Cofactorisation strategies for the number field sieve and an estimate for the sieving step for factoring 1024-bit integers

Thorsten Kleinjung

University of Bonn, Department of Mathematics,
Beringstraße 1, D-53115 Bonn, Germany
thor@math.uni-bonn.de

Abstract

The relation collection step for the number field sieve (NFS) is usually done by a combination of sieving techniques and methods for factoring the cofactors (cofactorisation). In this article a simplified model for the cofactorisation step is analysed and a method for finding good cofactorisation strategies is developed. This can be used to optimise hardware and software implementations. As an application we give an upper bound for the cost of the sieving step for a 1024-bit integer; using 12 million standard PCs it can be done within a year.

1 Introduction

The number field sieve (see [LL], [Coh]; the latter also describes all other factoring algorithms mentioned henceforth) is asymptotically the best known algorithm to factor numbers with large factors. In particular, it seems to be the best algorithm for factoring numbers which appear in RSA based cryptographic protocols. It has two expensive parts: the relation collection part and the matrix step.

In the relation collection part one searches among a lot of integers for those which completely split into small primes. Most hardware and software architectures do this in two steps (e. g., see [S05]). In the first step the smallest primes are identified and in the second step more sophisticated methods are used for the composites, which are not too big, to check whether they split into small primes. We call the first step sieving and the second step cofactorisation. This paper analyses the cofactorisation step (for another analysis see [Pom]) and describes a procedure to improve its efficiency. Usually this leads to a larger amount of relations such that less sieving needs to be done. Therefore it also improves implementations where the cofactorisation step is negligible.

This procedure has been used in the factorisation of RSA640 and in an earlier form also for factoring RSA200 ([RSA]). In the last section we apply it to estimate the cost of the sieving step for a 1024-bit integer and give details of the sieving experiments.

2 The sieving step

In NFS we are given two homogeneous polynomials $F_i \in \mathbb{Z}[X, Y]$, $i = 1, 2$, satisfying certain conditions. The task of the sieving step is to collect sufficiently many coprime pairs of integers (a, b), $b > 0$, such that both integers $F_i(a, b)$ are L_i-smooth for a given bound L_i, i. e., they decompose into prime factors $\leq L_i$. Such pairs (a, b) are called relations. The number of relations needed depends on the bounds L_i, collecting $\pi(L_1) + \pi(L_2)$ relations is usually more than necessary ($\pi(x)$ being the number of primes up to x).

The collection of relations is usually done by a combination of a sieving technique and a method for factoring smaller numbers, e. g., ECM or MPQS (see [Coh]). For this purpose we choose two factor bases $\mathcal{F}_i$ each consisting of pairs (p, r), where $p \leq B_i$ is a prime and r an integer such that p divides $F_i(a, b)$ whenever $p \mid a - br$. We write $F_i(a, b) = S_i^{(a,b)} R_i^{(a,b)}$, where $S_i^{(a,b)}$ is B_i-smooth and $R_i^{(a,b)}$ has no prime divisor $\leq B_i$. For this paper we divide the relation collection phase into the following steps:

Sieve Approximations of $\log R_i^{(a,b)}$ are calculated using a sieve.

Evaluation The pairs (a, b) for which both approximations of $\log R_i^{(a,b)}$, $i = 1, 2$, are below given bounds are identified. These are called candidates.

TD For each candidate (a, b) the prime divisors of $S_i^{(a,b)}$ are identified, e. g., by trial division, and the values $R_i^{(a,b)}$ are calculated. Optionally, a candidate is removed from the candidate list if the values $R_i^{(a,b)}$ are too big.

PSP If $R_i^{(a,b)} > L_i$ a fast compositeness test is performed, removing the candidate if $R_i^{(a,b)} > L_i$ is pseudo-prime.

Cofactorisation For each $R_i^{(a,b)} > L_i$ a factorisation attempt is done. If both values $R_i^{(a,b)}$ are known to be L_i-smooth, respectively, (a, b) is output as a relation.

Notice that in practice the steps are more complex, e. g., the first step is usually broken into several steps. However, we will mainly be concerned with the last step which is unaffected by this simplification.

For the following it is not necessary that the relation collection phase works as described above. We only need a black box which produces candidates, respectively pairs $(R_1^{(a,b)}, R_2^{(a,b)})$, and we assume that the bounds B_i are not too small.

3 Cofactorisation

In this section we discuss the cofactorisation step, i. e., we ask how much and which effort one should put in the factorisation attempts of $R_i^{(a,b)}$ for a candidate (a, b).

