Algorithms for Large Integer Matrix Problems

Mark Giesbrecht[1], Michael Jacobson, Jr.[2], and Arne Storjohann[1]

[1] Ontario Research Centre for Computer Algebra, University of Western Ontario, London, ON, N6A 5B7, Canada
{mwg,storjoha}@scl.csd.uwo.ca

[2] Dept. of Computer Science, University of Manitoba, Winnipeg, MB, R3T 2N2, Canada
jacobs@cs.umanitoba.ca

Abstract. New algorithms are described and analysed for solving various problems associated with a large integer matrix: computing the Hermite form, computing a kernel basis, and solving a system of linear diophantine equations. The algorithms are space-efficient and for certain types of input matrices — for example, those arising during the computation of class groups and regulators — are faster than previous methods. Experiments with a prototype implementation support the running time analyses.

1 Introduction

Let $A \in \mathbb{Z}^{n\times(n+k)}$ with full row-rank be given. The lattice $\mathcal{L}(A)$ is the set of all $\mathbb{Z}$-linear combinations of columns of A. This paper describes new algorithms for solving the following problems involving $\mathcal{L}(A)$: computing the Hermite basis, computing a kernel basis, and given an integer vector b, computing a diophantine solution x (if one exists) to the linear system $Ax = b$.

By Hermite basis of A we mean the unique lower-triangular matrix $H \in \mathbb{Z}^{n\times n}$ such that $\mathcal{L}(H) = \mathcal{L}(A)$ and each off-diagonal entry is nonnegative and strictly smaller than the positive diagonal entry in the same row. A kernel for A is an $N \in \mathbb{Z}^{(n+k)\times k}$ such that $\mathcal{L}(N) = \{v \in \mathbb{Z}^{n+k} \mid Av = 0\}$. The problem of computing H and N often occurs as a subproblem of a larger number-theoretic computation, and the input matrices arising in these applications often have some special properties. The algorithms we give here are designed to be especially efficient for an input matrix $A \in \mathbb{Z}^{n\times(n+k)}$ which satisfies the following properties:

- A is sparse. More precisely, let μ be the number of nonzero entries in A. Then $\mu = O(n^{1+\epsilon})$ for some $0 \le \epsilon < 1$.
- The dimension k of the kernel is small compared with n.
- Let l be the smallest index such that the principal $(n-l)\times(n-l)$ submatrix of the Hermite basis H of $\mathcal{L}(A)$ is the identity. Then l is small compared with n.

Sparse input-matrices which satisfy these conditions on k and l are typical in computations for computing class groups and regulators of quadratic fields

S. Boztaş and I.E. Shparlinski (Eds.): AAECC-14, LNCS 2227, pp. 297–307, 2001.

using the algorithm described in [4,7]. The diagonal elements of the Smith form of the matrix yield the elementary divisors of the class group (i.e., they give the class group as a product of cyclic groups), and the kernel (in the case of real quadratic fields) is used to compute the regulator. In practice, the number of diagonal elements of the Hermite basis which are not one is rarely larger than the rank of the class group. Since class groups are often cyclic or very close to being cyclic (as predicted by the Cohen-Lenstra heuristics [1]), l is small as well. Thus, the algorithms described in this paper are especially effective for these types of input.

Many algorithms have been proposed for computing the Hermite basis; for a survey we refer to [12]. The algorithm proposed in [12] — which is deterministic and computes a unimodular transformation-matrix, but does not exploit the sparsity of A or the fact that l may be small — requires about $O(n^4(\log||A||)^2)$ bit operations where $||A|| = \max_{ij}|A_{ij}|$. Moreover, that algorithm requires intermediate storage for about $O(n^3(\log||A||))$ bits. The algorithm we propose computes H in an expected number of about $O(\mu n^2(\log||A|)+n^3(\log||A||)^2(l^2+k\log||A||))$ bit operations. When A is sparse and k and l are small compared to n we essentially obtain an algorithm which requires about $O(n^3(\log||A||)^2)$ bit operations. Moreover, the algorithm requires intermediate space for only about $O(n^2\log||A||)$ bits, for both sparse and dense input matrices. However, in practice, when A is sparse the storage requirements are reduced by a factor of two.

Table 1. Running times: A constant size entries and $k < n$.

