Recent advances in quantum factoring

Greg Kahanamoku-Meyer

May 26, 2025

> Can the reader say what two numbers multiplied together will produce the number 8,616,460,799?
>
> I think it unlikely that anyone but myself will ever know.
>
> -William Stanley Jevons, 1874

Why work on factoring?

Why work on factoring?

Noisy qubits needed to factor 2048-bit RSA:

[Gidney + Ekera ’19] $\sim$ 20 million

Why work on factoring?

Noisy qubits needed to factor 2048-bit RSA:

[Gidney + Ekera '19] $\sim$ **20 million**

[Gidney last week] **< 1 million**

Why work on factoring?

Noisy qubits needed to factor 2048-bit RSA:

[Gidney + Ekera ’19] $\sim$ **20 million**

[Gidney last week] **< 1 million**

Two hypothetical futures:

Why work on factoring?

Noisy qubits needed to factor 2048-bit RSA:

[Gidney + Ekera '19] $\sim$ **20 million**

[Gidney last week] **< 1 million**

Two hypothetical futures:

Future A

2027: Circuit discovered needing only 50,000 qubits

2032: Device with 50,000 qubits constructed

Why work on factoring?

Noisy qubits needed to factor 2048-bit RSA:

[Gidney + Ekera '19] $\sim$ **20 million**

[Gidney last week] **< 1 million**

Two hypothetical futures:

Future A

2027: Circuit discovered needing only 50,000 qubits

2032: Device with 50,000 qubits constructed

Future B

2032: Device with 50,000 qubits constructed

2033: Circuit discovered needing only 50,000 qubits

Why work on factoring?

Noisy qubits needed to factor 2048-bit RSA:

[Gidney + Ekera '19] $\sim$ **20 million**

[Gidney last week] **< 1 million**

Two hypothetical futures:

Future A	Future B
2027: Circuit discovered needing only 50,000 qubits	**2032:** Device with 50,000 qubits constructed
2032: Device with 50,000 qubits constructed	**2033:** Circuit discovered needing only 50,000 qubits

I want to live in Future A!

Why work on factoring?

Other less important reasons:

Why work on factoring?

Other less important reasons:

- Factoring makes a really straightforward efficiently-verifiable proof of quantumness

Why work on factoring?

Other less important reasons:

- Factoring makes a really straightforward efficiently-verifiable proof of quantumness
- The math is really fun

What should we optimize for?

What should we optimize for?

2019 Greg

"Let's make a proof of quantumness so efficient we can run it on physical qubits!"

What should we optimize for?

2019 Greg

"Let's make a proof of quantumness so efficient we can run it on physical qubits!"

2025 Greg

"We will need quantum error correction to do any nontrivial cryptography."

What should we optimize for?

2019 Greg

"Let's make a proof of quantumness so efficient we can run it on physical qubits!"

2025 Greg

"We will need quantum error correction to do any nontrivial cryptography."

Fact: Logical error rate is exponential with code distance.

What should we optimize for?

2019 Greg

"Let's make a proof of quantumness so efficient we can run it on physical qubits!"

2025 Greg

"We will need quantum error correction to do any nontrivial cryptography."

Fact: Logical error rate is exponential with code distance.

Imagine two algorithms, A and B.
B uses half as many qubits as A, but 10 times as many gates.

Which will we be able to run first?

What should we optimize for?

2019 Greg

"Let's make a proof of quantumness so efficient we can run it on physical qubits!"

2025 Greg

"We will need quantum error correction to do any nontrivial cryptography."

Fact: Logical error rate is exponential with code distance.

Imagine two algorithms, A and B.
B uses half as many qubits as A, but 10 times as many gates.

Which will we be able to run first?

With some set number of physical qubits below EC threshold, can double code distance if we use algorithm B---exponential decrease in logical error rate!

What should we optimize for?

2019 Greg

"Let's make a proof of quantumness so efficient we can run it on physical qubits!"

2025 Greg

"We will need quantum error correction to do any nontrivial cryptography."

Fact: Logical error rate is exponential with code distance.

Imagine two algorithms, A and B.
B uses half as many qubits as A, but 10 times as many gates.

Which will we be able to run first?

Hot take: Right now, we should only really care about logical qubit count.

Should we ever care about gate count and depth?

Should we ever care about gate count and depth?

We should optimize for depth if:

We have very good devices and we care about wall time.

Should we ever care about gate count and depth?

We should optimize for depth if:

We have very good devices and we care about wall time.

We should optimize for gate count if:

We have pretty good devices that are limited by magic production.

All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors

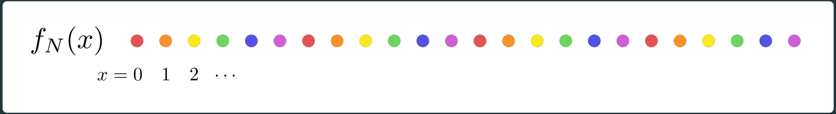
All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors

$\rightarrow$

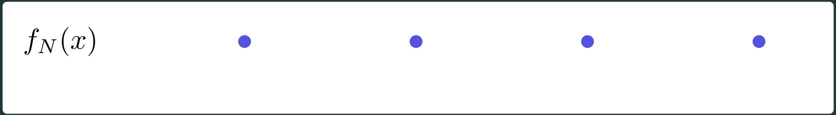
Generate

$$\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle\, |f_N(x)\rangle$$



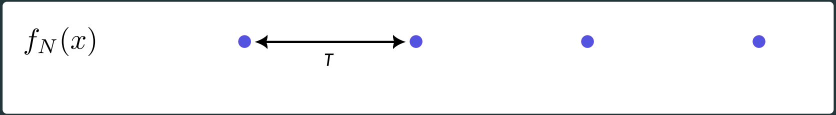
All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle\,|f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset



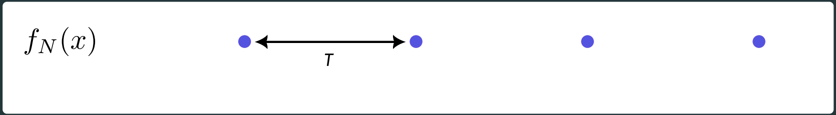
All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle\,|f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T

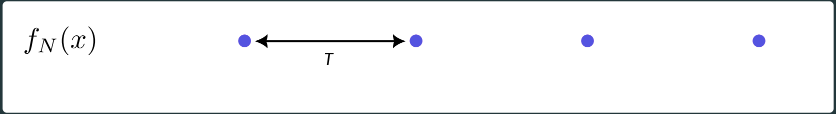


Qubit cost

- Input $|x\rangle$
- Output $|f_N(x)\rangle$
- Workspace

All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\text{poly}(T)} |x\rangle\, |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Qubit cost

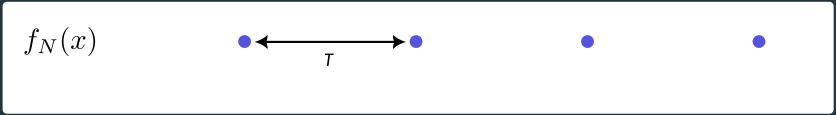
- Input $|x\rangle$
- Output $|f_N(x)\rangle$
- Workspace

Gate/depth cost

- Cost of $|f_N(x)\rangle$

All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle\,|f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Qubit cost

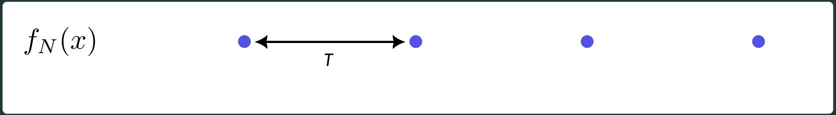
- Input $|x\rangle$: $O(\log T)$ qubits
- Output $|f_N(x)\rangle$
- Workspace

Gate/depth cost

- Cost of $|f_N(x)\rangle$

All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle\, |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Qubit cost

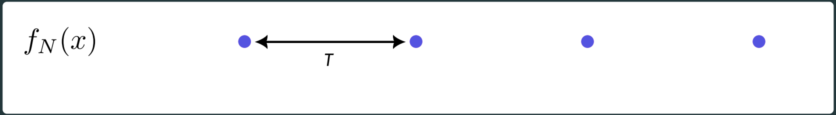
- Input $|x\rangle$: $O(\log T)$ qubits
- Output $|f_N(x)\rangle$: $O(\log T)$ qubits
- Workspace

Gate/depth cost

- Cost of $|f_N(x)\rangle$

All (poly-time) quantum factoring algorithms: period finding

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Qubit cost

- Input $|x\rangle$: $O(\log T)$ qubits
- Output $|f_N(x)\rangle$: $O(\log T)$ qubits
- Workspace: ???

Gate/depth cost

- Cost of $|f_N(x)\rangle$: ???

Shor’s algorithm

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\text{poly}(T)} |x\rangle\,|f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T

Function: $f_N(x) = a^x \bmod N$ **Period:** $T = \operatorname{ord}_N(a) \sim \mathcal{O}(N)$

Shor's algorithm

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\text{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T

Function: $f_N(x) = a^x \bmod N$

Period: $T = \mathrm{ord}_N(a) \sim \mathcal{O}(N)$

Let $n = \lceil \log N \rceil$:

Qubit cost

- Input $|x\rangle$: $2n$ qubits
- Output $|f_N(x)\rangle$: n qubits
- Workspace: ???

Gate/depth cost

- Cost of $|f_N(x)\rangle$: ???

