A Parallel GNFS Algorithm with the Biorthogonal Block Lanczos Method for Integer Factorization

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

[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. Currently, RSA is a very popular, widely used and secure public key cryptosystem, but the security of the RSA cryptosystem is based on the difficulty of factoring large integers. The General Number Field Sieve (GNFS) algorithm is 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 algorithm to solve the large and sparse linear systems over GF(2), has one major disadvantage, namely the input has to be symmetric (we have to symmetrize the input for nonsymmetric case and this will shrink the rank).

In this paper, we successfully implement the parallel General Number Field Sieve (GNFS) algorithm and integrate with a new algorithm called the biorthogonal block Lanczos algorithm for solving large and sparse linear systems over GF(2). This new algorithm is based on the biothorgonal technique, can find more solutions or dependencies than Montgomery's block Lanczos method with less iterations. The detailed experimental results on a SUN cluster will be presented as well.

1 Introduction

Currently, Rivest-Shamir-Adleman (RSA) algorithm [16] is the most popular algorithm in public-key cryptosystem. It also has been widely used in the real-world applications such as: internet explorer, email systems, online banking, critical electronic transactions and many other places. The security of this algorithm mainly relies on the difficulty of factoring large integers. So far, many integer factorization algorithms have been developed such as: Trial division [18], Pollard's p-1 algorithm [14], Lenstra Elliptic Curve Factorization (ECM) [9], Quadratic Sieve (QS) [15] and General Number Field Sieve (GNFS) [1,2,3,11] algorithm.

Although GNFS algorithm is the fastest integer factoring algorithm over 110 digits so far, it still takes a long time to factor large integers. In order to reduce

L.T. Yang et al. (Eds.): ATC 2006, LNCS 4158, pp. 428–438, 2006.

the execution time, one natural solution is to distribute jobs to parallel computers. The GNFS algorithm contains several time consuming steps. The most time consuming one is the sieving part which is used to generate enough relations. This step is very suitable for parallelization because the relation generations are independent. Another possible step is, in GNFS, the Montgomery's block Lanczos algorithm [12]. It is used to solve large and sparse linear systems over GF(2) generated by the GNFS algorithm. This block Lanczos algorithm has two drawbacks: *1.* The input of this algorithm is restricted to a symmetric matrix. For the nonsymmetric inputs, we have to make them symmetric first by multiplying the coefficient matrix A with A^T. However over GF(2), the rank of the product A^TA 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. *2.* It will break down for some case. The biorthogonal block Lanczos [5] has overcome the first drawback and can find more solutions or dependencies than Montgomery's block Lanczos method with less iterations. In this paper we have successfully implemented the biorthogonal block Lanczos algorithm and integrated together with the GNFS algorithm.

The rest of this paper is organized as follows: First we briefly describe the original GNFS algorithm in section 2. Then we present two block Lanczos algorithms, namely the Montgomery's block Lanczos algorithm [12] and the biorthogonal block Lanczos algorithm [5] 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 [2,3,11] is derived from the number fields sieve (NFS) algorithm, developed by Lenstra et al [10]. It is the fastest known algorithm for integer factorization. The idea of GNFS is from the congruence of squares algorithm [8].

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 the factoring n by computing the $gcd(n, s+r)$ and $gcd(n, s\text{-}r)$. There are six major steps [11]:

1. Selecting parameters: Choose an integer $m \in Z$ and a polynomial f which satisfy $f(m) \equiv 0 \ (mod\ n)$.

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

Name	Number
$tst100_{30}$	727563736353655223147641208603 = 743774339337499•978204944528897
$F7_{39}$	680564733841876926926749214863536422914 = 570468920068512905472 1•59649589127497217
$tst150_{45}$	799356282580692644127991443712991753990450969 = 3282311129325785189315 3•24353458617583497303673
$Briggs_{51}$	556158012756522140970101270050308458769458529626977 = 123640512800012087077584622835411918439 7•449818591141
$tst200_{61}$	124144515376516209037603246156473075708513733445081712801007 3 = 112719200713769737292395116697 9•110136085591805264981340691518 7
$tst250_{76}$	367504189473903940553325919721154884614311010915232376166537750553852083027 3 = 691198557808156253909979745422248943 23•531691198313966349161522824373742626 51

2. Defining three factor bases: rational factor base R, algebraic factor base A and quadratic character base Q.
3. Sieving: Generate enough pairs *(a,b)* (relations) to build a linear dependence.
4. Processing relations: Filter out useful pairs *(a,b)* that were found from sieving.
5. Building up a large and sparse linear system over GF(2) and solve it.
6. Squaring root, use the results from the previous step to generate two perfect squares, then factor n.

