A Parallel GNFS Algorithm Based on a Reliable Look-Ahead Block Lanczos Method for Integer Factorization

Laurence T. Yang[1,2], Li Xu[2], Man Lin[2], and John Quinn[2]

[1] Department of Computer Science and Engineering
Jiangsu Polytechnic University
Changzhou, Jiangsu Province, 213164, P.R. China
[2] Department of Computer Science
St. Francis Xavier University
Antigonish, Nova Scotia, B2G 2W5, Canada
{lyang, x2002uwf, mlin, jquinn}@stfx.ca

Abstract. The Rivest-Shamir-Adleman (RSA) algorithm is a very popular and secure public key cryptosystem, but its security relies on the difficulty of factoring large integers. The General Number Field Sieve (GNFS) algorithm is currently the best known method for factoring large integers over 110 digits. Our previous work on the parallel GNFS algorithm, which integrated the Montgomery's block Lanczos method to solve large and sparse linear systems over GF(2), is less reliable. In this paper, we have successfully implemented and integrated the parallel General Number Field Sieve (GNFS) algorithm with the new look-ahead block Lanczos method for solving large and sparse linear systems generated by the GNFS algorithm. This new look-ahead block Lanczos method is based on the look-ahead technique, which is more reliable, avoiding the break-down of the algorithm due to the domain of GF(2). The algorithm can find more dependencies than Montgomery's block Lanczos method with less iterations. The detailed experimental results on a SUN cluster will be presented in this paper as well.

1 Introduction

Today, the Rivest-Shamir-Adleman (RSA) algorithm [21] is the most popular algorithm in public-key cryptosystem and it also has been widely used in real-world applications such as: internet explorer, email systems, online banking, cell phones etc. The security of this algorithm mainly relies on the difficulty of factoring large integers. Many integer factorization algorithms have been developed. Examples are: Trial division [22], Pollard's p-1 algorithm [19], Lenstra Elliptic Curve Factorization (ECM) [13], Quadratic Sieve (QS) [20] and General Number Field Sieve (GNFS) [1,2,3,15] algorithm. GNFS is the best known method for factoring large composite numbers over 110 digits so far.

Although the GNFS algorithm is the fastest algorithm so far, it still takes a long time to factor large integers. In order to reduce the execution time, one natural solution is to use parallel computers. The GNFS algorithm contains several

E. Sha et al. (Eds.): EUC 2006, LNCS 4096, pp. 110–120, 2006.

steps. The most time consuming step is sieving which is used to generate enough relations. This step is very suitable for parallelization because the relation generations are independent. Another step that could benefit from parallel processing is the Montgomery's block Lanczos method [16]. It is used to solve large and sparse linear systems over GF(2) generated by the GNFS algorithm. The disadvantage of this block Lanczos method is its unreliability. The look-ahead block Lanczos method proposed in [8] has overcome this disadvantage and improved the overall reliability of block Lanczos algorithm. There are numerous references available on the look-ahead block Lanczos method [6,7,9,18], but none of those methods can be applied to GF(2) field directly. The algorithm we are developing and implementing is very suitable for solving the generated large and sparse linear systems over small finite fields such as GF(2). In this paper we have successfully developed and implemented the look-ahead block Lanczos method, and integrated together with the GNFS algorithm for integer factorization.

The rest of the paper is organized as follows: we first briefly describe the original GNFS algorithm in section 2. Then we present two block Lanczos methods, namely Montgomery's block Lanczos method [16] and look-ahead block Lanczos method [8] in section 3 and 4 respectively. Section 5 shows the detailed implementation and corresponding parallel performance results.

2 The GNFS Algorithm

The General Number Field Sieve (GNFS) algorithm [2,3,15] is derived from the number fields sieve (NFS) algorithm, developed by Lenstra et al. [14]. It is the fastest known algorithm for integer factorization. The idea of GNFS is from the congruence of squares algorithm [12].