Reducing input to 1 (reused) qubit

Key observation (Zalka '98, and others):

$$a^x \bmod N = \prod_i c_i^{x_i} \bmod N$$

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

Reducing input to 1 (reused) qubit

Key observation (Zalka '98, and others):

$$a^x \bmod N = \prod_i c_i^{x_i} \bmod N$$

where $c_i = a^{2^i} \bmod N$. **Reuse one qubit for all bits of x.**

Reducing input to 1 (reused) qubit

Key observation (Zalka '98, and others):

$$a^x \bmod N = \prod_i c_i^{x_i} \bmod N$$

where $c_i = a^{2^i} \bmod N$. **Reuse one qubit for all bits of x.**

Key operation: $|x_i\rangle\,|w\rangle \to |x_i\rangle\,\left|c_i^{x_i} w\right\rangle$

Reducing input to 1 (reused) qubit

Key observation (Zalka '98, and others):

$$a^x \bmod N = \prod_i c_i^{x_i} \bmod N$$

where $c_i = a^{2^i} \bmod N$. **Reuse one qubit for all bits of x.**

Key operation: $|x_i\rangle\,|w\rangle \to |x_i\rangle\,\left|c_i^{x_i} w\right\rangle$

Qubit cost

- Input $|x\rangle$: **1 qubit** (reused)
- Output $|f_N(x)\rangle$: n qubits
- Workspace: mult. workspace

Gate/depth cost

- Cost of $|f_N(x)\rangle$:
  - $2n$ multiplications

Reducing input to 1 (reused) qubit

Key observation (Zalka '98, and others):

$$a^x \mod N = \prod_i c_i^{x_i} \mod N$$

where $c_i = a^{2^i} \mod N$. **Reuse one qubit for all bits of x.**

Key operation: $|x_i\rangle |w\rangle \to |x_i\rangle \left|c_i^{x_i} w\right\rangle$

Qubit cost

- Input $|x\rangle$: **1 qubit** (reused)
- Output $|f_N(x)\rangle$: n qubits
- Workspace: mult. workspace

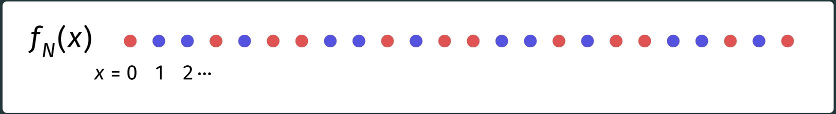
Gate/depth cost

- Cost of $|f_N(x)\rangle$:
  - $2n$ multiplications

[Gidney + Ekera '19]: Factoring 2048-bit $N = pq$ in 8 hours with 20 million physical qubits

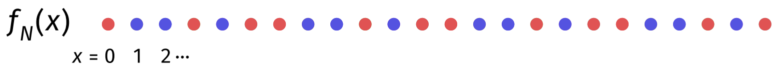
Reducing output to 1 qubit: what if $f_N(x) : \{0,1\}^* \rightarrow \{0,1\}$?

Find some $f_N(x)$ w/ period T, where T can be used to find factors



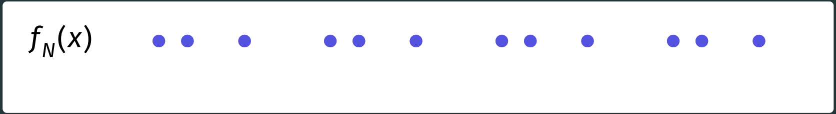
Reducing output to 1 qubit: what if $f_N(x) : \{0,1\}^* \to \{0,1\}$?

Find some $f_N(x)$ w/ period T, where T can be used to find factors $\rightarrow$ Generate $\sum_{x=0}^{\text{poly}(T)} |x\rangle\,|f_N(x)\rangle$



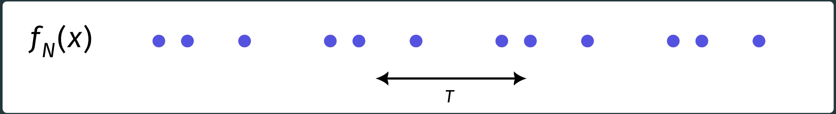
Reducing output to 1 qubit: what if $f_N(x) : \{0,1\}^* \to \{0,1\}$?

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle\,|f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset



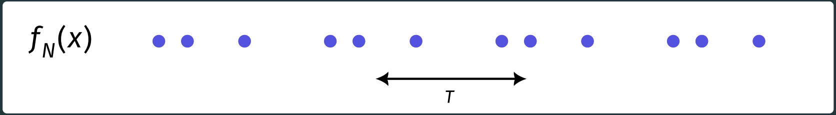
Reducing output to 1 qubit: what if $f_N(x) : \{0,1\}^* \to \{0,1\}$?

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Reducing output to 1 qubit: what if $f_N(x) : \{0,1\}^* \to \{0,1\}$?

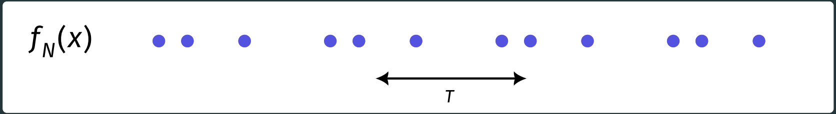
Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\text{poly}(T)} |x\rangle\,|f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Is this going to actually help (or even work)?

Reducing output to 1 qubit: what if $f_N(x) : \{0,1\}^* \to \{0,1\}$?

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T

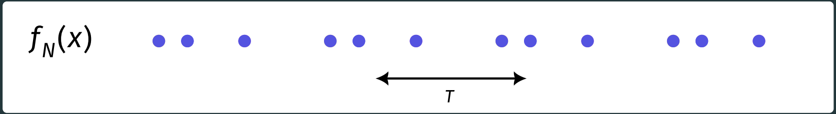


Is this going to actually help (or even work)?

- There could be *smaller* periods than T [Hales + Hallgren '00]

Reducing output to 1 qubit: what if $f_N(x) : \{0,1\}^* \to \{0,1\}$?

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\text{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T

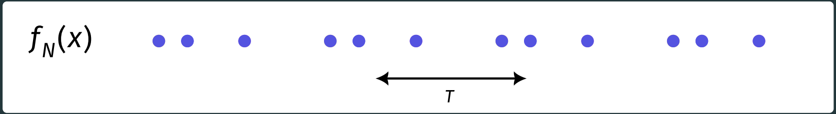


Is this going to actually help (or even work)?

- There could be *smaller* periods than T [Hales + Hallgren '00]
- Need $\sim \log T$ qubits for *input*

Reducing output to 1 qubit: what if $f_N(x) : \{0,1\}^* \to \{0,1\}$?

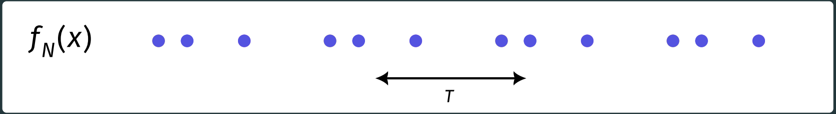
Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\text{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Is this going to actually help (or even work)?

- There could be *smaller* periods than T [Hales + Hallgren '00]
- Need $\sim \log T$ qubits for *input*
- Need workspace to compute $f_N(x)$

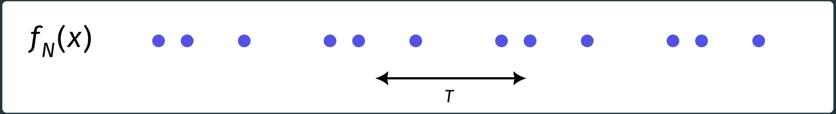
Avoid periods smaller than T



[May + Schlieper '22]: For some hash function $h : \{0,1\}^n \to \{0,1\}$, use

$$f_N(x) = h(a^x \bmod N)$$

Avoid periods smaller than T



[May + Schlieper '22]: For some hash function $h : \{0,1\}^n \to \{0,1\}$, use

$$f_N(x) = h(a^x \bmod N)$$

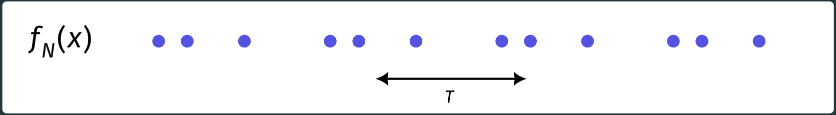
Qubit cost

- Input $|x\rangle$: 2n qubits
- Output $|f_N(x)\rangle$: **1 qubit**
- Workspace: $O(n)$ qubits

Gate/depth cost

- Cost of $|f_N(x)\rangle$:
  - $2n$ multiplications

Reduce period, reduce qubits for input



[Ekera + Hastad ’17]: Can factor $N = pq$ via discrete log with period $O(\sqrt{N})$

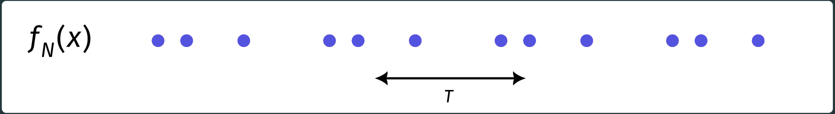
<u>Qubit cost</u>

- Input $|x\rangle$: **n/2 qubits**
- Output $|f_N(x)\rangle$: **1 qubit**
- Workspace: $O(n)$ qubits

<u>Gate/depth cost</u>

- Cost of $|f_N(x)\rangle$:
  - $n/2$ multiplications

Reduce workspace



[Chevignard et al. '24]: Used residue number system to cut workspace to $\sim O(\log n)$ qubits

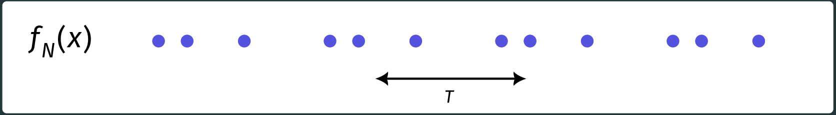
Qubit cost

- Input $|x\rangle$: **n/2 qubits**
- Output $|f_N(x)\rangle$: **1 qubit**
- Workspace: **$O(\log n)$ qubits**

Gate/depth cost

- Cost of $|f_N(x)\rangle$:
  - $\sim$ 2 trillion Toffoli gates

Putting it all together



[Gidney last week]: Arithmetic + fault tolerance optimizations

Factor 2048-bit $N = pq$ using < 1 million physical qubits in ~ 1 week

Some things I've worked on

A sublinear space and depth factoring algorithm

For integers $N = P^2 Q$:
Gate count $\tilde{O}(n)$
Qubits and depth $\tilde{O}(n^{2/3})$

GDKM, S. Ragavan, V. Vaikuntanathan, K. Van Kirk. arXiv:2412.12558

Some things I've worked on

A sublinear space and depth factoring algorithm

For integers $N = P^2 Q$:
Gate count $\tilde{O}(n)$
Qubits and depth $\tilde{O}(n^{2/3})$

GDKM, S. Ragavan, V. Vaikuntanathan, K. Van Kirk. arXiv:2412.12558

Log-depth "optimistic" QFT with no ancillas

Error bounded by ϵ on all but $O(\epsilon) \cdot 2^n$ basis states

GDKM, J. Blue, T. Bergamaschi, C. Gidney, I. Chuang. arXiv:2505.00701

Some things I've worked on

A sublinear space and depth factoring algorithm

For integers $N = P^2Q$:
Gate count $\tilde{O}(n)$
Qubits and depth $\tilde{O}(n^{2/3})$

GDKM, S. Ragavan, V. Vaikuntanathan, K. Van Kirk. arXiv:2412.12558

Log-depth "optimistic" QFT with no ancillas

Error bounded by ϵ on all but $O(\epsilon) \cdot 2^n$ basis states

GDKM, J. Blue, T. Bergamaschi, C. Gidney, I. Chuang. arXiv:2505.00701

Fast quantum integer multiplication

$O(n^{1+\varepsilon})$ gates
No ancilla qubits

GDKM, N. Yao. arXiv:2403.18006

Some things I've worked on

A sublinear space and depth factoring algorithm

For integers $N = P^2 Q$:
Gate count $\tilde{O}(n)$
Qubits and depth $\tilde{O}(n^{2/3})$

