Parallel computation of the rank of large sparse matrices from algebraic K-theory

Jean-Guillaume Dumas, Philippe Elbaz-Vincent, Pascal Giorgi, Anna Urbanska

To cite this version:

Jean-Guillaume Dumas, Philippe Elbaz-Vincent, Pascal Giorgi, Anna Urbanska. Parallel computation of the rank of large sparse matrices from algebraic K-theory. PASCO ’07 - International Workshop on Parallel Symbolic Computation, Jul 2007, London, Canada. pp.43-52, 10.1145/1278177.1278186. hal-00142141v2

HAL Id: hal-00142141

https://hal.science/hal-00142141v2

Submitted on 24 Apr 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Parallel computation of the rank of large sparse matrices from algebraic K-theory

Jean-Guillaume Dumas
Laboratoire Jean Kuntzmann, UMR CNRS 5224
Université Joseph Fourier, B.P. 53 X, 38041 Grenoble, France.
Jean-Guillaume.Dumas@imag.fr

Philippe Elbaz-Vincent
Institut de Mathématiques et de Modélisation de Montpellier,
UMR CNRS 5149
Université Montpellier II, CC051, Place E. Bataillon, 34095 Montpellier cedex 5, FRANCE.
piev@math.univ-montp2.fr

Pascal Giorgi
Laboratoire LP2A, Université de Perpignan Via Domitia, 52, avenue Paul Alduy 66860 Perpignan France.
pascal.giorgi@univ-perp.fr

Anna Urbańska
Laboratoire Jean Kuntzmann, UMR CNRS 5224
Université Joseph Fourier, B.P. 53 X, 38041 Grenoble, France.
Anna.Urbanska@imag.fr

ABSTRACT
This paper deals with the computation of the rank and some integer Smith forms of a series of sparse matrices arising in algebraic K-theory. The number of non-zero entries in the considered matrices ranges from 8 to 37 millions. The largest rank computation took more than 35 days on 50 processors. We report on the actual algorithms we used to build the matrices, their link to the motivic cohomology and the linear algebra and parallelizations required to perform such huge computations. In particular, these results are part of the first computation of the cohomology of the linear group $GL_7(\mathbb{Z})$.

1. INTRODUCTION
1.1 Motivation from K-theory
Numerous problems in modern number theory could be solved or at least better understood, if we have a good knowledge on the algebraic K-theory (or motivic cohomology) of integers of number fields or the cohomology of arithmetic groups (i.e. subgroups of finite index of $GL_N(\mathbb{Z})$). As a short list, we could mention:

- modular forms and special values of $L$ functions,
- Iwasawa theory and understanding of the “cyclotomy”,
- Galois representations (or automorphic representations).

Let us explain first what is algebraic K-theory: to a commutative ring $R$ we can associate (functorially) an infinite family of abelian groups $K_n(R)$ which encodes a huge amount of information on its arithmetic, geometric and algebraic structures. These groups extend some classical notions and give higher dimensional analogues of some well known results. For instance if $R$ is a ring of integers or a polynomial ring over a field, we have

- $K_0(R)$ is the classical Grothendieck group (classifying finitely generated $R$-modules),
- $K_1(R)$ is the group of invertibles of $R$,
- $K_2(R)$ classifies the universal extensions of $SL(R)$ and is related to the Brauer group in the case of a field.

We can give a general abstract definition of $K_n$ for $n > 0$,

$$K_n(R) = \pi_n(BGL(R)^+),$$

where $BGL(R)^+$ is the Quillen $+$-construction applied to the classifying space $BGL(R)$ and $\pi_n$ denotes the $n$th homotopy group $[2]$. We also can see the $K$-groups as a way to understand $GL(R)$. For instance, $K_1(R)$ is isomorphic to $GL(R)$ modulo elementary relations. For a more detailed background on K-theory and its applications see $[2]$.

Fact: these groups are hard to compute.

1.1.1 The Vandiver conjecture as an illustration
The Vandiver conjecture plays an important role in the understanding of the “cyclotomy” in number theory. Its statement is as follows:

Conjecture of Vandiver: Let $p$ be an odd prime, $\zeta_p = e^{2\pi i / p}$, $C$ the $p$-Sylow subgroup of the class group of $\mathbb{Q}(\zeta_p)$
and $C^+$ the subgroup fixed by the complex conjugation. The Vandiver conjecture is the statement that $C^+ = 0$.

To illustrate its interplay with K-theory, we have

**Fact (Kurihara, 1992)**: If $K_{2n} (\mathbb{Z}) = 0$ for all $n > 0$, then the conjecture of Vandiver is true.

Some partial results on this conjecture and its connection with the cohomology of $SL_N (\mathbb{Z})$ are given in [2].

**General problem**: Find explicit methods for computing (co)homologies of arithmetic groups and the K-theory of number fields (or their ring of integers).

Our first task will be the computation of the (co)homologies of linear groups (mainly $GL_N (\mathbb{Z})$ and $SL_N (\mathbb{Z})$). We can show [2] that the computation of those groups is the key point for computing K-groups with our method. We will begin to recall some facts from topology.

### 1.1.2 Topological Excursion: Cellular complexes, or how to simply describe the "combinatorial" structure of a topological space

The notion of cellular complex is a generalization of a graph to several dimensions. We call $n$-cell a topological space homeomorphic to the open unit ball of $\mathbb{R}^n$ and such that its closure is homeomorphic to the closed unit ball. A cellular complex (or cell complex or also cellular decomposition or cellular space) is a family of sets $X^n$ (with $n \in \mathbb{N}$), such that each $X^n$ is a collection (eventually infinite) of $n$-cells. Usually we work with cell complexes with a finite number of cells.