Suppose we want to factor an integer n where n has two prime factors p and q. Let's assume we have two integers s and r, such that s^2 and r^2 are perfect squares and satisfy the constraint $s^2 \equiv r^2 (mod\ n)$. Since $n = pq$, the following conditions must hold [2]:

$$\begin{aligned} pq|(s^2\text{-}r^2) \ &\Rightarrow pq|(s\text{-}r)(s+r) \\ &\Rightarrow p|(s\text{-}r)(s+r) \ and \ q|(s\text{-}r)(s+r). \end{aligned}$$

We know that, if $c|ab$ and $gcd(b,c) = 1$, then $c|a$. So p, q, r *and* s must satisfy $p|(s\text{-}r)$ or $p|(s+r)$ and $q|(s\text{-}r)$ or $q|(s+r)$. Based on this, it can be proved that we can find factors of n by computing the greatest common divisor $gcd(n,(s+r))$ and $gcd(n,(s\text{-}r))$ with the possibility of $2/3$ (see [2]).

Therefore, the essence of GNFS algorithm is based on the idea of factoring n by computing the $gcd(n,\ s+r)$ and $gcd(n,\ s\text{-}r)$. There are six major steps [15]:

1. Selecting parameters: choose an integer $m \in Z$ and a polynomial f which satisfies $f(m) \equiv 0\ (mod\ n)$.
2. Defining three factor bases: rational factor base R, algebraic factor base A and quadratic character base Q.

Table 1. The composite number n and the results after integer factorization

name	number
tst100$_{30}$	727563736353655223147641208603 = 743774339337499•978204944528897
F7$_{39}$	680564733841876926926749214863536422914 = 5704689200685129054721•59649589127497217
tst150$_{45}$	799356282580692644127991443712991753990450969 = 32823111293257851893153•24353458617583497303673
Briggs$_{51}$	556158012756522140970101270050308458769458529626977 = 1236405128000120870775846228354119184397•449818591141
tst200$_{61}$	1241445153765162090376032461564730757085137334450817128010073 = 1127192007137697372923951166979•1101360855918052649813406915187
tst250$_{76}$	3675041894739039405533259197211548846143110109152323761665377505538520830273 = 69119855780815625390997974542224894323•53169119831396634916152282437374262651

3. Sieving: generate enough pairs *(a,b)* (relations) to build a linear dependence.
4. Processing relations: filter out useful pairs *(a,b)* found from sieving.
5. Building up and solve a large and sparse linear system over GF(2).
6. Squaring root: use the results from the previous step to generate two perfect squares, then factor *n*.

Based on the previous studies, the most time consuming step is step 3, sieving. In our previous work [23,24], we have successfully implemented the sieving in parallel with very scalable performance. In this paper, we are focusing on another time consuming part, namely solving the large and sparse linear systems over GF(2) in parallel.

3 Montgomery's Block Lanczos Method

Montgomery's block Lanczos method was proposed by Montgomery in 1995 [16]. This block Lanczos method is a variant of the standard Lanczozs method [10,11]. Both Lanczos methods are used to solve large and sparse linear systems. In the standard Lanczos method, suppose we have a symmetric matrix $A \in R^{n\times n}$. Based on the notations used in [16], the method can be described as follows:

$$
\begin{aligned}
w_0 &= b,\\
w_i &= Aw_{i-1} - \sum_{j=0}^{i-1} \frac{w_j^T A^2 w_{i-1}}{w_j^T A w_j}.
\end{aligned}
\tag{1}
$$

The iteration will stop when w_i*=0.* $\{w_0,\ w_1,\ \ldots\ w_{i-1}\}$ are a basis of *span*$\{b,\ Ab,\ A^2b,\ \ldots\}$ with the properties:

$$\forall 0 \le i < m, \qquad w_i^T A w_i \neq 0, \tag{2}$$

$$\forall 0 \le i < j < m, \qquad w_i^T A w_j = w_j^T A w_i = 0. \tag{3}$$

The solution x can be computed as follows:

$$x = \sum_{j=0}^{m-1} \frac{w_j^T b}{w_j^T A w_j} w_j. \tag{4}$$

Furthermore the iteration of w_i can be simplified as follows:

$$w_i = Aw_{i-1} - \frac{(Aw_{i-1})^T (Aw_{i-1})}{w_{i-1}^T (Aw_{i-1})} w_{i-1} - \frac{(Aw_{i-2})^T (Aw_{i-1})}{w_{i-2}^T (Aw_{i-2})} w_{i-2}.$$

The total time for the standard Lanczos method iss *O(dn^2)+O(n^2)*, d is the average number of nonzero entries per column.