3 Montgomery's Block Lanczos Algorithm

In 1995, Montgomery proposed an algorithm for solving linear systems over GF(2) named Montgomery's block Lanczos algorithm [12]. This block Lanczos algorithm is a variant of standard Lancozs method [6,7]. Both Lanczos algorithms are used to solve linear systems. In the standard Lanczos algorithm, suppose we have a symmetric matrix $A \in \mathbb{R}^{n \times n}$. Based on the notations used in [12], the algorithm 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 basis of $span\{b, Ab, A^2b, \ldots\}$ with the properties:

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

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

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:

$$w_i = Aw_{i-1} - \frac{(Aw_{i-1})^T (Aw_{i-1})}{w_{i-1}^T (Aw_{i-1})} w_{i-1} - \frac{(Aw_{i-2})^T (Aw_{i-1})}{w_{i-2}^T (Aw_{i-2})} w_{i-2}. \tag{5}$$

The time complexity of the Standard Lanczos algorithm is $O(dn^2)+O(n^2)$, d is the average nonzero entries per row or column.

The Montgomery's block Lanczos algorithm is an extension of the Standard Lanczos algorithm by applying it over GF(2). There are some good properties on GF(2), for example, we can apply matrix to N vectors at a time (N is the length of computer word) and we can also apply bitwise operations. Instead of using vectors for iteration, we using subspace instead. First we generate subspace:

$$\begin{aligned} \mathcal{W}_i \qquad & is\ A - invertible, \\ \mathcal{W}_j^T A \mathcal{W}_i = \{0\}, & \{i \neq j\}, \\ A\mathcal{W} \subseteq \mathcal{W}, \quad & \mathcal{W} = \mathcal{W}_0 + \mathcal{W}_1 + \ldots + \mathcal{W}_{m-1}. \end{aligned} \tag{6}$$

Then we define x to be:

$$x = \sum_{j=0}^{m-1} W_j (W_j^T A W_j)^{-1} W_j^T b, \tag{7}$$

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

$$\begin{aligned} W_i &= V_i S_i, \\ V_{i+1} &= AW_i S_i^T + V_i - \sum_{j=0}^{i} W_j C_{i+1,j} \quad (i \geq 0), \\ \mathcal{W}_i &= \langle W_i \rangle, \end{aligned} \tag{8}$$

in which

$$C_{i+1,j} = (W_j^T A W_j)^{-1} W_j^T A (AW_i S_i^T + V_i). \tag{9}$$

This iteration will stop when ${V_i}^T A V_i = 0$ where $i = m$. The iteration can also be 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 $D_{i+1}, E_{i+1}, F_{i+1}$ can be computed:

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

S_i is an $N \times N_i$ projection matrix (N is the length of computer word and $N_i < N$). We can reduce the iteration time from $O(n^2)$ to $O(n^2/N)$ (n is the matrix's row or column size) using the block Lanczos algorithm.

4 Biorthogonal Block Lanczos Algorithm

The Biorthogonal block Lanczos algorithm is from standard biorthogonal scalar Lancozs algorithm. The idea of standard biorthogonal scalar Lancozs algorithm is proposed by Lanczos [7]. Like the symmetric case we described in section 3, first we choose two vector u_0 and v_0 from $\mathbb{K}^n$ form two basis $\{u_0,...u_{m-1}\}$ and $\{v_0,...v_{m-1}\}$, and the following conditions must be held [5]:

$$\begin{aligned} \forall 0 \le i < m \qquad & u_i^T A v_i \neq 0, \\ \forall 0 \le i < j < m \quad & u_i^T A v_j = u_j^T A v_i = 0. \end{aligned} \tag{10}$$

Then the solution will be:

$$x = \sum_{i=0}^{m-1} \frac{u_i^T b}{u_i^T A v_i} v_i. \tag{11}$$

With the condition (10), now we are ready to give the new iterations:

$$\begin{aligned} u_{i+1} &:= A^T u_i - \sum_{k=0}^{i} \frac{(A^T u_i)^T A v_k}{u_k^T A v_k} u_k, \\ v_{i+1} &:= A v_i - \sum_{k=0}^{i} \frac{(A^T u_k)^T A v_i}{u_k^T A v_k} v_k. \end{aligned} \tag{12}$$