If we consider only one candidate we drop the superscript $^{(a,b)}$ and simply write R_i.

Example

As before let B_i be the factor base bounds and L_i the large prime bounds. Suppose we have a candidate with non-prime cofactors satisfying $L_i < R_i$, $i = 1, 2$. Furthermore we restrict the set of factorisation methods to MPQS (or a similar method like SQUFOF) which has the property that the time it takes depends roughly on the size of the input number and which has a negligible probability of not finding a factorisation. This restriction implies that we only have two strategies: first applying MPQS to R_1 and, if R_1 is smooth, applying MPQS to R_2 or vice versa.

As a special case let us consider the situation where $R_1 \leq B_1 L_1$ and $B_2 L_2 < R_2$ holds. Then R_1 is L_1-smooth whereas R_2 might not be L_2-smooth. So the better strategy is to begin with factoring R_2 since we save the cost for factoring R_1 if R_2 is not L_2-smooth.

In general let p_i be the probability that a number of size R_i is L_i-smooth and c_i the cost for factoring a number of size R_i. Of course, for both strategies the probability that we get a relation is $p_1 p_2$. If we first apply MPQS to R_1 the cost is on average $c_1 + p_1 c_2$ whereas first applying MPQS to R_2 gives a cost of $c_2 + p_2 c_1$. Using approximations for c_i and p_i we get a good guess on which strategy is more efficient on average.

In the following we also consider factorisation methods like ECM for which the probability of success depends on deeper information about the input number like the probability that the input number has a prime divisor of a certain size. Moreover these factorisation methods can change the deeper information, e. g., many unsuccessful ECM attempts on a number decrease the probability that it has a small prime divisor.

Our simplified model

Factoring a composite number n is a recursive process which splits it in $n = n_1 n_2$, $n_i > 1$, and, if n_1 or n_2 is not prime, another factorisation attempt is made on the smaller composite(s). In our model we assume that most of the work is spent in the first splitting and that the time for primality tests and for subsequent splittings is negligible. Notice that we only want to check for L-smoothness such that we do not need to split numbers $n_i \leq L$ nor test their primality. Moreover we can stop the subsequent splittings if we encounter a prime divisor $> L$ of n. Furthermore in our situation n has no prime divisors $\leq B$, the factor base bound, such that a splitting considerably reduces the size of the number to be factored. However, there exist situations in which the time spent for subsequent splittings is significant. For these situations a more refined model has to be used.

We call an algorithm which has as only input a number n and outputs a divisor of n, possibly trivial, a factorisation method. For example, ECM with given bounds B_1 and B_2 for its two stages is a factorisation method and ECM with different bounds B_1' and B_2' is a different factorisation method.

A strategy is a finite sequence of pairs (FM, s) where FM is a factorisation method and $s \in \{1, 2\}$. A strategy (FM_j, s_j) can be applied to (R_1, R_2). Applying a strategy (FM_j, s_j) to (R_1, R_2) means that for $j = 1, 2, \ldots$ the factorisation method FM_j is applied to R_{s_j} respecting the following rules:

- FM_j is not applied to R_{s_j} if R_{s_j} has already been split.
- If a (pseudo-)prime factor $> L_i$ of R_i, $i = 1$ or $i = 2$, has been found the process is aborted.
- If R_i, $i = 1$ or $i = 2$, has not been split yet and $s_k \neq i$ holds for all $k \geq j$ the process is aborted.

In other words, we apply the factorisation methods successively to their targets avoiding unnecessary work. We call the application of the strategy to (R_1, R_2) successful, if R_i is L_i-smooth, $i = 1, 2$, and this is detected by the strategy, i. e., R_i is either $\leq L_i$ or split by one factorisation method of the strategy, $i = 1, 2$.

We also allow a strategy of length 0 meaning that no factorisation attempt is done. This strategy is by definition never successful, even if R_1 and R_2 are L_1-smooth and L_2-smooth, respectively.

Finding good strategies

In the cofactorisation step we want to quickly find a good factorisation strategy for a pair (R_1, R_2), as before R_i has no prime divisor $\leq B_i$. Denoting by r_i the bit length of R_i we will select a strategy depending on (r_1, r_2). It is possible to choose a finer or coarser graining leading to a larger or smaller effort for precomputations.