The Montgomery's block Lanczos method is an extension of the standard Lanczos method applied over field GF(2). The major problem for working on GF(2) is that inner products are very likely to become zero because of the binary entries, then the algorithm breaks down accordingly, can not proceed easily. The Montgomery's block Lanczos method is the first attempt to avoid such break down by using N vectors at a time (N is the length of the computer word). Instead of using vectors for iteration which easily leads to inner products to zero, we are using the subspace instead. First we generate the subspace:

$$\begin{aligned} \mathcal{W}_i \qquad & isA - invertible, \\ \mathcal{W}_j^T A \mathcal{W}_i = \{0\}, & \{i \neq j\}, \\ A\mathcal{W} \subseteq \mathcal{W}, \quad & \mathcal{W} = \mathcal{W}_0 + \mathcal{W}_1 + \ldots + \mathcal{W}_{m-1}. \end{aligned} \tag{5}$$

Then we define x to be:

$$x = \sum_{j=0}^{m-1} W_j (W_j^T A W_j)^{-1} W_j^T b, \tag{6}$$

where W is a basis of $\mathcal{W}$. The iteration in the standard Lanczos method will be changed to:

$$\begin{aligned} W_i &= V_i S_i, \\ V_{i+1} &= AW_i S_i^T + V_i - \sum_{j=0}^{i} W_j C_{i+1,j} \qquad (i \geq 0), \\ \mathcal{W}_i &= \langle W_i \rangle, \end{aligned} \tag{7}$$

in which

$$C_{i+1,j} = (W_j^T A W_j)^{-1} W_j^T A (AW_i S_i^T + V_i). \tag{8}$$

This iteration will stop when $V_i^T AV_i$=0 where $i = m$. The iteration can also be further simplified as follows:

$$V_{i+1} = AV_i S_i S_i^T + V_i D_{i+1} + V_{i-1} E_{i+1} + V_{i-2} F_{i+1}.$$

where $D_{i+1}, E_{i+1}, F_{i+1}$ are:

$$D_{i+1} = I_N - W_i^{inv}(V_i^T A^2 V_i S_i S_i^T + V_i^T A V_i),$$
$$E_{i+1} = -W_{i-1}^{inv} V_i^T A V_i S_i S_i^T,$$
$$F_{i+1} = -W_{i-2}^{inv}(I_N - V_{i-1}^T A V_{i-1} W_{i-1}^{inv})(V_{i-1}^T A^2 V_{i-1} S_{i-1} S_{i-1}^T + V_{i-1}^T A V_{i-1}) S_i S_i^T.$$

$\mathbf{S}_i$ is an $N \times N_i$ projection matrix (N is the length of computer word and $N_i < N$). The cost of the Montgomery's block Lanczos method will be reduced to *$O(n^2)+O(dn^2/N)$.*

4 Look-Ahead Block Lanczos Method

In this paper, the look-ahead block Lanczos method over small finite fields such as GF(2) we are developing is mainly based on the method proposed in [8]. There are some advantages of such look-ahead block Lanczos method compared with Montgomery's block Lanczos method: first of all, this method is bi-orthogonalizing, so the input matrix generated from GNFS does not need to be symmetric. In order to apply Montgomery's block Lanczos method, we need to multiply the coefficient matrix A with A^T. However over GF(2), the rank of the product $A^T A$ is, in general, much less than that of A. Thus, when applied to find elements of the nullspace of A, the Montgomery's block Lanczos method may find many spurious vectors. Secondly, also more importantly, it solves the problem of break down we mentioned before, namely ($\mathcal{W}_i{}^T A \mathcal{W}_i$=$\{0\}$).