Proposition 1. *If $u_0,....,u_i$ and $v_0,....,v_i$ are defined in (12), then $u_i^T A^2 v_k = u_k^T A^2 v_i = 0$ for all $0 \le k < i$-1 [5].*

According to the Proposition *1*, all the projections will be vanished except the last two, the iterations are simplified to follows:

$$u_{i+1} := A^T u_i - \frac{(A^T u_i)^T A v_i}{u_i^T A v_i} u_i - \frac{(A^T u_i)^T A v_{i-1}}{u_{i-1}^T A v_{i-1}} u_{i-1}, \tag{13}$$

$$v_{i+1} := A v_i - \frac{(A^T u_i)^T A v_i}{u_i^T A v_i} v_i - \frac{(A^T u_{i-1})^T A v_i}{u_{i-1}^T A v_{i-1}} v_{i-1}. \tag{14}$$

Instead of using scalars in real fields, now we extend this standard biorthogonal scalar Lancozs algorithm to our problem over GF(2). The input of this algorithm can be either symmetric or nonsymmetric. Montgomery's block Lanczos algorithm only takes a symmetric matrix as the input. For the nonsymmetric matrix, some preconditioning must be performanced first, such as A^TA. Generally speaking, The rank of A^TA is much less than the rank of A, Thus, when applied to find elements of the nullspace of A, the Montgomery's block Lanczos method may find many spurious vectors, then lose some solutions accordingly.

The procedure of biothogonal block Lanczos algorithm are: first we choose u_0, $v_0 \in \mathbb{K}^{n\times N}$ randomly and uniformly. (here u and v are block vector). Then we compute u_1, u_1,...u_{m-1} and v_1, v_1,...v_{m-1}. Two matrices ξ_i and ω_i are also computed by the columns of u_i and v_i.

The following conditions must be hold through the whole algorithm:

1. $K(A^T,\ u_0) = \bigoplus_{i=0}^{m-1}\langle\xi_i\rangle$.
2. $K(A,v_0) = \bigoplus_{i=0}^{m-1}\langle\omega_i\rangle$.
3. for all $0\le i<m$, $\xi_i^T A\omega_i$ is invertible.
4. for all $0\le i,\ j<m$ with $i \ne j$, $\xi_j^T Av_i = u_i^T A\omega_j = 0$.

Assuming all the conditions are held, we can give the biorthogonal Block Lanczos iterations. Let ξ_i and ω_i be two projection matrices and define by $\xi_i = u_i s_i$ and $\omega_i = v_i t_i$.

The iterations for u_{i+1}, v_{i+1} are:

$$u_{i+1} := A^T\xi_i s_i^T + u_i - \sum_{k=0}^{i}\xi_k(\omega_k^T A\xi_k)^{-1}\omega_k^T A^T(A^T\xi_i s_i^T + u_i), \tag{15}$$

$$v_{i+1} := A\omega_i t_i^T + v_i - \sum_{k=0}^{i}\omega_k(\xi_k^T A\omega_k)^{-1}\xi_k^T A(A\omega_i t_i^T + v_i). \tag{16}$$

We also want to simply the iterations like what we did in standard biorthogonal scalar Lanczos algorithm. In every iteration, we pick out as many columns as possible from the previous iteration results u_i, v_i then project them into the Krylov basis [5].

We also have: when $0\le k<i<m$,

$$\begin{aligned}
\xi_k^T A^2\omega_i &= s_k^T s_k \xi_k^T A^2\omega_i \\
&= s_k^T(A^T\xi_k s_k^T)^T A\omega_i \\
&= s_k^T(u_{k+1} - u_k + \sum_{j=0}^{k}\xi_j(\omega_j^T A\xi_j)^{-1}\omega_j^T A^T(A^T\xi_k s_k^T + u_k))^T A\omega_i \\
&= s_k^T u_{k+1}^T A\omega_i - s_k^T u_k^T A\omega_i \\
&= s_k^T u_{k+1}^T A\omega_i.
\end{aligned} \tag{17}$$

So $\xi_k^T A^2\omega_i$ can be simplify to $s_k^T u_{k+1}^T A\omega_i$, and similarly, we can simplify $\omega_k^T(A^T)^2\xi_i$ *to* $t_k^T v_{k+1}^T A^T\xi_i$. This tells us that for some $j<i$, if all the columns

of u_{k+1} have been chosen and projected into ξ_{k+1},...ξ_j, this term will be vanished which means $u_{k+1}^T A\omega_i = 0$.

