Accelerating lattice sieving in FPGAs

Michał Andrzejczak

Institute of Mathematics and Cryptology,
Military University of Technology in Warsaw

June 25, 2019

What is a lattice?

Lattice

Given n linearly independent vectors $b_1, b_2, \ldots, b_n \in \mathbb{Z}^n$, the lattice generated by them is defined as

$$\mathcal{L}(b_1, b_2, ..., b_n) = \left\{ \sum x_i b_i \mid x_i \in \mathbb{Z} \right\}. \tag{1}$$

Where $b_1, b_2, ..., b_n$ is a *basis B* of a lattice.

Shortest Vector Problem - SVP

For a given basis B belonging to $\mathbb{Z}^{n \times n}$ the shortest vector is a non-zero vector v, such that for every x belonging to $\mathcal{L}(B) \setminus \{0\}$

$$\|v\| \leq \|x\|$$

: where $\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$ is an Euclidean norm.

Shortest Vector - Estimation

For every lattice, the shortest vector can be estimated using

SVP Challenge restriction, TU Darmstadt

$$1.05 \cdot \frac{(\Gamma(n/2+1))^{1/n}}{\sqrt{\pi}} \cdot (det\mathcal{L})^{1/n} \tag{2}$$

- n — lattice dimension
- Γ — Euler's gamma function
- det $\mathcal{L}$ — determinant of lattice basis
- restriction comes as a combination of the Hadamard inequality, Hermite constant, and Minkowsky theorem

How to solve SVP? — Lattice sieving

How a lattice sieving works? Universal framework:

Algorithm 1 $\textsc{Sieve}(\mathcal{L})$

```
$L \leftarrow$ a set of $N$ random vectors from $\mathcal{L}$ where $N \approx (4/3)^{n/2}$.
while $\exists(\mathbf{v}, \mathbf{w}) \in L^2$ such that $\|\mathbf{v} - \mathbf{w}\| < \|\mathbf{v}\|$ do
    $\mathbf{v} \leftarrow \mathbf{v} - \mathbf{w}$
end while
return $L$
```

GaussSieve - the most well known sieve

- **Proposed by:** Micciancio and Voulgaris
- **Memory:** $2^{0.21n}$
- **Operations:** $\approx (2^{0.21n})^2$
- **Basic idea:** reducing new vectors by previously reduced. Set of vectors is Gauss reduced (no more reduction available).

Algorithm 2: GaussSieve(**B**,c)

```
Data: $B$ - lattice basis, $c$ - number of generated vectors,
$\lambda_1(B)$ - targeted norm
Result: $v : v \in \mathcal{L}(B) \wedge \|v\| \leqslant \lambda_1(B)$
begin
    $L \longleftarrow \{0\}$, $S \longleftarrow \{\}$, $K \longleftarrow 0$
    while $K < c$ do
        if $S \neq \varnothing$ then
            $v_{new} \leftarrow S.pop()$
        else
            $v_{new} \leftarrow SampleGaussian(B)$;
        end if
        $v_{new} \leftarrow GaussReduce(v_{new}, L, S)$
        if $v_{new} = 0$ then
            $K \leftarrow K + 1$
        else
            $L \leftarrow L \cup \{v_{new}\}$
        end if
    end while
end
```

Vector reduction - a critical operation

Algorithm 3: GaussReduce(**p, L, S**)

Data: p - lattice vector, L - list of already reduced vectors, S - list of vectors to reduce

Result: p - reduced vector

```
begin
    while $(\exists \mathbf{v_i} \in L : \|\mathbf{v_i}\| \leqslant \|\mathbf{p}\| \wedge \|\mathbf{p} - \mathbf{v_i}\| \leqslant \|\mathbf{p}\|)$ do
        $\mathbf{p} \leftarrow \mathbf{p} - \mathbf{v_i}$
    end while
    while $(\exists \mathbf{v_i} \in L : \|\mathbf{v_i}\| > \|\mathbf{p}\| \wedge \|\mathbf{v_i} - \mathbf{p}\| \leqslant \|\mathbf{v_i}\|)$ do
        $L \leftarrow L \backslash \{\mathbf{v_i}\}$
        $S.push(\mathbf{v_i} - \mathbf{p})$
    end while
    return p
end
```