A classical result [2] shows that any (reasonable) topological space can be approximated by such cell complexes.

### 1.1.3 Computation of the homology of a cell complex

To a cell complex, we can associate a family $C_n$ ($n \in \mathbb{N}$) of free $\mathbb{Z}$-modules and a family $d_n : C_n \rightarrow C_{n-1}$ of linear maps.

The module $C_n$ is the free module with basis the $n$-cells (modulo a choice of orientation).

If the complex is finite, then all the modules are of finite rank and we will denote by $\{b^n_\lambda\}_{\lambda \in \Lambda_n}$ a basis of $C_n$, $\Lambda_n$ being an index set for the $n$-cells. Then the map $d_n$ is defined by

$$d_n (b^n_\lambda) = \sum_{\mu} \langle b^n_\lambda : b^{n-1}_\mu \rangle b^{n-1}_\mu,$$

and the integer number $[b^n_\lambda : b^{n-1}_\mu]$ is called the incidence number of the cell $c^n_\lambda$ inside the cell $c^{n-1}_\mu$. The relation $d_n \circ d_{n+1} = 0$ (i.e., formally $d^2 = 0$) should hold for any $n$.

If the complex is regular (i.e., always at most one cell of dimension $n + 1$ between two cells of dimension $n$), then we can build the incidences inductively starting from the 0-cells up to the maximal cells using the $d^2 = 0$ condition and moreover the incidence numbers will be 0, or $\pm 1$.

The $n$th homology group of the complex is defined as the quotient of $\text{Ker}(d_n)$ by $\text{Im}(d_{n+1})$. This construction is functorial (in the category of cell complexes). As a consequence, we can determine the homology groups effectively by computing the Smith form of the integral matrices of the $d_n$ (relatively to the fixed basis). Notice that the Smith form gives both the rank of the free part and the explicit description of the torsion part. In case the computation of the torsion is unnecessary (or to difficult to achieve), we can tensorize by $\mathbb{Z} \otimes \mathbb{Q}$ and the homology groups become $\mathbb{Q}$-vector spaces with their dimensions given by the ranks of the matrices of the differentials.

In general, the matrices of the differentials can be very large even for a relatively simple cell complex. However, they are also very sparse and we may look for some other favorable properties which would enable the computation despite the size of the problem.

### 1.1.4 How to use such settings for the computation of linear groups?

If $G$ is a group acting on a cell space $X$ (i.e., $G$ sends $n$-cells to $n$-cells), then, under some technical assumptions on $X$ and on the action, we can show [2] that roughly computing the homology of $G$ (as group homology) is the same as computing the homology of the cell space $X / G$. Hence, if $X / G$ can be calculated effectively, we can compute explicitly its homology, and from this the homology of $G$ (similarly for the cohomology). Notice that in general the space $X / G$ will not be regular anymore. The main difficulty is to find a cell space $X$ such that $X / G$ will be effective. We will discuss in section [3] how we can construct such cellular space for linear groups.

### 1.2 Parallelism motivations

This first idea to deal with very large sparse matrices is to use them as blackboxes, i.e. only using the matrix-vector product. This will let the matrix remain sparse all along the computation of the sequence. Then the algorithm candidate is Wiedemann’s as shown in [2]. This algorithm computes a sequence of scalars of the form $u^tA^tv$ ($u$ and $v$ are vectors) with $i$ matrix vector products and dot products. It has been shown to successfully deal with large sparse matrix problems from homology, see e.g. [4]. Nevertheless, when matrices are very large (e.g millions on non-zero entries) computations would require months or years. This is due to the low practical efficiency of the computation of a sparse matrix-vector product. For instance, in our case of homology computation, one would need 300 days of CPU to compute the sequence involving matrix of $GL_7 (\mathbb{Z})$ with $n = 19$ ($GL_7d19$ matrix). To achieve computations of many homologies in a realistic time we then need to parallelize the computation of the sequence. Then the algorithm candidate...
is the block Wiedemann method, which computes a sequence $X^T A' Y$ where $X$ and $Y$ are blocks of vectors. This step can be easily parallelized by distributing vectors of block $Y$ to several processors. We thus have several objectives with regard to the parallelism in this paper:

- We want to solve large problems coming from homology computation.
- We want to experimentally validate our parallel implementation of the block Wiedemann rank algorithm.
- We want to show the parallel scaling of block Wiedemann approaches.

### 1.3 Summary of the paper

In section 1.3 the algorithms and optimizations we used to generate the matrices and compute with them are discussed. Then, section 1.4 shows our experimental results on these large sparse matrices coming from homology.

## 2. ALGORITHMS AND OPTIMIZATIONS

As seen in section 1.3, we can effectively compute the homology of a cellular space and of a group which acts "nicely" on a cellular space. The main difficulty remains to find such explicit cellular space. In section 2, we present the process of matrix generation and optimization. Then in the following sections we give a description of the algorithms used for the computation of the rank and the Smith form of those matrices.

### 2.1 Matrices generation

In the case of subgroups of $\text{GL}_n(\mathbb{Z})$, we have an "obvious" action on $\mathbb{Z}^n$. We can then capture the topology by regarding $\mathbb{Z}^n$ not as a free $\mathbb{Z}$-module but as a lattice (or equivalently as quadratic forms), and see if this leads to some interesting topological construction. We will describe this approach below and the results that we can get.

