Parallel Spooky Pebbling Makes Regev Factoring More Practical

[arXiv:2510.08432]

Greg Kahanamoku-Meyer [1] Seyoon Ragavan [1] Katherine Van Kirk [2]

[1]MIT

[2]Harvard

March 5, 2026

Thanks Greg and Katherine for many of these slides!


PEBBLERS
ENDGAME

Quantum factoring algorithms

Given an n-bit integer N, compute its prime factorisation in poly(n) time.

[1] $O(\cdot)$ hides constant factors. $\widetilde{O}(\cdot)$ hides poly($\log n$) factors.

Quantum factoring algorithms

Given an n-bit integer N, compute its prime factorisation in poly(n) time.

- Believed to be classically hard (current fastest algorithms take time $2^{\widetilde{O}(n^{1/3})}$)[1]

[1] $O(\cdot)$ hides constant factors. $\widetilde{O}(\cdot)$ hides poly($\log n$) factors.

Quantum factoring algorithms

Given an n-bit integer N, compute its prime factorisation in poly(n) time.

- Believed to be classically hard (current fastest algorithms take time $2^{\widetilde{O}(n^{1/3})}$)[1]
- Shor's algorithm: this is quantumly doable

[1] $O(\cdot)$ hides constant factors. $\widetilde{O}(\cdot)$ hides poly($\log n$) factors.

Quantum factoring algorithms

Given an n-bit integer N, compute its prime factorisation in poly(n) time.

- Believed to be classically hard (current fastest algorithms take time $2^{\widetilde{O}(n^{1/3})}$)[1]
- Shor's algorithm: this is quantumly doable
- Key idea: run quantum period finding on $g(x) = a^x \bmod N$

[1] $O(\cdot)$ hides constant factors. $\widetilde{O}(\cdot)$ hides poly($\log n$) factors.

Bottleneck: computing $g(x)$

Task: compute $|x\rangle|0\rangle \mapsto |x\rangle|g(x)\rangle = |x\rangle|a^x \bmod N\rangle$

Shor

- Shor, Beauregard: doable with:
  - Gates: $O(n^3)$
  - Qubits: 1 for x, n for $a^x \bmod N$, and $n + O(1)$ ancilla qubits

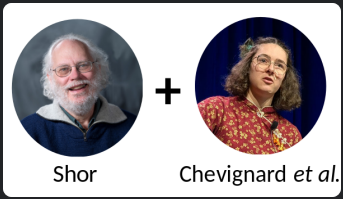
Bottleneck: computing $g(x)$

Task: compute $|x\rangle|0\rangle \mapsto |x\rangle|g(x)\rangle = |x\rangle|a^x \bmod N\rangle$

- Shor, Beauregard: doable with:
  - Gates: $O(n^3)$
  - Qubits: 1 for x, n for $a^x \bmod N$, and $n + O(1)$ ancilla qubits
- Meyer et al.: gates down to $O(n^{2+\epsilon})$!

Bottleneck: computing $g(x)$

Task: compute $|x\rangle|0\rangle \mapsto |x\rangle|g(x)\rangle = |x\rangle|a^x \bmod N\rangle$



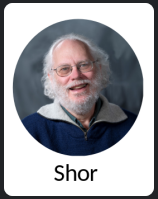
- Shor, Beauregard: doable with:
  - Gates: $O(n^3)$
  - Qubits: 1 for x, n for $a^x \bmod N$, and $n + O(1)$ ancilla qubits
- Meyer et al.: gates down to $O(n^{2+\epsilon})$!
- Chevignard et al.:
  - Gates: back to $O(n^3)$
  - Qubits: $(1/2 + \epsilon)n$ for x and $O(\log n)$ output bits and ancillas (just compute lowest-order bits of $a^x \bmod N$)

Bottleneck: computing $g(x)$

Task: compute $|x\rangle|0\rangle \mapsto |x\rangle|g(x)\rangle = |x\rangle|a^x \bmod N\rangle$

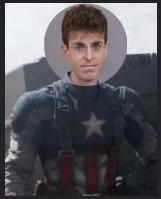
- Shor, Beauregard: doable with:
  - Gates: $O(n^3)$
  - Qubits: 1 for x, n for $a^x \bmod N$, and $n + O(1)$ ancilla qubits
- Meyer et al.: gates down to $O(n^{2+\epsilon})$!
- Chevignard et al.:
  - Gates: back to $O(n^3)$
  - Qubits: $(1/2 + \epsilon)n$ for x and $O(\log n)$ output bits and ancillas (just compute lowest-order bits of $a^x \bmod N$)
- Gidney: concrete improvements to Chevignard et al.

Enter Regev



Core of Shor's algorithm

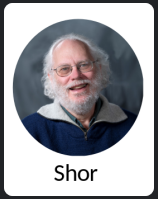
$g(x) = a^x \bmod N$, where
$x \leq 2^{O(n)}$.



Core of Regev's algorithm

$$g(\vec{x}) = \prod_{i=1}^{d} a_i^{x_i} \bmod N, \text{ where}$$
$d = O(\sqrt{n})$ and $x_i \leq 2^{O(\sqrt{n})}$

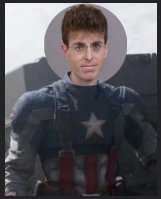
Enter Regev


Shor

Core of Shor's algorithm

$g(x) = a^x \bmod N$, where
$x \leq 2^{O(n)}$.

Gates: $\widetilde{O}(n^2)$ (per shot)

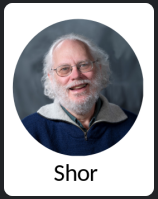


Core of Regev's algorithm

$$g(\vec{x}) = \prod_{i=1}^{d} a_i^{x_i} \bmod N, \text{ where}$$
$d = O(\sqrt{n})$ and $x_i \leq 2^{O(\sqrt{n})}$

Gates: $\widetilde{O}(n^{3/2})$ (per shot)

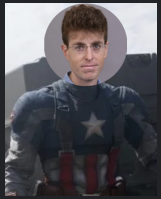
Enter Regev


Shor

Core of Shor's algorithm

$g(x) = a^x \bmod N$, where
$x \leq 2^{O(n)}$.

Gates: $\widetilde{O}(n^2)$ (per shot)
Qubits: $\widetilde{O}(n)$



Core of Regev's algorithm

$$g(\vec{x}) = \prod_{i=1}^{d} a_i^{x_i} \bmod N, \text{ where}$$
$d = O(\sqrt{n})$ and $x_i \leq 2^{O(\sqrt{n})}$

Gates: $\widetilde{O}(n^{3/2})$ (per shot)
Qubits: $O(n^{3/2})$

Roadmap

- Previous approaches to Regev
  - Original algorithm: fewer gates, but irreversible so more qubits
  - Making things reversible but concretely inefficient: Fibonacci arithmetic

- Pebbling!
  - Pebble games: accepting irreversibility
  - Previous work: adding parallelism *or* spookiness

- Our work: parallelism and spookiness join forces
  - Example
  - Pebbling results
  - Factoring results

Previous approaches to Regev

Roadmap

- Previous approaches to Regev
  - Original algorithm: fewer gates, but irreversible so more qubits
  - Making things reversible but concretely inefficient: Fibonacci arithmetic

- Pebbling!
  - Pebble games: accepting irreversibility
  - Previous work: adding parallelism *or* spookiness

- Our work: parallelism and spookiness join forces
  - Example
  - Pebbling results
  - Factoring results

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Repeated squaring: (everything is mod N)

$$a$$

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Repeated squaring: (everything is mod N)

$$a \to a^2$$

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Repeated squaring: (everything is mod N)

$$a \to a^2 \to a^4$$

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Repeated squaring: (everything is mod N)

$$a \to a^2 \to a^4 \to a^8$$

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Repeated squaring: (everything is mod N)

$$a \to a^2 \to a^4 \to a^8 \to a^{16}$$

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Repeated squaring: (everything is mod N)

$$a \to a^2 \to a^4 \to a^8 \to a^{16} \to \cdots$$

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Repeated squaring: (everything is mod N)

$$a \to a^2 \to a^4 \to a^8 \to a^{16} \to \cdots \to a^x$$

Gate count of modular exponentiation

Intuition: how to compute a^x mod N when x is a power of two?