Section	Word operations	Type
§3 Permutation conditioning	$O(n^3)$	LV
§4 Leading minor computation	$O(\mu n^2(\log n))$	LV
§5 Lattice conditioning	$O(kn^3(\log n)^3)$	DET
§6 Kernel basis computation	$O(k^2n^3(\log n)^2)$	DET
§7 Hermite basis computation	$O(kn^3(\log n)^3+l^2n^3(\log n)^2)$	DET
§8 System solving	$O(n^3(\log n)^2)$	DET

For the analyses of our algorithms we assume we are working on a binary computer which has words of length ω, and if we are working with an input matrix $A \in \mathbb{Z}^{n+(n+k)}$, that ω satisfies

$$\omega > \max\left(6+\log\log\left((\sqrt{n}||A||)^n\right), 1+\log(2(n^2+n))\right). \qquad (1)$$

Primes in the range $2^{\omega-1}$ and 2^ω are called *wordsize* primes. We assume that a wordsize prime can be chosen uniformly and randomly at unit cost. Complexity results will be given in terms of word operations. For a more thorough discussion of this model see the text [13].

The computation is divided into a number of phases. The first three phases (described in Sections 3, 4 and 5) can be viewed as precomputation. Once these are complete, computing a kernel and Hermite basis, as well as solving diophantine systems involving A, can be accomplished deterministically in the running times indicated in Table 1.

The first phase – permutation conditioning – is to find a wordsize prime p for which A has full row-rank modulo p and permute the columns via a permutation matrix P such that the principal $n \times n$ submatrix B_1 has generic rank-profile: $B = AP = \left[B_1 \middle| B_2 \right]$. The inverse modulo p of B_1 is also computed during this phase.

The second phase – leading minor computation – is to compute the determinant d of B_1. This is the only phase where we exploit the possible sparseness of A to get a better asymptotic running-time bound. In practice, we use Wiedemann's algorithm modulo a collection of distinct primes; this is easy to parallelize.

The third phase – lattice conditioning – is to compute a $Q \in \mathbb{Z}^{k \times n}$ which is used to compress the information from the columns of B_2 with B_1 to obtain a single $n \times n$ matrix $B_1 + B_2 Q$ from which the Hermite basis of B can be recovered.

2 Preliminaries

We recall the notion of a recursive and iterated inverse. Let R be a commutative ring with identity.

Recursive Inverse

Suppose that $A \in \mathsf{R}^{n \times n}$ enjoys the special property that each principal minor is invertible over R. The recursive inverse is a data structure that requires space for only n^2 ring elements but gives us the inverse of all principal minors of A. By "gives us" the inverse we mean that we can compute a given inverse×vector or vector×inverse product in quadratic time — just as if we had the inverse explicitly.

For $i = 1, \ldots, n$ let A_i denote the principal $i \times i$ submatrix of A. Let d_i be the i-th diagonal entry of A. For $i = 2, \ldots, n$ let $u_i \in \mathsf{R}^{1 \times (i-1)}$ and $v_i \in \mathsf{R}^{(i-1) \times 1}$ be the submatrices of A comprised of the first $i-1$ entries in row i and column i, respectively. In other words, for $i > 1$ we have

$$A_i = \left[\begin{array}{c|c} A_{i-1} & v_i \\ \hline u_i & d_i \end{array} \right].$$

The *recursive inverse* of A is the expansion

$$A^{-1} = V_n D_n U_n \cdots V_2 D_2 U_2 V_1 D_1 U_0 D_0, \tag{2}$$

where V_i, D_i and U_i are $n \times n$ matrix defined as follows. For $i = 1, 2, \ldots, n$ let $B_i = \operatorname{diag}(A_i^{-1}, I_{n-i}) \in \mathsf{R}^{n \times n}$. Then

$$B_1 = \overbrace{\left[\begin{array}{c|c} d_1^{-1} & \\ \hline & I_{n-1} \end{array} \right]}^{D_0},$$

and for $i > 1$ we have

$$B_i = \overbrace{\left[\begin{array}{c|c|c} I_{i-1} & -B_{i-1}v_i & \\ \hline & 1 & \\ \hline & & I_{n-i} \end{array}\right]}^{V_i} \overbrace{\left[\begin{array}{c|c|c} I_{i-1} & & \\ \hline & (d_i - v_i u_i)^{-1} & \\ \hline & & I_{n-i} \end{array}\right]}^{D_i} \overbrace{\left[\begin{array}{c|c|c} I_{i-1} & & \\ \hline -u_i & 1 & \\ \hline & & I_{n-r} \end{array}\right]}^{U_i} B_{i-1}.$$