#### 2.1.1 Voronoi’s reduction theory

Let $N \geq 2$ be an integer. We let $C_N$ be the set of positive definite real quadratic forms in $N$ variables. Given $h \in C_N$, let $m(h)$ be the finite set of minimal vectors of $h$, i.e. vectors $v \in \mathbb{Z}^N$, $v \neq 0$, such that $h(v)$ is minimal. A form $h$ is called perfect when $m(h)$ determines $h$ up to scalar: if $h' \in C_N$ is such that $m(h') = m(h)$, then $h'$ is proportional to $h$.

**Example 2.1.** The form $q(x, y) = x^2 + y^2$ has minimum 1 and minimal vectors $\pm(1,0)$ and $\pm(0,1)$. Nevertheless this form is not perfect, because there is an infinite number of definite positive quadratic forms having these minimal vectors. On the other hand, the form $q(x, y) = x^2 + xy + y^2$ has also minimum 1 and has exactly 3 minimal vectors (up to sign), the one above and $\pm(1,-1)$. This form is perfect, the associated lattice is the "honeycomb lattice" (with optimal spheres packing in the plane), it is the only one.

Denote by $C_N^+$ the set of non negative real quadratic forms on $\mathbb{R}^n$ the kernel of which is spanned by a proper linear subspace of $C_N^\perp$, by $X_N^+$ the quotient of $C_N^+$ by positive real homotheties, and by $\pi : C_N^+ \to X_N^+$ the projection. Let $X_N = \pi(C_N)$ and $\partial X_N^+ = X_N^+ - X_N$. Let $\Gamma$ be either $\text{GL}_n(\mathbb{Z})$ or $\text{SL}_n(\mathbb{Z})$. The group $\Gamma$ acts on $C_N$ and $X_N$ on the right by the formula

$$h \cdot \gamma = \gamma^t h \gamma, \quad \gamma \in \Gamma, \quad h \in C_N,$$

where $h$ is viewed as a symmetric matrix and $\gamma^t$ is the transposed of the matrix $\gamma$. Voronoi proved that there are only finitely many perfect forms modulo the action of $\Gamma$ and multiplication by positive real numbers ([14], Th. p. 110).

Given $v \in \mathbb{Z}^N - \{0\}$ we let $\hat{v} \in C_N^+$ be the form defined by

$$\hat{v}(x) = (v \mid x)^2, \quad x \in \mathbb{R}^N,$$

where $(v \mid x)$ is the scalar product of $v$ and $x$. The *convex hull* of a finite subset $B \subset \mathbb{Z}^N - \{0\}$ is the subset of $X_N^+$ image by $\pi$ of the elements $\sum \lambda_j v_j, v_j \in B, \lambda_j \geq 0$. For any perfect form $h$, we let $\sigma(h) \subset X_N^+$ be the convex hull of the set $m(h)$ of its minimal vectors. Voronoi proved in ([2], § 8-15), that the cells $\sigma(h)$ and their intersections, as $h$ runs over all perfect forms, define a cell decomposition of $X_N^+$, which is invariant by the action of $\Gamma$. We endow $X_N^+$ with the corresponding $CW$-topology. If $\tau$ is a closed cell in $X_N^+$ and $h$ a perfect form with $\tau \cap \sigma(h)$, we let $m(\tau)$ be the set of vectors $v$ in $m(h)$ such that $\hat{v}$ lies in $\tau$. Any closed cell $\tau$ is the convex hull of $m(\tau)$ and $m(\tau) \cap m(\tau') = m(\tau \cap \tau')$.

#### 2.1.2 Voronoi’s complex

Let $d(N) = N(N + 1)/2 - 1$ be the dimension of $X_N^+$ and $n \leq d(N)$ a natural integer. We define by $\Sigma_n$ a set of representatives, modulo the action of $\Gamma$, of those cells of dimension $n$ in $X_N^+$ which meet $X_N$, and by $\Sigma_n \subset \Sigma_n$ the cells $\sigma$ such that the stabilizer $\Gamma_\sigma$ of $\sigma$ in $\Gamma$ preserves its orientation. Let $V_n$ be the free abelian group generated by $\Sigma_n$. We define as follows a map

$$d_n : V_n \to V_{n-1}.$$  

For each closed cell $\sigma$ in $X_N^+$ we fix an orientation of $\sigma$, i.e. an orientation of the real vector space $\text{R}(\sigma)$ of symmetric matrices spanned by the forms $v, v \in m(\sigma)$. Let $\sigma \in \Sigma_n$ and let $\tau'$ be a face of $\sigma$. Given a positive basis $B'$ of $\text{R}(\tau')$ we get a basis $B$ of $\text{R}(\sigma)$ by adding after $B'$ a vector $\hat{v}, v \in m(\sigma) - m(\tau')$. We let $\varepsilon(\tau', \sigma) = \pm 1$ be the sign of the orientation of $B$ in the oriented vector space $\text{R}(\sigma)$ (this sign does not depend on the choice of $v$).

Next, let $\tau \in \Sigma_{n-1}$ be the cell equivalent to $\tau' = \tau \cdot \gamma$. We define $\eta(\tau, \tau') = 1$ (resp. $\eta(\tau, \tau') = -1$) when $\gamma$ is compatible (resp. incompatible) with the chosen orientations of $\text{R}(\tau)$ and $\text{R}(\tau')$.

Finally we define

$$d_n(\sigma) = \sum_{\tau \in \Sigma_{n-1}} \sum_{\tau'} \eta(\tau, \tau') \varepsilon(\tau', \sigma) \tau,$$

where $\tau'$ runs through the set of faces of $\sigma$ which are equivalent to $\tau$.

It is shown in ([14], up to $p$-torsions with $p \leq N + 1$, the homology of this complex computes the cohomology of $G$. For $N = 5, 6, 7$ we get the following results for $\Sigma_n$. 
The main challenge was then the computation of the ranks of matrices involved in the computation of the homology.