Due to the limited space, we only outline the algorithm in the paper. First we choose v_0 and u_0 from $\mathbb{K}^{n\times N}$. Then we will compute $v_1,\ v_2,\ \cdots\ ,\ v_{m-1}$ and $u_1,\ u_2,\ \cdots\ ,\ u_{m-1}$. We try to achieve the following conditions:

- $K(A^T,u_0)$=$\bigoplus_{i=0}^{m-1}\langle \mathrm{u}_i\rangle$ and $K(A,v_0)$=$\bigoplus_{i=0}^{m-1}\langle v_i\rangle$.
- Each subspace $\langle u_i\rangle$ and $\langle v_i\rangle$ is of dimension at most N.
- For all $0\leq i<m$, $u_i^T A v_i$ is invertible.
- For all $0 \leq i, j \leq m$ with $i \neq j$, $u_i^T A v_j = 0$ and $u_j^T A v_i = 0$.

Then we can decompose the vector spaces $\langle u_i\rangle$ and $\langle v_i\rangle$. Define variables $\bar{v}_i$, $\bar{u}_i$, $\hat{v}_i$, $\hat{u}_i$, $\check{v}_i^i$, $\check{u}_i^i$, σ_i^v and σ_i^u have the properties:

- $\hat{v}_i^T A v_i = 0$.
- $u_i^T A \hat{v}_i = 0$.
- $\bar{u}_i^T A \bar{v}_i$ is invertible.

and

$$\check{u}_i^i := \{\bar{u}_i^i | \hat{u}_i^i\} = u_i \sigma_i^u, \tag{9}$$

$$\check{v}_i^i := \{\bar{v}_i^i | \hat{v}_i^i\} = v_i \sigma_i^v, \tag{10}$$

σ_i^v and σ_i^u are two invertible matrices in $\mathbb{K}^{N\times N}$. This may be computed by performing a Gauess-Jordan decomposition of the matrix $u_i^T A v_i$ and using the output to select the independent row and column vectors, which then correspond to the columns of $\bar{v}_i$ and $\bar{u}_i$, respectively. The matrices σ_i^u and σ_i^v permute these columns to the front and apply row and column dependencies, respectively, to give $\hat{u}_i$ and $\hat{v}_i$. We define $\check{u}_i$ and $\check{v}_i$ to be the matrices representing this decomposition: $\check{v}_i = v_i \sigma_i^v$, $\check{u}_i = u_i \sigma_i^u$. Through this, then v_{i+1} and u_{i+1} can be computed by:

$$v_{i+1} = Av_i - \sum_{k=0}^{i} \bar{v}_k (\bar{u}_k^T A \bar{v}_k)^{-1} \bar{u}_k^T A^2 v_i, \tag{11}$$

$$u_{i+1} = A^T u_i - \sum_{k=0}^{i} \bar{u}_k (\bar{v}_k^T A^T \bar{u}_k)^{-1} \bar{v}_k^T (A^T)^2 u_i. \tag{12}$$

In computing u_{i+2} and v_{i+2} in next iteration, we have the following situations:

$$(\check{u}_{i-1}|\check{u}_{i-1})^T A(\check{v}_{i-1}|\check{v}_i|v_{i+1}|Av_{i+1}) = \left(\begin{array}{c|c|c|c} r_{i-1,i-1} & & & u_{i-1}^T A^2 v_{i+1} \\ \hline & r_{ii} & & s_{i+1,i+2} \\ & 0 & r_{i,i+1} & r_{i+1,i+2} \end{array}\right) \tag{13}$$

Since $r_{i-1,i-1}$ and $r_{i,i}$ are assumed invertible (modifying to operate in the case where it is not invertible is straightforward), elimination steps to zero $u_{i-1}^T A^2 v_{i+1}$ and $s_{i+1,i+2}$ are performed. For the cases of $r_{i,i+1}$ has full rank or not, we cope differently. We continue the same manner until all rows corresponding to u_i have an associated invertible minor. The iterative formula has been simplified as follows:

$$u_{i+1} = A^T u_i - \sum_{k=0}^{i} \dot{u}_k^i ((\bar{v}_k^i)^T A^T \dot{u}_k^i)^{-1} (\bar{v}_k^i)^T (A^T)^2 u_i, \tag{14}$$

$$v_{i+1} = Av_i - \sum_{k=0}^{i} \dot{v}_k^i ((\bar{u}_k^i)^T A \dot{v}_k^i)^{-1} (\bar{u}_k^i)^T A^2 v_i. \tag{15}$$