Let $\mathcal{FM}$ be a set of factorisation methods and $\mathcal{S}$ a finite set of strategies built from elements of $\mathcal{FM}$ and containing the strategy of length 0. For a strategy $S \in \mathcal{S}$ and a pair of bit lengths (r_1, r_2) we denote by $p(S; r_1, r_2)$ the probability that S is applied successfully to a pair (R_1, R_2) of bit lengths (r_1, r_2) and by $c(S; r_1, r_2)$ the average cost of such an application of S (see below for the calculation of these quantities).

Let C_0 be the cost for sieving, evaluation, trial division and compositeness tests, producing a set $\mathcal{C}$ of candidates. We assume that $\mathcal{C}$ is representative with respect to the distribution of the sizes of $(r_1^{(a,b)}, r_2^{(a,b)})$, the bit lengths of $(R_1^{(a,b)}, R_2^{(a,b)})$, for $(a, b) \in \mathcal{C}$. I. e., we assume that sufficiently many candidates with the same distribution can be produced at the same rate. In practice, if we use lattice sieving ([LL]), we generate $\mathcal{C}$ by sieving over some special q, uniformly distributed in the anticipated range of special q.

If we choose strategy S_{r_1,r_2} for cofactors of bit length (r_1, r_2) the average number of relations will be

$$Y = \sum_{(a,b)\in\mathcal{C}} p(S_{r_1^{(a,b)},r_2^{(a,b)}}; r_1^{(a,b)}, r_2^{(a,b)})$$

generated in average time

$$T = C_0 + \sum_{(a,b)\in\mathcal{C}} c(S_{r_1^{(a,b)},r_2^{(a,b)}}; r_1^{(a,b)}, r_2^{(a,b)}).$$

Y and T depend on the set of candidates $\mathcal{C}$ and on the collection $\{S_{r_1,r_2}\}$ of strategies which we suppressed in the notation.

Our goal is to choose a collection $\{S_{r_1,r_2}\}$ such that $\frac{Y}{T}$ is maximised for a given set of candidates $\mathcal{C}$. Since r_1 and r_2 are bounded and $\mathcal{S}$ is finite this is a finite calculation which can be considerably simplified by the following observations.

Observation 1 *Let the notations be as above. There exists a maximum s of $\frac{Y}{T}$ which is obtained by choosing for each (r_1, r_2) a strategy $S^{opt}_{r_1,r_2}$ such that the value $p(S; r_1, r_2) - s \cdot c(S; r_1, r_2)$ attains a maximum for $S = S^{opt}_{r_1,r_2}$.*

Because of the finiteness of $\mathcal{S}$ and the boundedness of r_1 and r_2 there exists a maximum s for $\frac{Y}{T}$. For a collection $\{S_{r_1,r_2}\}$ we have $s \geq \frac{Y}{T}$ or

$$sC_0 \geq \sum_{(a,b)\in\mathcal{C}} \left(p(S_{r_1^{(a,b)},r_2^{(a,b)}}; r_1^{(a,b)}, r_2^{(a,b)}) - s \cdot c(S_{r_1^{(a,b)},r_2^{(a,b)}}; r_1^{(a,b)}, r_2^{(a,b)}) \right).$$

Equality holds if each summand on the right hand side is maximal, resulting in the condition of the observation.

Consider the following set

$$P_{r_1,r_2} = \left\{ \begin{pmatrix} c(S; r_1, r_2) \\ p(S; r_1, r_2) \end{pmatrix} \mid S \in \mathcal{S} \right\}$$

of points in the plane. For finding the maximum of $p(S; r_1, r_2) - s \cdot c(S; r_1, r_2)$ we have the following obvious observation:

Observation 2 *Denote the vertices of the ascending part of the boundary of the convex hull of P_{r_1,r_2} by $h^{(0)}_{r_1,r_2} = \binom{0}{0}, h^{(1)}_{r_1,r_2}, \ldots, h^{(h)}_{r_1,r_2}$ (in clockwise order) and let $s^{(i)}_{r_1,r_2}$ be the slope of the edge connecting $h^{(i-1)}_{r_1,r_2}$ and $h^{(i)}_{r_1,r_2}$. Set $s^{(0)}_{r_1,r_2} = \infty$ and $s^{(h+1)}_{r_1,r_2} = 0$.*

The maximum of $y - sx$, $\binom{x}{y} \in P_{r_1,r_2}$ is always attained at a point of the convex hull of P_{r_1,r_2}. If no slope equals s the maximum is attained at vertex $h^{(i)}_{r_1,r_2}$ such that $s^{(i)}_{r_1,r_2} > s > s^{(i+1)}_{r_1,r_2}$, otherwise it is attained at every point of P_{r_1,r_2} lying on the edge with slope s.