The previous result gives the precise size of the matrices and the computation of the Smith forms (which gives the relevant arithmetical information of the homology).

In the following paragraphs we will discuss the different methods chosen for the computations and to take up the challenge.

### 2.2 Coppersmith Block Wiedemann

One successful approach to deal with linear algebra computations on large sparse matrices is to rely on Lanczos/Krylov block-box methods. In particular, block versions of Wiedemann method are well suited for parallel computation. This technique has been first proposed by Coppersmith in for computation over $GF(2)$ and then analyzed and proved by Kaltofen and Villard and Lobo. The idea is for a matrix $A \in F_n^n$ to compute the minimal generating matrix polynomial of the matrix sequence $\{XA^iY \}_{i=0}^\infty \in F^{s \times s}$, where $X, Y$ are blocks of $s$ vectors (instead of vectors in the original Wiedemann’s algorithm). Therefore, an intuitive parallelization of this method is to distribute the vectors of the block $X, Y$ to several processors. Thus, the computation of the sequence, which is the major performance bottleneck of this method, can be done in parallel and then allow for better performance.

Lots of implementations and practical experiments have been developed on parallel block Wiedemann. For instance, in 1996, Kaltofen and Lobo have proposed a coarse grain parallelization of this method. For instance, in 1996, Kaltofen and Lobo have proposed a coarse grain parallelization of this method. For instance, in 1996, Kaltofen and Lobo have proposed a coarse grain parallelization of this method.

In the following paragraphs we will discuss the different methods chosen for the computations and to take up the challenge.

### 2.3 Block symmetry

In order to reduce the number of dot products, we used a symmetric projection. In other words, we set $X = Y^T$ in the $X, Y^T$ sequence. Indeed, in this case the probability of success is reduced but the obtained block is symmetric as
soon as $A$ is symmetric. This is always the case when the preconditioners of $\tilde{A}$ are used (they are of the form $A^T \tilde{A}$). This reduces the dot product part of the computation of the sequence by a factor of two. For instance column $i$ can be deduced from its top $i$ elements and row $i$. This induces some load balancing issues when one process owns the computation of one column. Note also that we use BLAS level-2 for the computation of this dot products. In other words we perform them by blocks.

### 2.4 $\sigma$-basis computations

In order to efficiently compute $\sigma$-basis we rely on algorithm PM-Basis of [25] which reduces this computation to polynomial matrix multiplication. One can multiply two polynomial matrices $A, B \in \mathbb{F}^{n \times n}[x]$ of degree $d$ in $O(n^3 d + n^d \log d)$ finite field operations if $d$-th primitive roots of unity are available in $\mathbb{F}$. Consequently, we decided for our computations to define $\mathbb{F}$ as a prime field with primes of the form $c \times 2^d + 1$ such that $c \times 2^d \equiv 0 \mod d$. These primes are commonly called FFT primes since they allow the use of FFT evaluation/interpolation for polynomials of degree $d$. We refer the reader to [4, 26] and references therein for further informations on fast polynomial matrix arithmetic.

When finite fields not having $d$-th primitive roots of unity are used, polynomial matrix multiplication is still be done efficiently by using Chinese Remainder Theorem with few FFT primes. Let be $\mathbb{F}$ a prime field of cardinality $p$, then the multiplication of $A, B \in \mathbb{F}^{n \times n}[x]$ of degree $d$ can be efficiently done by using CRT with FFT primes $p_i$ satisfying $\prod p_i > d \times n \times p^2$. This is equivalent to perform the multiplication over the integers and then reduce the result in $\mathbb{F}$. The overall performance of the multiplication, and then of the $\sigma$-basis, is dependent on the numbers of FFT primes needed.

### 2.5 Rank

Our main interest in the block Wiedemann approach is to compute the rank of large sparse matrices given by the Homology group determination problem explained in section 2.6. Hence, we rely on Kaltofen-Saunders‘ rank algorithm [27] and its block version [28] to achieve efficient parallel computation.

The Kaltofen-Saunders approach is based on the fact that if $\tilde{A}$ is a good preconditioned matrix of $A$ then its rank is equal to the degree of its minimal polynomial minus its valuation (or co-degree) [29]. Thus, by using well chosen preconditioners and Wiedemann algorithm one can easily compute the rank of a sparse matrix over a finite field. The block version of this method is presented e.g. in [29, §4]. We recall now the basic outline of this algorithm.

**Block Wiedemann Rank Algorithm**

1. **form** $\tilde{A}$ from $A$ with good preconditioners (e.g. those of [14]).
2. **choose** random block $Y \in \mathbb{F}^{n \times s}$ and compute the matrix sequence $S = \{Y^T \tilde{A} Y\}$ for $i = 0 \ldots 2n/s + O(1)$.
3. **compute** the minimal matrix generator $F^{\tilde{A}} \in \mathbb{F}^{s \times s}[x]$ of the matrix sequence $S$.

4. **return** the rank $r$ as $r = \deg(\det(F^{\tilde{A}})) - \text{codeg}(\det(F^{\tilde{A}}))$.

Note that if the minimal matrix of step 3 is in Popov form (e.g. computed using the $\sigma$-basis of [14]), then the degree of $\det(F^{\tilde{A}})$ is simply the sum of the row degrees of the matrix $F^{\tilde{A}}$. Then the co-degree is zero if the determinant of the constant term of $F^{\tilde{A}}$, seen as a matrix polynomial, is non-zero. In the latter case the computation of the determinant of the whole polynomial matrix can be avoided.

When this fails, this determinant is computed by a massively parallel evaluation/interpolation. It could be interesting, though, to interpolate only the lower coefficients of this polynomial incrementally. This was not required for the matrices we considered and we therefore did not investigate more on these speed improvements.

