Resource counts for quantum floating-point multiplication and division

Shaun Miller
Department of Mathematical Sciences
Florida Atlantic University

Outline

Floating-Point Arithmetic in Lattice Based Cryptography

Resource Counts for High Precision

Floating-Point Values

$$x \approx (-1)^{x_s} x_M 2^{x_E}$$

$x \in \mathbb{Z}$ has binary represetation $x_{(0)}, x_{(1)}, \ldots, x_{(n-1)}$

$x_M \in [1, 2)$ so $x_{M(n-1)} = 1$

- precision[1]: $k = \text{width} - \lceil(4 * log_2(\text{width}))\rfloor + 13$
- exponent width: $\ell = \text{width} - k - 1$
  - embedded in $\ell + 1$ qubit register
- 1 qubit needed for sign

[1] IEEE std.754-2019. *IEEE Standard for Floating-Point Arithmetic.* 2019, pp. 1–23.

Lattice-Based Cryptography

- many post-quantum public-key proposals reduce to hard lattice problems[2]
- primal attack creates a lattice with a shortest vector that reveals key information

[2] Joppe Bos et al. "Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1006–1018; Erdem Alkim et al. "Post-quantum key exchange—a new hope". In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016, pp. 327–343; M Seo et al. "Proposal for NIST post-quantum cryptography standard: EMBLEM and R". In: *EMBLEM. NIST PQC Round* 1 (), p. 4.

b

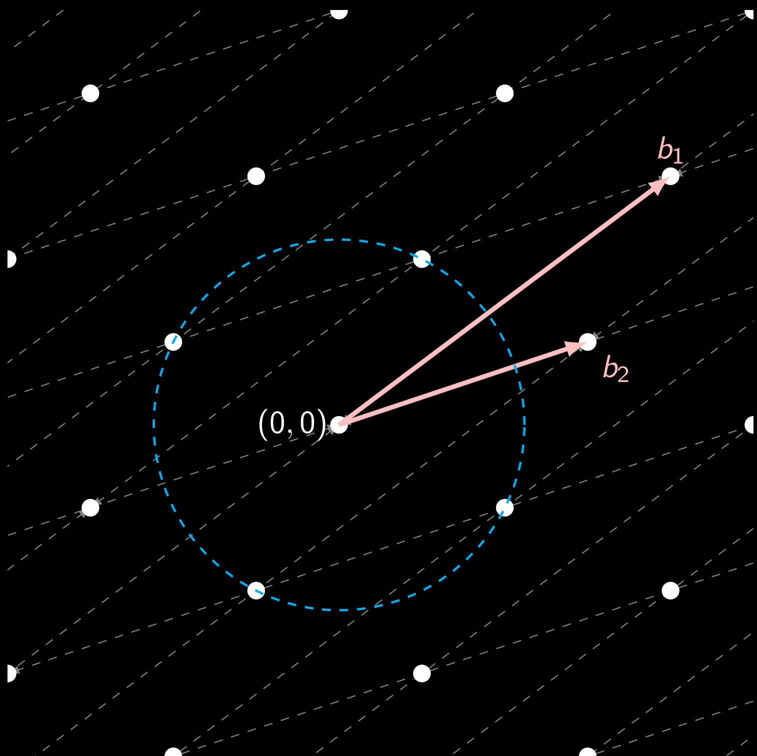


Figure: Lattice Generated by $b_1 = (4,3)$ and $b_2 = (3,1)$ with $\lambda_1(\Lambda) = \sqrt{5}$.

Lattice Reduction

- LLL[3]
- fp-arithmetic is required for large dimensional lattices[4]
  - precision of $\log_2(3) * d$
- a variant of BKZ-β[5] most successful
  - focuses resources on smaller sublattices or "blocks"

---

[3] Hendrik Willem Lenstra, Arjen K Lenstra, L Lovfiasz, et al. "Factoring polynomials with rational coefficients". In: (1982).