The elimination and decomposition steps presented above do not yield sufficient orthogonality conditions to allow computation of a candidate system solution easily. We would continue the elimination and decomposition until it has a permuted block diagonal structure, in which the non-zero parts are as closely clustered around the diagonal as possible. Please refer to [8] for details. Eventually, the solution x can be calculated by:

$$x = \sum_{i=0}^{m-1} \dot{v}_i^m ((\bar{u}_i^m)^T A \dot{v}_i^m)^{-1} (\bar{u}_i^m)^T b. \tag{16}$$

5 Parallel Implementation Details

As we mentioned before, the most time consuming part in GNFS is sieving. This part has already been parallelized in our previous work [23,24]. This paper is build on top of the our previous parallel implementation. Our overall parallel code is built on the sequential source GNFS code from Monico [15].

5.1 Hardware and Programming Environment

The whole implementation is built on two software packages, the sequential GNFS code from Monico [15] (Written in ANSI C) and the sequential Lookahead block Lanczos code from Hovinen [8] (Written in C++). For parallel implementation, MPICH1 (Message Passing Interface) [5] library is used, version 1.2.5.2. The GMP 4.x is also used [4] for precision arithmetic calculations. We use GUN compiler to compile whole program and MPICH1 [17] for our MPI library. The cluster we use is a Sun cluster from University of New Brunswick Canada whose system configurations is:

- Model: Sun Microsystems V60.
- Architecture: x86 cluster.
- Processor count: 164.
- Master processor: 3 GB registered DDR-266 ECC SDRAM.
- Slave processor: 2 to 3 GB registered DDR-266 ECC SDRAM.

In the program, each slave processor only communicates with the master processor. Figure 1 shows the flow chart of our parallel program.

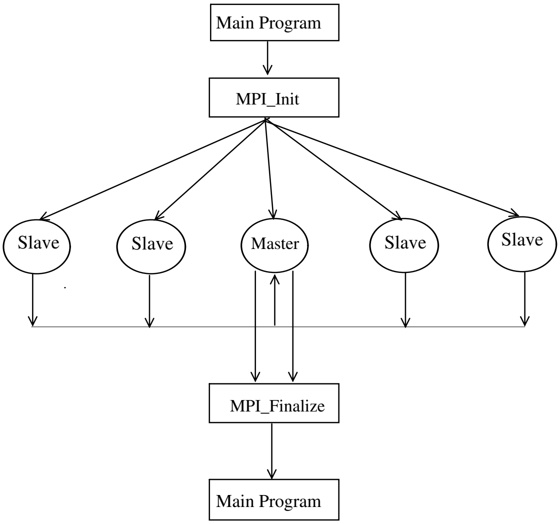

Fig. 1. Each processors do the sieving at the same time, and all the slave nodes send the result back to master node

6 Performance Evaluation

We have six test cases, each test case have a different size of n, all are listed in Table 1.

The sieving time increases when the size of n increases. Table 2 shows the average sieving time for each n with one processor. Table 3 shows the number

Table 2. Average sieving time for each n

name	number of sieve	average sieve time(s)
tst100$_{30}$	1	35.6
F7$_{39}$	1	28.8
tst150$_{45}$	5	50.6
Briggs$_{51}$	3	85.67
tst200$_{61}$	7	560.6
tst250$_{76}$	7	4757.91

Table 3. Number of processors for each test case

name	number of slave processors
tst100$_{30}$	1,2,4,8,16
F7$_{39}$	1,2,4,8,16
tst150$_{45}$	1,2,4,8,16
Briggs$_{51}$	1,2,4,8,16
tst200$_{61}$	1,2,4,8,16
tst250$_{76}$	1,2,4,8,16