Note that to probabilistically compute the rank over the integers, it is sufficient to choose several primes at random and take the largest obtained value, see e.g. [14] for more details. Moreover, one can choose the primes at random among primes not dividing the determinant (and thus preserving the rank). In order to ensure this property it is sufficient to select primes not dividing the valence or last invariant factor computed by one of the methods of next section.

### 2.6 Smith form

The computation of the Smith form for the matrices of $GL_d(\mathbb{Z})$ turned out to be a very challenging problem.

#### 2.6.1 Smith for via the Valence

Prior experience with sparse homology matrices led us to try the SmithViaValence algorithm of [22]. The idea is to compute the minimal valence (the coefficient of the smallest non zero monomial of the minimal polynomial) of the product $A^T A$ to determine the primes $p$ which divide the invariant factors of the Smith form of $A$. When the primes have been found, one can compute the local Smith forms of $A$ at each $p$ separately and return the resulting Smith form $S$ as the product of the local Smith forms $S_p$ over all $p$. Local Smith form computation can be done by a repeated Gauss elimination modulo $p^\epsilon$ where the exponent $\epsilon$ is adjusted automatically during the course of the algorithm.

This algorithm works very efficiently for sparse matrices provided that the minimal polynomial of the product $AA^T$ has a small degree. Unfortunately, the latter condition does not hold in the case of $GL_d(\mathbb{Z})$ matrices. Moreover, some early experiments with small matrices showed that much more primes occur in the computed valence than in the Smith form of the original matrix.

#### 2.6.2 Saunders and Wan’s adaptive algorithm

Thus, we decided to apply the adaptive algorithm of Saunders and Wan [24] which is a modified version of Eberly-Giesbrecht-Villard algorithm [23]. In [24] the authors proposed a procedure OneInvariantFactor($r, A$) (OIF) which computes the $i$th invariant factor of a $n \times n$ matrix $A$. Then the binary search for distinct invariant factors allows them to find the Smith form of $A$. OIF reduces the $i$th factor
computation to the computation of the last \((n)\)th invariant factor (LIF) of a preconditioned matrix \(A + U_i V_i\), where \(U_i, V_i\) are random \(n \times (n - i)\) matrices. In \([2]\) the method was extended to handle the rectangular case of \(m \times n\) matrix. It is done by computing the last \((i)\)th invariant factor of a preconditioned matrix \(L_i R_i\), where \(L_i\) is a \(m \times i\) and \(R_i\) is a \(i \times n\) matrix.

The procedure OIF is of Monte Carlo probabilistic type where the probability of correctness is controlled by repeating the choice of preconditioners. Assuming the correctness of LIF computation, it gives a multiple of the \(i\)th invariant factor. In practice, LIF is also of randomized Monte Carlo type. The idea is to get a divisor of the last invariant factor by solving a linear equation \(Mx = b\) with random right-hand side. After several solvings we get the last invariant factor with large probability. Thus, the overall situation is more complex and we cannot exclude the possibility that some primes are omitted or unnecessary in the output of OIF. However, the probability that a prime is omitted or is unnecessary in this output can be controlled for each prime separately and is smaller for bigger primes.

Therefore in \([2]\) the authors introduce a notion of smooth and rough parts of the Smith form. The idea is to compute the local Smith form for smaller primes by for example the SmithViaValence or OIF algorithm and to recover only large factors. The idea is to compute the last \((i)\)th invariant factor of \(\times n\) matrix. It is done by computing the last \((i)\)th invariant factor of a preconditioned matrix \(L_i R_i\), where \(L_i\) is a \(m \times i\) and \(R_i\) is a \(i \times n\) matrix. Theorem 3.3.3 More adaptiveness

As we did not want to compute the valence, we introduced some minor changes to the algorithm, which at the end is as follows:

1. \(r = \text{rank}(A)\)
2. For primes \(1 < p < 100\) compute the local Smith form \(S_p\) of \(A\);
3. Compute \(s_i(A)\) by OneInvariantFactor algorithm;
4. \(P = \text{all primes } p > 100\) which divide \(s_r(A)\);
5. If \(P = \emptyset\) return \(S = \prod_{p \in P} S_p\);

One advantage of this method is that we get the information on the smooth form of the matrix very quickly. Moreover, the OIF computation acts as a certification phase which allows us to prove that no other primes are present with a sufficiently large probability. This probability is explicit in the following theorem:

**Theorem 2.3.** The probability that there exists a prime \(p > P\) that divides the \(i\)th invariant factor but does not divide the output of OIF which uses \(M\) random preconditioners \(L_i, R_i\) and \(N\) random vectors \(b\), with \(b \in \{0, 1, \ldots, \beta - 1\}\) and \(\beta > s_i(A)\), in the LIF procedure is bounded by

\[
M \sum_{p > P} \left(\frac{2}{p}\right)^N.
\]

**Proof.** As we take the gcd of the result with different preconditioners \(L_i, R_i\), is suffices that the LIF computation fails in one case to spoil the computation. We are free to choose a large bound for \(|b|\) such that \(|b| > s_i(A)\) without increasing the complexity of LIF computation. Then the probability that a prime \(p < |b|\) is omitted in LIF is less than or equal to \(\frac{1}{\prod p_i} < \frac{1}{2}\), see \([3]\). Finally, we bound the probability that any prime \(p > P\) \(|s_i(A)|\) is missing by taking a sum over all primes. \(\square\)

The limits of this method are imposed by the available memory. For example, it was possible to use the dense Dixon solver only for the six smallest matrices. Furthermore, sparse elimination reached its limits for matrices of size greater than 171375 \(\times 47271\) and 21074 \(\times 105054\) when the filling of the matrices started to be impossible to handle. For the 460261 \(\times 171375\) matrix GL7d15 and 105054 \(\times 349443\) matrix GL7d23, specialized space-efficient elimination procedures mod 2, 3 and 5 allowed us to compute the rank mod 2, 3 and 5 respectively.