[4] Phong Q Nguyen and Damien Stehlé. "An LLL algorithm with quadratic complexity". In: *SIAM Journal on Computing* 39.3 (2009), pp. 874–903.

[5] Claus-Peter Schnorr and Martin Euchner. "Lattice basis reduction: Improved practical algorithms and solving subset sum problems". In: *Mathematical programming* 66.1-3 (1994), pp. 181–199.

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} \overbrace{b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5}^{\beta} & b_6 & b_7 & b_8 & b_9 & b_{10} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} \overbrace{b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5}^{\beta} & b_6 & b_7 & b_8 & b_9 & b_{10} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\left[\; b_1 \quad \overbrace{b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6}^{\beta} \quad b_7 \quad b_8 \quad b_9 \quad b_{10} \;\right]$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{array}{cccccccccc} & \multicolumn{5}{c}{\overbrace{\hspace{8em}}^{\beta}} & & & & \end{array}$$

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} b_1 \quad b_2 \quad \overbrace{b_3 \quad b_4 \quad b_5 \quad b_6 \quad b_7}^{\beta} \quad b_8 \quad b_9 \quad b_{10} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} b_1 \quad b_2 \quad \overbrace{b_3 \quad b_4 \quad b_5 \quad b_6 \quad b_7}^{\beta} \quad b_8 \quad b_9 \quad b_{10} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} b_1 & b_2 & b_3 & \overbrace{b_4 & b_5 & b_6 & b_7 & b_8}^{\beta} & b_9 & b_{10} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\left[\; b_1 \quad b_2 \quad b_3 \quad \overbrace{b_4 \quad b_5 \quad b_6 \quad b_7 \quad b_8}^{\beta} \quad b_9 \quad b_{10} \;\right]$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & \overbrace{b_5 \quad b_6 \quad b_7 \quad b_8 \quad b_9}^{\beta} & b_{10} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & \overbrace{b_5 & b_6 & b_7 & b_8 & b_9}^{\beta} & b_{10} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & \overbrace{b_6 \quad b_7 \quad b_8 \quad b_9 \quad b_{10}}^{\beta} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & \overbrace{b_6 \quad b_7 \quad b_8 \quad b_9 \quad b_{10}}^{\beta} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\left[\begin{array}{cccccccccc} & & & & & & \multicolumn{4}{c}{\overbrace{\hphantom{b_7 \quad b_8 \quad b_9 \quad b_{10}}}^{\beta - 1}} \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} \end{array}\right]$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{array}{cccccccccc} & & & & & & & \multicolumn{3}{c}{\overbrace{\hspace{8em}}^{\beta - 1}} \end{array}$$

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & \color{green}{b_7} & \color{red}{b_8} & \color{red}{b_9} & \color{red}{b_{10}} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & \overbrace{b_8 \quad b_9 \quad b_{10}}^{\beta - 2} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{array}{r} \overbrace{\hphantom{b_9 \quad b_{10}}}^{\beta - 2} \quad \\ \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & b_{10} \end{bmatrix} \end{array}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{array}{cccccccccc} & & & & & & & & & \overbrace{\phantom{b_{10}}}^{\beta - 3} \end{array}$$

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & \color{red}{b_9} & \color{red}{b_{10}} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\overbrace{\phantom{b_{10}}}^{\beta - 3}$$

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & \color{green}{b_9} & \color{red}{b_{10}} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\beta - 4$$

$$\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & \color{red}{b_{10}} \end{bmatrix}$$

BKZ-β algorithm calls SVP solver on "blocks" of size β

$$\begin{array}{cccccccccc} & & & & & & & & & \beta - 4 \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 & {\color{green}b_{10}} \end{array}$$

analysis of BKZ-β

cost of BKZ-β dominated by SVP solver

sieving runs in time $2^{O(n)}$ but requires exponential space

enumeration runs in $2^{O(n \log(n))}$

[2] suggests sieving more efficient for $n \geq 250$

lower bound for attack cost is given as $2^{c\beta}$

Scheme	β for Primal Attack[6]	$\log_2(3) * \beta$	ℓ	width
NewHope	386	612	23	636
Frodo	481	763	24	788
Emblem	337	535	22	558