Vector reduction - pseudocode

Algorithm 4: Reduce(**v, u**)

```
Data: $v, u$ - lattice vectors
Result: $v$ - reduced vector, $\|v\|^2$ - the squared norm of reduced vector
begin
    $dot = \sum v_i * u_i$
    if $2 * |dot| \leq \|u\|^2$ then
        return false
    else
        $q = \left\lfloor \frac{dot}{\|u\|^2} \right\rceil$
        for $i = 0; i < n; i++$ do
            $v_i -= q * u_i$
        end
        $\|v\|^2 += q^2 * \|u\|^2 - 2 * q * dot$
        return true
    end
end
```

Reduce method — dot product

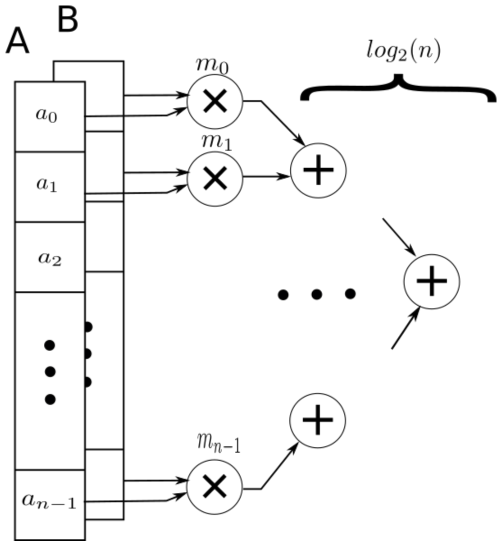


Figure: Hardware module for computing an inner product of two vectors

Division with rounding

Division with rounding to the nearest integer is the second operation in *Reduce* method and is computed as follows:

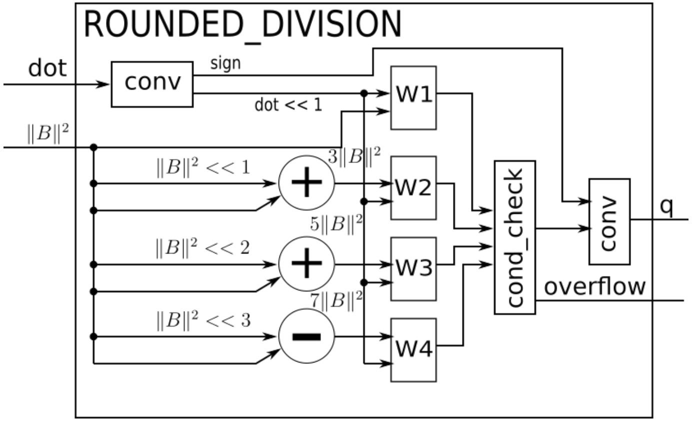
$$q = \left\lfloor \frac{dot}{\|B\|^2} \right\rceil \tag{3}$$

Taking into account the statistical experiments, division with rounding can be computed by checking several conditions on the absolute values. The sign of the dot product is removed at the beginning and applied at the end to the result.

q value	**Value of rounding**	**Condition**
$< 0; 0.5)$	0	$2 \cdot dot < \|B\|^2$
$< 0.5; 1.5)$	1	$\|B\|^2 \leq 2 \cdot dot < 3 \cdot \|B\|^2$
$< 1.5; 2.5)$	2	$3 \cdot \|B\|^2 \leq 2 \cdot dot < 5 \cdot \|B\|^2$
$< 2.5; 3.5)$	3	$5 \cdot \|B\|^2 \leq 2 \cdot dot < 7 \cdot \|B\|^2$
$< 3.5; 4.5)$	4	$7 \cdot \|B\|^2 \leq 2 \cdot dot < 9 \cdot \|B\|^2$