Repeated squaring: (everything is mod N)

$$a \to a^2 \to a^4 \to a^8 \to a^{16} \to \cdots \to a^x$$

General case: multiply in extra factors of a along the way...

Gate count of modular exponentiation

Intuition: how to compute $a^x \bmod N$ when x is a power of two?

Repeated squaring: (everything is mod N)

$$a \to a^2 \to a^4 \to a^8 \to a^{16} \to \cdots \to a^x$$

General case: multiply in extra factors of a along the way...
Gate count: $O(\log x)$ multiplications of n-bit integers $\Rightarrow \widetilde{O}(n \log x)$

Regev's gate savings

Recall: computing $a^x \bmod N$ takes $\widetilde{O}(n \log x)$ gates

Shor

- Goal: compute $a^x \bmod N$ for $x \leq 2^{O(n)}$

Regev

- Goal: compute $\prod_{i=1}^{\sqrt{n}} a_i^{x_i} \bmod N$ for $x_i \leq 2^{O(\sqrt{n})}$

Regev's gate savings

Recall: computing $a^x \bmod N$ takes $\widetilde{O}(n \log x)$ gates

Shor

- Goal: compute $a^x \bmod N$ for $x \leq 2^{O(n)}$
- Gates: $\widetilde{O}(n \log 2^{O(n)}) = \widetilde{O}(n^2)$

Regev

- Goal: compute $\prod_{i=1}^{\sqrt{n}} a_i^{x_i} \bmod N$ for $x_i \leq 2^{O(\sqrt{n})}$

Regev's gate savings

Recall: computing $a^x \bmod N$ takes $\widetilde{O}(n \log x)$ gates

Shor

- Goal: compute $a^x \bmod N$ for $x \leq 2^{O(n)}$
- Gates: $\widetilde{O}(n \log 2^{O(n)}) = \widetilde{O}(n^2)$

Regev

- Goal: compute $\prod_{i=1}^{\sqrt{n}} a_i^{x_i} \bmod N$ for $x_i \leq 2^{O(\sqrt{n})}$
- Gates to compute just one $a_i^{x_i}$: $\widetilde{O}(n \log 2^{O(\sqrt{n})}) = \widetilde{O}(n^{3/2})$

Regev's gate savings

Recall: computing $a^x \bmod N$ takes $\widetilde{O}(n \log x)$ gates

Shor

- Goal: compute $a^x \bmod N$ for $x \leq 2^{O(n)}$
- Gates: $\widetilde{O}(n \log 2^{O(n)}) = \widetilde{O}(n^2)$

Regev

- Goal: compute $\prod_{i=1}^{\sqrt{n}} a_i^{x_i} \bmod N$ for $x_i \leq 2^{O(\sqrt{n})}$
- Gates to compute just one $a_i^{x_i}$: $\widetilde{O}(n \log 2^{O(\sqrt{n})}) = \widetilde{O}(n^{3/2})$
  - Gates to naively compute all $\sqrt{n}$ terms: $\sqrt{n} \cdot \widetilde{O}(n^{3/2}) = \widetilde{O}(n^2)$ 😢

Regev's gate savings

Recall: computing $a^x \bmod N$ takes $\widetilde{O}(n \log x)$ gates

Shor

- Goal: compute $a^x \bmod N$ for $x \leq 2^{O(n)}$
- Gates: $\widetilde{O}(n \log 2^{O(n)}) = \widetilde{O}(n^2)$

Regev

- Goal: compute $\prod_{i=1}^{\sqrt{n}} a_i^{x_i} \bmod N$ for $x_i \leq 2^{O(\sqrt{n})}$
- Gates to compute just one $a_i^{x_i}$: $\widetilde{O}(n \log 2^{O(\sqrt{n})}) = \widetilde{O}(n^{3/2})$
  - Gates to naively compute all $\sqrt{n}$ terms: $\sqrt{n} \cdot \widetilde{O}(n^{3/2}) = \widetilde{O}(n^2)$ 😢
- Regev magic: if the a_i's are all small, can essentially just pay the cost of computing one term! 😃

Regev's qubit problem

How to compute $|x\rangle|0\rangle \rightarrow |x\rangle|a^x \bmod N\rangle$ quantumly?

Regev's qubit problem

How to compute $|x\rangle|0\rangle \to |x\rangle|a^x \bmod N\rangle$ quantumly?

Repeated squaring?

$$|x\rangle|a\rangle \to |x\rangle|a^2\rangle \to |x\rangle|a^4\rangle \to |x\rangle|a^8\rangle \to |x\rangle|a^{16}\rangle \to \cdots \to |x\rangle|a^x\rangle$$

Regev's qubit problem

How to compute $|x\rangle|0\rangle \to |x\rangle|a^x \bmod N\rangle$ quantumly?

Repeated squaring?

$$|x\rangle|a\rangle \to |x\rangle|a^2\rangle \to |x\rangle|a^4\rangle \to |x\rangle|a^8\rangle \to |x\rangle|a^{16}\rangle \to \cdots \to |x\rangle|a^x\rangle$$

- **Issue:** squaring mod N is not reversible!

Regev's qubit problem

How to compute $|x\rangle|0\rangle \to |x\rangle|a^x \bmod N\rangle$ quantumly?

Repeated squaring?

$$|x\rangle|a\rangle \to |x\rangle|a^2\rangle \to |x\rangle|a^4\rangle \to |x\rangle|a^8\rangle \to |x\rangle|a^{16}\rangle \to \cdots \to |x\rangle|a^x\rangle$$

- **Issue:** squaring mod N is not reversible!

Regev's qubit problem

How to compute $|x\rangle|0\rangle \to |x\rangle|a^x \bmod N\rangle$ quantumly?

Repeated squaring?

$$|x\rangle|a\rangle \to |x\rangle|a^2\rangle \to |x\rangle|a^4\rangle \to |x\rangle|a^8\rangle \to |x\rangle|a^{16}\rangle \to \cdots \to |x\rangle|a^x\rangle$$

- **Issue:** squaring mod N is not reversible!
- Regev's original approach: square out of place to get $|a\rangle|a^2\rangle|a^4\rangle \ldots |a^x\rangle \Rightarrow O(n \log x)$ qubits 😢

Roadmap

- Previous approaches to Regev
  - Original algorithm: fewer gates, but irreversible so more qubits
  - Making things reversible but concretely inefficient: Fibonacci arithmetic

- Pebbling!
  - Pebble games: accepting irreversibility
  - Previous work: adding parallelism *or* spookiness

- Our work: parallelism and spookiness join forces
  - Example
  - Pebbling results
  - Factoring results

Making Shor reversible: classical precomputation

What if we restructure our computation so each step is reversible?