GDKM, S. Ragavan, V. Vaikuntanathan, K. Van Kirk. arXiv:2412.12558

Log-depth "optimistic" QFT with no ancillas

Error bounded by ϵ on all but $O(\epsilon) \cdot 2^n$ basis states

GDKM, J. Blue, T. Bergamaschi, C. Gidney, I. Chuang. arXiv:2505.00701

Fast quantum integer multiplication

$O(n^{1+\varepsilon})$ gates
No ancilla qubits

GDKM, N. Yao. arXiv:2403.18006

Shor's algorithm with:

$O(n^{2+\varepsilon})$ gates
$O(n^{1+\varepsilon})$ depth
$2n + O(n / \log n)$ total qubits

Factoring in sublinear space and depth

Greg Kahanamoku-Meyer

Seyoon Ragavan

Vinod Vaikuntanathan

Katherine Van Kirk

Asymptotic costs

Main result: Circuit for factoring n-bit integers $N = p^2 q$, with m-bit q

Schoolbook mult. + standard GCD:

Gates: $\mathcal{O}(nm)$
Depth: $\mathcal{O}(n + m)$
Space: $\mathcal{O}(m)$

Fast mult. + fast GCD:

Gates: $\widetilde{\mathcal{O}}(n)$
Depth: $\widetilde{\mathcal{O}}(n/m + m)$
Space: $\widetilde{\mathcal{O}}(m)$

Asymptotic costs

Main result: Circuit for factoring n-bit integers $N = p^2 q$, with m-bit q

<u>Schoolbook mult. + standard GCD:</u>

Gates: $\mathcal{O}(nm)$
Depth: $\mathcal{O}(n + m)$
Space: $\mathcal{O}(m)$

<u>Fast mult. + fast GCD:</u>

Gates: $\widetilde{\mathcal{O}}(n)$
Depth: $\widetilde{\mathcal{O}}(n/m + m)$
Space: $\widetilde{\mathcal{O}}(m)$

What should we set m to?

What should we set m to?

Classical factoring: for integers $N = p^2 q$, with $n = \log N$ and $m = \log q$

General Number Field Sieve:

Used for RSA integers

Costs roughly $\exp\left(\mathcal{O}(\sqrt[3]{n})\right)$

Lenstra ECM/Mulder:

Used for integers with small factors

Costs roughly $\exp\left(\mathcal{O}(\sqrt{m})\right)$

What should we set m to?

Classical factoring: for integers $N = p^2 q$, with $n = \log N$ and $m = \log q$

General Number Field Sieve:

Used for RSA integers

Costs roughly $\exp\left(\mathcal{O}(\sqrt[3]{n})\right)$

Lenstra ECM/Mulder:

Used for integers with small factors

Costs roughly $\exp\left(\mathcal{O}(\sqrt{m})\right)$

Set $m = \mathcal{O}(n^{2/3})$ for the *cheapest* quantum circuit classically as hard as RSA

Asymptotic costs

Main result: Circuit for factoring n-bit integers $N = p^2q$, with $\log q = m = O(n^{2/3})$

<u>Schoolbook mult. + standard GCD:</u>

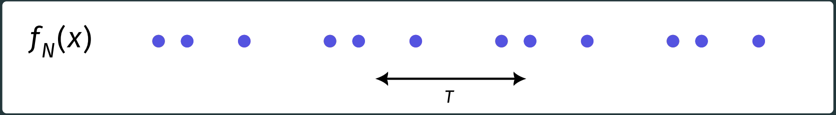
Gates: $\mathcal{O}(n^{5/3})$
Depth: $\mathcal{O}(n)$
Space: $\mathcal{O}(n^{2/3})$

<u>Fast mult. + fast GCD:</u>

Gates: $\widetilde{\mathcal{O}}(n)$
Depth: $\widetilde{\mathcal{O}}(n^{2/3})$
Space: $\widetilde{\mathcal{O}}(n^{2/3})$

Reducing output to 1 qubit: $f_N(x) : \{0,1\}^* \to \{0,1\}$

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Things we need for low qubit count:

- Small period T
- Avoid *smaller* periods than T
- Low workspace to compute $f_N(x)$

Some number theory

Legendre symbol

Legendre symbol

For a prime p:

$$\left(\frac{x}{p}\right) = \begin{cases} 0 & \text{if } x \equiv 0 \pmod{p} \\ 1 & \text{if } \exists\, w \text{ s.t. } w^2 \equiv x \pmod{p} \\ -1 & \text{otherwise} \end{cases}$$

Legendre symbol

For a prime p:

$$\left(\frac{x}{p}\right) = \begin{cases} 0 & \text{if } x \equiv 0 \pmod{p} \\ 1 & \text{if } \exists\, w \text{ s.t. } w^2 \equiv x \pmod{p} \\ -1 & \text{otherwise} \end{cases}$$

Legendre symbol is 1) **efficient to compute** given x and p, 2) **periodic** with period p

Jacobi symbol

For a composite number $N = \prod_i p_i$:

$$\left(\frac{x}{N}\right) = \prod_i \left(\frac{x}{p_i}\right)$$

Jacobi symbol is 1) **efficient to compute** given x and N, 2) **periodic** with period...?

Some number theory

Jacobi symbol is 1) **efficient to compute** given x and N, 2) **periodic** with period...?

For $N = pq$:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right)\left(\frac{x}{q}\right)$$

Jacobi symbol is 1) **efficient to compute** given x and N, 2) **periodic** with period...?

For $N = pq$:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right)\left(\frac{x}{q}\right)$$

Period is N—not helpful for factoring!

Jacobi symbol is 1) **efficient to compute** given x and N, 2) **periodic** with period...?

For $N = p^2 q$:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right)^2 \left(\frac{x}{q}\right)$$

Jacobi symbol is 1) **efficient to compute** given x and N, 2) **periodic** with period...?

For $N = p^2 q$:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right)^2 \left(\frac{x}{q}\right) = \left(\frac{x}{q}\right)$$

Some number theory

Jacobi symbol is 1) **efficient to compute** given x and N, 2) **periodic** with period...?

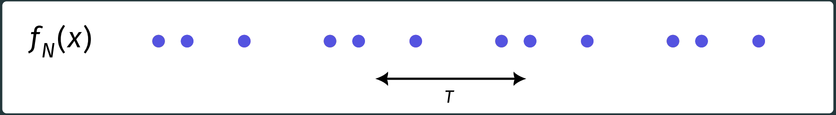
For $N = p^2q$:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right)^2 \left(\frac{x}{q}\right) = \left(\frac{x}{q}\right)$$

Period is q—exactly what we need!!

Reducing output to 1 qubit: $f_N(x) : \{0,1\}^* \to \{0,1\}$

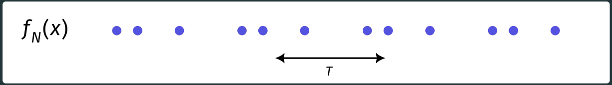
Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Things we need for low qubit count:

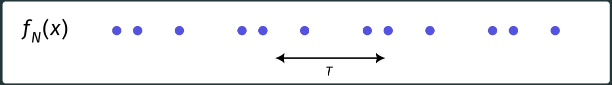
✓ Small period T
- Avoid *smaller* periods than T
- Low workspace to compute $f_N(x)$

Avoiding smaller periods



Need to compute the Fourier transform of the Jacobi symbol.

Avoiding smaller periods

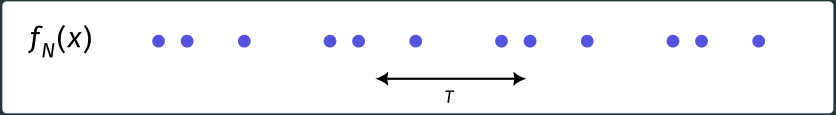


Need to compute the Fourier transform of the Jacobi symbol.

Carl Friedrich Gauss, early 1800s: this function has the *ideal* Fourier spectrum for us!

Reducing output to 1 qubit: $f_N(x) : \{0,1\}^* \to \{0,1\}$

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle |f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Things we need for low qubit count:

- ✓ Small period T
- ✓ Avoid *smaller* periods than T
- Low workspace to compute $f_N(x)$

People knew this!

"An Efficient Exact Quantum Algorithm for the Integer Square-free Decomposition Problem." Li, Peng, Du, Suter. Nature Scientific Reports, 2012.

Decomposing $N \to P^2 Q$ via Jacobi symbol was known in the literature a decade ago!

People knew this!

"An Efficient Exact Quantum Algorithm for the Integer Square-free Decomposition Problem." Li, Peng, Du, Suter. Nature Scientific Reports, 2012.

Decomposing $N \to P^2Q$ via Jacobi symbol was known in the literature a decade ago!

Their results:

- Jacobi symbol can be computed via standard circuits, using $O(n)$ space
- Quantum period finding yields Q *exactly* if we take a superposition $x \in [0, N-1]$

People knew this!

"An Efficient Exact Quantum Algorithm for the Integer Square-free Decomposition Problem." Li, Peng, Du, Suter. Nature Scientific Reports, 2012.

Decomposing $N \to P^2 Q$ via Jacobi symbol was known in the literature a decade ago!

Our contributions:

- Jacobi symbol can be computed via standard circuits, using $O(n)$ space
    - When quantum input is small, **extremely** efficient quantum circuits exist!
- Quantum period finding yields Q *exactly* if we take a superposition $x \in [0, N-1]$

People knew this!

"An Efficient Exact Quantum Algorithm for the Integer Square-free Decomposition Problem." Li, Peng, Du, Suter. Nature Scientific Reports, 2012.

Decomposing $N \to P^2 Q$ via Jacobi symbol was known in the literature a decade ago!

Our contributions:

- Jacobi symbol can be computed via standard circuits, using $O(n)$ space
  - When quantum input is small, **extremely** efficient quantum circuits exist!
- Quantum period finding yields Q *exactly* if we take a superposition $x \in [0, N-1]$
  - With superposition only to $\mathrm{poly}(Q)$, we still succeed $\to$ x needs only $\mathcal{O}(\log Q)$ qubits

Computing the Jacobi symbol

Goal: Compute $\left(\frac{x}{N}\right)$

Computing the Jacobi symbol

Goal: Compute $\left(\frac{x}{N}\right)$

$$\left(\frac{a}{b}\right) \in \{-1, 0, 1\} \tag{1}$$

Computing the Jacobi symbol

Goal: Compute $|x\rangle \to \left(\frac{x}{N}\right)|x\rangle$

$$\left(\frac{a}{b}\right) \widetilde{\in} \{-1, 1\} \tag{1}$$

Computing the Jacobi symbol

Goal: Compute $|x\rangle \to \left(\frac{x}{N}\right)|x\rangle$

$$\left(\frac{a}{b}\right) \widetilde{\in} \{-1, 1\} \tag{1}$$

Recall: N is classical, n bits; $|x\rangle$ is quantum, m qubits—and potentially $m \ll n$.