Table: Conditions for rounded division in a finite set

Reduce method — rounded division



Rounded division — proof

For the Euclidean inner product $< \cdot, \cdot >$ and any $x, y \in \mathbb{R}^n$:

$$|\langle x, y\rangle| \leq \|x\|\|y\|$$

so if we reject vectors with the norm k times larger than the estimated shortest one y, the maximum difference will be

$$\|x\| = k\|y\|$$

then the result of division of the dot product by the norm of the second vector will be

$$\frac{|\langle x, y\rangle|}{\|y\|^2} \leq k$$

For every vector of the length (Euclid norm) between $\|y\|$ and $4\|y\|$ this inequality is valid.

Reduce method — values update

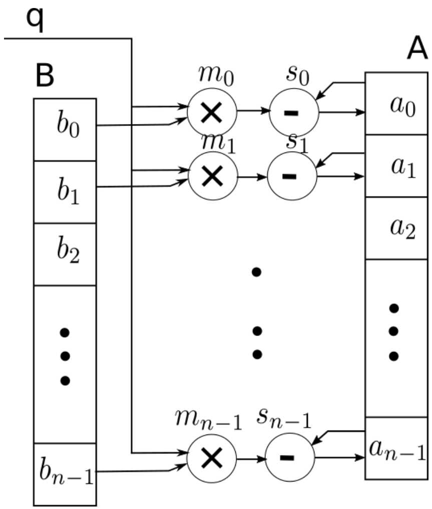


Figure: Hardware module for updating values of reduced vector

Reduce method — top view

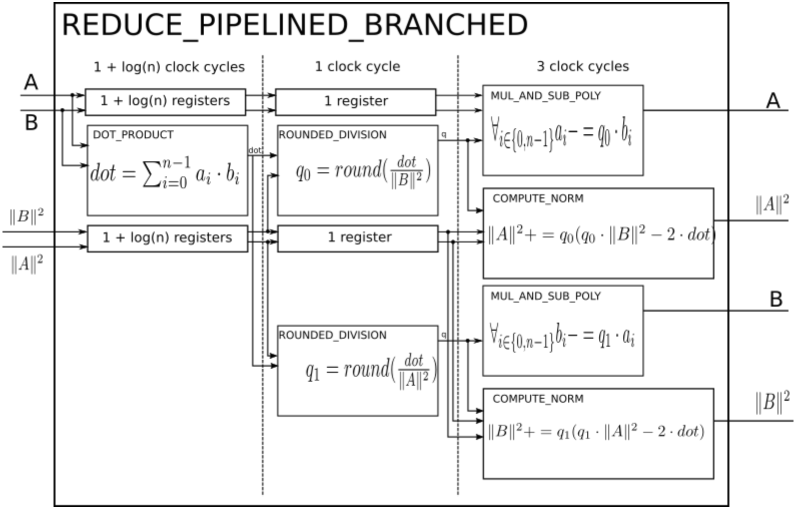

REDUCE_PIPELINED_BRANCHED
1 + log(n) clock cycles
1 clock cycle
3 clock cycles
A
B
1 + log(n) registers
DOT_PRODUCT
$dot = \sum_{i=0}^{n-1} a_i \cdot b_i$
$\|B\|^2$
$\|A\|^2$
1 + log(n) registers
dot
1 register
ROUNDED_DIVISION
$q_0 = round(\frac{dot}{\|B\|^2})$
q
1 register
ROUNDED_DIVISION
$q_1 = round(\frac{dot}{\|A\|^2})$
q
MUL_AND_SUB_POLY
$\forall_{i\in\{0,n-1\}} a_i - = q_0 \cdot b_i$
COMPUTE_NORM
$\|A\|^2 + = q_0(q_0 \cdot \|B\|^2 - 2 \cdot dot)$
MUL_AND_SUB_POLY
$\forall_{i\in\{0,n-1\}} b_i - = q_1 \cdot a_i$
COMPUTE_NORM
$\|B\|^2 + = q_1(q_1 \cdot \|A\|^2 - 2 \cdot dot)$
A
$\|A\|^2$
B
$\|B\|^2$

