Aalto University

Algorithm for calculating HNF of a matrix $M \in \mathbb{Z}^{n \times m}$ in C++

Kristian Wasastjerna
kristian.wasastjerna@aalto.fi

Pseudo code

- You create a matrix L to change the leftmost non lower triangular columns lowest (positionally) non zero element into zero
- Proceed upwards until the column is in lower triangular form
- Move onto the next column
- Continue until the matrix is lower triangular form

Algorithm 1: HNF(M)

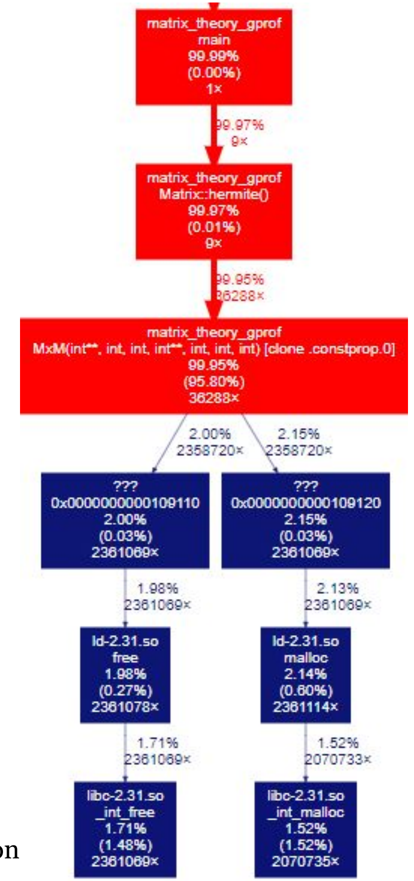
```
[n,m] $\leftarrow$ size($M$)
$U \leftarrow IMat(n)$
$L \leftarrow IMat(n)$
$H \leftarrow M$
for $j \leftarrow 1...m$ do
    for $i \leftarrow n...j+1$ do
        if $H_{i,j} = 0$
            Nothing
        else if $H_{i-1,j} = 0$
            Swap rows $i-1$ and $i$ in $L$
        else
            $a \leftarrow H_{i-1,j}, b \leftarrow H_{i,j}$
            $g \leftarrow gcd(a,b)$
            $[x,y] = \text{DiophantesSolver}(a,b,g)$
            $\alpha \leftarrow \frac{a}{g}, \beta \leftarrow \frac{b}{g}$
            $L_{i,i} \leftarrow \alpha,\ L_{i,i-1} \leftarrow -\beta,\ L_{i-1,i} \leftarrow y,\ L_{i-1,i-1} \leftarrow x$
        end
        $U \leftarrow LU, H \leftarrow LH$
        $L \leftarrow IMat(n)$
    end
end
return $[U,H]$
```

Pseudo code complexity

- Current complexity is $O(n^5)$
  - Since Matrix multiplication is $O(n^3)$
- Everything else is negligible time
  - O(log(max(A, B))) time complexity for Diophante solver
  - O(n) time for swapping rows of L
  - O(log(min(a, b)) time for gcd

Optimizing the algorithm

- Running profiling we can detect bottlenecks
  - Clearly only relevant bottleneck is matrix multiplication