Computing the Jacobi symbol

Goal: Compute $|x\rangle \to \left(\frac{x}{N}\right)|x\rangle$

$$\left(\frac{a}{b}\right) \widetilde{\in} \{-1, 1\} \tag{1}$$

Recall: N is classical, n bits; $|x\rangle$ is quantum, m qubits—and potentially $m \ll n$.

The "big" input is entirely classical.
Can we implement this circuit using only $O(m)$ qubits?

Computing the Jacobi symbol

Goal: Compute $|x\rangle \to \left(\frac{x}{N}\right)|x\rangle$

$$\left(\frac{a}{b}\right) \widetilde{\in} \{-1, 1\} \tag{1}$$

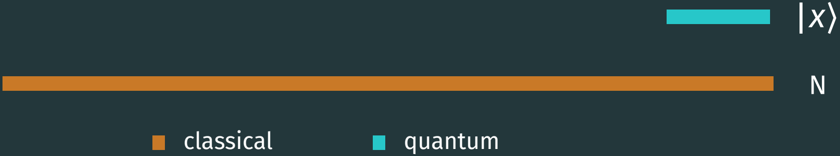
Recall: N is classical, n bits; $|x\rangle$ is quantum, m qubits—and potentially $m \ll n$.

The "big" input is entirely classical.
Can we implement this circuit using only $O(m)$ qubits?

Yes!

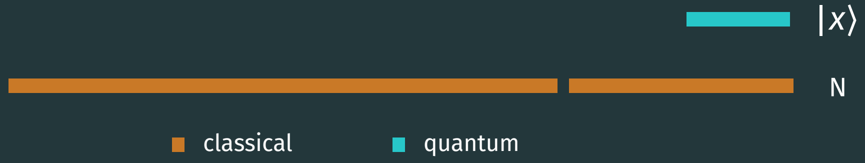
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



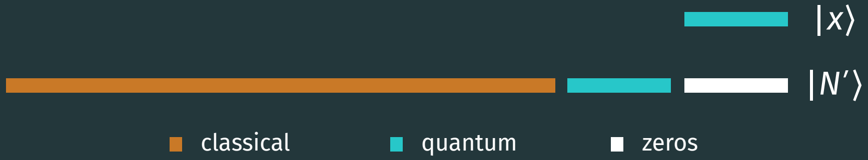
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



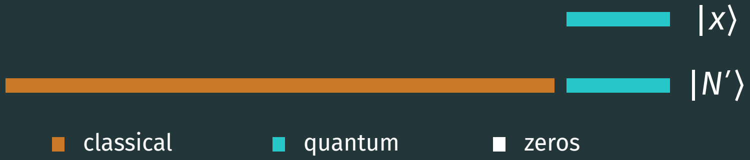
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



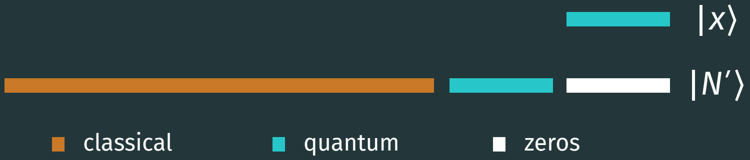
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



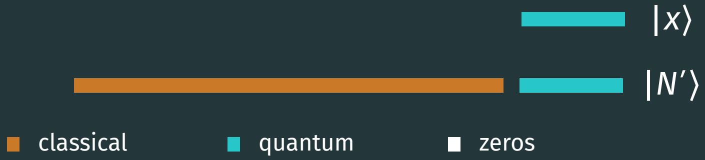
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



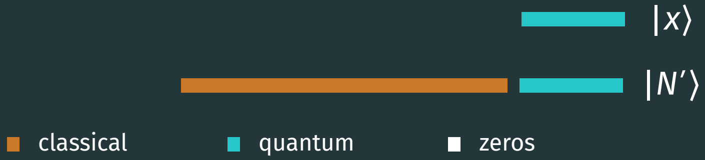
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



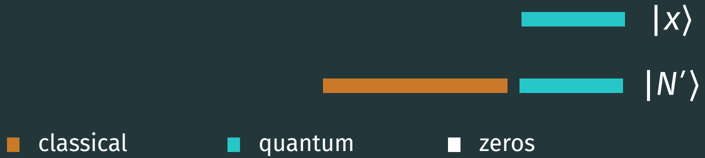
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



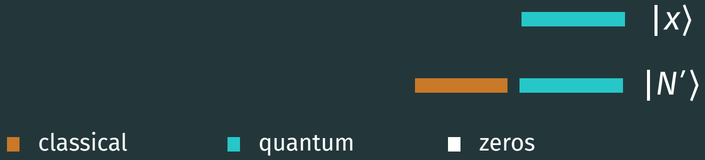
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N



Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N

classical quantum zeros

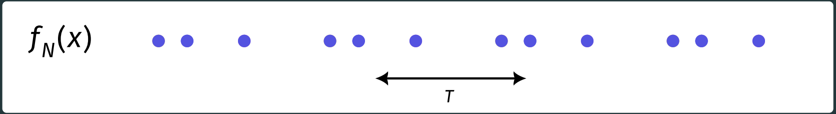
Streaming classical bits to a quantum circuit

Key idea: can *stream* through the classical bits of N

Now with two length-m inputs,
standard circuits for $\left(\frac{x}{N'}\right)$ have depth and qubits $\widetilde{O}(m)$

Reducing output to 1 qubit: $f_N(x) : \{0,1\}^* \to \{0,1\}$

Find some $f_N(x)$ w/ period T, where T can be used to find factors → Generate $\sum_{x=0}^{\mathrm{poly}(T)} |x\rangle\,|f_N(x)\rangle$ → Measure $f_N(x)$, yielding superposition over a coset → QFT on x register, measure, postprocess to find T



Things we need for low qubit count:

- ✓ Small period T
- ✓ Avoid *smaller* periods than T
- ✓ Low workspace to compute $f_N(x)$

Putting it all together: asymptotic costs

Main result: Circuit for factoring n-bit integers $N = p^2 q$, with $\log q = m = O(n^{2/3})$

<u>Schoolbook mult. + standard GCD:</u>

Gates: $\mathcal{O}(nm)$
Depth: $\mathcal{O}(n + m)$
Space: $\mathcal{O}(m)$

<u>Fast mult. + fast GCD:</u>

Gates: $\widetilde{\mathcal{O}}(n)$
Depth: $\widetilde{\mathcal{O}}(n/m + m)$
Space: $\widetilde{\mathcal{O}}(m)$

Putting it all together: asymptotic costs

Main result: Circuit for factoring n-bit integers $N = p^2 q$, with $\log q = m = O(n^{2/3})$

<u>Schoolbook mult. + standard GCD:</u>

Gates: $\mathcal{O}(nm)$
Depth: $\mathcal{O}(n + m)$
Space: $\mathcal{O}(m)$

<u>Fast mult. + fast GCD:</u>

Gates: $\widetilde{\mathcal{O}}(n)$
Depth: $\widetilde{\mathcal{O}}(n/m + m)$
Space: $\widetilde{\mathcal{O}}(m)$

Space $\sim 2m$ seems achievable, $m \sim 300$ seems classically hard.
Classically-hard factoring with a few hundred qubits?

Recent results

A sublinear space and depth factoring algorithm

For integers $N = P^2Q$:
Gate count $\tilde{O}(n)$
Qubits and depth $\tilde{O}(n^{2/3})$

GDKM, S. Ragavan, V. Vaikuntanathan, K. Van Kirk. arXiv:2412.12558

Log-depth "optimistic" QFT with no ancillas

Error bounded by ϵ on all but $O(\epsilon) \cdot 2^n$ basis states

GDKM, J. Blue, T. Bergamaschi, C. Gidney, I. Chuang. arXiv:2505.00701

Fast quantum integer multiplication

$O(n^{1+\varepsilon})$ gates
No ancilla qubits

GDKM, N. Yao. arXiv:2403.18006

Shor's algorithm with:

$O(n^{2+\varepsilon})$ gates
$O(n^{1+\varepsilon})$ depth
$2n + O(n/\log n)$ total qubits

Multiplication on quantum computers

Goal: Implement $\mathcal{U}_{c\times q}(a)\,|x\rangle\,|0\rangle = |x\rangle\,|ax\rangle$, for n-bit a and x

Multiplication on quantum computers

Goal: Implement $\mathcal{U}_{c\times q}(a)\,|x\rangle\,|0\rangle = |x\rangle\,|ax\rangle$, for n-bit a and x

[Draper '00]: Arithmetic in Fourier space

$$\mathrm{QFT}\,|ax\rangle = \sum_z \exp\left(\frac{2\pi i a x z}{2^n}\right)|z\rangle$$

Multiplication on quantum computers

Goal: Implement $\mathcal{U}_{c\times q}(a)\,|x\rangle\,|0\rangle = |x\rangle\,|ax\rangle$, for n-bit a and x

[Draper '00]: Arithmetic in Fourier space

$$\mathrm{QFT}\,|ax\rangle = \sum_z \exp\left(\frac{2\pi i a x z}{2^n}\right)|z\rangle$$

Plan:

Multiplication on quantum computers

Goal: Implement $\mathcal{U}_{c\times q}(a)\,|x\rangle\,|0\rangle = |x\rangle\,|ax\rangle$, for n-bit a and x

[Draper '00]: Arithmetic in Fourier space

$$\mathrm{QFT}\,|ax\rangle = \sum_z \exp\left(\frac{2\pi i a x z}{2^n}\right)|z\rangle$$

Plan: 1) Generate $|x\rangle \sum_z |z\rangle$

Multiplication on quantum computers

Goal: Implement $\mathcal{U}_{c\times q}(a)\left|x\right\rangle\left|0\right\rangle = \left|x\right\rangle\left|ax\right\rangle$, for n-bit a and x

[Draper '00]: Arithmetic in Fourier space

$$\mathrm{QFT}\left|ax\right\rangle = \sum_z \exp\left(\frac{2\pi i a x z}{2^n}\right)\left|z\right\rangle$$

Plan: 1) Generate $\left|x\right\rangle \sum_z \left|z\right\rangle$, 2) apply a phase rotation of $\exp\left(\frac{2\pi i a x z}{2^n}\right)$

Multiplication on quantum computers

Goal: Implement $\mathcal{U}_{c\times q}(a)\,|x\rangle\,|0\rangle = |x\rangle\,|ax\rangle$, for n-bit a and x