After having calculated all values $c(S; r_1, r_2)$ and $p(S; r_1, r_2)$ (see below) we compute for each (r_1, r_2) the convex hull and sort all slopes. This is independent of the set of candidates $\mathcal{C}$. Having generated a representative set $\mathcal{C}$ we can find the optimal s by a binary search. Indeed, given an s we find $S_{(r_1,r_2)}$ maximising $p(S; r_1, r_2) - s \cdot c(S; r_1, r_2)$ by observation 2. Then we can calculate Y and T, and compare $\frac{Y}{T}$ with s. Since a lot of simplifications have been made, the optimal s in practice will usually differ from the theoretically optimal s. So it is better to choose a few reasonable values for s and measure $\frac{Y}{T}$ directly.

We now show how to compute $c(S; r_1, r_2)$ and $p(S; r_1, r_2)$.

Local calculations

We restrict ourselves to the discussion of three factorisation methods, namely the $p-1$-method, ECM and MPQS. Here MPQS stands synonymously for SQUFOF, the continued fractions method or GNFS whose run time also only depends on the size of the input number. These methods heuristically always yield a splitting of the input number. In contrast, the $p-1$-method and ECM with fixed parameters B_1 and B_2 find a prime divisor p of the input number with a certain probability depending on the size of p. The time these methods need depends on the size of the

input number. A difference between the $p-1$-method and ECM is that applying the $p-1$-method with the same parameters to the same number always gives the same result, whereas several ECM runs have a higher probability of finding a factor than one ECM run. For this reason we only consider strategies in which the $p-1$-method occurs at most once per side. This is a bit more restrictive than necessary. We could also allow several applications of the $p-1$-method per side which would slightly increase the complexity of the analysis below.

In the following we will consider the probability that a number has prime divisors of certain bit lengths. For this we will count numbers of a certain size having a certain decomposition type. For $m, n, n_i \in \mathbb{Z}_{\geq 0}$ let

$$\Psi^{(n;n_1,\ldots,n_m)} = \left\{x = q_1 \cdot \ldots \cdot q_m \in \mathbb{Z}_{>0} \,|\, q_1 \leq \ldots \leq q_m, \text{prime}, q_i \in [2^{n_i-1}, 2^{n_i}[, x \in [2^{n-1}, 2^n[\right\}$$

be the set of n-bit numbers which are products of m prime numbers such that the i-th smallest prime divisor has exactly n_i bit. For an arbitrary set $X \subset \mathbb{Z}_{\geq 0}$ we write $X^{(n;n_1,\ldots,n_m)} = X \cap \Psi^{(n;n_1,\ldots,n_m)}$ such that we get a disjoint decomposition

$$X = \dot{\bigcup_{m,n,n_i}} X^{(n;n_1,\ldots,n_m)}.$$

Let M be the set of composite natural numbers having no prime divisor $\leq B$. The cardinality of $M^{(n;n_1,\ldots,n_m)}$ can be approximated using the prime number theorem and some integration. Since the bound B usually is big enough, the approximation is quite good.

As a next step we gather data about the factorisation methods. For MPQS we measure the average run time for factoring n-bit integers, n ranging over the possible bit lengths of the R_i. For the $p-1$-method and ECM with fixed parameters we do the same and additionally we compute the probability that a k-bit prime divisor will be discovered. It is sufficient to do this for all k up to the bit length of the largest R_i, but it is also possible to stop if the probability is very near to 0 and set the probability for larger k to 0.

We now turn to the computation of $c(S; r_1, r_2)$ and $p(S; r_1, r_2)$. Let M_i initially be the set of composite natural numbers having no prime divisor $\leq B_i$. We denote the cardinality of $M_i^{(n;n_1,\ldots,n_m)}$ by $m_i^{(n;n_1,\ldots,n_m)}$ and compute it for n up to the maximal bit lengths of the R_i. Notice that this is a finite computation since for each n there are only finitely many m-tuples $(n_1, \ldots, n_m)$ such that $\Psi^{(n;n_1,\ldots,n_m)}$ is non empty.