5 Implementation Details

As we mentioned before, the most time consuming part in GNFS is the sieving part. This part has already been parallelized in our previous work [19,20]. Another part in GNFS program can be improved is to solve the large and sparse linear systems over GF(2) by biorthogonal block Lanczos algorithm, instead of using the Montgomery's block Lanczos algorithm which only takes symmetric input. With the new algorithm, we would not lose any solutions. Our parallel code is built on the sequential source GNFS code from Monico [11].

5.1 Parallel Sieving

The sieving step in sequential GNFS is very suitable for parallel. The purpose of sieving is to find enough *(a,b)* pairs. The way we do sieving is: fixing b, let a change from -N to N (N is a integer, usually larger than *500*), then we check each *(a,b)* pair whether it smooth over factor bases. The sieving for next b can not start until the previous sieving is finished. After we got enough relations from the sieving step, we start building a linear system over GF(2). This linear system's coefficient could be either symmetric or nonsymmetric, both can be solved by the biorthogonal block Lanczos algorithm.

In parallel, we use several processors do sieving simultaneously, each slave node takes a small range of b, then send results back to master node. The detailed parallel sieving implementation can be found in [19,20].

5.2 Hardware and Programming Environment

The whole implementation uses two software packages, the sequential GNFS code from C. Monico [11] (Written in ANSI C) and sequential biorthogonal block Lanczos code from Hovinen [5] (Written in C++). For parallel implementation, MPICH1 (Message Passing Interface) [17] library is used. In order to do arbitrary precision arithmetic, the GMP 4.x is also used [4]. We use GUN compiler to compile whole program and MPICH1 [13] for our MPI library. The version of MPICH1 is 1.2.5.2. 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 node: 3 GB registered DDR-266 ECC SDRAM.
- Slave node: 2 to 3 GB registered DDR-266 ECC SDRAM.

In the program, Each slave node only communicates with the master node. Fig. 1 shows the flow chart of our parallel program.

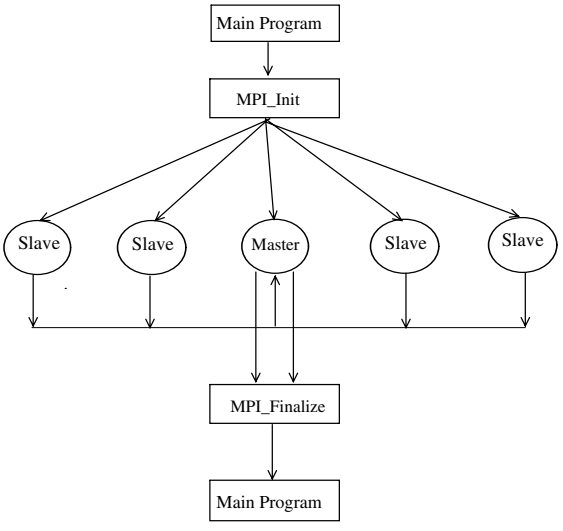

Fig. 1. Each processors do the sieving at the same time, and all the slave nodes send the result back to master node

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 of processors we use for each test case. Fig. 2 and Fig. 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 Fig. 4. Fig. 5 gives the total execution time, sieve time, speed-up and parallel efficiency with different processor numbers for test case tst250. Fig. 6 gives the speed-up and parallel efficiency for each test case with different processor numbers.