### 2.6.4 Chain Reductions

The encountered problems have shown a need for a more elaborated reduction algorithm. We focused our attention on the algebraic reduction algorithm for chain complexes of \([11]\). We implemented a simplified version of the algorithm in the language of matrices using the LinBox library. The heuristic behind this algorithm is that Gaussian elimination can propagate from one matrix of a chain complex to the next thanks to the exactness of the differential map (i.e. \(d^2 = 0\) condition).

The motivations come from the geometric properties of homologies. By a free face we refer to a \((k - 1)\) cell which is in the differential of only one \(k\)-dimensional cell \(b\). By removing the pair \((a, b)\) we obtain a retract of the initial cell complex (viewed as a geometrical object), see Figure \([1]\). The process can be repeated. From the homology theory we know that the groups of homology are the same for the set and its retract. The removal of pairs leads to a reduction of the basis of the cell complex. We refer to \([1]\), Ch.4 for a full description of the procedure.

If the differential map is represented by matrices whose rows represent the cells of dimension \(k - 1\) and columns - dimension \(k\), the removal of a pair \((a, b)\) can be interpreted as the
removal of a row with only one non-zero entry (1-row) and
the column it points to. In the general case, the algebraic
reduction of \((a,b)\) such that \(d_k(b) = \lambda a + u\) is possible iff \(\lambda\)
is invertible in the ring of computation \(\mathbb{Q}[Z, \mathbb{Z}_{p^k}]\) - de-
pending on the problem. A modification \(\hat{d}\) of the differential
given by the formula
\[
d_k v = d_k(v) - \lambda^{-1} [v : a] d_k(b).
\]
Thus, in the basic case of free face removal no modification
is needed. In the case of matrices, the formula describes a
step of Gauss elimination where the reduced row is removed and
not permuted. This proves that the Smith form (or the
rank) of the initial and reduced matrix will be the same,
provided we add a number of trivial invariant factors equal
to the number of rows reduced to the reduced Smith form.

The important characteristic of this methods is that, thanks
to the exactness of the matrix sequence, we can also remove
row \(b\) and column \(a\) from the neighboring matrices. In this
way, elimination in one matrix can propagate on the others.

Due to the format of data (large files with matrix entries)
we decided to implement only the simplest case of 1-rows
removal which led to entries removal but no modifications.
We removed the empty rows/columns at the same time and
performed the whole reduction phase at the moment of read-
ing the files. This lead to vast matrix reduction in the case of
\(GL_7(\mathbb{Z})\) matrices from the beginning of the sequence. The
propagation of reductions unfortunately burned out near
\(GL_{7d14}\) matrix and stopped completely on \(GL_{7d19}\). Ap-
plying the process for the transposed sequence did not im-
prove the solution. Next step would be to implement the
propagation of Gauss elimination steps as in Eq. (2). It
would be interesting to examine whether the burn-out can
be connected to the loss of regularity for \(X/G\) and/or a huge
rectangularity of the input matrix \(GL_{7d14}\).

3. EXPERIMENTAL RESULTS

3.1 Parallel implementation

For the sake of simplicity in our experimental validation,
we decided to develop our parallel implementation using
shell tasks distribution on SMP architecture. Thus, a sim-
ple script code is used to distribute all different tasks over
all the processors and files are used to gather up the com-
puted results. Our parallel implementation has been done
as follow:

1. The block of vectors \(Y\) and the sparse matrix \(A\) are
   broadcasted on every processors.

2. Each process takes one column of the blackbox \(A^T.A\)
   and compute the corresponding column’s sequence using
   the first \(i\)th columns of \(Y\). Each process writes the
   result in a file labeled with the corresponding index of
   the column sequence.

3. When all previous processes have terminated, the \(\sigma\)-
   basis computation is sequentially performed after load-
ing the sequence from the generated files.

Despite the naïve approach used for the parallelization, our
implementation authorized us to perform very large computa-
tions as show in next section. However, our experiments
show a need for at least a more robust parallel computation
scheduler.

3.2 Rank and Smith form

All our computation have been done on a SGI Altix 3700

7 gathering 64 Itanium2 processors with 192Gb memory and

running SuSE Linux Enterprise System 10. Further infor-
mations on this platform are available at

waterloo.ca/mcf/c/pilatus/ncf/pilatus.

In Table 3 we include the information about the dimensions
of the \(GL_7(\mathbb{Z})\) matrices and their sparsity. The matrices are
very sparse which is illustrated by the fact that less than
1% of the entries are non-zero except for matrices \(GL_{7d10}\)
and \(GL_{7d11}\). This value drops to less than 0,2% in the case
of the largest matrices. Also in Table 3 we give the results
for the rank and the Smith form computations. We have
obtained a full information on the rank of \(GL_7(\mathbb{Z})\) ma-
trices. For the computation of the Smith form, full result has
been obtained in the case of matrices \(10,11,12,13,25,26\). For
matrices 14 and 24 only the smooth part of the Smith form
has been computed. For matrices 15 and 23 we have proved
the existence of a non-trivial local Smith form at 2 and 3
and a triviality of the local Smith form at 5. As the result
for these matrices we give the number of invariant factors
divisible by 2 and 3.

In Table 3 we give the times for the Smith form algorithms
used. For cases with * no data are available or relevant.

