Fast quantum integer multiplication with very few ancillas

Gregory D. Kahanamoku-Meyer, Norman Y. Yao

October 19, 2023

Motivation

Arithmetic on quantum computers: why do we care?

Motivation

Arithmetic on quantum computers: why do we care?

OPEN

Classically verifiable quantum advantage from a computational Bell test

Gregory D. Kahanamoku-Meyer[1], Soonwon Choi[1], Umesh V. Vazirani[2] and Norman Y. Yao[1]

Existing experimental demonstrations of quantum computational advantage have had the limitation that verifying the correctness of the quantum device requires exponentially costly classical computations. Here we propose and analyse an interactive protocol for demonstrating quantum computational advantage, which is efficiently classically verifiable. Our protocol relies on a class of cryptographic tools called trapdoor claw-free functions. Although this type of function has been applied to quantum advantage protocols before, our protocol employs a surprising connection to Bell's inequality to avoid the need for a demanding cryptographic property called the adaptive hardcore bit, while maintaining essentially no increase in the quantum circuit complexity and no extra assumptions. Leveraging the relaxed cryptographic requirements of the protocol, we present two trapdoor claw-free function constructions, based on Rabin's function and the Diffie-Hellman problem, which have not been used in this context before. We also

Motivation

Arithmetic on quantum computers: why do we care?

OPEN

Classically verifiable quantum advantage from a computational Bell test

Gregory D. Kahanamoku-Meyer[1], Soonwon Choi[1], Umesh V. Vazirani[2] and Norman Y. Yao[1]

Existing experimental demonstrations of quantum computational advantage have had the limitation that verifying the correctness of the quantum device requires exponentially costly classical computations. Here we propose and analyse an interactive protocol for demonstrating quantum computational advantage, which is efficiently classically verifiable. Our protocol relies on a class of cryptographic tools called trapdoor claw-free functions. Although this type of function has been applied to quantum advantage protocols before, our protocol employs a surprising connection to Bell's inequality to avoid the need for a demanding cryptographic property called the adaptive hardcore bit, while maintaining essentially no increase in the quantum circuit complexity and no extra assumptions. Leveraging the relaxed cryptographic requirements of the protocol, we present two trapdoor claw-free function constructions, based on Rabin's function and the Diffie–Hellman problem, which have not been used in this context before. We also

Efficiently-verifiable advantage, using $n = 1024$ bit factoring

Motivation

Arithmetic on quantum computers: why do we care?

OPEN

Classically verifiable quantum advantage from a computational Bell test

Gregory D. Kahanamoku-Meyer[1], Soonwon Choi[1], Umesh V. Vazirani[2] and Norman Y. Yao[1]

Existing experimental demonstrations of quantum computational advantage have had the limitation that verifying the correctness of the quantum device requires exponentially costly classical computations. Here we propose and analyse an interactive protocol for demonstrating quantum computational advantage, which is efficiently classically verifiable. Our protocol relies on a class of cryptographic tools called trapdoor claw-free functions. Although this type of function has been applied to quantum advantage protocols before, our protocol employs a surprising connection to Bell's inequality to avoid the need for a demanding cryptographic property called the adaptive hardcore bit, while maintaining essentially no increase in the quantum circuit complexity and no extra assumptions. Leveraging the relaxed cryptographic requirements of the protocol, we present two trapdoor claw-free function constructions, based on Rabin's function and the Diffie–Hellman problem, which have not been used in this context before. We also

Efficiently-verifiable advantage, using $n = 1024$ bit factoring

Shor's algorithm: $10^{10}+$ gates

Arithmetic on quantum computers: why do we care?

OPEN

Classically verifiable quantum advantage from a computational Bell test

Gregory D. Kahanamoku-Meyer[1], Soonwon Choi[1], Umesh V. Vazirani[2] and Norman Y. Yao[1]

Existing experimental demonstrations of quantum computational advantage have had the limitation that verifying the correctness of the quantum device requires exponentially costly classical computations. Here we propose and analyse an interactive protocol for demonstrating quantum computational advantage, which is efficiently classically verifiable. Our protocol relies on a class of cryptographic tools called trapdoor claw-free functions. Although this type of function has been applied to quantum advantage protocols before, our protocol employs a surprising connection to Bell's inequality to avoid the need for a demanding cryptographic property called the adaptive hardcore bit, while maintaining essentially no increase in the quantum circuit complexity and no extra assumptions. Leveraging the relaxed cryptographic requirements of the protocol, we present two trapdoor claw-free function constructions, based on Rabin's function and the Diffie-Hellman problem, which have not been used in this context before. We also

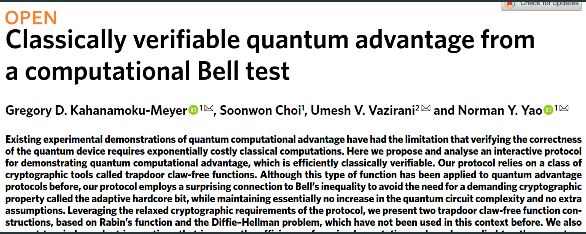
Efficiently-verifiable advantage, using $n = 1024$ bit factoring

Shor's algorithm: $10^{10}+$ gates

$\leftarrow$ Original $x^2 \bmod N$ proposal: $\lesssim 10^7$ gates, 7000 qubits

Motivation

Arithmetic on quantum computers: why do we care?