Then we successively consider the factorisation methods of $S = (FM_j, s_j)$, virtually applying them to (M_1, M_2) but only keeping track of the cardinalities $m_i^{(n;n_1,\ldots,n_m)}$. If FM_j is the $p-1$-method or ECM it will change M_{s_j} by removing those $x \in M_{s_j}$ such that FM_j splits x. Denoting by $\text{prob}(FM_j, n_k)$ the probability that FM_j discovers a prime divisor of n_k-bit, the cardinality $m_i^{(n;n_1,\ldots,n_m)}$ will change to

$$m_i^{(n;n_1,\ldots,n_m)} \cdot \prod_k (1 - \text{prob}(FM_j, n_k)).$$

Analogously we compute how many smooth numbers are detected by FM_j. For MPQS we compute the number of smooth numbers and set M_{s_j} to $\emptyset$. Putting together all these information we obtain approximations for $c(S; r_1, r_2)$ and $p(S; r_1, r_2)$.

Note that we made several assumptions which heuristically seem to be true, for example the assumption that the probabilities of several ECM attempts are independent. We also neglected some strange cases like ECM finding all factors simultaneously.

4 Practical considerations

Here we discuss implementation details and problems of the methods in the previous section.

Reducing TD work

Usually for a lot of bit lengths (r_1, r_2) no factorisation attempt is done. For example, we often have $L_1^2 < B_1^3$ and a number between L_1^2 and B_1^3 not divisible by primes $\leq B_1$ is not L_1-smooth. To avoid a lot of useless trial divisions we would like to discard such candidates as soon as possible.

If the sieving is very exact we can approximate $r_1^{(a,b)}$ and $r_2^{(a,b)}$ from the sieve approximations at (a, b). We then discard the candidate (a, b) if in a small neighbourhood of $(r_1^{(a,b)}, r_2^{(a,b)})$ no factorisation attempt is done. In this way we might loose a few relations but the gain in reducing the trial division work seems to outweigh the loss. Notice also that bit lengths (r_1, r_2) for which no factorisation attempt is done seem to be clustered such that the procedure above discards a good part of unwanted candidates.

Reducing precomputations

If we consider n factorisation methods and all strategies of length $\leq l$ built out of these, we have to evaluate $\frac{(2n)^{l+1}-1}{2n-1}$ strategies for each pair (r_1, r_2). Even for small n and l this might be too big.

We can reduce this by discarding nonsense strategies, for example strategies where MPQS is applied to one side and afterwards another factorisation method to the same side. But the number of strategies to be checked will still be huge.

Since we only want to compute the convex hull of P_{r_1,r_2} we only need to consider the (unknown) strategies corresponding to the edges in the convex hull. Therefore we will make a guess which strategies are likely to contribute to the convex hull and only evaluate these strategies. By doing this we might miss some good strategies but we can consider much longer strategies built from a larger set of factorisation methods.

There are several ways for guessing which strategies are good. We have implemented the following procedure. First consider all strategies of small length, ≤ 4 say, built from the n chosen factorisation methods. For those strategies which contribute to the convex hull consider also mutations of these strategies which arise by inserting a factorisation method (always discarding nonsense strategies) and iterate this several times.

Auxiliary factorisations

In the case $R_i \leq B_i^3$ the first splitting always gives a complete factorisation. For $B_i^3 < R_i \leq B_i^4$ the first splitting may give one prime factor and a composite which is $\leq \frac{R_i}{B_i}$. Usually factoring this composite is much cheaper than the first splitting of R_i. For larger R_i the situation is more difficult.

The number R_i might be a product of a small prime and two larger primes of almost the same size which are bigger than L_i. If the small prime factor is discovered by the $p-1$-method or ECM we have to factor the product of the two larger primes. In this case it is best to use MPQS which will use a lot of time, only to discover that R_i is not L_i-smooth. On the other hand R_i might be a product of several (≥ 4) small primes $\leq L_i$. If the first splitting discovers one of these we might use ECM on the cofactor which will find the other small prime factors faster than MPQS.

Therefore we have implemented the following procedure for the auxiliary factorisation. A composite smaller than 2^{96} is factored by MPQS. For a larger composite a few ECM attempts are done, stopping, if its size falls below 2^{96}. If the size of the remaining composite is smaller than 2^{128} MPQS is applied, otherwise we discard the composite. Since smooth composites contain many small prime factors the probability that ECM finds a factor and reduces the size of the composite is much higher than for general composites. So we expect that smooth composites are very rarely discarded by this procedure. Nevertheless, if very large R_i occur frequently the handling of auxiliary factorisations should be investigated further.