[Draper '00]: Arithmetic in Fourier space

$$\mathrm{QFT}\,|ax\rangle = \sum_z \exp\left(\frac{2\pi i a x z}{2^n}\right)|z\rangle$$

Plan: 1) Generate $|x\rangle \sum_z |z\rangle$, 2) apply a phase rotation of $\exp\left(\frac{2\pi i a x z}{2^n}\right)$, 3) apply QFT^{-1}

Multiplication on quantum computers

Goal: Implement $\mathcal{U}_{c\times q}(a)\,|x\rangle\,|0\rangle = |x\rangle\,|ax\rangle$, for n-bit a and x

[Draper '00]: Arithmetic in Fourier space

$$\mathrm{QFT}\,|ax\rangle = \sum_z \exp\left(\frac{2\pi i a x z}{2^n}\right)|z\rangle$$

Plan: 1) Generate $|x\rangle \sum_z |z\rangle$, 2) apply a phase rotation of $\exp\left(\frac{2\pi i a x z}{2^n}\right)$, 3) apply QFT^{-1}

[GDKM, Yao '24]: Can apply phase using:

$O(n^{1+\epsilon})$ gates $\quad O(n^{\epsilon})$ depth $\quad O(n/\log n)$ ancillas

A log-depth "optimistic" QFT with no ancillas

Greg
Kahanamoku-Meyer

John
Blue

Thiago
Bergamaschi

Craig
Gidney

Ike
Chuang

Structure of the QFT

The quantum Fourier transform on n qubits (dropping normalization):

$$\mathrm{QFT}\,|x\rangle \equiv |\Phi_x\rangle = \sum_{y=0}^{2^n-1} e^{2\pi i x y/2^n}\,|y\rangle$$

Structure of the QFT

The quantum Fourier transform on n qubits (dropping normalization):

$$\mathrm{QFT}\,|x\rangle \equiv |\Phi_x\rangle = \bigotimes_{i=0}^{n-1} \Big(|0\rangle + e^{2\pi i \; 0.x_i x_{i+1}\cdots} |1\rangle \Big)$$

where $0.x_i x_{i+1} \cdots = 2^i x / 2^n \mod 1$

Structure of the QFT

The quantum Fourier transform on n qubits (dropping normalization):

$$\mathrm{QFT}\,|x\rangle \equiv |\Phi_x\rangle = \bigotimes_{i=0}^{n-1} \Big(|0\rangle + e^{2\pi i \; 0.x_i x_{i+1}\cdots} |1\rangle \Big)$$

where $0.x_i x_{i+1}\cdots = 2^i x/2^n \mod 1$

ϵ-approximate QFT: truncate $0.x_i x_{i+1}\cdots$ after $m \sim O(\log(n/\epsilon))$ bits

Structure of the QFT

The quantum Fourier transform on n qubits (dropping normalization):

$$\mathrm{QFT}\,|x\rangle \equiv |\Phi_x\rangle = \bigotimes_{i=0}^{n-1} \left(\sum_{y_j \in \{0,1\}} e^{2\pi i y_j \cdot 0.x_i x_{i+1}\cdots} \left|y_j\right\rangle \right)$$

where $0.x_i x_{i+1} \cdots = 2^i x / 2^n \mod 1$

ϵ-approximate QFT: truncate $0.x_i x_{i+1}\cdots$ after $m \sim O(\log(n/\epsilon))$ bits

Structure of the QFT

The quantum Fourier transform on n qubits (dropping normalization):

$$\mathrm{QFT}\,|x\rangle \equiv |\Phi_x\rangle = \bigotimes_{i=0}^{n-1}\left(\sum_{y_j\in\{0,1\}} e^{2\pi i y_j\cdot 0.x_i x_{i+1}\cdots}\,|y_j\rangle\right)$$

where $0.x_i x_{i+1}\cdots = 2^i x/2^n \mod 1$

ϵ-approximate QFT: truncate $0.x_i x_{i+1}\cdots$ after $m \sim O(\log(n/\epsilon))$ bits

Let's do a similar trick, in base $b = 2^m$

QFT, block version

In base $b = 2^m$ we have $x = \sum_i 2^{mi} X_i$.

QFT, block version

In base $b = 2^m$ we have $x = \sum_i 2^{mi} X_i$.

$$\mathrm{QFT}\,|x\rangle \equiv |\Phi_x\rangle = \bigotimes_{i=0}^{n/m-1} \left[|\Phi_x\rangle\right]_i \approx \bigotimes_{i=0}^{n/m-1} \left(\sum_{Y_j=0}^{2^m-1} e^{2\pi i Y_j \cdot 0.X_i X_{i+1} \cdots} \left|Y_j\right\rangle \right)$$

QFT, block version

In base $b = 2^m$ we have $x = \sum_i 2^{mi} X_i$.

$$\mathrm{QFT}\,|x\rangle \equiv |\Phi_x\rangle = \bigotimes_{i=0}^{n/m-1} [|\Phi_x\rangle]_i \approx \bigotimes_{i=0}^{n/m-1} \left(\sum_{Y_j=0}^{2^m-1} e^{2\pi i Y_j \cdot 0.X_i X_{i+1} \cdots} \left| Y_j \right\rangle \right)$$

ϵ-approximate QFT: since $m \sim O(\log(n/\epsilon))$, truncate to $0.X_i X_{i+1}$

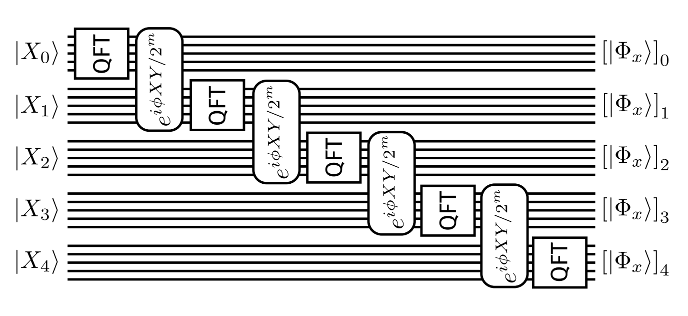
Approximate QFT, block version

In base $b = 2^m$ we have $x = \sum_i 2^{mi} X_i$.

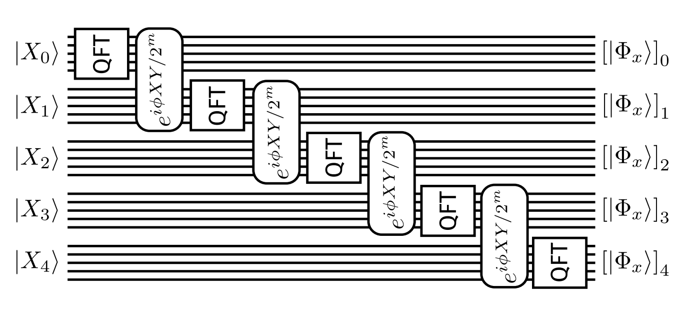
Approximate QFT, block version

In base $b = 2^m$ we have $x = \sum_i 2^{mi} X_i$. With $\phi = 2\pi/2^m$:

$$\mathrm{QFT}\,|x\rangle \equiv |\Phi_x\rangle \approx \bigotimes_{i=0}^{n/m-1} \left(\sum_{Y_j=0}^{2^m-1} e^{i\phi(X_i + X_{i+1}/2^m)Y_j} \left|Y_j\right\rangle \right)$$



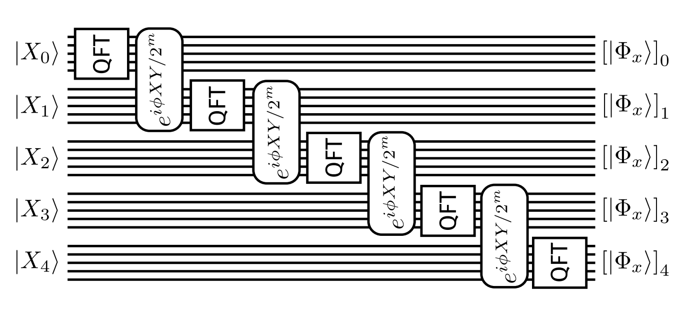
Approximate QFT, block version



Gate count: $\mathcal{O}(n \log n)$

Space-time product: $\mathcal{O}(n^2)$

Approximate QFT, block version



Gate count: $\mathcal{O}(n \log n)$

Space-time product: $\mathcal{O}(n^2)$

Why are we stuck with linear depth here?

How to do better than linear depth

What happens if you apply $\mathrm{QFT}^\dagger$ to the following (remember $\phi = 2\pi/2^m$)

$$\mathrm{QFT}^\dagger \sum_{Y_j} e^{i\phi X_i Y_j} \left|Y_j\right\rangle = ?$$

How to do better than linear depth

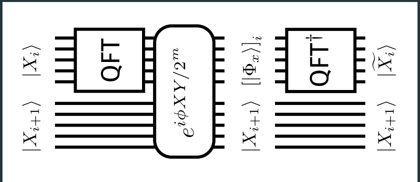
What happens if you apply $\text{QFT}^\dagger$ to the following (remember $\phi = 2\pi/2^m$)

$$\text{QFT}^\dagger \sum_{Y_j} e^{i\phi X_i Y_j} \left|Y_j\right\rangle = \left|X_i\right\rangle$$

How to do better than linear depth

What happens if you apply $\mathrm{QFT}^\dagger$ to the following (remember $\phi = 2\pi/2^m$)

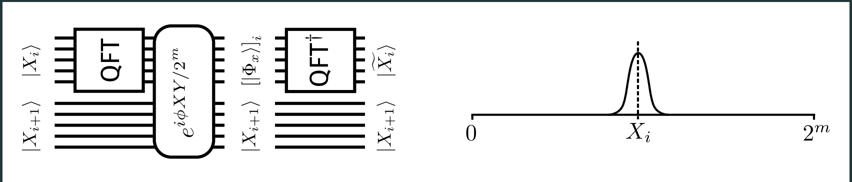
$$\mathrm{QFT}^\dagger \left[|\Phi_x\rangle\right]_i = \mathrm{QFT}^\dagger \sum_{Y_j} e^{i\phi(X_i + X_{i+1}/2^m)Y_j} \left|Y_j\right\rangle = ?$$



How to do better than linear depth

What happens if you apply QFT† to the following (remember $\phi = 2\pi/2^m$)

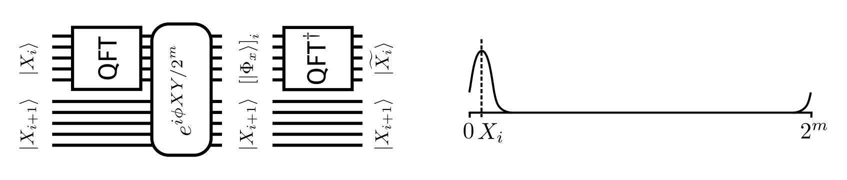
$$\mathrm{QFT}^\dagger\left[|\Phi_x\rangle\right]_i = \mathrm{QFT}^\dagger \sum_{Y_j} e^{i\phi(X_i + X_{i+1}/2^m)Y_j}\left|Y_j\right\rangle = \left|\widetilde{X}_i\right\rangle$$



A subtlety

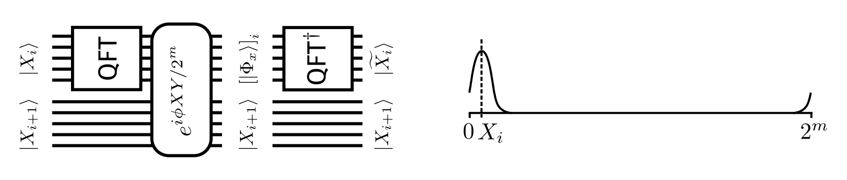
A subtlety

What happens if X_i is too close to $0 \pmod{2^m}$?