Making Shor reversible: classical precomputation

What if we restructure our computation so each step is reversible?

Shor

Replace **squaring** with **multiplication by constants**, which is reversible!

Making Shor reversible: classical precomputation

What if we restructure our computation so each step is reversible?

Shor

To compute $a^x \bmod N$

Replace **squaring** with **multiplication by constants**, which is reversible!

Making Shor reversible: classical precomputation

What if we restructure our computation so each step is reversible?

Shor

Replace **squaring** with **multiplication by constants**, which is reversible!

To compute $a^x \bmod N = \prod_i \left(a^{2^i}\right)^{x_i} \bmod N$:

Making Shor reversible: classical precomputation

What if we restructure our computation so each step is reversible?


Shor

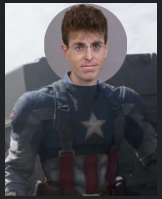
Replace **squaring** with **multiplication by constants**, which is reversible!

To compute $a^x \bmod N = \prod_i \left(a^{2^i}\right)^{x_i} \bmod N$:

- **classically** precompute $a, a^2, a^4, a^8, \cdots \pmod N$
- Multiply together for all i where bit $x_i = 1$

Making Shor reversible: classical precomputation

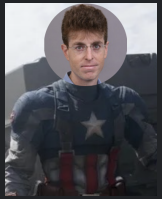
What if we restructure our computation so each step is reversible?



Regev's factoring speedup comes from multiplied values being **small**

Making Shor reversible: classical precomputation

What if we restructure our computation so each step is reversible?



Regev's factoring speedup comes from multiplied values being **small**

😢 $a_j^{2^i}$ mod N in general *not* small

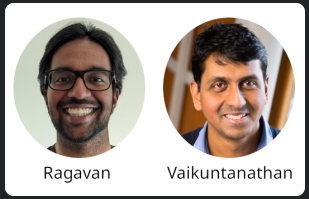
- Applying Shor's rearrangement would **kill Regev's improvement** in gate count

Making Regev reversible: Fibonacci arithmetic

What if we restructure our computation so each step is reversible?

Making Regev reversible: Fibonacci arithmetic

What if we restructure our computation so each step is reversible?



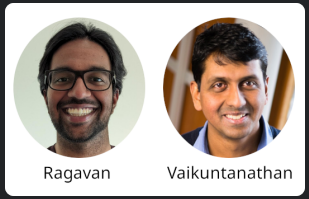
Let's do "repeated squaring," but with **Fibonacci number** exponents instead of powers of 2

😃 "Squaring" is **reversible** in this representation!!

- Maintains Regev's gate count of $\widetilde{O}(n^{3/2})$
- Reduces qubit count to $\widetilde{O}(n)$

Making Regev reversible: Fibonacci arithmetic

What if we restructure our computation so each step is reversible?



Let's do "repeated squaring," but with **Fibonacci number** exponents instead of powers of 2

😃 "Squaring" is **reversible** in this representation!!

- Maintains Regev's gate count of $\widetilde{O}(n^{3/2})$
- Reduces qubit count to $\widetilde{O}(n)$

😢 Large constant factors → Shor still wins in practice

When life gives you concretely inefficient reversible arithmetic...

When life gives you concretely inefficient reversible arithmetic...

... give up on reversibility and win anyway.

When life gives you concretely inefficient reversible arithmetic...

... give up on reversibility and win anyway.

Pebbling!

Roadmap

- Previous approaches to Regev
  - Original algorithm: fewer gates, but irreversible so more qubits
  - Making things reversible but concretely inefficient: Fibonacci arithmetic

- Pebbling!
  - Pebble games: accepting irreversibility
  - Previous work: adding parallelism *or* spookiness

- Our work: parallelism and spookiness join forces
  - Example
  - Pebbling results
  - Factoring results

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$x$$

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f_1(x)$$

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f_2(f_1(x))$$

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f_{k-1}(\cdots f_2(f_1(x))\cdots)$$

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

The general problem

General problem: How to do $|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$x$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$$|x\rangle$$

The general problem

General problem: How to do $|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f_1(x)$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$$|x\rangle\ |z_1\rangle$$

The general problem

General problem: How to do $|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f_2(f_1(x))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$$|x\rangle\ |z_1\rangle\ |z_2\rangle$$

The general problem

General problem: How to do $|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f_{k-1}(\cdots f_2(f_1(x))\cdots)$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$$|x\rangle\ |z_1\rangle\ |z_2\rangle\ \cdots\ |z_{k-1}\rangle$$

The general problem

General problem: How to do $|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$$|x\rangle\,|z_1\rangle\,|z_2\rangle\,\cdots\,|z_{k-1}\rangle\,|f(x)\rangle$$

The general problem

General problem: How to do $|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$$|x\rangle\ |z_1\rangle\ |z_2\rangle\ \cdots \qquad\qquad |f(x)\rangle$$

The general problem

General problem: How to do $|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$$|x\rangle\,|z_1\rangle\,|z_2\rangle \qquad\qquad |f(x)\rangle$$

The general problem

General problem: How to do $|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$$|x\rangle\,|z_1\rangle \qquad\qquad |f(x)\rangle$$

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

😃 Total quantum steps: $2k-1$ (optimal)

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$|x\rangle$ $|f(x)\rangle$

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$|x\rangle$ $|f(x)\rangle$

😃 Total quantum steps: $2k-1$ (optimal)

😢 Requires $O(k)$ ancilla registers...

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$|x\rangle$ $|f(x)\rangle$

😃 Total quantum steps: $2k-1$ (optimal)

😢 Requires $O(k)$ ancilla registers...

😳 **Regev:** $k = O(\sqrt{n})$, each pebble is n qubits $\Rightarrow O(n^{3/2})$ qubits

The general problem

General problem: How to do $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$ efficiently when f has many irreversible steps?

Consider an algorithm f with k steps f_i.

$$f(x) := f_k(f_{k-1}(\cdots f_2(f_1(x))\cdots))$$

Let $z_i = f_i(\cdots)$. One quantum way to do it:

$|x\rangle$ $|f(x)\rangle$

😃 Total quantum steps: $2k-1$ (optimal)

😢 Requires $O(k)$ ancilla registers...

😳 **Regev:** $k = O(\sqrt{n})$, each pebble is n qubits $\Rightarrow O(n^{3/2})$ qubits

Is it possible to do better?

Pebble games for reversible computation

Bennett ’89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$

Pebble games for reversible computation

Bennett ’89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)

$|x\rangle \quad |z_1\rangle \quad |z_2\rangle \quad |z_3\rangle \quad |f(x)\rangle$

Pebble games for reversible computation

Bennett ’89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble

$$|x\rangle \quad |z_1\rangle \quad |z_2\rangle \quad |z_3\rangle \quad |f(x)\rangle$$

Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

$|x\rangle \quad |z_1\rangle \quad |z_2\rangle \quad |z_3\rangle \quad |f(x)\rangle$

Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

$$|x\rangle \quad |z_1\rangle \quad |z_2\rangle \quad |z_3\rangle \quad |f(x)\rangle$$

Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$

Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$

Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$

Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

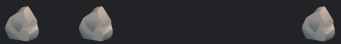
$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$

Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$



Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

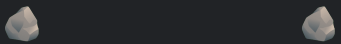
$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$


Pebble games for reversible computation

Bennett '89 introduced pebble games:

- Placing a pebble → computing a value (uses new space)
- Removing a pebble → uncomputing that value (frees space)
- **Rule:** Can only pebble/unpebble if value to the left has a pebble
- **Goal:** place pebble on $f(x)$; leave no pebbles on z_i