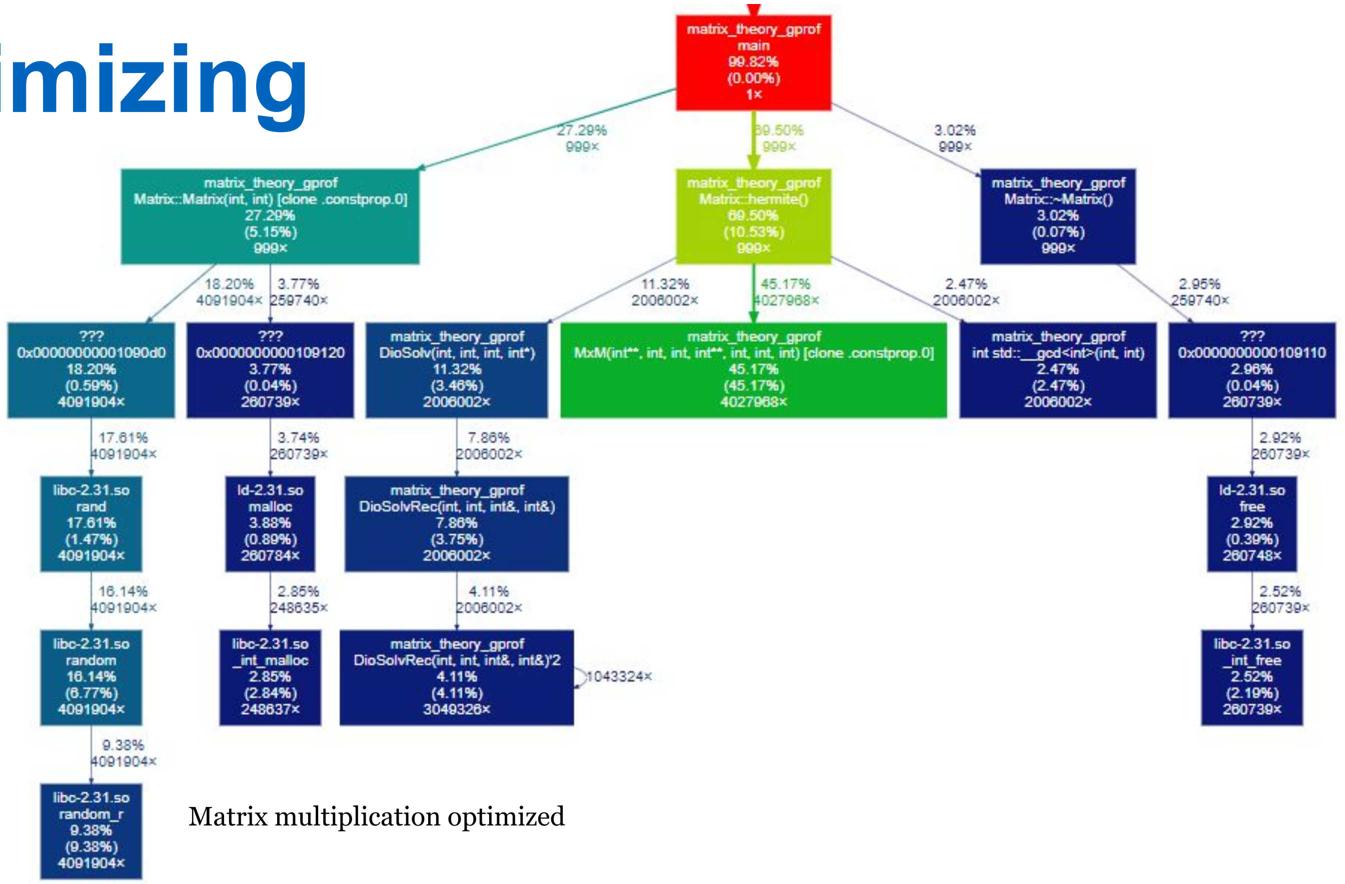


Naive implementation

Optimizing the algorithm

- Since L is a sparse matrix, with the form $I \oplus G \oplus I = L$, were $G \in \mathbb{Z}^{2\times2}$, we see that in LU, L only acts on the rows where G is located
- This means that $LU = [U_1, GU_2, U_3]^T$ were $U_2 \in \mathbb{Z}^{2\times m}$
- Hence we can change the matrix multiplication used in the algorithm to a specific case with complexity $O(m)$
- Also resetting L to a identity matrix can be done by replacing G with a identity matrix $I \in \mathbb{Z}^{2\times2}$