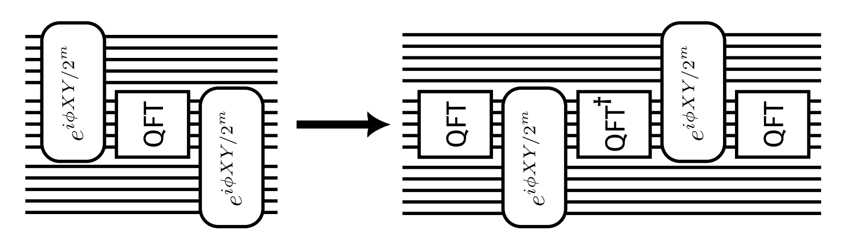
A subtlety

What happens if X_i is too close to $0 \pmod{2^m}$?



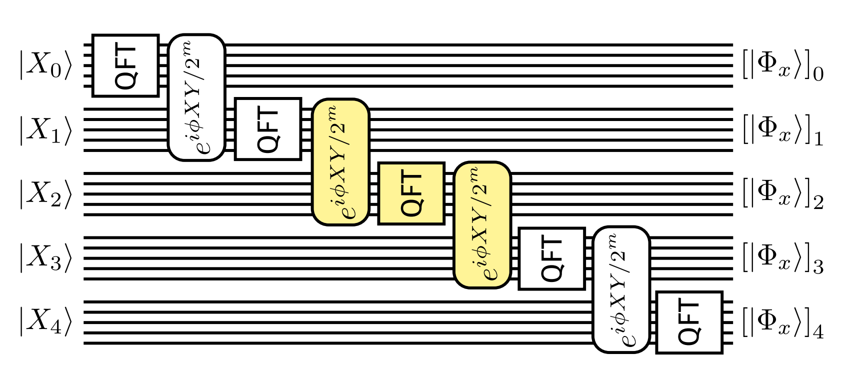
Part of the phase rotation "controlled off" $\left|\widetilde{X}_i\right\rangle$ will be off by 2^m!

Proposed replacement:



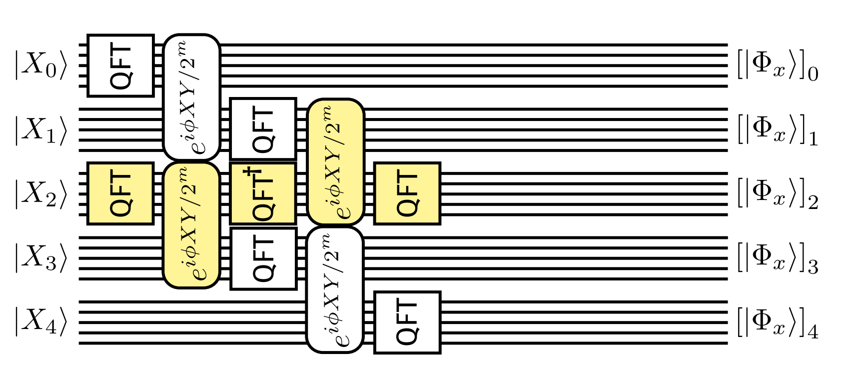
Rearranging gates

Proposed replacement:

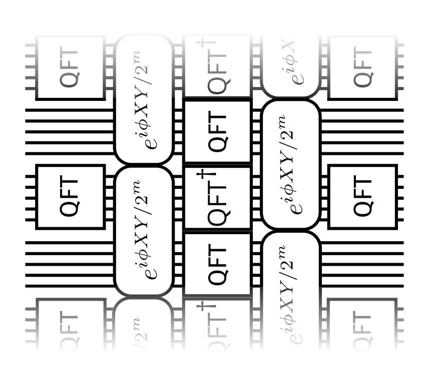


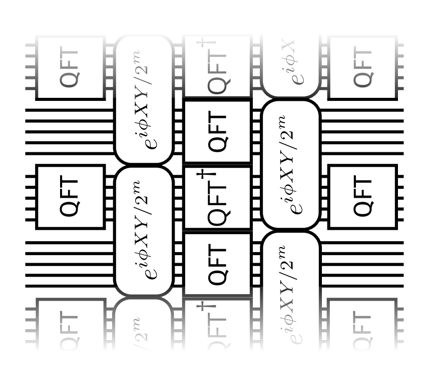
Rearranging gates

Proposed replacement:



Looking a bit closer

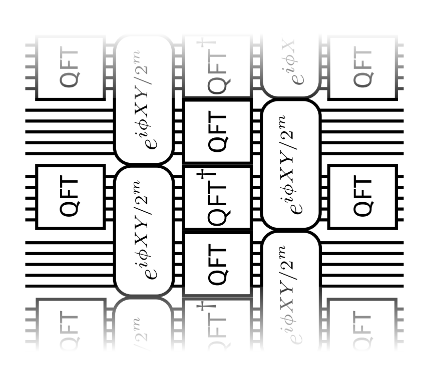




Features:

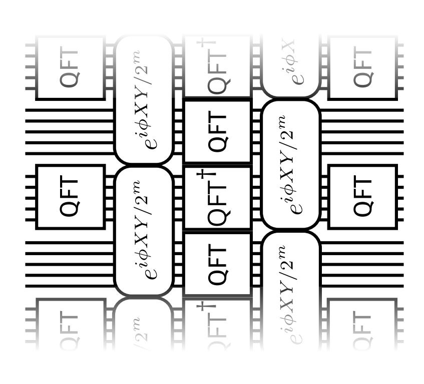
- 5 layers, each layer has depth $\mathcal{O}(\log n)$

Looking a bit closer



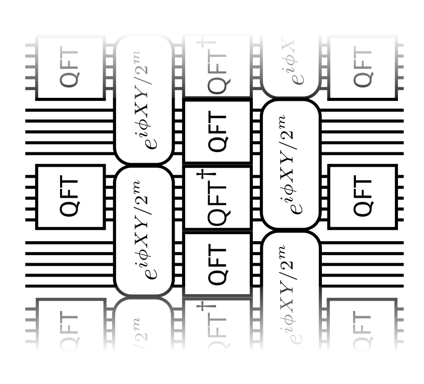
Features:

- 5 layers, each layer has depth $\mathcal{O}(\log n)$
- No ancilla qubits



Features:

- 5 layers, each layer has depth $\mathcal{O}(\log n)$
- No ancilla qubits
- All gates have range at most $\mathcal{O}(\log n)$

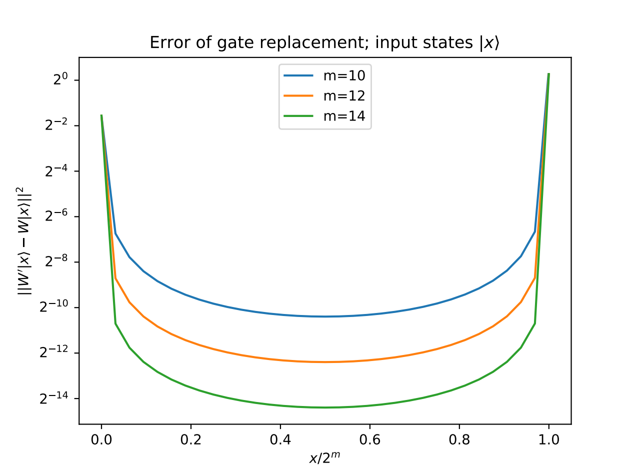


Features:

- 5 layers, each layer has depth $\mathcal{O}(\log n)$
- No ancilla qubits
- All gates have range at most $\mathcal{O}(\log n)$
- Doesn't give the right answer (sometimes)

“Optimistic” QFT

We have a good approximation on most basis states, with super nice properties!



“Optimistic” QFT

We have a good approximation on the vast majority of basis states, with super nice properties!

What should we do with it?

“Optimistic” QFT

We have a good approximation on the vast majority of basis states, with super nice properties!

What should we do with it?

- Use it anyway (on “random” inputs)

“Optimistic” QFT

We have a good approximation on the vast majority of basis states, with super nice properties!

What should we do with it?

- Use it anyway (on “random” inputs)
- Bootstrap it into a slightly more expensive circuit that approximates QFT well on all basis states

Some things I've worked on

A sublinear space and depth factoring algorithm

For integers $N = P^2 Q$:
Gate count $\tilde{O}(n)$
Qubits and depth $\tilde{O}(n^{2/3})$

GDKM, S. Ragavan, V. Vaikuntanathan, K. Van Kirk. arXiv:2412.12558

Log-depth "optimistic" QFT with no ancillas

Error bounded by ϵ on all but $O(\epsilon) \cdot 2^n$ basis states

GDKM, J. Blue, T. Bergamaschi, C. Gidney, I. Chuang. arXiv:2505.00701

Fast quantum integer multiplication

$O(n^{1+\varepsilon})$ gates
No ancilla qubits

GDKM, N. Yao. arXiv:2403.18006

Shor's algorithm with:

$O(n^{2+\varepsilon})$ gates
$O(n^{1+\varepsilon})$ depth
$2n + O(n/\log n)$ total qubits

Fast quantum multiplication without ancillas

Greg
Kahanamoku-Meyer

Norm
Yao

Background: fast multiplication

Given two n-bit numbers x and y, write them as ”two digit” numbers in base $b = 2^{n/2}$.

$$
\begin{array}{rccc}
 & & x_1 & x_0 \\
\times & & y_1 & y_0 \\
\hline
 & & & x_0 y_0 \\
 & & x_1 y_0 & \\
 & & x_0 y_1 & \\
+ & x_1 y_1 & & \\
\hline
\end{array}
$$

Background: fast multiplication

Given two n-bit numbers x and y, write them as "two digit" numbers in base $b = 2^{n/2}$.