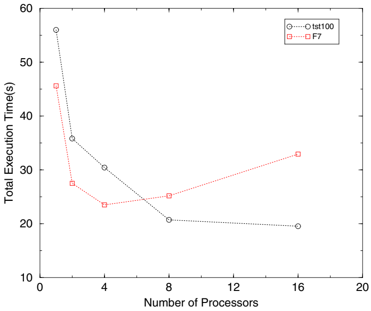

Fig. 2. Execution time for tst100 and F7

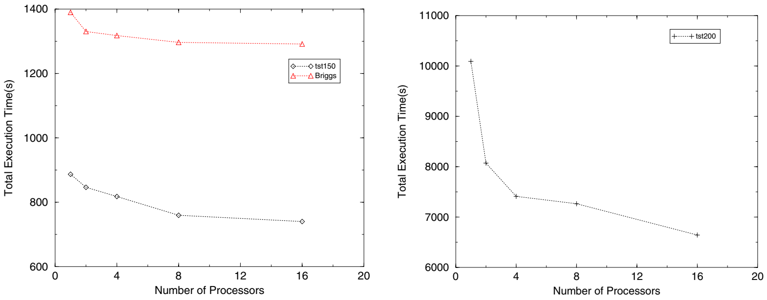

Fig. 3. Execution time for tst150, Briggs and tst200

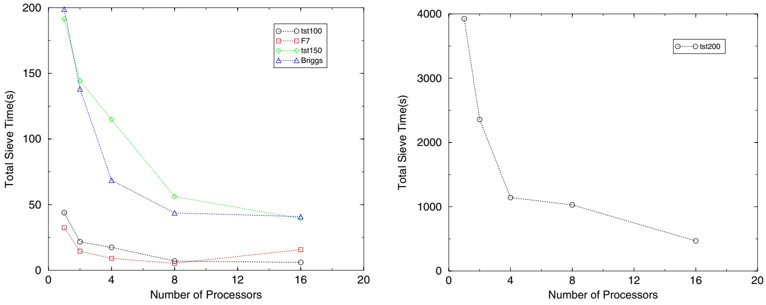

Fig. 4. Sieve time for tst100, F7, tst150, Briggs and tst200

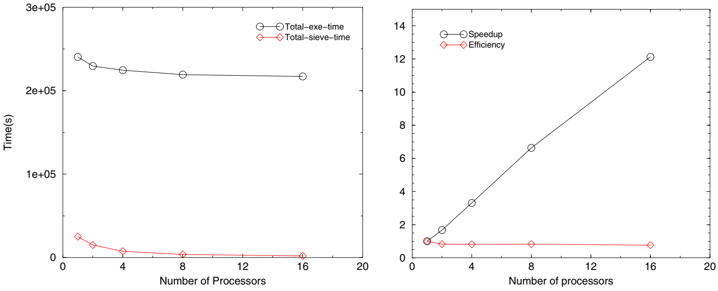

Fig. 5. Total execution time, sieve time, speedup and efficiency for test case tst250

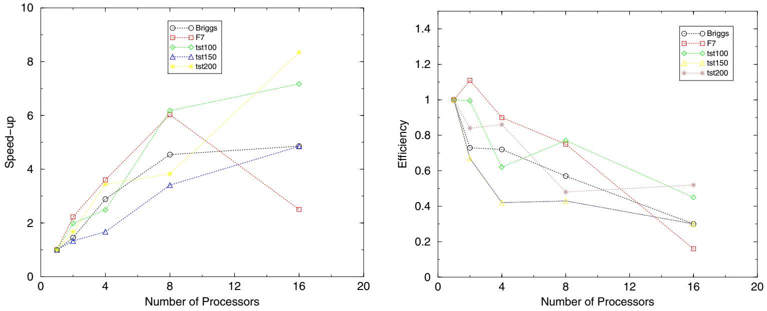

Fig. 6. Speedups and parallel efficiency

of processors we use for each test case. Figure 2 and 3 show the total execution time for each test case in seconds.

The total sieve time for test case: tst100, F7, tst150, Briggs and tst200 are presented in Figure 4. Figure 5 gives the total execution time, sieve time, speedups and parallel efficiency with different processor numbers. Figure 6 gives the speed-ups and parallel efficiency for each test case with different processor numbers.