$|x\rangle \quad |z_1\rangle \quad |z_2\rangle \quad |z_3\rangle \quad |f(x)\rangle$



Space usage (max # pebbles): $O(k)$ registers
Time cost (# of steps): $2k - 1$ steps (optimal)

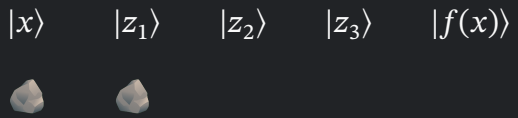
Using less space: a recursive strategy

1. Pebble from start to $k/2$

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$

Using less space: a recursive strategy

1. Pebble from start to $k/2$

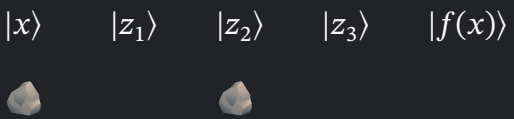


Using less space: a recursive strategy

1. Pebble from start to $k/2$

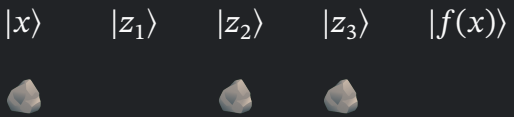
Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k



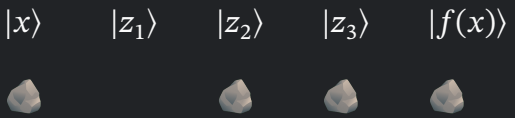
Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k



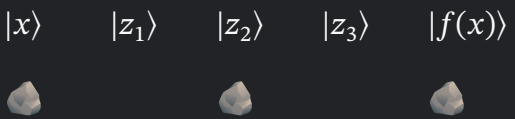
Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k



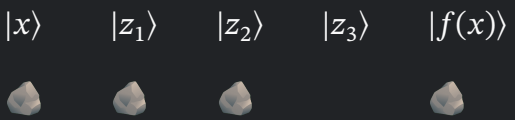
Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$



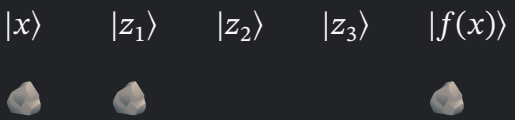
Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$



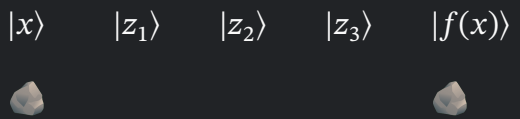
Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$



Using less space: a recursive strategy

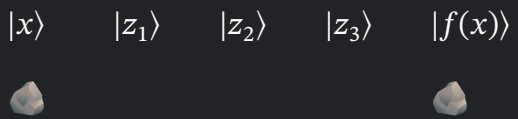
1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$



Space usage (max # pebbles): $O(\log k)$ registers 😃

Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

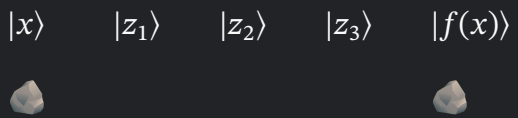


Space usage (max # pebbles): $O(\log k)$ registers 😃

Time cost (# of steps): $T(k) = 3T(k/2)$

Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

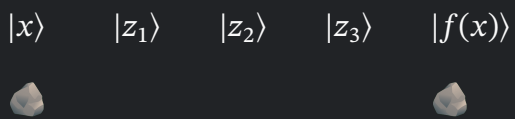


Space usage (max # pebbles): $O(\log k)$ registers 😃

Time cost (# of steps): $O(k^{\log_2 3}) \approx O(k^{1.58\cdots})$ steps 😢

Using less space: a recursive strategy

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$



Space usage (max # pebbles): $O(\log k)$ registers 😃

Time cost (# of steps): $O(k^{\log_2 3}) \approx O(k^{1.58\cdots})$ steps 😢

Generalizing this strategy: $O(2^{1/\epsilon} \log k)$ pebbles, pebbling time $O(k^{1+\epsilon})$ 😐

Roadmap

- Previous approaches to Regev
  - Original algorithm: fewer gates, but irreversible so more qubits
  - Making things reversible but concretely inefficient: Fibonacci arithmetic

- Pebbling!
  - Pebble games: accepting irreversibility
  - Previous work: adding parallelism *or* spookiness

- Our work: parallelism and spookiness join forces
  - Example
  - Pebbling results
  - Factoring results

Resource 1: parallelism

Setting:

- Pebble operations in parallel are allowed as long as they act on disjoint pebbles

Resource 1: parallelism

Setting:

- Pebble operations in parallel are allowed as long as they act on disjoint pebbles
- Time cost is the depth rather than the total number of pebbling operations

Resource 1: parallelism

Setting:

- Pebble operations in parallel are allowed as long as they act on disjoint pebbles
- Time cost is the depth rather than the total number of pebbling operations

Blocki, Holman, Lee ’22: this offers minor improvements!

Resource 1: parallelism

Setting:

- Pebble operations in parallel are allowed as long as they act on disjoint pebbles
- Time cost is the depth rather than the total number of pebbling operations

Blocki, Holman, Lee '22: this offers minor improvements!
Example: space usage $O(4^{\sqrt{\log k}})$ pebbles, $O(k)$ pebbling depth

Resource 1: parallelism

Setting:

- Pebble operations in parallel are allowed as long as they act on disjoint pebbles
- Time cost is the depth rather than the total number of pebbling operations

Blocki, Holman, Lee '22: this offers minor improvements!
Example: space usage $O(4^{\sqrt{\log k}})$ pebbles, $O(k)$ pebbling depth

Depth is optimal up to a constant factor!

Resource 2: quantum spookiness

“Spooky Pebble Games and Irreversible Uncomputation”
algassert.com/post/1905

Resource 2: quantum spookiness

"Spooky Pebble Games and Irreversible Uncomputation"
algassert.com/post/1905

Measurement-based uncomputation

Given an intermediate value

$$|z_i\rangle$$

Resource 2: quantum spookiness

"Spooky Pebble Games and Irreversible Uncomputation"
algassert.com/post/1905

Measurement-based uncomputation

Given an intermediate value

$$|z_i\rangle$$

what if we apply $H^{\otimes n}$ and then measure it?

Resource 2: quantum spookiness

"Spooky Pebble Games and Irreversible Uncomputation"
algassert.com/post/1905

Measurement-based uncomputation

Given an intermediate value

$$|z_i\rangle$$

what if we apply $H^{\otimes n}$ and then measure it? Get phase

$$(-1)^{d \cdot z_i}$$

where d is classically-known measurement outcome.

Resource 2: quantum spookiness

"Spooky Pebble Games and Irreversible Uncomputation"
algassert.com/post/1905

Measurement-based uncomputation

Given an intermediate value

$$|z_i\rangle$$

what if we apply $H^{\otimes n}$ and then measure it? Get phase

$$(-1)^{d \cdot z_i}$$

where d is classically-known measurement outcome.

We've turned $|z_i\rangle$ into a ghost!

Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left

Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time

Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

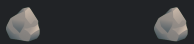
$|x\rangle$

Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

$|x\rangle$ $|z_1\rangle$



Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$

Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$

Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

$|x\rangle$ $(-1)^{d_1 \cdot z_1}$ $|z_2\rangle$

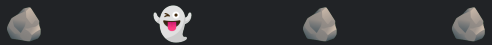


Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

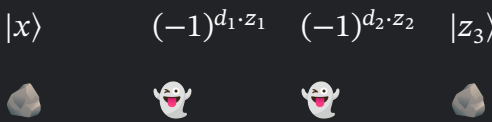
$|x\rangle$ $(-1)^{d_1 \cdot z_1}$ $|z_2\rangle$ $|z_3\rangle$



Spooky pebble games

Rules:

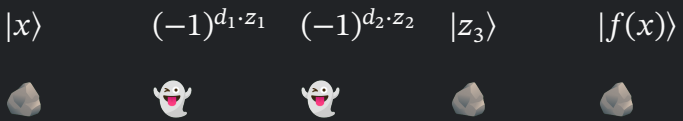
- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)



Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)



Spooky pebble games

Rules:

- can only place or remove a pebble if there is a pebble to its left
- can ghost a pebble at any time—but must exorcise later by placing a pebble again
- no parallelism allowed (for now)

$|x\rangle$ $(-1)^{d_1 \cdot z_1}$ $(-1)^{d_2 \cdot z_2}$ $(-1)^{d_3 \cdot z_3}$ $|f(x)\rangle$

We've been tempted too strongly by the power of dark magic...

Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts

$|x\rangle$

Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts

$|x\rangle$ $|z_1\rangle$


Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$

Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k

$|x\rangle$ $(-1)^{d_1 \cdot z_1}$ $|z_2\rangle$



Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k

$|x\rangle$ $(-1)^{d_1 \cdot z_1}$ $|z_2\rangle$ $|z_3\rangle$




Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k

$|x\rangle$ $(-1)^{d_1 \cdot z_1}$ $|z_2\rangle$ $|z_3\rangle$ $|f(x)\rangle$





Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

$|x\rangle$ $(-1)^{d_1 \cdot z_1}$ $|z_2\rangle$ $|f(x)\rangle$




Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

$|x\rangle$ $(-1)^{d_1 \cdot z_1} |z_1\rangle$ $|z_2\rangle$ $|f(x)\rangle$




Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

$|x\rangle$ $(-1)^{d_1 \cdot z_1} |z_1\rangle$ $|z_2\rangle$ $|f(x)\rangle$

Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|f(x)\rangle$




Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

$|x\rangle$ $|z_1\rangle$ $|f(x)\rangle$



Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

$|x\rangle$

$|f(x)\rangle$

Space: $O(\log k)$ pebbles 😃

Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

$|x\rangle$

$|f(x)\rangle$

Space: $O(\log k)$ pebbles 😃

Time cost (# steps): $T(k) = O(k) + 2T(k/2)$

Spooky pebble games

1. Blast straight to $k/2$, leaving ghosts
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

$|x\rangle$

$|f(x)\rangle$

Space: $O(\log k)$ pebbles 😃

Time cost (# steps): $O(k \log k)$ steps 😍

Why is spookiness so powerful?

Without spookiness

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

With spookiness

1. Blast straight from start to $k/2$
2. Pebble from $k/2$ to k
3. Remove pebble at $k/2$

Why is spookiness so powerful?

Without spookiness

1. Pebble from start to $k/2$
2. Pebble from $k/2$ to k: $T(k/2)$
3. Remove pebble at $k/2$: $T(k/2)$

With spookiness

1. Blast straight from start to $k/2$
2. Pebble from $k/2$ to k: $T(k/2)$
3. Remove pebble at $k/2$: $T(k/2)$

Why is spookiness so powerful?

Without spookiness

1. Pebble from start to $k/2$: $T(k/2)$
2. Pebble from $k/2$ to k: $T(k/2)$
3. Remove pebble at $k/2$: $T(k/2)$

With spookiness

1. Blast straight from start to $k/2$: $k/2$
2. Pebble from $k/2$ to k: $T(k/2)$
3. Remove pebble at $k/2$: $T(k/2)$

Why is spookiness so powerful?

Without spookiness

1. Pebble from start to $k/2$: $T(k/2)$
2. Pebble from $k/2$ to k: $T(k/2)$
3. Remove pebble at $k/2$: $T(k/2)$

With spookiness

1. Blast straight from start to $k/2$: $k/2$
2. Pebble from $k/2$ to k: $T(k/2)$
3. Remove pebble at $k/2$: $T(k/2)$

- **What spookiness does NOT enable:** moving pebbles faster, without leaving a mess

Why is spookiness so powerful?

Without spookiness

1. Pebble from start to $k/2$: $T(k/2)$
2. Pebble from $k/2$ to k: $T(k/2)$
3. Remove pebble at $k/2$: $T(k/2)$

With spookiness

1. Blast straight from start to $k/2$: $k/2$
2. Pebble from $k/2$ to k: $T(k/2)$
3. Remove pebble at $k/2$: $T(k/2)$

- **What spookiness does NOT enable:** moving pebbles faster, without leaving a mess
- **What spookiness does enable:** if we're okay with making a mess (to be cleaned up later), can move pebbles faster

Our work: parallelism and spookiness join forces

Roadmap

- Previous approaches to Regev
  - Original algorithm: fewer gates, but irreversible so more qubits
  - Making things reversible but concretely inefficient: Fibonacci arithmetic

- Pebbling!
  - Pebble games: accepting irreversibility
  - Previous work: adding parallelism *or* spookiness

- Our work: parallelism and spookiness join forces
  - Example
  - Pebbling results
  - Factoring results

Our work: parallel spooky pebble games

Absolutely optimal depth for a length-k pebble game: $2k - 1$ steps
How many pebbles do we need to achieve this?

Our work: parallel spooky pebble games

Absolutely optimal depth for a length-k pebble game: $2k - 1$ steps
How many pebbles do we need to achieve this?

- No spookiness, no parallelism: need $O(k)$ pebbles (trivial strategy)

Our work: parallel spooky pebble games

Absolutely optimal depth for a length-k pebble game: $2k - 1$ steps
How many pebbles do we need to achieve this?

- No spookiness, no parallelism: need $O(k)$ pebbles (trivial strategy)
- Yes spookiness, no parallelism: still need $O(k)$ pebbles

Our work: parallel spooky pebble games

Absolutely optimal depth for a length-k pebble game: $2k - 1$ steps
How many pebbles do we need to achieve this?

- No spookiness, no parallelism: need $O(k)$ pebbles (trivial strategy)
- Yes spookiness, no parallelism: still need $O(k)$ pebbles
- No spookiness, yes parallelism: need at least $O(4^{\sqrt{\log k}})$ pebbles

Our work: parallel spooky pebble games

Absolutely optimal depth for a length-k pebble game: $2k - 1$ steps
How many pebbles do we need to achieve this?

- No spookiness, no parallelism: need $O(k)$ pebbles (trivial strategy)
- Yes spookiness, no parallelism: still need $O(k)$ pebbles
- No spookiness, yes parallelism: need at least $O(4^{\sqrt{\log k}})$ pebbles

Can we do better using spookiness and parallelism in tandem?

