery showed how to circumvent this problem by using a variant of the block Lanczos method.
- Many more iterations are required to find the exact solution (actually several exact dependencies) than an approximate solution.
- Preconditioning is useless (although other forms of preprocessing may be useful).
- The matrix is never symmetric.
- Operations over $\mathbf{GF}(2)$ can be parallelised using logical operations on words of (typically) 32 or 64 bits.

Strategy for Finding Dependencies

- If row i has a single nonzero a_{ij} then row i and column j can be eliminated.
- The time for Lanczos is $O(nw)$, where w is the number of nonzeros, so it pays to perform some steps of Gaussian elimination on columns (i.e. combine relations) to decrease n, even at the expense of increasing w slightly. This is sometimes called "filtering". After a certain point it pays to switch to the Lanczos algorithm.
- For efficiency (as we may need several dependencies) and to avoid problems with self-orthogonal vectors, we use Montgomery's block Lanczos algorithm.
- If the solution is to be computed on a parallel machine, it is important to distribute rows and columns to balance both the communication and computational loads on each processor (as far as possible).

Large NFS Factorisations

NFS was used to factor the 155-digit (512-bit) RSA Challenge number RSA155. It was split into the product of two 78-digit primes in August 1999, by a team coordinated from CWI, Amsterdam.

The record has been broken several times since 1999. Currently it stands at 200 decimal digits (RSA200) by Kleinjung *et al.* After removing duplicates there were 2.26×10^9 relations. Filtering produced a matrix with 6.4×10^7 rows and columns, having 1.1×10^{10} non-zero entries. This was solved by the Block-Wiedemann method (Lanczos could also have been used).

Some Statistics

For RSA155, the total amount of CPU time spent on sieving was 8000 Mips-years on assorted machines (calendar time 3.7 months). The resulting matrix had about 6.7×10^6 rows and weight (number of nonzeros) about 4.2×10^8 (about 62 nonzeros per row). Using Montgomery's block Lanczos program, it took almost 224 CPU-hours and 2 GB of memory on a Cray C916 to find 64 dependencies. Calendar time for this was 9.5 days.

For RSA200, sieving took about 120,000 Mips-years (actually about 55 processor-years on 2.2GHz Opteron processors). There were 2.26×10^9 relations and after "filtering" the matrix had 6.4×10^7 rows and columns, and 1.1×10^{10} nonzeros. Linear dependencies were found by the block-Wiedemann method. The linear algebra took 3 months using a cluster of 80 Opterons connected via a Gigabit network.

Summary – Large NFS Factorizations

In Table 1 we summarise the RSA130, RSA155 and RSA200 factorisations. Extrapolation is dangerous because there were algorithmic improvements over time.

Table 1: 130D to 200D Factorisations

	RSA130	RSA155	RSA200
Date	4/1996	8/1999	5/2005
T	500	8000	120000
R	3.5×10^6	6.7×10^6	6.4×10^7
NZ	1.4×10^8	4.2×10^8	1.1×10^{10}
NZ/R	39	62	171
LA	68 hr	224 hr	2160 hr

Here T is the total (estimated) time in Mips-years, R is the number of rows, NZ is the number of nonzeros, and LA is the wall-clock time for linear algebra. The linear algebra was done on a Cray C90 (RSA130), a Cray C916 (RSA155), and a gigabit network of eighty Opterons (RSA200).

References

[1] H. Boender and H. J. J. te Riele, *Factoring integers with large prime variations of the quadratic sieve,* Experimental Mathematics, **5** (1996), 257–273.

[2] R. P. Brent, Recent progress and prospects for integer factorisation algorithms, *LNCS*, Vol. 1858, Springer-Verlag, Berlin, 2000, 3–22. `http://wwwmaths.anu.edu.au/~brent/pub/pub196.html` .

[3] D. Coppersmith, A. Odlyzko and R. Schroeppel, Discrete logarithms in $\mathbf{GF}(p)$, *Algorithmica* **1** (1986), 1–15.

[4] M. Elkenbracht-Huizing, A multiple polynomial general number field sieve *Algorithmic Number Theory – ANTS III*, *LNCS* **1443**, Springer-Verlag, Berlin, 1998, 99–114.

[5] T. Kleinjung *et al*, RSA200, 9 May 2005, `http://www.loria.fr/~zimmerma/records/rsa200` .

[6] B. A. LaMacchia and A. M. Odlyzko, Solving large sparse systems over finite fields, *Advances in Cryptology, CRYPTO '90* (A. J. Menezes and S. A. Vanstone, eds.), *LNCS* **537**, Springer-Verlag, Berlin, 109–133.

[7] C. Lanczos, Solution of systems of linear equations by minimized iterations, *J. Res. Nat. Bureau of Standards* **49** (1952), 33–53.

[8] A. K. Lenstra and H. W. Lenstra, Jr. (editors), The development of the number field sieve, *Lecture Notes in Mathematics* **1554**, Springer-Verlag, Berlin, 1993.

[9] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, The factorization of the ninth Fermat number, *Math. Comp.* **61** (1993), 319–349.

[10] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997. `http://cacr.math.uwaterloo.ca/hac/` .

[11] P. L. Montgomery, A survey of modern integer factorization algorithms, *CWI Quarterly* **7** (1994), 337–366. `ftp://ftp.cwi.nl/pub/pmontgom/cwisurvey.psl.Z` .

[12] P. L. Montgomery, A block Lanczos algorithm for finding dependencies over $\mathbf{GF}(2)$, *Advances in Cryptology: Proc. Eurocrypt'95*, *LNCS* **921**, Springer-Verlag, Berlin, 1995, 106–120. `ftp://ftp.cwi.nl/pub/pmontgom/BlockLanczos.psa4.gz` .

[13] P. L. Montgomery, *Parallel block Lanczos*, Microsoft Research, Redmond, USA, 17 January 2000 (transparencies of a talk presented at RSA 2000).

[14] B. A. Murphy, *Polynomial selection for the number field sieve integer factorisation algorithm*, Ph. D. thesis, Australian National University, July 1999.

[15] A. M. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance, *Advances in Cryptology: Proc. Eurocrypt '84*, *LNCS* **209**, Springer-Verlag, Berlin, 1985, 224–314.

[16] A. M. Odlyzko, The future of integer factorization, *CryptoBytes* **1**, 2 (1995), 5–12. Available from `http://www.rsa.com/rsalabs/pubs/cryptobytes` .

[17] C. Pomerance, A tale of two sieves, *Notices Amer. Math. Soc.* **43** (1996), 1473–1485.

[18] H. te Riele *et al*, *Factorization of a 512-bits RSA key using the number field sieve*, announcement of 26 August 1999, `http://www.loria.fr/~zimmerma/records/RSA155` .

[19] R. L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM* **21** (1978), 120–126.

[20] P. W. Shor, Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Computing* **26** (1997), 1484–1509.

[21] R. D. Silverman, The multiple polynomial quadratic sieve, *Math. Comp.* **48** (1987), 329–339.

[22] Brendan Vastenhouw and Rob H. Bisseling, *A two-dimensional data distribution method for parallel sparse matrix-vector multiplication*, preprint #1238, Dept. of Mathematics, Univ. Utrecht, May 2002.

[23] D. H. Wiedemann, Solving sparse linear equations over finite fields, *IEEE Trans. Inform. Theory* **32** (1986), 54–62.

[24] J. Zayer, *Faktorisieren mit dem Number Field Sieve*, Ph. D. thesis, Universität des Saarlandes, 1995.