The expression (2) for A^{-1} as the product of structured matrices has some practical advantages in addition to giving us the inverse of all principal submatrices. Suppose that A is sparse, with $O(n^{1+\epsilon})$ entries for some $0 \leq \epsilon < 1$. Then the V_i will also be sparse and $A^{-1}v$ or $v^T A^{-1}$ for a given $v \in \mathsf{R}^{n\times 1}$ can be computed in $n^2/2 + O(n^{1+\epsilon})$ ring operations.

Iterated Inverse

Now, let $U \in \mathsf{R}^{n\times k}$ and $V \in \mathsf{R}^{k\times n}$ be given in addition to A. Suppose the perturbed matrix $A + UV$ is invertible. The iterated inverse is a data structure that gives us $(A + UV)^{-1}$ but requires only $O(n^2 k)$ ring operations to compute if we already have the inverse of A.

For $i = 0, 1, 2, \ldots, k$ let U_i and V_i be the submatrices of U and V comprised of the principal i columns and rows, respectively. Let u_i and v_i be the i-th column and row of U and V, respectively. Note that $u_i v_i$ is an $n \times n$ matrix over R while while $v_i u_i$ is a 1×1 matrix over R. For $i = 0, 1, \ldots, n$ suppose that $(A + U_i V_i)$ is invertible, and let $B_i = (A + U_i V_i)^{-1}$. Then $B_0 = A^{-1}$ and for $i > 0$ we have

$$B_i = (I + \bar{u} v_i) B_{i-1} \quad \text{where} \quad \bar{u}_i = -1/(1 + v_i B_{i-1} u_i) B_{i-1} u_i \in \mathsf{R}^{n\times 1}.$$

The vector $\bar{u}_i$ can be computed using B_{i-1} in $O(n^2 + ni)$ ring operations. Thus, if we start with B_0, we can compute the *iterated inverse* expansion

$$(A + UV)^{-1} = (I + \bar{u}_k v_k) \cdots (I + \bar{u}_2 v_2)(I + \bar{u}_1 v_1) A^{-1}$$

in $O(n^2 k + nk^2)$ ring operations. Using the iterated inverse, we can compute $(A + UV)^{-1} u$ or $u^T (A + UV)^{-1}$ for a given $u \in \mathsf{R}^{n\times 1}$ using $O(n^2 + nk)$ ring operations. Note that for our applications k is typically much smaller than n.

3 Permutation Conditioning

Let $A \in \mathbb{Z}^{n\times(n+k)}$ be given. Choose random wordsize primes in succession until a prime p is found for which A has full rank modulo p. The rank check is performed using gaussian elimination. The lower bound (1) on ω (the word length on the computer) ensures such a prime will be found in an expected constant number of iterations. Once a good prime is found, we can also compute a $(n + k) \times (n + k)$ permutation matrix P such that each principal submatrix of AP is nonsingular modulo p. Let $B = AP$. Let C be the modulo p recursive inverse of the principal $n \times n$ submatrix of B. We call the tuple (B, P, C, p) a *permutation conditioning of* A. Producing a permutation conditioning requires an expected number of $O(n^3 + n^2(\log ||A||))$ word operations.

4 Computation of Leading Minor

Let (B, P, C, p) be a permutation conditioning of $A \in \mathbb{Z}^{n+(n+k)}$. Let B_1 be the principal $n \times n$ submatrix of B. Let μ be a bound on the number of nonzero entries in B_1 and let $d = \det B_1$. For a wordsize prime p, the image $d \bmod p$ can be computed in an expected number of $O(\mu(n + (\log ||A||)))$ word operations using the method of Wiedemann [14]. Hadamard's bound gives $|d| \le (\sqrt{n}||A||)^n$, so if we have images for at least $\lceil n(\log_2 \sqrt{n}||A||)/(\omega - 1)\rceil + 1 = O(n(\log n + \log ||A||))$ distinct primes we can compute d using Chinese remaindering. We obtain the following.