Example: optimal-depth parallel spooky pebble game for $k = 12$

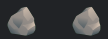
$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|z_4\rangle$ $|z_5\rangle$ $|z_6\rangle$ $|z_7\rangle$ $|z_8\rangle$ $|z_9\rangle$ $|z_{10}\rangle$ $|z_{11}\rangle$ $|f(x)\rangle$

Step number: 0

Max. pebble count: 1

Example: optimal-depth parallel spooky pebble game for $k = 12$

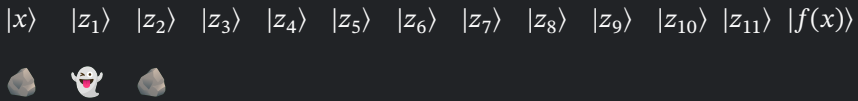
$|x\rangle$ $|z_1\rangle$ $|z_2\rangle$ $|z_3\rangle$ $|z_4\rangle$ $|z_5\rangle$ $|z_6\rangle$ $|z_7\rangle$ $|z_8\rangle$ $|z_9\rangle$ $|z_{10}\rangle$ $|z_{11}\rangle$ $|f(x)\rangle$



Step number: 1

Max. pebble count: 2

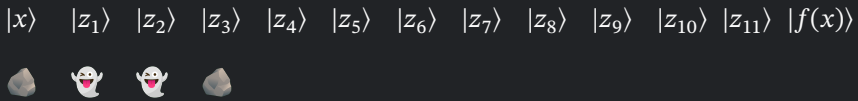
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 2

Max. pebble count: 2

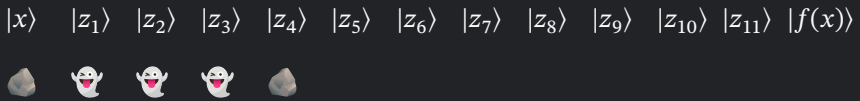
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 3

Max. pebble count: 2

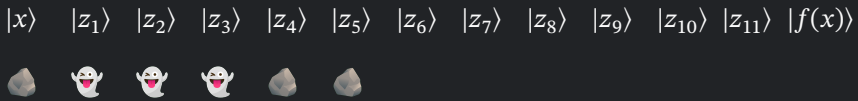
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 4

Max. pebble count: 2

Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 5

Max. pebble count: 3

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 6

Max. pebble count: 3

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 7

Max. pebble count: 3

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 8

Max. pebble count: 4

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 9

Max. pebble count: 4

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 10

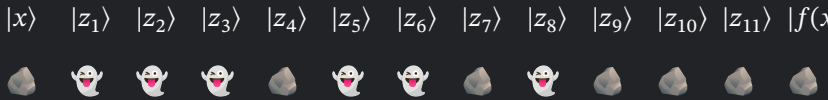
Max. pebble count: 5

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 11

Max. pebble count: 6

Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 12

Max. pebble count: 7

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 13

Max. pebble count: 7

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 14

Max. pebble count: 7

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 15

Max. pebble count: 7

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 16

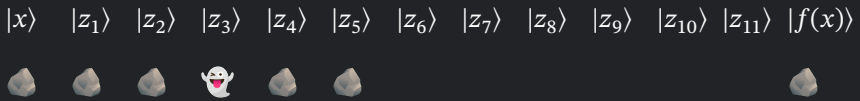
Max. pebble count: 7

Example: optimal-depth parallel spooky pebble game for $k = 12$

Step number: 17

Max. pebble count: 7

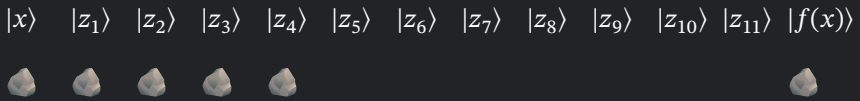
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 18

Max. pebble count: 7

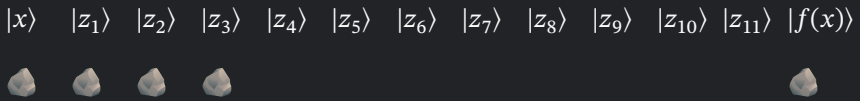
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 19

Max. pebble count: 7

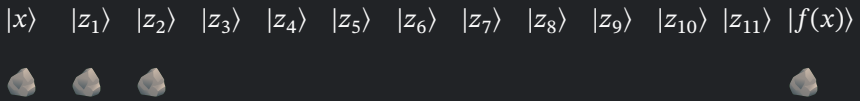
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 20

Max. pebble count: 7

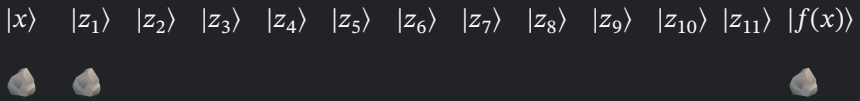
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 21

Max. pebble count: 7

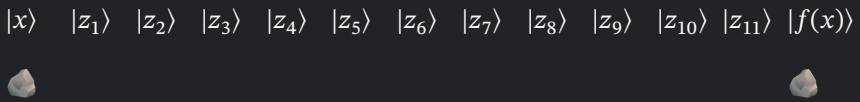
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 22

Max. pebble count: 7

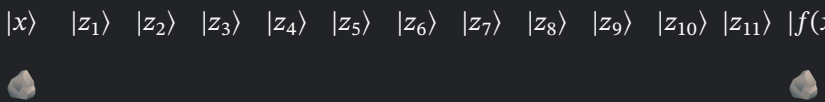
Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: 23

Max. pebble count: 7

Example: optimal-depth parallel spooky pebble game for $k = 12$



Step number: $23 = 2k - 1$, which is optimal 🤩

Max. pebble count: 7

Roadmap

- Previous approaches to Regev
  - Original algorithm: fewer gates, but irreversible so more qubits
  - Making things reversible but concretely inefficient: Fibonacci arithmetic

- Pebbling!
  - Pebble games: accepting irreversibility
  - Previous work: adding parallelism *or* spookiness

- Our work: parallelism and spookiness join forces
  - Example
  - Pebbling results
  - Factoring results

Our results

Explicit construction

Our results

Explicit construction

- Achieves optimal depth $2k - 1$

Our results

Explicit construction

- Achieves optimal depth $2k - 1$
- Our parallel construction uses only $2.47 \log k$ pebbles

Our results

Explicit construction

- Achieves optimal depth $2k - 1$
- Our parallel construction uses only $2.47 \log k$ pebbles
  - **Recall**: achieving optimal depth without parallelism required $O(k)$ pebbles, and without spookiness required $O(4^{\sqrt{\log k}})$ pebbles

Our results

Explicit construction

- Achieves optimal depth $2k - 1$
- Our parallel construction uses only $2.47 \log k$ pebbles
  - **Recall**: achieving optimal depth without parallelism required $O(k)$ pebbles, and without spookiness required $O(4^{\sqrt{\log k}})$ pebbles

Automated search

Our results

Explicit construction

- Achieves optimal depth $2k - 1$
- Our parallel construction uses only $2.47 \log k$ pebbles
  - **Recall**: achieving optimal depth without parallelism required $O(k)$ pebbles, and without spookiness required $O(4^{\sqrt{\log k}})$ pebbles

Automated search

- Highly optimized **A* search** written in Julia

Our results

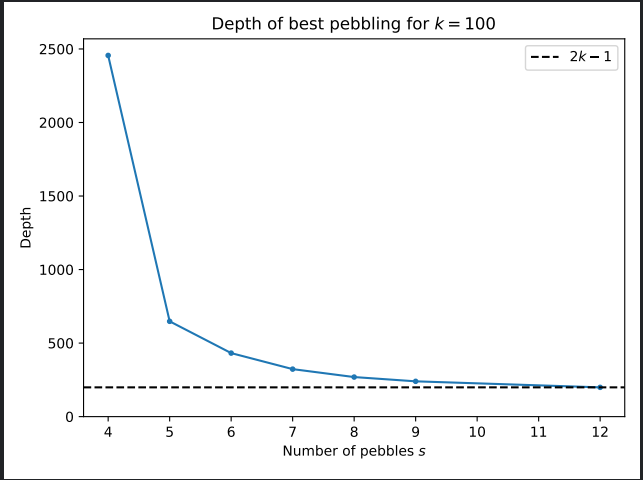
Explicit construction

- Achieves optimal depth $2k - 1$
- Our parallel construction uses only $2.47 \log k$ pebbles
  - **Recall**: achieving optimal depth without parallelism required $O(k)$ pebbles, and without spookiness required $O(4^{\sqrt{\log k}})$ pebbles