Figure: Top level view of hardware module for vector reduction with pipelined and branched execution

Use case 1 - simple reduction acceleration

- Basic idea - use FPGA for every vector reduction
- Most of the algorithm run on a PC. With every reduction, vectors are sent to FPGA.
- For 120 dimensional lattice: 102 clock cycles required (90 for communication, 12 for computation) — around 2 million operations per sec
- Comparing to CPU: 12 million ops/sec vs. 2.5 million ops/sec $\rightarrow$ nearly 6 times slower! (for 32 bit data bus)
- ... but CPU performance drops to 7.5 million ops/sec when memory size exceeds CPU cache
- Communication — the bottleneck (around 90% of execution time)
- Performance depends on the data bus width!

Performance comparison

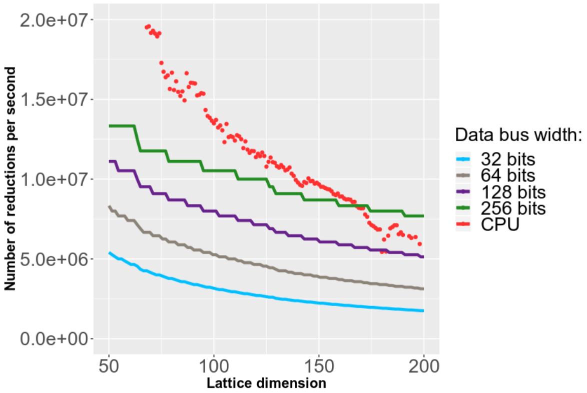


Figure: Performance comparison of accelerators with different data bus sizes and CPU (Core i7, 8th gen)

Let's minimize the communication!

Use case 2 - GaussSieve acceleration

- At first send large enough set of vectors to sieve to FPGA
- Execute at least $3n^2/2$ reductions
- Results are transferred back to CPU
- The biggest FPGAs can store only a small part of the required lattice points
- Offered speed-up depends only on available memory