5 An estimate for the cost of the sieving step for RSA1024 using PCs

In this section we will describe an experiment to estimate the cost for the sieving step for the 1024-bit number RSA1024 using standard PCs. We have chosen the following polynomial pair (f_1, f_2):

$$f_1 = 1000000001002023904806000x^6 + 2696978952367681630566064163 40x^5 - 6212838818608524196100227896844747498x^4$$

$$- 8471052513942755376507570481852462668136x^3 + 7386089168513102555044082528893786797012311 1795x^2$$

$$+ 1032395042584592690889615837724142616376240650532 06x - 1139431985616391987769376205036438729670911 71901277555912$$

and

$$f_2 = 5146620559617247177525524125973348 61x - 2265119830146382627844763723199431809702055345 45.$$

Only a few hours have been spent for the polynomial selection and one should expect to find much better polynomial pairs.

We use the lattice siever of Jens Franke, which was also used for factoring RSA200 and RSA640 ([RSA]), with a sieving area of size $2^{16} \times 2^{15}$. The factor bases contain all prime ideals below $1.1 \cdot 10^9$ and $3 \cdot 10^8$ for polynomial f_1 and f_2, respectively. On both sides the large prime bounds are set to 2^{42}. Cofactorisation strategies were calculated for $r_1 \leq 192$ and $r_2 \leq 160$. As special q we choose most numbers between

$8 \cdot 10^{12}$ and $56 \cdot 10^{12}$ consisting of prime factors between 2^8 and 2^{32}. Approximately 18% of the special q have a highly skewed lattice and were discarded.

We did lattice sieving for special q in the intervals $[i \cdot 10^{12}, i \cdot 10^{12} + 200]$ for $i = 8, 9, \ldots, 56$. There are 580 special q of which 105 were discarded. The remaining 475 special q produced 295 relations, so on average 0.62 relations per special q leading to at least 0.168 unique relations per special q. Using $1.876 \cdot 10^{12}$ special q we expect to get $3.16 \cdot 10^{11} > 2\pi(2^{42}) \approx 3.14 \cdot 10^{11}$ unique relations.

In the following table details about the timing and number of candidates for an average special q are given. Since the siever does not consider the part of the sieving area where both coordinates are even, we begin with $3 \cdot 2^{29} = 1\,610\,612\,736$ (a, b)-pairs. In the first step, sieving and a very crude evaluation are done. This reduces the number of (a, b)-pairs by a factor of about 500. The next step removes non-coprime (a, b)-pairs and does more accurate size checks as described in the first part of the previous section, reducing the number of candidates by more than 90%. In the third step $(R_1^{(a,b)}, R_2^{(a,b)})$ are calculated and size checks and compositeness tests are done. About 70% of the candidates are discarded. The last step is the cofactorisation step.

Step	# input	#output	time per input	time per original input
Sieve and evaluation	1 610 612 736	3 515 840	120 cycles	120 cycles
gcd and size checks	3 515 840	315 592	630 cycles	1.4 cycles
TD and PSP	315 592	88 840	179 180 cycles	35 cycles
Cofactorisation	88 840	0.62	1 912 032 cycles	105 cycles

Timing and number of candidates for an average special q

On a 2.2 GHz Athlon 64 with 2 GB main memory one special q needs on average 194 seconds such that 12 million of these PCs need less than a year for the total sieving step. The sieving program needed about 1.9 GB of main memory.

The upper bound above may be improved in several ways. One might try different parameters for the special q and the sieving area to reduce the number of duplicates. About half of the time is spent for compositeness tests and in the cofactorisation step. Since the memory used in these steps is of the size of the L1-cache, a dualcore processor might give a speedup close to 2. Furthermore the size of the main memory might be changed to 1 or 3 GB.

References

[Coh] H. Cohen, *A Course in Computational Algebraic Number Theory*, GTM 138, Springer, 1993.

[LL] A. K. Lenstra and H. W. Lenstra, Jr. (eds.), *The Development of the Number Field Sieve*, Lecture Notes in Math. 1554, Springer, 1993.

[Pom] C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, in: Computational Methods in Number Theory, Part I (ed. by H. W. Lenstra, Jr. and R. Tijdeman), Math. Centre Tract 154, Amsterdam, 1982, 89 – 139.

[RSA] J. Franke et al., E-mail announcements, 2005.
`http://www.crypto-world.com/announcements/rsa200.txt`
`http://www.crypto-world.com/announcements/rsa640.txt`

[S05] SHARCS 2005, *Workshop on Special Purpose Hardware for Attacking Cryptographic Systems*, Paris, 2005.