Optimizing

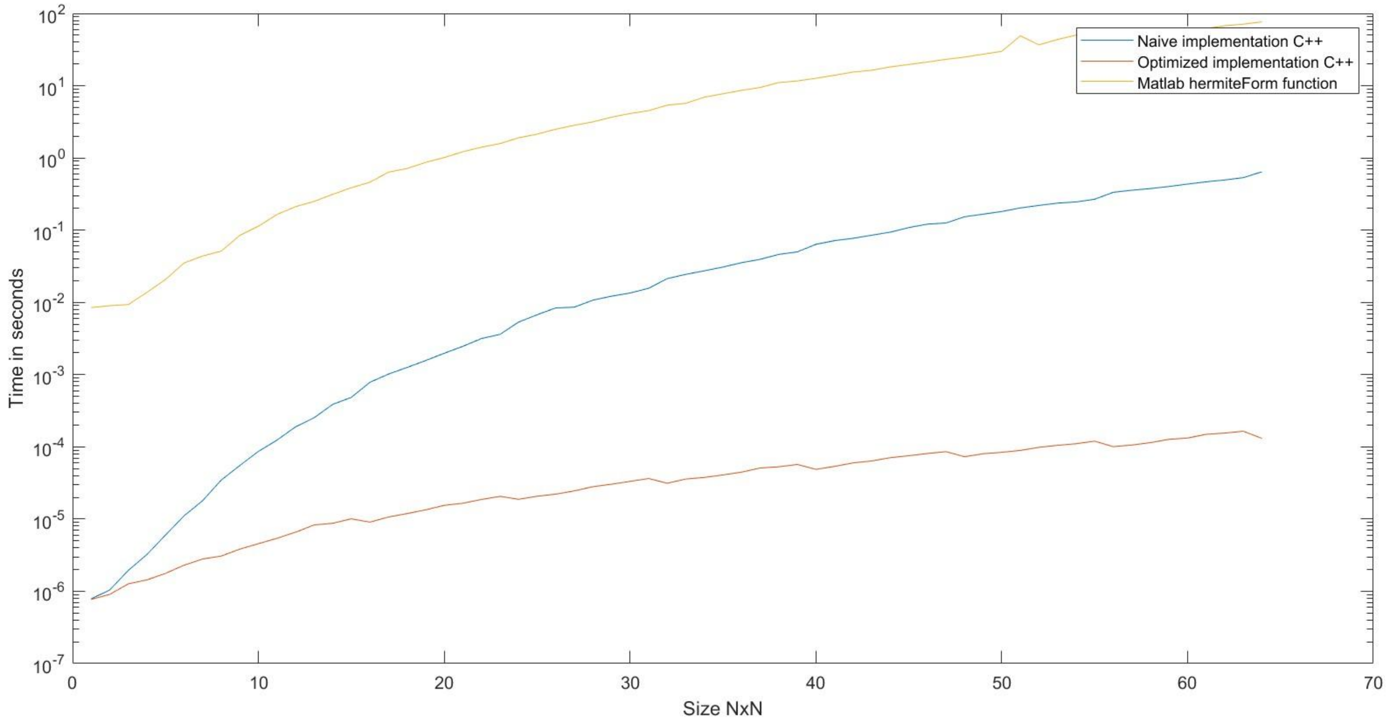

matrix_theory_gprof main 99.82% (0.00%) 1×
27.29% 999×
69.50% 999×
3.02% 999×
matrix_theory_gprof Matrix::Matrix(int, int) [clone .constprop.0] 27.29% (5.15%) 999×
matrix_theory_gprof Matrix::hermite() 69.50% (10.53%) 999×
matrix_theory_gprof Matrix::~Matrix() 3.02% (0.07%) 999×
18.20% 4091904×
3.77% 259740×
11.32% 2006002×
45.17% 4027968×
2.47% 2006002×
2.95% 259740×
??? 0x00000000001090d0 18.20% (0.59%) 4091904×
??? 0x0000000000109120 3.77% (0.04%) 260739×
matrix_theory_gprof DioSolv(int, int, int, int*) 11.32% (3.46%) 2006002×
matrix_theory_gprof MxM(int**, int, int, int**, int, int, int) [clone .constprop.0] 45.17% (45.17%) 4027968×
matrix_theory_gprof int std::__gcd<int>(int, int) 2.47% (2.47%) 2006002×
??? 0x0000000000109110 2.96% (0.04%) 260739×
17.61% 4091904×
3.74% 260739×
7.86% 2006002×
2.92% 260739×
libc-2.31.so rand 17.61% (1.47%) 4091904×
ld-2.31.so malloc 3.88% (0.89%) 260784×
matrix_theory_gprof DioSolvRec(int, int, int&, int&) 7.86% (3.75%) 2006002×
ld-2.31.so free 2.92% (0.39%) 260748×
16.14% 4091904×
2.85% 248635×
4.11% 2006002×
2.52% 260739×
libc-2.31.so random 16.14% (6.77%) 4091904×
libc-2.31.so _int_malloc 2.85% (2.84%) 248637×
matrix_theory_gprof DioSolvRec(int, int, int&, int&)'2 4.11% (4.11%) 3049326×
1043324×
libc-2.31.so _int_free 2.52% (2.19%) 260739×
9.38% 4091904×
libc-2.31.so random_r 9.38% (9.38%) 4091904×

Matrix multiplication optimized

Optimized algorithm

- Still over half of the time used in calculating the HNF is taken by matrix multiplication
- Diophantine equation solver takes around 15% of the time
- And around 15% is spent inside the HNF algorithm itself
    - Mainly updating variables and if statements

Comparing algorithms to Matlab



Comparing algorithms to Matlab

- Clearly the optimized version is a lot more efficient in runtime
- Compared to the matlab version both were vastly superior, why?
  - Possibly since Matlab automatically stores values as double precision floats
  - Matlab is used for mostly matrices over real and complex numbers
    - HNF implementation might be of little importance
  - Overflow checking, see next slide

Shortcomings of algorithm in C++

- There is no integer overflow guard in the algorithm
  - Since the values, especially at the bottom right grow large extremely quickly overflow errors happen and the result is not correct
  - This happens with elements in $M \in \mathbb{Z}^{n \times n}$ being between -10 and 10 and n>5
- Sometimes gives a U, H such that det(U)=-1 and UM=H
  - Bug for some unknown reason, fairly rare
    - 1 in 10
  - Fix is to just let $U \leftarrow -U$ and $H \leftarrow -H$

Sources

- https://github.com/KristianWasas/HNF-calc
- Chapter 2 of lecture notes
- https://github.com/google/benchmark
  - For benchmarking
- https://github.com/jrfonseca/gprof2dot
  - For visualization of profiling data
- Matlab R2022b for plotting