Proposition 1. *The principal $n \times n$ minor of B can be computed using an expected number of $O(\mu n^2(\log n + \log ||A||) + \mu n(\log ||A||)^2)$ word operations.*

Now assume we have computed $d = \det B_1$. Let $v \in \mathbb{Z}^{n\times 1}$ be the n-th column of I_n. Then the last entry of $B_1^{-1}dv$ will be the determinant of the principal $(n-1)\times(n-1)$ submatrix of B_1. The vector $B_1^{-1}dv$ is computed in $O(n^3(\log n + \log ||A||)^2)$ word operations using p-adic lifting as described in [2]. Because we have the recursive inverse of B_1, we get the following:

Proposition 2. *Let a permutation conditioning (B, P, C, p) together with the principal $t \times t$ minor of B be given, $t > 1$. Then the determinant of the principal $(t-1) \times (t-1)$ minor of B can be computed in $O(n^3(\log n + \log ||A||)^2)$ word operations.*

5 Lattice Conditioning

Let a permutation conditioning (B, P, C, p) of $A \in \mathbb{Z}^{n\times(n+k)}$ be given. Write $B = \left[\, B_1 | B_2 \,\right]$ where B_1 is $n \times n$. Assume $d = \det B_1$ is also given. Recall that $\det \mathcal{L}(B)$ is the product of diagonal entries in the Hermite basis of B.

Definition 1. *A* lattice conditioning *of B is a tuple (Q, W, c) such that:*

- $Q \in \mathbb{Z}^{k\times n}$,
- $\gcd(c, pd^2) = \det \mathcal{L}(B)$ *where* $c = \det(B_1 + B_2Q)$,
- *W is the modulo p iterated inverse of $B_1 + B_2Q$.*

The purpose of a lattice conditioning is to compress the information from the extra columns B_2 into the principal n columns. Note that

$$\left[\, B_1 | B_2 \,\right] \left[\begin{array}{c|c} I_n & \\ \hline Q & I_k \end{array}\right] = \left[\, B_1 + B_2Q | B_2 \,\right]$$

where the transforming matrix is unimodular. The condition $\gcd(c, pd^2) = \det \mathcal{L}(B)$ on c means that we can neglect the columns B_2 when computing the Hermite basis of B. Note that the condition $\gcd(c, d^2) = \det \mathcal{L}(B)$ would also suffice, but using the modulus pd^2 ensures that $B_1 + B_2Q$ is nonsingular modulo p.

We have the following result, which follows from the theory of modulo d computation of the Hermite form described in [3], see also [12, Proposition 5.14]. Let (Q, W, c) be a lattice conditioning of B. Then

Lemma 1. $\mathcal{L}\left(\left[\, B_1 + B_2 Q \middle| d^2 I \,\right]\right) = \mathcal{L}(B)$.

The algorithm to compute a lattice conditioning is easiest to describe recursively. Let $\bar{B}$ and $\bar{B}_2$ be the matrices B and B_2, respectively, but with the last column removed. Assume we have recursively computed a lattice conditioning $(\bar{Q}, \bar{W}, \bar{c})$ for $\bar{B}$. Let u be the last column of B. We need to compute a $v \in \mathbb{Z}^{1\times n}$ such that $\gcd(c, pd^2)$ is minimized, where $c = \det(B_1 + \bar{B}_2\bar{Q} + uv)$. Using the iterated inverse $\bar{W}$, compute $\bar{u} = (B_1 + \bar{B}_1\bar{Q})^{-1}\bar{c}u$ using linear p-adic lifting. This costs $O(n^3(\log n + \log ||A||)^2)$ word operations. It is easy to derive from elementary linear algebra that $c = \bar{c} + v\bar{u}$. We arrive at the problem of computing v such that

$$\gcd(\bar{c} + v_1\bar{u}_1 + v_2\bar{u}_2 + \cdots v_n\bar{u}_n, d^2) = \gcd(\bar{c}, \bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n, d^2). \tag{3}$$

This problem, the "modulo N extended gcd problem" with $N = pd^2$, is studied in [11]. From [6] we know that there exists a v with entries bounded in magnitude by $O((\log d)^2)$. We may assume (by induction) the same bound for entries in $\bar{Q}$. Then $||B_1 + \bar{B}_2\bar{Q}|| = O(n(\log d)^2(\log ||A||))$ and Hadamard's bound gives that $\max(d, \bar{c}, ||\bar{u}||) = O(n(\log n + \log ||A||))$.