Additionally, based on our comparisons on a few limited test cases, the method can find more dependencies than Montgomery's block Lanczos method with less iterations. We will report the details in future publications.

Acknowledgements

We would like to thank C. Monico of Texas Tech University and B. Hovinen of University of Waterloo. Our work is based on their sequential source codes. They also helped us with some technical problems through emails. Dr. Silva from IBM advised us many good ideas for our parallel program.

References

1. M. E. Briggs. An introduction to the general number field sieve. Master's thesis, Virginia Polytechnic Institute and State University, 1998.
2. M. Case. A beginner's guide to the general number field sieve. Oregon State University, ECE575 Data Security and Cryptography Project, 2003.
3. J. Dreibellbis. Implementing the general number field sieve. pages 5–14, June 2003.
4. T. Granlund. *The GNU Multiple Precision Arithmetic Library.* TMG Datakonsult, Boston, MA, USA, 2.0.2 edition, June 1996.
5. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994.
6. M. H. Gutknecht. Block krylov space methods for linear systems with multiple right-hand sides. In *The Joint Workshop on Computational Chemistry and Numerical Analysis (CCNA2005)*, Tokyo, Dec 2005.
7. M. H. Gutknecht and T. Schmelzer. A QR-decomposition of block tridiagonal matrices generated by the block lanczos process. In *Proceedings IMACS World Congress*, Paris, July 2005.
8. B. Hovinen. Blocked lanczos-style algorithms over small finite fields. Master Thesis of Mathematics, University of Waterloo, Canada, 2004.
9. R. Lambert. *Computational Aspects of Discrete Logarithms.* PhD thesis, University of Waterloo, 1996.
10. C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. In *Journal of Research of the National Bureau of Standards*, volume 45, pages 255–282, 1950.
11. C. Lanczos. Solutions of linread equations by minimized iterations. In *Journal of Research of the National Bureau of Standards*, volume 49, pages 33–53, 1952.
12. A. K. Lenstra. Integer factoring. *Designs, Codes and Cryptography*, 19(2-3):101–128, 2000.
13. H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics(2)*, 126:649–673, 1987.
14. H. W. Lenstra, C. Pomerance, and J. P. Buhler. Factoring integers with the number field sieve. In *The Development of the Number Field Sieve*, volume 1554, pages 50–94, New York, 1993. Lecture Notes in Mathematics, Springer-Verlag.
15. C. Monico. General number field sieve documentation. GGNFS Documentation, Nov 2004.
16. P. L. Montgomery. A block lanczos algorithm for finding dependencies over gf(2). In *Proceeding of the EUROCRYPT '95*, volume 921 of *LNCS*, pages 106–120. Springer, 1995.

17. MPICH. `http://www-unix.mcs.anl.gov/mpi/mpich/`.
18. B. N. Parlett, D. R. Taylor, and Z. A. Liu. A look-ahead lanczos algorithm for unsymetric matrics. *Mathematics of Computation*, 44:105–124, 1985.
19. J. M. Pollard. Theorems on factorization and primality testing. In *Proceedings of the Cambridge Philosophical Society*, pages 521–528, 1974.
20. C. Pomerance. The quadratic sieve factoring algorithm. In *Proceeding of the EUROCRYPT 84 Workshop on Advances in Cryptology: Theory and Applications of Cryptographic Techniques*, pages 169–182. Springer-Verlag, 1985.
21. R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. Technical Report MIT/LCS/TM-82, 1977.
22. M. C. Wunderlich and J. L. Selfridge. A design for a number theory package with an optimized trial division routine. *Communications of ACM*, 17(5):272–276, 1974.
23. L. Xu, L. T. Yang, and M. Lin. Parallel general number field sieve method for integer factorization. In *Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA-05)*, pages 1017–1023, Las Vegas, USA, June 2005.
24. L. T. Yang, L. Xu, and M. Lin. Integer factorization by a parallel gnfs algorithm for public key cryptosystem. In *Proceddings of the 2005 International Conference on Embedded Software and Systems (ICESS-05)*, pages 683–695, Xian, China, December 2005.