Figure: Precision Requirements for L^2 algorithm on sublattice of dimension β

[6] Martin R Albrecht et al. "Estimate all the {LWE, NTRU} schemes!" In: *International Conference on Security and Cryptography for Networks*. Springer. 2018, pp. 351–367.

operation	width	qubits	T-count
fp	32	2207	28672
int	32	95	14525
fp	64	8511	114688
int	64	191	57757
fp	n	$2n^2 + 5n - 1$	$28n^2$
int	n	$3n - 1$	$14n^2 + 7n + 7$

Figure: Comparison of Integer[7] and Floating-Point[8] Division Circuits

[7] Himanshu Thapliyal et al. "Quantum circuit designs of integer division optimizing T-count and T-depth". In: *IEEE Transactions on Emerging Topics in Computing* (2019).

[8] Lafifa Jamal and Hafiz Md Hasan Babu. "Efficient approaches to design a reversible floating point divider". In: *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*. IEEE. 2013, pp. 3004–3007.

- T-Count of 7 for Toffoli (CCX) and Fredkin (CSWAP) gates[9]
- the cost of adding a control to any controlled gate is at most 8 additional T-gates[10]
  - I assume a CCCX gate has a T-count of 7+8

[9] Matthew Amy et al. "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.6 (2013), pp. 818–830.

[10] Peter Selinger. "Quantum circuits of T-depth one". In: *Physical Review A* 87.4 (2013), p. 042302.

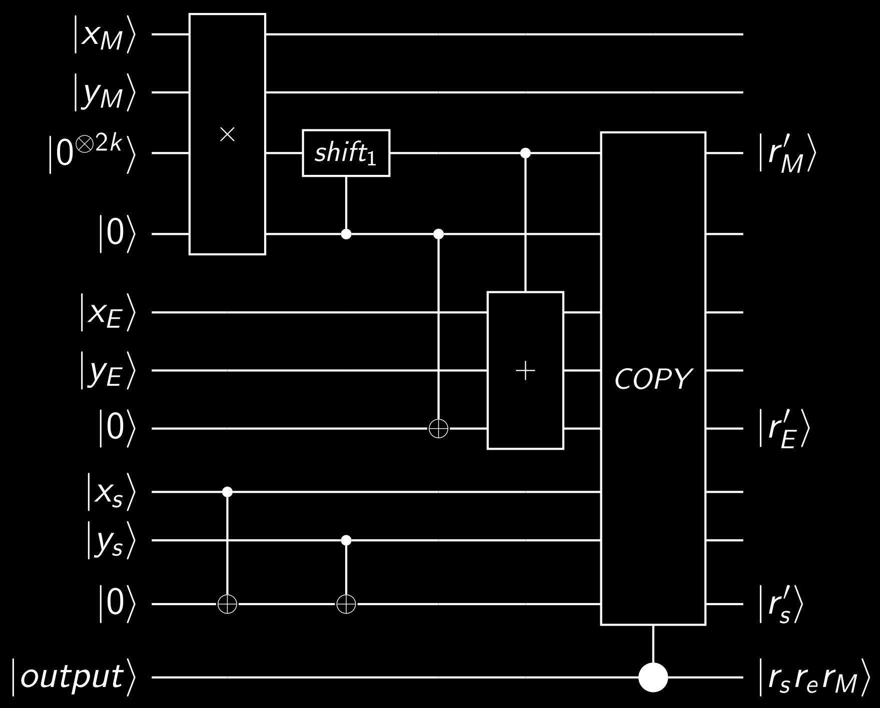


Figure: Floating-Point Multiplication Circuit[11]

[11] Thomas Häner et al. "Quantum circuits for floating-point arithmetic". In: *International Conference on Reversible Computation*. Springer. 2018, pp. 162–174.