$$
\begin{array}{cccc}
 & & x_1 & x_0 \\
\times & & y_1 & y_0 \\
\hline
 & & & x_0 y_0 \\
 & & x_1 y_0 & \\
 & & x_0 y_1 & \\
+ & x_1 y_1 & & \\
\hline
\end{array}
$$

$$xy = x_1 y_1 b^2 + x_0 y_1 b + x_1 y_0 b + x_0 y_0$$

Background: fast multiplication

Given two n-bit numbers x and y, write them as "two digit" numbers in base $b = 2^{n/2}$.

$$\begin{array}{rrrr}
 & & x_1 & x_0 \\
\times & & y_1 & y_0 \\
\hline
 & & & x_0 y_0 \\
 & & x_1 y_0 & \\
 & & x_0 y_1 & \\
+ & x_1 y_1 & & \\
\hline
\end{array}$$

$$xy = x_1 y_1 b^2 + x_0 y_1 b + x_1 y_0 b + x_0 y_0$$

Time remains $\mathcal{O}(n^2)$, because $4(n/2)^2 = n^2$

Background: Karatsuba multiplication

$$xy = x_1 y_1 b^2 + (x_0 y_1 + x_1 y_0) b + x_0 y_0$$

Background: Karatsuba multiplication

$$xy = x_1 y_1 b^2 + (x_0 y_1 + x_1 y_0) b + x_0 y_0$$

Observation: $x_0 y_1 + x_1 y_0 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$

Background: Karatsuba multiplication

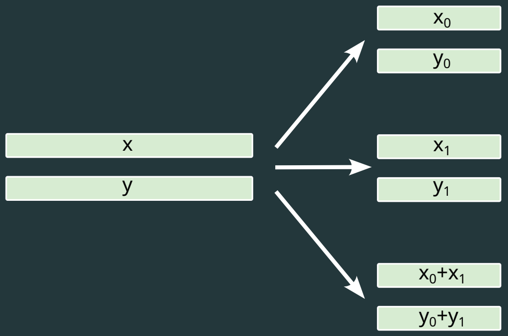
$$xy = x_1 y_1 b^2 + (x_0 y_1 + x_1 y_0) b + x_0 y_0$$

Observation: $x_0 y_1 + x_1 y_0 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$

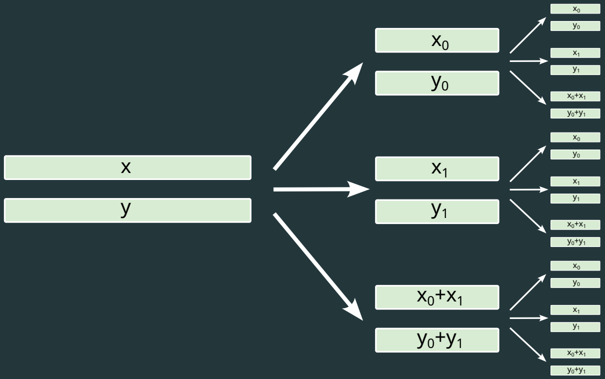
Can compute xy with only three multiplications of size $\log b = n/2$:

1. $x_1 y_1$
2. $x_0 y_0$
3. $(x_1 + x_0)(y_1 + y_0)$

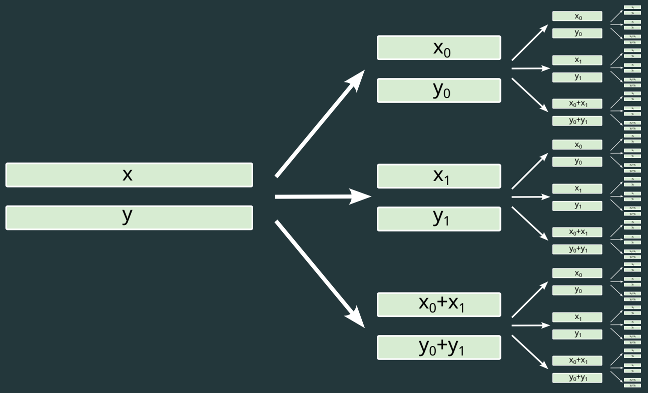
Background: Karatsuba multiplication



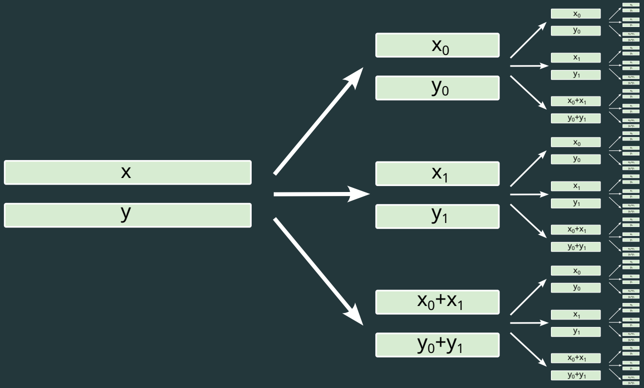
Background: Karatsuba multiplication



Background: Karatsuba multiplication

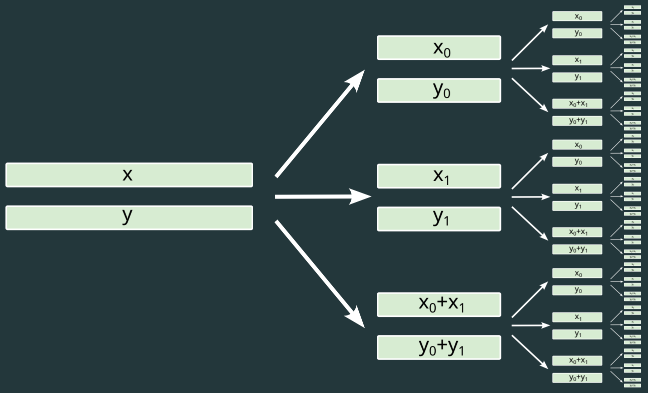


Background: Karatsuba multiplication



Depth: $d = \log_2 n$

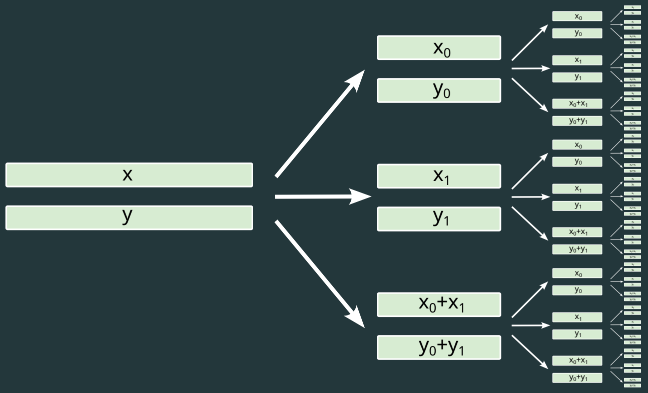
Background: Karatsuba multiplication



Depth: $d = \log_2 n$

Operations: 3^d

Background: Karatsuba multiplication



Depth: $d = \log_2 n$

Operations: 3^d

Cost: $\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.58\cdots})$

Fast classical-quantum multiplication

Goal: $\mathcal{U}(a)\,|x\rangle\,|0\rangle = |x\rangle\,|ax\rangle$

Fast classical-quantum multiplication

Goal: Apply phase ϕxz; x and z are quantum

Fast classical-quantum multiplication

Goal: Apply phase ϕxz; x and z are quantum

Karatsuba:

$$xz = 2^n x_1 z_1 + 2^{n/2}((x_0 + x_1)(z_0 + z_1) - x_0 z_0 - x_1 z_1) + x_0 z_0$$

Fast classical-quantum multiplication

Goal: Apply phase ϕxz; x and z are quantum

Plugging in Karatsuba:

$$\begin{aligned}\exp(i\phi xz) &= \exp(i\phi 2^n x_1 z_1)\\ &\cdot \exp(i\phi x_0 z_0)\\ &\cdot \exp\left(i\phi 2^{n/2}((x_0+x_1)(z_0+z_1) - x_0 z_0 - x_1 z_1)\right)\end{aligned}$$

Fast classical-quantum multiplication

Goal: Apply phase ϕxz; x and z are quantum

Plugging in Karatsuba:

$$\begin{aligned}\exp(i\phi xz) = {} & \exp(i\phi 2^n x_1 z_1) \\ & \cdot \exp(i\phi x_0 z_0) \\ & \cdot \exp\left(i\phi 2^{n/2}((x_0+x_1)(z_0+z_1) - x_0 z_0 - x_1 z_1)\right)\end{aligned}$$

How are we supposed to reuse values in the *phase*?

Fast classical-quantum multiplication

Goal: Implement PhaseProduct$(\phi) |x\rangle |z\rangle = \exp(i\phi xz) |x\rangle |z\rangle$

Karatsuba:

$$xz = 2^n x_1 z_1 + 2^{n/2}((x_0 + x_1)(z_0 + z_1) - x_0 z_0 - x_1 z_1) + x_0 z_0$$

Fast classical-quantum multiplication

Goal: Implement $\text{PhaseProduct}(\phi)\,|x\rangle\,|z\rangle = \exp(i\phi xz)\,|x\rangle\,|z\rangle$

Re-ordering Karatsuba:

$$xz = (2^n - 2^{n/2})x_1 z_1 + 2^{n/2}(x_0 + x_1)(z_0 + z_1) + (1 - 2^{n/2})x_0 z_0$$

Fast classical-quantum multiplication

Goal: Implement PhaseProduct$(\phi) |x\rangle |z\rangle = \exp(i\phi xz) |x\rangle |z\rangle$

Plugging in reordered Karatsuba:

$$\begin{aligned}\exp(i\phi xz) = \exp&\left(i\phi(2^n - 2^{n/2})x_1 z_1\right)\\ &\cdot \exp\left(i\phi(1 - 2^{n/2})x_0 z_0\right)\\ &\cdot \exp\left(i\phi 2^{n/2}(x_0 + x_1)(z_0 + z_1)\right)\end{aligned}$$

Fast classical-quantum multiplication

Goal: Implement PhaseProduct(ϕ) $|x\rangle\,|z\rangle = \exp(i\phi xz)\,|x\rangle\,|z\rangle$

Plugging in reordered Karatsuba:

$$
\begin{aligned}
\exp(i\phi xz) = &\exp(i\phi_1 x_1 z_1) \\
&\cdot \exp(i\phi_2 x_0 z_0) \\
&\cdot \exp(i\phi_3 (x_0 + x_1)(z_0 + z_1))
\end{aligned}
$$

$$
\begin{aligned}
\phi_1 &= (2^n - 2^{n/2})\phi \\
\phi_2 &= (1 - 2^{n/2})\phi \\
\phi_3 &= 2^{n/2}\phi
\end{aligned}
$$

Fast classical-quantum multiplication

Goal: Implement $\text{PhaseProduct}(\phi)\,|x\rangle\,|z\rangle = \exp(i\phi x z)\,|x\rangle\,|z\rangle$

Plugging in reordered Karatsuba:

$$\begin{aligned}\exp(i\phi xz) = &\exp(i\phi_1 x_1 z_1)\\ &\cdot \exp(i\phi_2 x_0 z_0)\\ &\cdot \exp(i\phi_3 (x_0 + x_1)(z_0 + z_1))\end{aligned}$$

$$\begin{aligned}\phi_1 &= (2^n - 2^{n/2})\phi\\ \phi_2 &= (1 - 2^{n/2})\phi\\ \phi_3 &= 2^{n/2}\phi\end{aligned}$$

Each of these has the same structure, but on half as many qubits $\rightarrow$ do it recursively!