Automated search

- Highly optimized **A* search** written in Julia
- Finds lowest-depth solution for *any* fixed number of pebbles s and length k

Numerical results



Pebbling summary

To pebble to length k:

Algorithm	Parallelism?	Spookiness?	Space	Depth
Bennett ’89	N	N	$O(2^{1/\epsilon} \cdot \log k)$	$O(k^{1+\epsilon})$

Pebbling summary

To pebble to length k:

Algorithm	Parallelism?	Spookiness?	Space	Depth
Bennett ’89	N	N	$O(2^{1/\epsilon} \cdot \log k)$	$O(k^{1+\epsilon})$

Pebbling summary

To pebble to length k:

Algorithm	Parallelism?	Spookiness?	Space	Depth
Bennett ’89	N	N	$O(2^{1/\epsilon} \cdot \log k)$	$O(k^{1+\epsilon})$
BHL’22	Y	N	$O(4^{\sqrt{\log k}})$	$O(k)$

Pebbling summary

To pebble to length k:

Algorithm	Parallelism?	Spookiness?	Space	Depth
Bennett ’89	N	N	$O(2^{1/\epsilon} \cdot \log k)$	$O(k^{1+\epsilon})$
BHL’22	Y	N	$O(4^{\sqrt{\log k}})$	$O(k)$
Gidney ’19, KSS’25	N	Y	$\approx \log k$	$\approx k \log k$

Pebbling summary

To pebble to length k:

Algorithm	Parallelism?	Spookiness?	Space	Depth
Bennett ’89	N	N	$O(2^{1/\epsilon} \cdot \log k)$	$O(k^{1+\epsilon})$
BHL’22	Y	N	$O(4^{\sqrt{\log k}})$	$O(k)$
Gidney ’19, KSS’25	N	Y	$\approx \log k$	$\approx k \log k$
Our work	Y	Y	$\approx 2.47 \log k$	$\approx 2k$

Roadmap

- Previous approaches to Regev
  - Original algorithm: fewer gates, but irreversible so more qubits
  - Making things reversible but concretely inefficient: Fibonacci arithmetic

- Pebbling!
  - Pebble games: accepting irreversibility
  - Previous work: adding parallelism *or* spookiness

- Our work: parallelism and spookiness join forces
  - Example
  - Pebbling results
  - Factoring results

Recap

Hope seemed lost...

Recap

Hope seemed lost...

Recap

But then Bennett broke through the reversibility barrier with pebbling!

Recap

And Blocki, Holman, and Lee parallelized this!

Recap

Using spookiness, Gidney, Kornerup, Sadun, and Soloveichik brought even more hope

Recap

Using spookiness, Gidney, Kornerup, Sadun, and Soloveichik brought even more hope

Recap

And a ragtag crew of friends came along to put it all together 😃

Recap

Pebblers! Assemble.

Recap

Pebblers! ~~Assemble~~ Estimate concrete costs for RSA-2048 and RSA-4096.

Factoring results

For factoring 4096-bit RSA:

- all metrics counted in n-bit multiplications and evaluated per shot
- for previous estimates see: Ekerå + Gärtner, arXiv:2405.14381

Factoring results

For factoring 4096-bit RSA:

- all metrics counted in n-bit multiplications and evaluated per shot
- for previous estimates see: Ekerå + Gärtner, arXiv:2405.14381

Circuit	Depth	Mults.	Qubits
Previous Regev + Fibonacci	700	1400	$\geq 13n$

Factoring results

For factoring 4096-bit RSA:

- all metrics counted in n-bit multiplications and evaluated per shot
- for previous estimates see: Ekerå + Gärtner, arXiv:2405.14381

Circuit	Depth	Mults.	Qubits
Previous Regev + Fibonacci	700	1400	$\geq 13n$
Standard Shor (e.g. Gidney + Ekerå '19)	444	444	$2n - 3n$

Factoring results

For factoring 4096-bit RSA:

- all metrics counted in n-bit multiplications and evaluated per shot
- for previous estimates see: Ekerå + Gärtner, arXiv:2405.14381

Circuit	Depth	Mults.	Qubits
Previous Regev + Fibonacci	700	1400	$\geq 13n$
Standard Shor (e.g. Gidney + Ekerå '19)	444	444	$2n - 3n$
Standard Shor + some parallelism	186	896	$\approx 14n$

Factoring results

For factoring 4096-bit RSA:

- all metrics counted in n-bit multiplications and evaluated per shot
- for previous estimates see: Ekerå + Gärtner, arXiv:2405.14381

Circuit	Depth	Mults.	Qubits
Previous Regev + Fibonacci	700	1400	$\geq 13n$
Standard Shor (e.g. Gidney + Ekerå '19)	444	444	$2n - 3n$
Standard Shor + some parallelism	186	896	$\approx 14n$
Our results (more pebbles)	200	381	$15n$

Factoring results

For factoring 4096-bit RSA:

- all metrics counted in n-bit multiplications and evaluated per shot
- for previous estimates see: Ekerå + Gärtner, arXiv:2405.14381

Circuit	Depth	Mults.	Qubits
Previous Regev + Fibonacci	700	1400	$\geq 13n$
Standard Shor (e.g. Gidney + Ekerå '19)	444	444	$2n - 3n$
Standard Shor + some parallelism	186	896	$\approx 14n$
Our results (more pebbles)	200	381	$15n$
Our results (fewer pebbles)	561	600	$8n$

Note: this is shamelessly focusing on **per-shot depth/gates**, our best metrics...

Takeaways

- Concretely the best Regev variant to date, but doesn’t quite match up to Shor (even in per-shot depth)

Takeaways

- Concretely the best Regev variant to date, but doesn't quite match up to Shor (even in per-shot depth)
- Additionally, qubit count, not depth, seems most important in *near-term*

Takeaways

- Concretely the best Regev variant to date, but doesn't quite match up to Shor (even in per-shot depth)
- Additionally, qubit count, not depth, seems most important in *near-term*
  - So this will likely not be the way we factor the first cryptographic-size integers

Takeaways

- Concretely the best Regev variant to date, but doesn't quite match up to Shor (even in per-shot depth)
- Additionally, qubit count, not depth, seems most important in *near-term*
  - So this will likely not be the way we factor the first cryptographic-size integers
- "This paper creates **slack**" for further Regev modifications in the future

Takeaways

- Concretely the best Regev variant to date, but doesn't quite match up to Shor (even in per-shot depth)
- Additionally, qubit count, not depth, seems most important in *near-term*
  - So this will likely not be the way we factor the first cryptographic-size integers
- "This paper creates **slack**" for further Regev modifications in the future
- Efficiently evaluating sequential algorithms is **widely applicable** beyond factoring