Minimal size of the set

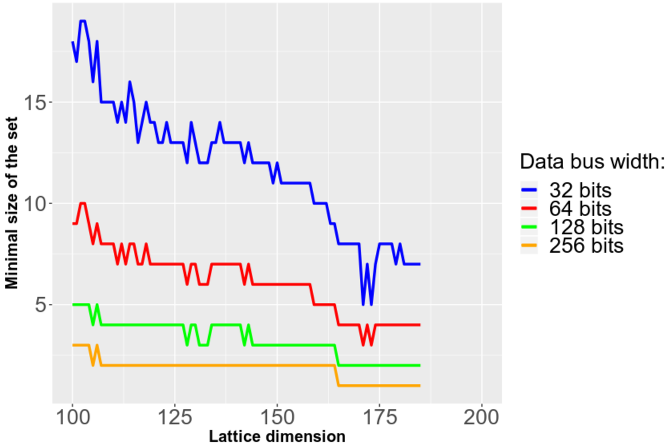


Figure: The minimal size of the set able to offer any acceleration

Maximal possible set

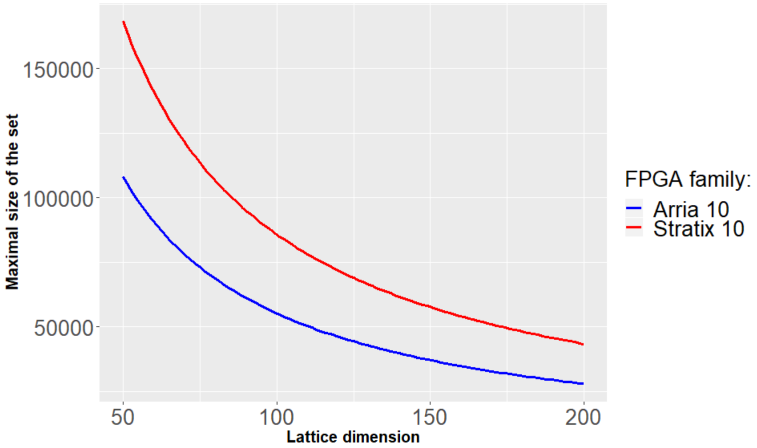


Figure: The maximal size of the set able to be fit on the largest device from Stratix 10 and Arria 10 families

Maximal possible acceleration

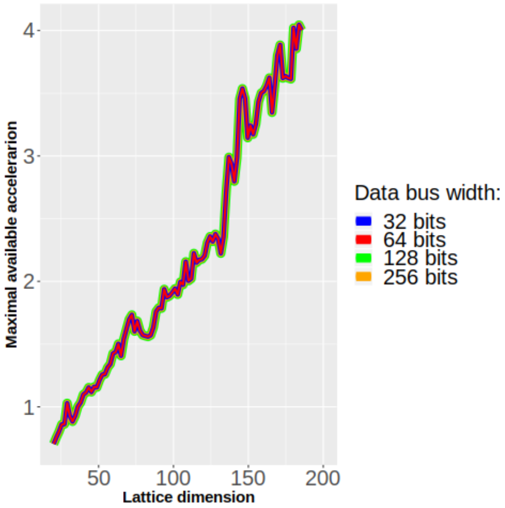


Figure: The maximal possible acceleration to obtain with the largest set able to be fit in the biggest Stratix 10 FPGA

Use case 3 - Paraller GaussSieve acceleration

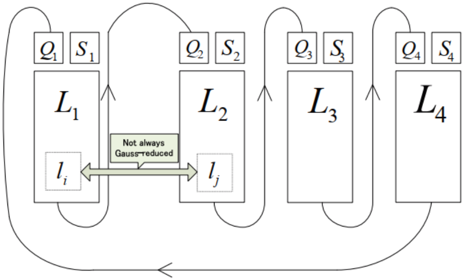


Figure: Graph representing an idea proposed by Milde and Schneider

- modified idea from paper by Milde and Schneider
- utilize pipelined execution of vector reduction
- compute Gauss-reduced subset of vectors
- can achieve up to **12x** more speed-up

Other ideas for improving results

- sampling vectors in hardware to minimize communication overhead
- implementing TupleSieve algorithm (n^3 reduce operations on smaller set than in GaussSieve)
- make changes in addition tree to sum up more than 2 elements by one addition unit — decrease the number of addition layers

Initilal results for Intel Arria 10

Lattice rank	Parameter	Version of module	
		base	*pipelined*
70	Logic	≈ 3500 *ALM*	≈ 7000 *ALM*
	DSP	144	218
	Clock	235 MHz	230 MHz
	ops/sec	$21 \cdot 10^6$	$460 \cdot 10^6$
120	Logic	≈ 5000 *ALM*	≈ 12500 *ALM*
	DSP	368	496
	Clock	212 MHz	202 MHz
	ops/sec	$19 \cdot 10^6$	$404 \cdot 10^6$

Table: Comparison of hardware modules conducting *Reduce* operation

Conclusions

- It is first reported so far attempt to accelerate lattice sieving
- The acceleration ratio depends on use case scenario and data bus width
- The acceleration ratio is limited by FPGA memory size, not by number of logic elements
- The acceleration for currently attacked dimension (120-160) is between 2-3.5x
- It is possible to achieve better results with more complex algorithms (like TupleSieve)

Thank you for your attention.