The rank computation for \(GL_7(\mathbb{Z})\) matrices was performed
modulo 65537. This FFT prime allowed us to use both
BLAS routines and sigma-basis reconstruction using fast
polynomial multiplication. In Table 3 we give the timings
for different operations used in the computation i.e. the
sparse matrix-vector product, BLAS-based matrix-vector pro-
duct and, for the sake of comparison, the time of scalar dot
product equivalent to the BLAS computation.

In Table 3 we give the estimation of sequential and paral-
lel cpu time of rank computation and compare it with the
real time of parallel computation. The times are estimated
based on the number of iterations and the times of one step
which can be computed from Table 3 (notice, that in the
scalar case we use 1 dot product instead of BLAS). The real
time of computation includes the time of writing and read-
ing the data which was considerable. The difference of the
real and estimated running times may also be due to the
loadable of the computation cluster. Moreover, long com-
putations suffered from system crashes and/or shutdowns.

Figure 3: Retraction for a square
Table 1: Results of the rank and Smith form computation for $\text{GL}_7(\mathbb{Z})$ matrix $A$ of dimension $n \times m$ with $\Omega$ non-zero entries. For (*) the information is incomplete - only divisors of the invariant factors were determined based on the rank mod 2 and 3 computation.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$n$</th>
<th>$m$</th>
<th>rank</th>
<th>ker</th>
<th>Smith form</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL7d10</td>
<td>8</td>
<td>60</td>
<td>1</td>
<td>1</td>
<td>59</td>
</tr>
<tr>
<td>GL7d11</td>
<td>1513</td>
<td>1019</td>
<td>60</td>
<td>59</td>
<td>960</td>
</tr>
<tr>
<td>GL7d12</td>
<td>37519</td>
<td>8899</td>
<td>1019</td>
<td>960</td>
<td>7939</td>
</tr>
<tr>
<td>GL7d13</td>
<td>356232</td>
<td>47241</td>
<td>8899</td>
<td>7938</td>
<td>39333</td>
</tr>
<tr>
<td>GL7d14</td>
<td>181183</td>
<td>171375</td>
<td>47241</td>
<td>39332</td>
<td>132043</td>
</tr>
<tr>
<td>GL7d15</td>
<td>609384</td>
<td>460260</td>
<td>171375</td>
<td>132043</td>
<td>28218</td>
</tr>
<tr>
<td>GL7d16</td>
<td>1488884</td>
<td>955128</td>
<td>460260</td>
<td>626910</td>
<td>921740</td>
</tr>
<tr>
<td>GL7d17</td>
<td>25978098</td>
<td>1548650</td>
<td>955128</td>
<td>626910</td>
<td>921740</td>
</tr>
<tr>
<td>GL7d18</td>
<td>35590540</td>
<td>1955309</td>
<td>1548650</td>
<td>921740</td>
<td>1033569</td>
</tr>
<tr>
<td>GL7d19</td>
<td>37322725</td>
<td>1911130</td>
<td>1955309</td>
<td>1033568</td>
<td>877562</td>
</tr>
<tr>
<td>GL7d20</td>
<td>29893084</td>
<td>1437547</td>
<td>1911130</td>
<td>877562</td>
<td>559985</td>
</tr>
<tr>
<td>GL7d21</td>
<td>18174775</td>
<td>822922</td>
<td>1437547</td>
<td>559985</td>
<td>262937</td>
</tr>
<tr>
<td>GL7d22</td>
<td>8251000</td>
<td>349443</td>
<td>822922</td>
<td>262937</td>
<td>86506</td>
</tr>
<tr>
<td>GL7d23</td>
<td>2695430</td>
<td>105054</td>
<td>349443</td>
<td>86505</td>
<td>18549</td>
</tr>
<tr>
<td>GL7d24</td>
<td>593892</td>
<td>21074</td>
<td>105054</td>
<td>18549</td>
<td>2525</td>
</tr>
<tr>
<td>GL7d25</td>
<td>81671</td>
<td>2798</td>
<td>21074</td>
<td>2525</td>
<td>273</td>
</tr>
<tr>
<td>GL7d26</td>
<td>7412</td>
<td>305</td>
<td>2798</td>
<td>273</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 2: Times for Smith Form computation for $\text{GL}_7(\mathbb{Z})$ matrices. From left to right: the dimensions of the matrix after reductions, rank approximation by reductions, time of reading and reducing the matrix, time for the adaptive algorithm for a reduced matrix; times for the original matrix: smooth form computation, adaptive algorithm; valence computation in parallel - sequential time equivalent.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$\tilde{n}$</th>
<th>$\tilde{m}$</th>
<th>$\tilde{r}$</th>
<th>Red</th>
<th>RAdaptive</th>
<th>SmoothSF</th>
<th>AdaptiveSF</th>
<th>SFValence</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL7d11</td>
<td>39</td>
<td>6</td>
<td>0.01s</td>
<td>0.01s</td>
<td>0.01s</td>
<td>$&lt;10^{-4}$s</td>
<td>0.69s</td>
<td>0.26s</td>
</tr>
<tr>
<td>GL7d12</td>
<td>289</td>
<td>58</td>
<td>0.30s</td>
<td>0.16s</td>
<td>0.30s</td>
<td>0.16s</td>
<td>9.75s</td>
<td>218.68s</td>
</tr>
<tr>
<td>GL7d13</td>
<td>7938</td>
<td>740</td>
<td>105054</td>
<td>18549</td>
<td>2525</td>
<td>1 (86488), 2 ($\cdot 12$), 6 ($\cdot 5$) (*)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GL7d14</td>
<td>165450</td>
<td>35741</td>
<td>4279</td>
<td>21.62s</td>
<td>*</td>
<td>9.75s</td>
<td>218.68s</td>
<td>4.04h</td>
</tr>
<tr>
<td>GL7d25</td>
<td>2797</td>
<td>20990</td>
<td>0</td>
<td>0.14s</td>
<td>0.14s</td>
<td>*</td>
<td>0.29s</td>
<td>26.81s</td>
</tr>
</tbody>
</table>