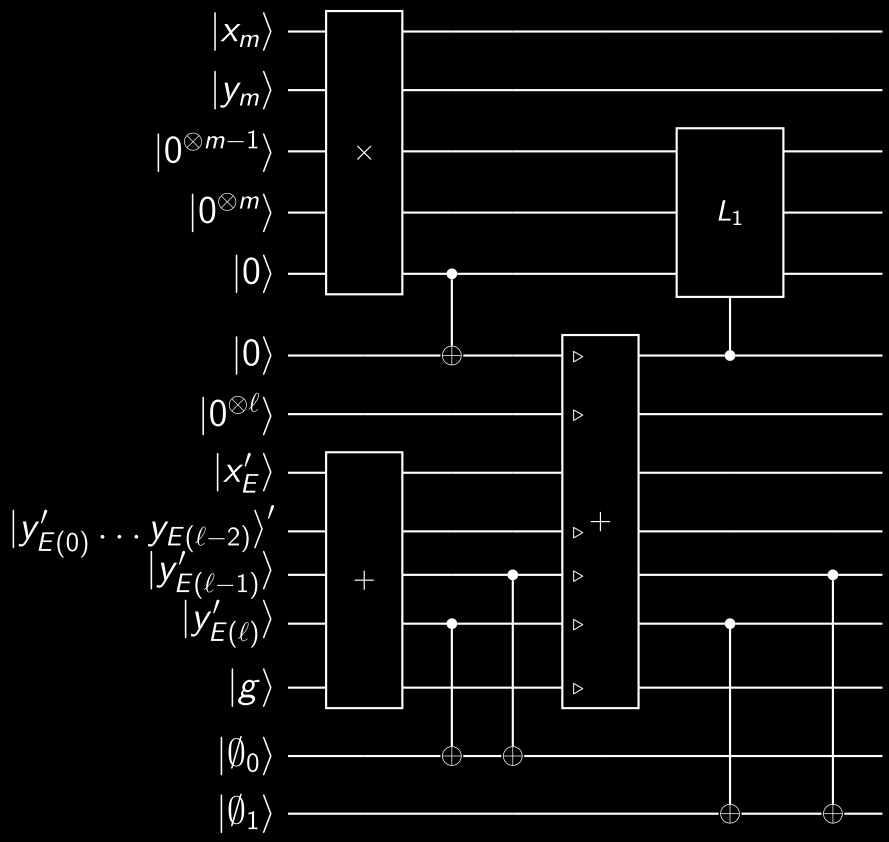


Figure: Floating-Point Multiplication Circuit (Modified). High $|\emptyset_i\rangle$ signals an underflow or overflow has occurred.