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

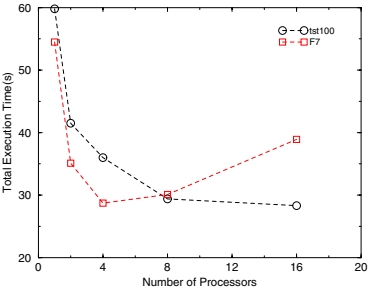

Fig. 2. Execution time for tst100 and F7

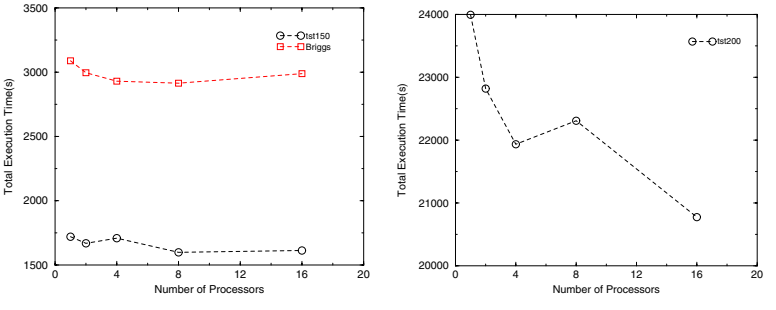

Fig. 3. Execution time for tst150, Briggs and tst200

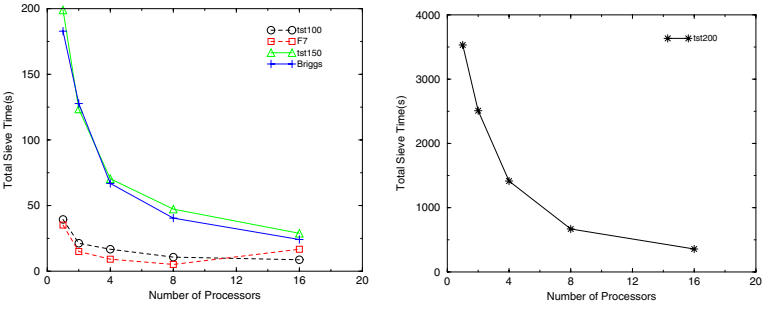

Fig. 4. Sieve time for tst100, F7, tst150, Briggs and tst200

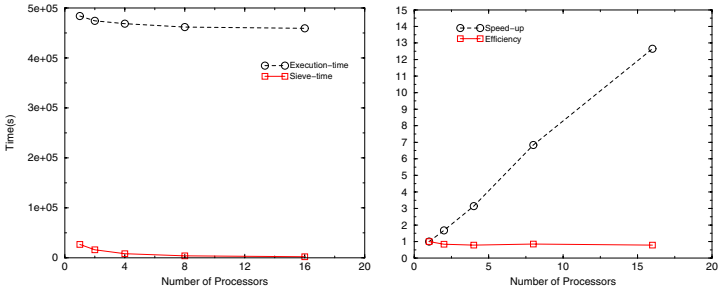

Fig. 5. Total execution time, sieve time, speedup and efficiency for test case tst250

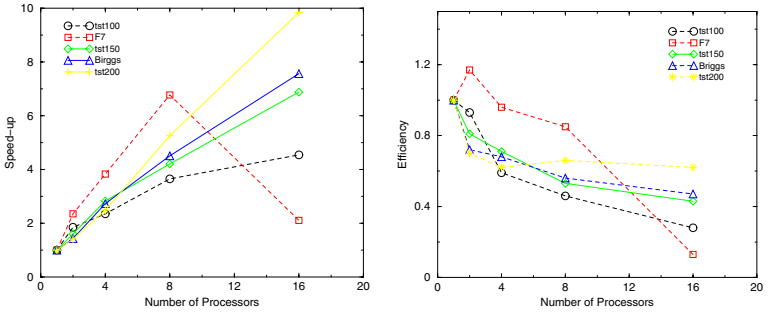

Fig. 6. Speedups and parallel efficiency

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. B. Hovinen. Blocked lanczos-style algorithms over small finite fields. Master Thesis of Mathematics, University of Waterloo, Canada, 2004.
6. 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.

7. 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.
8. A. K. Lenstra. Integer factoring. *Designs, Codes and Cryptography*, 19(2-3):101–128, 2000.
9. H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics(2)*, 126:649–673, 1987.
10. 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.
11. C. Monico. General number field sieve documentation. GGNFS Documentation, Nov 2004.
12. 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.
13. MPICH. `http://www-unix.mcs.anl.gov/mpi/mpich/`.
14. J. M. Pollard. Theorems on factorization and primality testing. In *Proceedings of the Cambridge Philosophical Society*, pages 521–528, 1974.
15. 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.
16. 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.
17. W.Gropp, E.Lusk, and A.Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, 1994.
18. 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.
19. 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.
20. 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.