Lemma 2. *A solution $v \in \mathbb{Z}^{1\times n}$ to the modulo pd^2 extended gcd problem (3) which satisfies $||v|| = O((\log d)^2)$ can be computed in $O(n^2(\log n + \log ||A||)^2 + n^3(\log n + \log ||A||)^3)$ word operations.*

We obtain the following result.

Proposition 3. *Let a permutation conditioning (B, P, C, p) for $A \in \mathbb{Z}^{n+(n+k)}$ together with the principal $n\times n$ minor d of B be given. Suppose that $k < n$. Then a lattice conditioning (Q, W, c) for (B, P, C, p) which satisfies $||Q|| = O((\log d)^2)$ can be computed in $O(kn^3(\log n + \log ||A||)^3)$ word operations.*

In practice, the code fragment below will compute a suitable $v \in \mathbb{Z}^{n\times 1}$ and c quickly. Correctness is easy to verify.

$c \leftarrow \bar{c}$; $\quad g \leftarrow \gcd(c, pd^2)$;
for i **from** 1 **to** n **do**
$\quad v[i] \leftarrow 0$; $g \leftarrow \gcd(g, \bar{u}[i])$;
$\quad$ **while** $\gcd(c, pd^2) \neq g$ **do** $c \leftarrow c + \bar{u}[i]$; $\quad v[i] \leftarrow v[i] + 1$

6 Kernel Basis Computation

Let a permutation conditioning (B, P, C, p) of $A \in \mathbb{Z}^{n+(n+k)}$ be given. Write $B = \left[\, B_1 \middle| B_2 \,\right]$ where $B_1 \in \mathbb{Z}^{n\times n}$. Assume $d = \det B_1$ is also given. We want to compute a basis of the kernel of A, i.e., an $N \in \mathbb{Z}^{(n+k)\times k}$ such that $\mathcal{L}(N) = \{v \in \mathbb{Z}^{n+k} \mid Bv = 0\}$. Noting that $AN = 0$ if and only if $BP^{-1}N = 0$ shows it will be sufficient to compute a kernel basis of B.

The construction given in the next fact is classical. The bound is also easy to derive. See for example [12].

Fact 1. *Let* $X = B_1^{\mathsf{adj}} B_2$ *and let* H *be the trailing* $k \times k$ *submatrix of the Hermite basis of* $\left[\begin{array}{c|c} B_1 & B_2 \\ \hline & I \end{array}\right]$. *Then a kernel basis for* B *is given by* $N = \left[\begin{array}{c} -XH(1/d) \\ \hline H \end{array}\right]$. *Moreover,* $||N|| \leq (\sqrt{n}||A||)^n$.

A happy feature of the basis given by Fact 1 is that it is canonical; it is the only basis which has trailing $k \times k$ submatrix in Hermite form. Suppose we had some other kernel basis $\bar{N}$ for B. Then we could construct H by transforming the trailing $k \times k$ block of $\bar{N}$ to Hermite form. We will use this observation in our construction of N. Recover X by solving the matrix system $B_1 X = dB_2$ using linear p-adic lifting. Let $M = \left[\begin{array}{c} -X \\ \hline dI \end{array}\right] \in \mathbb{Z}^{(n+k)\times k}$. Then $BM = 0$. The following observation is well known.

Fact 2. *Let* $M \in \mathbb{Z}^{(n+k)\times k}$ *have rank* k *and satisfy* $BM = 0$. *If* $G \in \mathbb{Z}^{k\times k}$ *is such that* $\mathcal{L}(G^T) = \mathcal{L}(N^T)$ *then* MG^{-1} *is a basis for the kernel for* B.

Compute the Hermite basis G^T of M^T. Then MG^{-1} is a basis for the kernel of B. In particular dG^{-1} is integral and has each diagonal entry a divisor of d. Recover H by computing the Hermite form of of dG^{-1}. Recovering G and H is accomplished using the modulo d algorithm as described in [3] or [5]. The cost is $O(nk^2)$ operations with integers bounded in length by $\log|d| = O(n(\log n + \log||A||))$ bits, or $O(n^3k^2(\log n + \log||A||)^2)$ word operations. This also bounds the cost of constructing X and post-multiplying X by $H(1/d)$.