Takeaways

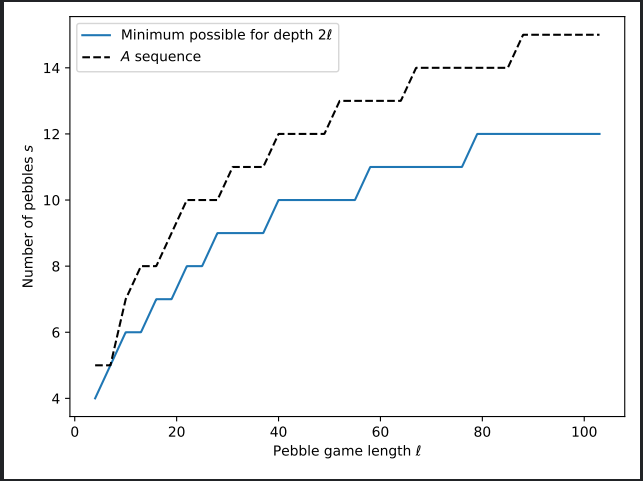
- Concretely the best Regev variant to date, but doesn't quite match up to Shor (even in per-shot depth)
- Additionally, qubit count, not depth, seems most important in *near-term*
  - So this will likely not be the way we factor the first cryptographic-size integers
- "This paper creates **slack**" for further Regev modifications in the future
- Efficiently evaluating sequential algorithms is **widely applicable** beyond factoring
- **Generalization:** parallel spooky pebbling on arbitrary graphs

Thank you!

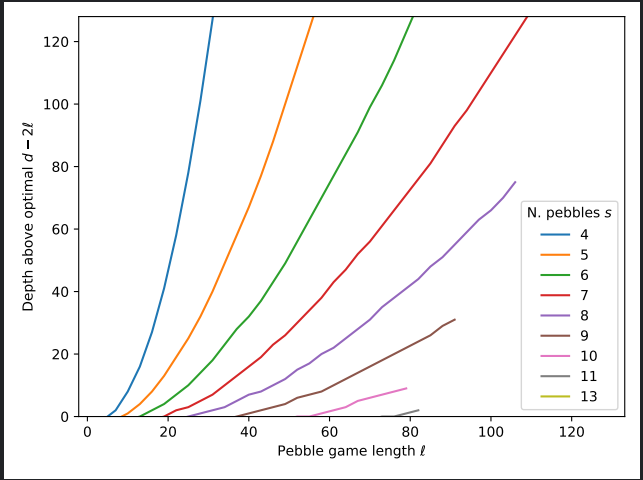
arXiv:2510.08432

Backup

Numerical results



Numerical results



Making Shor reversible

Break up x into its individual bits x_i:

$a^x \bmod N$

Making Shor reversible

Break up x into its individual bits x_i:

$$a^x \bmod N = a^{\sum_i 2^i x_i} \bmod N$$

Making Shor reversible

Break up x into its individual bits x_i:

$$a^x \bmod N = a^{\sum_i 2^i x_i} \bmod N$$
$$= \prod_i a^{2^i x_i} \bmod N$$

Making Shor reversible

Break up x into its individual bits x_i:

$$
\begin{aligned}
a^x \bmod N &= a^{\sum_i 2^i x_i} \bmod N \\
&= \prod_i a^{2^i x_i} \bmod N \\
&= \prod_i \left(a^{2^i}\right)^{x_i} \bmod N
\end{aligned}
$$

Making Shor reversible

Break up x into its individual bits x_i:

$$
\begin{aligned}
a^x \bmod N &= a^{\sum_i 2^i x_i} \bmod N \\
&= \prod_i a^{2^i x_i} \bmod N \\
&= \prod_i \left(a^{2^i}\right)^{x_i} \bmod N \\
&= \prod_i c_i^{x_i} \bmod N
\end{aligned}
$$

where classical $c_i = a^{2^i} \bmod N$

Making Shor reversible

Break up x into its individual bits x_i:

$$
\begin{aligned}
a^x \bmod N &= a^{\sum_i 2^i x_i} \bmod N \\
&= \prod_i a^{2^i x_i} \bmod N \\
&= \prod_i \left(a^{2^i}\right)^{x_i} \bmod N \\
&= \prod_i c_i^{x_i} \bmod N
\end{aligned}
$$

$|1\rangle$

where classical $c_i = a^{2^i} \bmod N$

Making Shor reversible

Break up x into its individual bits x_i:

$$
\begin{aligned}
a^x \bmod N &= a^{\sum_i 2^i x_i} \bmod N \\
&= \prod_i a^{2^i x_i} \bmod N \\
&= \prod_i \left(a^{2^i}\right)^{x_i} \bmod N \\
&= \prod_i c_i^{x_i} \bmod N
\end{aligned}
$$

$$|1\rangle \rightarrow |c_0^{x_0}\rangle$$

where classical $c_i = a^{2^i} \bmod N$

Making Shor reversible

Break up x into its individual bits x_i:

$$
\begin{aligned}
a^x \bmod N &= a^{\sum_i 2^i x_i} \bmod N \\
&= \prod_i a^{2^i x_i} \bmod N \\
&= \prod_i \left(a^{2^i}\right)^{x_i} \bmod N \\
&= \prod_i c_i^{x_i} \bmod N
\end{aligned}
$$

$$|1\rangle \rightarrow |c_0^{x_0}\rangle \rightarrow |c_1^{x_1} c_0^{x_0}\rangle$$

where classical $c_i = a^{2^i} \bmod N$

Making Shor reversible

Break up x into its individual bits x_i:

$$
\begin{aligned}
a^x \bmod N &= a^{\sum_i 2^i x_i} \bmod N \\
&= \prod_i a^{2^i x_i} \bmod N \\
&= \prod_i \left(a^{2^i}\right)^{x_i} \bmod N \\
&= \prod_i c_i^{x_i} \bmod N
\end{aligned}
$$

$$|1\rangle \rightarrow |c_0^{x_0}\rangle \rightarrow |c_1^{x_1} c_0^{x_0}\rangle \rightarrow |c_2^{x_2} c_1^{x_1} c_0^{x_0}\rangle$$

where classical $c_i = a^{2^i} \bmod N$

Making Shor reversible

Break up x into its individual bits x_i:

$$
\begin{aligned}
a^x \bmod N &= a^{\sum_i 2^i x_i} \bmod N \\
&= \prod_i a^{2^i x_i} \bmod N \\
&= \prod_i \left(a^{2^i}\right)^{x_i} \bmod N \\
&= \prod_i c_i^{x_i} \bmod N
\end{aligned}
$$

$$|1\rangle \rightarrow |c_0^{x_0}\rangle \rightarrow |c_1^{x_1} c_0^{x_0}\rangle \rightarrow |c_2^{x_2} c_1^{x_1} c_0^{x_0}\rangle \rightarrow \cdots$$

where classical $c_i = a^{2^i} \bmod N$

Making Shor reversible

Break up x into its individual bits x_i:

$$
\begin{aligned}
a^x \bmod N &= a^{\sum_i 2^i x_i} \bmod N \\
&= \prod_i a^{2^i x_i} \bmod N \\
&= \prod_i \left(a^{2^i}\right)^{x_i} \bmod N \\
&= \prod_i c_i^{x_i} \bmod N
\end{aligned}
$$

where classical $c_i = a^{2^i} \bmod N$

$$|1\rangle \to |c_0^{x_0}\rangle \to |c_1^{x_1} c_0^{x_0}\rangle \to |c_2^{x_2} c_1^{x_1} c_0^{x_0}\rangle \to \cdots$$

Each iteration is a controlled multiplication by classical c_i—which is reversible!