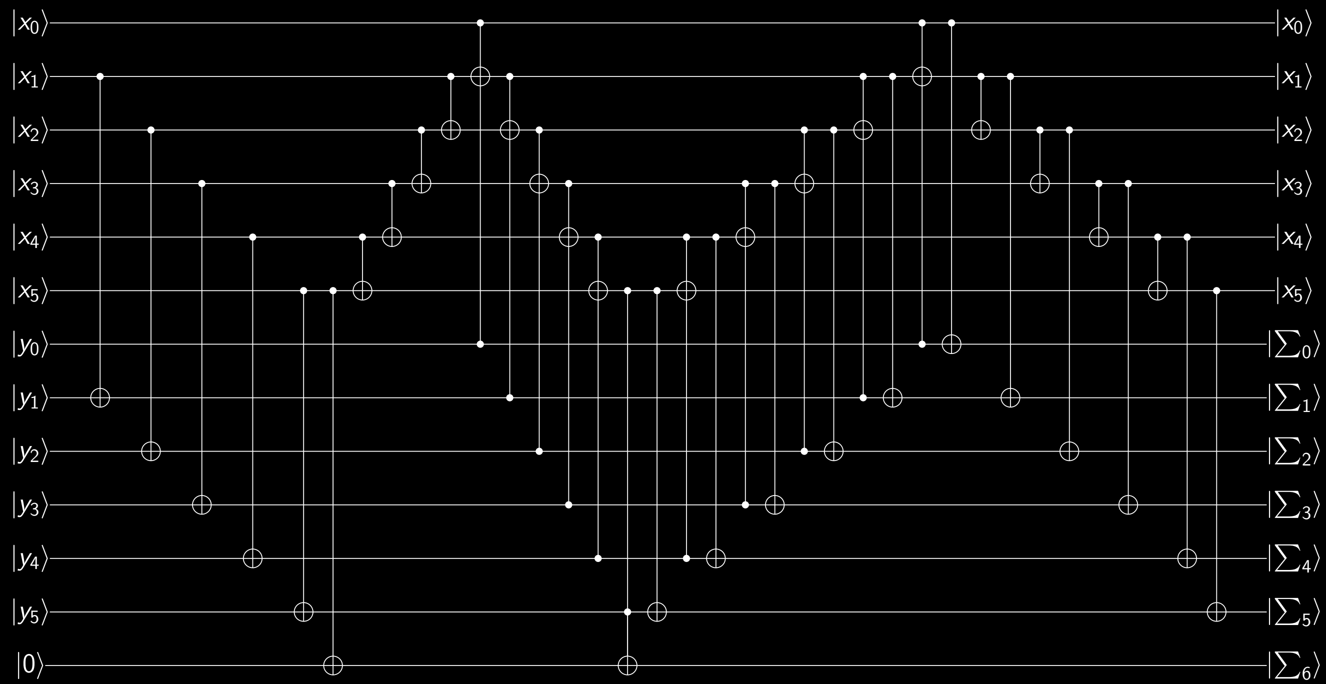


Figure: 6-qubit Addition Circuit[12]

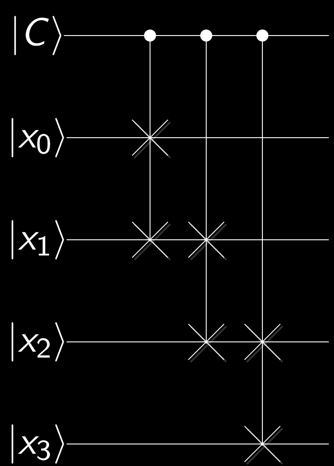
[12] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. "Quantum addition circuits and unbounded fan-out". In: *arXiv preprint arXiv:0910.2530* (2009).

	width	qubits	CCX	CCCX
$\times$	763	3052	2907030	1163575
$\times$	612	2448	1869660	748476
$\times$	535	2140	571915	1428450
$+$	25	51	120	49
$+$	24	49	115	47
$+$	23	47	110	45

Figure: Resource Counts[14] for Fixed-Point Operations Required for High Precision FP Arithmetic

[13] Damian S. Steiger, Thomas Häner, and Matthias Troyer. "ProjectQ: an open source software framework for quantum computing". In: *Quantum* 2 (2018), p. 49; Thomas Häner et al. "A software methodology for compiling quantum programs". In: *Quantum Science and Technology* 3.2 (2018), p. 020501.

[14] Steiger, Häner, and Troyer, "ProjectQ: an open source software framework for quantum computing"; Häner et al., "A software methodology for compiling quantum programs".



width	qubits	CSWAP
1526	1527	1525
1224	1225	1223
1070	1071	1069

Figure: L1 Circuit and Resource Requirements

	width	k	ℓ	qubits	T-count
fp $\times$	788	763	24	3133	37816660
fp $\times$	636	612	23	2526	24326341
fp $\times$	558	535	22	2215	25440528
fp $\div$	788	-	-	1245827	17386432
fp $\div$	636	-	-	812171	11325888
fp $\div$	558	-	-	625517	8718192

Figure: Resource Counts for High Precision Floating-Point Arithmetic

- The resource requirements for high-precision fp arithmetic are demanding
- qubit cost seems to be drastically improved by hand-optimized circuits
    - a hand-optimized fp circuit would probably benefit similarly
- for component circuit implementations see https://github.com/shaunmillerc1010