Proposition 4. *Let a permutation conditioning* (B, P, C, p) *for* $A \in \mathbb{Z}^{n+(n+k)}$ *together with the principal* $n \times n$ *minor of* B *be given. Then a kernel basis for* A *can be computed in* $O(k^2n^3(\log n + \log||A||)^2)$ *word operations.*

7 Hermite Basis Computation

Recall that l is the minimal index such that the principal $(n-l) \times (n-l)$ submatrix of the Hermite basis of A is the identity. Our result is:

Proposition 5. *Let a permutation conditioning* (B, P, C, p) *for* $A \in \mathbb{Z}^{n+(n+k)}$ *together with the principal* $n \times n$ *minor* d *of* B *be given. Suppose* $k < n$. *Then the Hermite basis of* A *can be computed in in* $O(kn^3(\log n + \log||A||)^3 + l^2n^3(\log n + \log||A||)^2)$ *word operations.*

Proof. (Sketch) Let $\bar{B}$ be the first l rows of $B_1 + QB_2$. Write $\bar{B}$ as $\left[\,\bar{B}_1 | \bar{B}_2\,\right]$ where $\bar{B}_1$ is $(n-l) \times (n-l)$. Find $\bar{d} = \det \bar{B}_1$ using $l-1$ applications of Proposition 2. Let $\bar{C}$ be the recursive inverse of $\bar{B}_1$. (Note that we get $\bar{C}$ for free from C.) Compute a lattice conditioning $(\bar{Q}, \bar{W}, \bar{c})$ for $(\bar{B}, I_{n+k}, \bar{C}, p)$. Then $\gcd(\bar{c}, p\bar{d}^2) = 1$. Furthermore:

$$\overset{B}{\left[\begin{array}{c|c} \bar{B}_1 & \bar{B}_2 \\ \hline * & * \end{array}\right]} \left[\begin{array}{c|c} I_{n-l} & \\ \hline \bar{Q} & I_{k+l} \end{array}\right] \left[\begin{array}{c|c} (\bar{B}_1 + \bar{B}_2\bar{Q})^{-1}\bar{c} & \\ \hline & I_{k+l} \end{array}\right] = \left[\begin{array}{c|c} \bar{c}I_{n-l} & \bar{B}_2 \\ \hline * & * \end{array}\right].$$

where the transformed matrix on the right can be computed in $O(kn^3(\log n + \log ||A||)^2)$ word operations using p-adic lifting. By an extension of Lemma 1, the Hermite basis of this matrix augmented with d^2I will be the Hermite basis of B. The basis is computed using $O(nl^2)$ operations with integers bounded in length by $\log |d| = O(n(\log n + \log ||A||))$ bits.

8 System Solving

Our result is:

Proposition 6. *Let the following (associated to an $A \in \mathbb{Z}^{n+(n+k)}$) be given:*

- *a permutation conditioning (B, P, C, p),*
- *the principal $n \times n$ minor d of B, and*
- *a lattice conditioning (Q, W, c) for (B, P, C, p) which satisfies $||Q|| = O((\log d)^2)$.*

Then given a column vector $b \in \mathbb{Z}^{n+k}$, a minimal denominator solution to the system $Ax = b$ can be computed in $O(n^3(\log n + \log ||A||)^2)$ word operations.

Proof. The technique is essentially that used in [9]; we only give the construction here. Write B as $B = \left[B_1 | B_2 \right]$ where B_1 is $n \times n$. Compute $v = B_1^{-1}db$ and $w = (B_1 + B_2Q)^{-1}cb$. Find $s, t \in \mathbb{Z}$ such that $sd + tc = \gcd(d, c)$. Then

$$x = sP \left[\frac{I_n}{} \right] v + tP \left[\frac{I_n}{Q} \right] w$$

is a solution to $Ax = b$ with minimal denominator.

Note that there exists a diophantine solution to the system if and only if the minimal denominator is one.