Fast classical-quantum multiplication

Goal: Implement PhaseProduct(ϕ) $|x\rangle\,|z\rangle = \exp(i\phi xz)\,|x\rangle\,|z\rangle$

$$\begin{aligned}\exp(i\phi xz) = &\exp(i\phi_1 x_1 z_1)\\ &\cdot \exp(i\phi_2 x_0 z_0)\\ &\cdot \exp(i\phi_3 (x_0 + x_1)(z_0 + z_1))\end{aligned}$$

$$\begin{aligned}\phi_1 &= (2^n - 2^{n/2})\phi\\ \phi_2 &= (1 - 2^{n/2})\phi\\ \phi_3 &= 2^{n/2}\phi\end{aligned}$$

Recursion relation: $T(n) = 3T(n/2)$

Fast classical-quantum multiplication

Goal: Implement PhaseProduct$(\phi)\,|x\rangle\,|z\rangle = \exp(i\phi xz)\,|x\rangle\,|z\rangle$

$$\begin{aligned}\exp(i\phi xz) = &\exp(i\phi_1 x_1 z_1)\\ &\cdot \exp(i\phi_2 x_0 z_0)\\ &\cdot \exp(i\phi_3 (x_0 + x_1)(z_0 + z_1))\end{aligned}$$

$$\begin{aligned}\phi_1 &= (2^n - 2^{n/2})\phi\\ \phi_2 &= (1 - 2^{n/2})\phi\\ \phi_3 &= 2^{n/2}\phi\end{aligned}$$

Recursion relation: $T(n) = 3T(n/2) \Rightarrow \mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.58\cdots})$ gates!

How many qubits do we need?

Splitting registers $|x\rangle \to |x_1\rangle |x_0\rangle$ and $|z\rangle \to |z_1\rangle |z_0\rangle$, can immediately do

- $\exp(i\phi_1 x_1 z_1)$
- $\exp(i\phi_2 x_0 z_0)$

How many qubits do we need?

Splitting registers $|x\rangle \to |x_1\rangle |x_0\rangle$ and $|z\rangle \to |z_1\rangle |z_0\rangle$, can immediately do

- $\exp(i\phi_1 x_1 z_1)$
- $\exp(i\phi_2 x_0 z_0)$

What about $\exp(i\phi_3 (x_0 + x_1)(z_0 + z_1))$?

How many qubits do we need?

Splitting registers $|x\rangle \to |x_1\rangle |x_0\rangle$ and $|z\rangle \to |z_1\rangle |z_0\rangle$, can immediately do

- $\exp(i\phi_1 x_1 z_1)$
- $\exp(i\phi_2 x_0 z_0)$

What about $\exp(i\phi_3(x_0 + x_1)(z_0 + z_1))$?

Use quantum addition circuits.

How many qubits do we need?

Splitting registers $|x\rangle \rightarrow |x_1\rangle |x_0\rangle$ and $|z\rangle \rightarrow |z_1\rangle |z_0\rangle$, can immediately do

- $\exp(i\phi_1 x_1 z_1)$
- $\exp(i\phi_2 x_0 z_0)$

What about $\exp(i\phi_3(x_0 + x_1)(z_0 + z_1))$?

Use quantum addition circuits.

But, **addition is reversible** $\rightarrow$ do it *in-place*! E.g. $|x_1\rangle |x_0\rangle \rightarrow |x_1\rangle |x_0 + x_1\rangle$

How many qubits do we need?

Splitting registers $|x\rangle \rightarrow |x_1\rangle |x_0\rangle$ and $|z\rangle \rightarrow |z_1\rangle |z_0\rangle$, can immediately do

- $\exp(i\phi_1 x_1 z_1)$
- $\exp(i\phi_2 x_0 z_0)$

What about $\exp(i\phi_3 (x_0 + x_1)(z_0 + z_1))$?

Use quantum addition circuits.

But, **addition is reversible** $\rightarrow$ do it *in-place*! E.g. $|x_1\rangle |x_0\rangle \rightarrow |x_1\rangle |x_0 + x_1\rangle$

With a few tricks, can use zero ancillas.

Karatsuba

Multiply n-bit numbers via
3 multiplications of size $n/2$

$\mathcal{O}(n^{\log_2 3})$ gates

Making it go faster

Karatsuba

Multiply n-bit numbers via
3 multiplications of size $n/2$

$\mathcal{O}(n^{\log_2 3})$ gates

Toom-Cook

Multiply n-bit numbers via
$2k-1$ multiplications of size n/k

$\mathcal{O}(n^{\log_k (2k-1)})$ gates

Complexity vs. k

Toom-Cook has asymptotic complexity $\mathcal{O}(n^{log_k(2k-1)})$

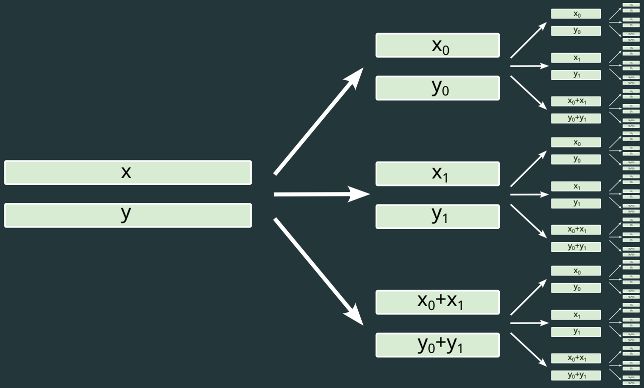
Complexity vs. k

Toom-Cook has asymptotic complexity $\mathcal{O}(n^{log_k(2k-1)})$

Algorithm	Gate count
Schoolbook	$\mathcal{O}(n^2)$
$k = 2$	$\mathcal{O}(n^{1.58\cdots})$
$k = 3$	$\mathcal{O}(n^{1.46\cdots})$
$k = 4$	$\mathcal{O}(n^{1.40\cdots})$
$\vdots$	$\vdots$

Depth

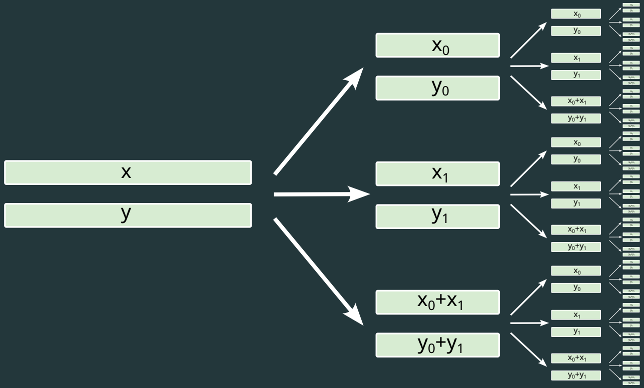
Parallelization is natural.



Depth

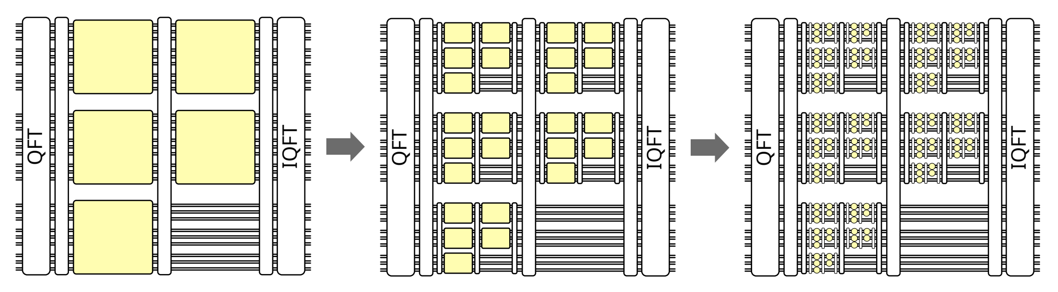
Parallelization is natural.

We have k sub-registers to work with—can do k sub-products in parallel.



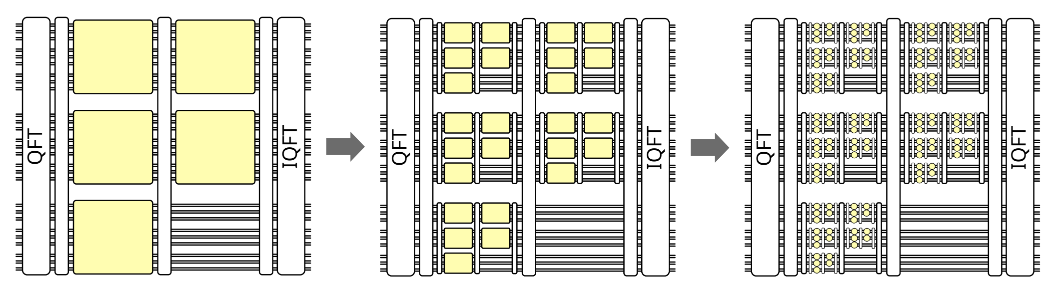
Depth

$k = 3$:



Depth

$k = 3$:



Depth is $\mathcal{O}(n^\epsilon)$ where $\epsilon = \log_k 2$, using $\mathcal{O}(n/\log n)$ ancillas.

Some things I've worked on

A sublinear space and depth factoring algorithm

For integers $N = P^2 Q$:
Gate count $\tilde{O}(n)$
Qubits and depth $\tilde{O}(n^{2/3})$

GDKM, S. Ragavan, V. Vaikuntanathan, K. Van Kirk. arXiv:2412.12558

Log-depth "optimistic" QFT with no ancillas

Error bounded by ϵ on all but $O(\epsilon) \cdot 2^n$ basis states

GDKM, J. Blue, T. Bergamaschi, C. Gidney, I. Chuang. arXiv:2505.00701

Fast quantum integer multiplication

$O(n^{1+\varepsilon})$ gates
No ancilla qubits

GDKM, N. Yao. arXiv:2403.18006

Shor's algorithm with:

$O(n^{2+\varepsilon})$ gates
$O(n^{1+\varepsilon})$ depth
$2n + O(n/\log n)$ total qubits