Table 4: A summary of large-scale parallel rank computation. From left to right: number of iteration in the scalar case, time estimation in this case, number of iterations on $p$ processors computed as $2 + 2 \cdot r/p$, average (real) time of sequence computation, estimated time on $p$ processors, the time of the $\sigma$ basis computation for a sequence of length $\text{iter}[p]$ of $p \times p$ matrices.
Table 3: CPU timings (in sec.) for different operations used in large-scale parallel rank computation. All times in seconds. From left to right: time of a matrix-vector product, a BLAS multiplication of a vector and a $30 \times \min(n,m)$ matrix $U$ and 30 dot products.

<table>
<thead>
<tr>
<th>A</th>
<th>$1A^T Au$ [s]</th>
<th>$1U^T v$ [s]</th>
<th>$30u^T v$ [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL7d11</td>
<td>0.0002</td>
<td>$&lt; 10^{-4}$</td>
<td>$&lt; 10^{-4}$</td>
</tr>
<tr>
<td>GL7d12</td>
<td>0.0038</td>
<td>0.0001</td>
<td>0.0002</td>
</tr>
<tr>
<td>GL7d13</td>
<td>0.0550</td>
<td>0.0005</td>
<td>0.0036</td>
</tr>
<tr>
<td>GL7d14</td>
<td>0.2677</td>
<td>0.0025</td>
<td>0.0190</td>
</tr>
<tr>
<td>GL7d15</td>
<td>0.9048</td>
<td>0.0082</td>
<td>0.0708</td>
</tr>
<tr>
<td>GL7d16</td>
<td>2.7724</td>
<td>0.0234</td>
<td>0.2641</td>
</tr>
<tr>
<td>GL7d17</td>
<td>7.8003</td>
<td>0.0485</td>
<td>0.5052</td>
</tr>
<tr>
<td>GL7d18</td>
<td>11.9457</td>
<td>0.0729</td>
<td>0.8710</td>
</tr>
<tr>
<td>GL7d19</td>
<td>13.3591</td>
<td>0.0948</td>
<td>1.0710</td>
</tr>
<tr>
<td>GL7d20</td>
<td>10.4056</td>
<td>0.0711</td>
<td>0.8587</td>
</tr>
<tr>
<td>GL7d21</td>
<td>5.7461</td>
<td>0.0408</td>
<td>0.4049</td>
</tr>
<tr>
<td>GL7d22</td>
<td>1.9919</td>
<td>0.0180</td>
<td>0.2082</td>
</tr>
<tr>
<td>GL7d23</td>
<td>0.4354</td>
<td>0.0052</td>
<td>0.0450</td>
</tr>
<tr>
<td>GL7d24</td>
<td>0.0843</td>
<td>0.0012</td>
<td>0.0085</td>
</tr>
<tr>
<td>GL7d25</td>
<td>0.0078</td>
<td>0.0002</td>
<td>0.0008</td>
</tr>
<tr>
<td>GL7d26</td>
<td>0.0007</td>
<td>$&lt; 10^{-4}$</td>
<td>$&lt; 10^{-4}$</td>
</tr>
</tbody>
</table>

Some restoration scripts were used to recover the data which unfortunately required re-running some part of the computation. Thus, the real time given in Table 3 Col.5 should be treated as a rough approximation.

4. CONCLUSION

Using the previous methods and computations, we get the following new result for the rational cohomology of $GL_7(\mathbb{Z})$.

**Theorem 4.1.** (Elbaz-Vincent/Gangl/Soulé) \[ H^m(GL_7(\mathbb{Z}), \mathbb{Q}) = \begin{cases} \mathbb{Q} & \text{if } m = 0, 5, 11, 14, 15, \\ 0 & \text{otherwise.} \end{cases} \]

Clearly the simple parallelization we used together with the highly optimized routines were the key to enable these computations.

To go further and solve even larger problems, it is mandatory to improve the parallelism. On SMP we can split the matrices into blocks and perform the matrix-vector products with different threads. This, and the unbalanced load we had when we choose to assign one vector to one process, advocate for the use of more advanced scheduling. We are experimenting KAAPI\footnote{Kernel for Adaptive, Asynchronous Parallel and Interactive programming. \texttt{kaapi.gforge.inria.fr}} but were not ready for the computation of $GL_7$.

Other improvements are of algorithmic type. They include the use of the sparse projections of $[2]$ for the matrix sequence. But then we lose the symmetry of the projections and therefore must pay a factor of two for the number of iterations. We could also use an early termination strategy to stop the iteration earlier, but up to now this require to lose the fast algorithm for the sigma bases. Then if a good structure for the sparsity of the matrices could be found, e.g. an adapted reordering technique, this would enable an efficient clustering and therefore faster and more scalable matrix-vector products.

In order to have the relevant part of the torsion of the integral cohomology of $GL_7(\mathbb{Z})$, we would need the complete description of the Smith forms of all the matrices described above. Our experiments have shown that this can be an enormously difficult task. The computation remains to be done but would have applications in number theory.

5. ACKNOWLEDGMENTS

We are grateful to Arne Storjohann and to the Computer Science Computing Facilities of the University of Waterloo for letting us fill up their SMP machine to perform our parallel computations.

6. REFERENCES