9 Massaging and Machine Word Lifting

The algorithms in previous sections make heavy use of p-adic lifting to solve linear systems. For efficiency, we would like to always choose p to be a power of two. That is, $p = 2^\omega$ where ω is the length of a word on the particular architecture we are using, for example $\omega = 32,\ 64,\ 128$. Then the lion's share of computation will involve machine arithmetic.

Unfortunately, the input matrix A may not have full rank modulo two, causing the permutation conditioning described in Section 2 to fail. In this section we show how to transform A to a "massaged" matrix B of the same dimension as A but such that all leading minors of B are nonsingular modulo two. The massaged B can then be used as input in lieu of A.

The construction described here is in the same spirit as the Smith form algorithm for integer matrices proposed by [8] and analogous to the massaging process used to solve a linear polynomial system described in [10].

Definition 2. *A* massaging *of A is tuple (B, P, G, C) such that:*

- *$G \in \mathbb{Z}^{n\times n}$ is in Hermite form with each diagonal entry a power of two,*
- *$G^{-1}A$ is an integer matrix of full rank modulo two,*
- *$(B, P, C, 2^{\omega})$ is a permutation conditioning of A.*

Now we describe an algorithm to compute a massaging. Let $\bar{A}$ be the submatrix of A comprised of the first $n-1$ rows. Recursively compute a massaging $(\bar{P}, \bar{G}, \bar{B}, \bar{C})$ for $\bar{A}$. Write $\bar{B} = \left[\bar{B}_1 | \bar{B}_2\right]$ where $\bar{B}_1$ has dimension $(n-1)\times(n-1)$. Let $b = \left[b_1 | b_2\right]$ be the last row of A where b_1 has dimension $n-1$. Consider the over-determined linear system $x\left[\bar{B}_1 | \bar{B}_2\right] = \left[b_1 | b_2\right]$. This system is necessarily inconsistent since we assumed that A has full row rank. But for maximal t, we want to compute an $x \in \{0, 1, \ldots, 2^t - 1\}^{n-1}$ such that $xB_1 \equiv b_1 \bmod 2^t$, $xB_2 \equiv b_2 \bmod 2^{t-1}$ and $xB_2 \not\equiv b_2 \bmod 2^t$. At the same time find an elementary permutation matrix E such that the first component of $(b_2 - xB_2)E$ is not divisible by 2^t. The computation of x and E is accomplished using linear p-adic lifting with $p = 2^{\omega}$; for a description of this see [2] or [9]. Set

$$G = \left[\begin{array}{c|c} \bar{G} & \\ \hline x & 2^t \end{array}\right], \quad P = \bar{P}\left[\begin{array}{c|c} I_{n-1} & \\ \hline & E \end{array}\right], \quad B = \left[\begin{array}{c|c} I_{n-1} & \\ \hline & 1/2^t \end{array}\right]\left[\begin{array}{c|c} I_{n-1} & \\ \hline -x & 1 \end{array}\right]\left[\begin{array}{c} B \\ \hline b \end{array}\right].$$

Update the recursive inverse to produce C as described in Section 2.

We now estimate the complexity of computing a massaging. By Hadamard's bound, $\log_2 \det G \le n(\log_2 \sqrt{n} + \log_2 ||A||)$ which gives the worst-case bound

$$\lceil n + n\log_2(\sqrt{n}||A||)/\omega\rceil = O(n(\log n + \log ||A||))$$

on the number of lifting steps. This a worst-case factor of only $O(\log n + \log ||A||)$ more lifting steps than required to compute only a permutation conditioning.

The only quibble with massaging is that entries in B might be larger than entries in A. Recall that the parameter l is used to denote the smallest index such that the Hermite basis of A has principal $(n-l)\times(n-l)$ submatrix the identity. Then entries in the first $n-l$ rows of B are bounded by $||A||$. The bound

$$||G^{-1}|| \le (l+1)^{(l+1)/2} \tag{4}$$

is easy to derive. It follows that $||G^{-1}B|| \le n(l+1)^{(l+1)/2}||A||$. We remark that the bound (4) is pessimistic but difficult to improve substantially in the worst case. It is an unfortunate byproduct of the fact that the ring $\mathbb{Z}$ is archimedian. In practice, $||G^{-1}||$ is much smaller.

10 Implementation and Execution

All the algorithms described in the previous sections have been implemented in C using the GNU MP large integer package. While the implementation is still experimental, preliminary results are very encouraging for computing the determinant, kernel and Hermite form of matrices with the small k and l.

We have employed this code on matrices generated during the computation of class groups and regulators of quadratic fields using the algorithm described in [7]. This algorithm uses the index-calculus approach and is based largely on the self-initializing quadratic-sieve integer-factorization algorithm. As in the factoring algorithm, the matrices generated are very sparse, with on the order of only 0.5% of entries nonzero.

The kernel of the matrix is required to compute the regulator of a real quadratic field. In practice, only a few vectors in the kernel are sufficient for this purpose, so the dimension of the kernel is small. As noted earlier, the expected number of diagonal elements of the Hermite basis which are not 1 is also small. The algorithms described in this paper are especially effective for this type of input.

Timings

The following table summarizes some of the execution timings on input as described above. Times are in hours and minutes.

Input				Timings HH:MM				
n	$n+k$	l	% μ	Massaging	Det	Cond	Kernel	Hermite
6000	6178	1	.373	00:14	05:50	02:40	–	00:03
6000	6220	1	.460	00:17	06:33	03:10	–	00:03
5000	5183	0	.542	00:09	07:55	00:02	02:50	–
6000	6181	0	.473	00:15	27:15	00:04	05:07	–
8600	8908	0	.308	00:38	20:30	00:14	19:15	–
10500	10780	0	.208	01:09	68:06	00:15	36:40	–

All computations were performed on 866Mhz Pentium III processors with 256Mb of RAM. Machine word lifting was used. The times for the determinant computation represent total work done; each determinant was computed in parallel on a cluster of ten such machines.

The first two rows in the table correspond to input matrices from the computation of the class groups of two imaginary quadratic orders. In this case, there is no regulator and hence the kernel does not have to be computed. The remaining examples all arise from real quadratic fields. The Hermite basis was trivial for all theses examples, a fact which was immediately detected once the lattice determinant had been computed. The second example and the last example correspond to quadratic orders with 90 and 101 decimal-digit discriminants, respectively. These are the largest discriminants for which the class group and regulator have been computed to date.

For comparison, previous methods described in [7], and run on a 550Mhz Pentium, required 5.2 days to compute the determinant and Hermite form of the 6000×6220 matrix. The 6000×6181 matrix required 12.8 days of computing time to find the determinant, Hermite form and kernel on the same machine. Computation of the Hermite form of the 10500×10780 matrix required 12.1 days. In this latter case, the computation of the kernel was not possible without the new methods described in this paper.

References

1. H. Cohen and H. Lenstra, Jr. Heuristics on class groups of number fields. In *Number Theory, Lecture notes in Math.*, volume 1068, pages 33–62. Springer-Verlag, New York, 1983.
2. J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40:137–141, 1982.
3. P. D. Domich, R. Kannan, and L. E. Trotter, Jr. Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research*, 12(1):50–59, 1987.
4. J. L. Hafner and K. S. McCurley. A rigorous subexponential algorithm for computation of class groups. *J. Amer. Math. Soc.*, 2:837–850, 1989.
5. C. S. Iliopoulos. Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and Smith normal forms of an integer matrix. *SIAM Journal of Computing*, 18(4):658–669, 1989.
6. H. Iwaniec. On the problem of Jacobsthal. *Demonstratio Mathematica*, 11(1):225—231, 1978.
7. M. J. Jacobson, Jr. *Subexponential Class Group Computation in Quadratic Orders.* PhD thesis, Technischen Universität Darmstadt, 1999.
8. F. Lübeck. On the computation of elementary divisors of integer matrices. *Journal of Symbolic Computation*, 2001. To appear.
9. T. Mulders and A. Storjohann. Diophantine linear system solving. In S. Dooley, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '99*, pages 281–288. ACM Press, 1999.
10. T. Mulders and A. Storjohann. Rational solutions of singular linear systems. In C. Traverso, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '00*, pages 242–249. ACM Press, 2000.
11. A. Storjohann. A solution to the extended gcd problem with applications. In W. W. Küchlin, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '97*, pages 109–116. ACM Press, 1997.
12. A. Storjohann. *Algorithms for Matrix Canonical Forms.* PhD thesis, ETH – Swiss Federal Institute of Technology, 2000.
13. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.
14. D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, IT-32:54–62, 1986.