OPEN

Classically verifiable quantum advantage from a computational Bell test

Gregory D. Kahanamoku-Meyer[1], Soonwon Choi[1], Umesh V. Vazirani[2] and Norman Y. Yao[1]

Existing experimental demonstrations of quantum computational advantage have had the limitation that verifying the correctness of the quantum device requires exponentially costly classical computations. Here we propose and analyse an interactive protocol for demonstrating quantum computational advantage, which is efficiently classically verifiable. Our protocol relies on a class of cryptographic tools called trapdoor claw-free functions. Although this type of function has been applied to quantum advantage protocols before, our protocol employs a surprising connection to Bell's inequality to avoid the need for a demanding cryptographic property called the adaptive hardcore bit, while maintaining essentially no increase in the quantum circuit complexity and no extra assumptions. Leveraging the relaxed cryptographic requirements of the protocol, we present two trapdoor claw-free function constructions, based on Rabin's function and the Diffie-Hellman problem, which have not been used in this context before. We also

Efficiently-verifiable advantage, using $n = 1024$ bit factoring

Shor's algorithm: 10^{10}+ gates

$\leftarrow$ Original $x^2 \bmod N$ proposal: $\lesssim 10^7$ gates, 7000 qubits

$x^2 \bmod N$ with this result: $\gtrsim 10^6$ gates, 2000 qubits

Motivation

Arithmetic on quantum computers: why do we care?

Check for updates

OPEN

Classically verifiable quantum advantage from a compu

Gregory D. Kahar

Existing experimental of the quantum device for demonstrating qua cryptographic tools ca protocols before, our p property called the ad assumptions. Leveragi structions, based on R

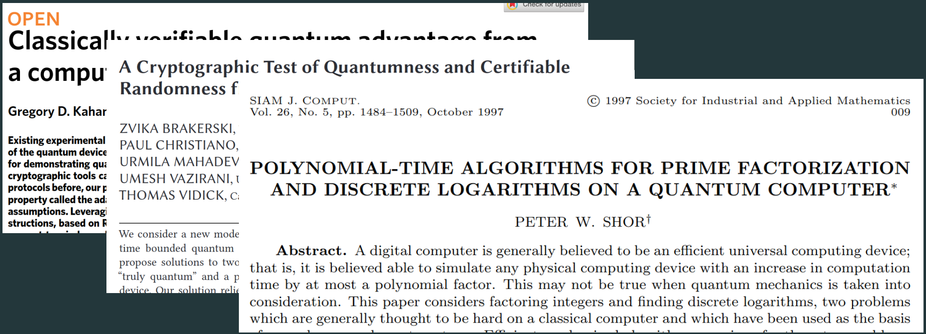
A Cryptographic Test of Quantumness and Certifiable Randomness from a Single Quantum Device

ZVIKA BRAKERSKI, Weizmann Institute of Science, Israel
PAUL CHRISTIANO, OpenAI, USA
URMILA MAHADEV, California Institute of Technology, USA
UMESH VAZIRANI, UC Berkeley, USA
THOMAS VIDICK, California Institute of Technology, USA

We consider a new model for the testing of untrusted quantum devices, consisting of a single polynomial time bounded quantum device interacting with a classical polynomial time verifier. In this model, we propose solutions to two tasks—a protocol for efficient classical verification that the untrusted device is "truly quantum" and a protocol for producing certifiable randomness from a single untrusted quantum device. Our solution relies on the existence of a new cryptographic primitive for constraining the power

Motivation

Arithmetic on quantum computers: why do we care?



OPEN

Classically verifiable quantum advantage from a compu

Gregory D. Kahar

Existing experimental of the quantum device for demonstrating qu cryptographic tools ca protocols before, our p property called the ad assumptions. Leveragi structions, based on R

A Cryptographic Test of Quantumness and Certifiable Randomness f

ZVIKA BRAKERSKI,
PAUL CHRISTIANO,
URMILA MAHADEV
UMESH VAZIRANI,
THOMAS VIDICK,

We consider a new mode time bounded quantum propose solutions to two "truly quantum" and a p device. Our solution relie

SIAM J. COMPUT. © 1997 Society for Industrial and Applied Mathematics
Vol. 26, No. 5, pp. 1484–1509, October 1997 009

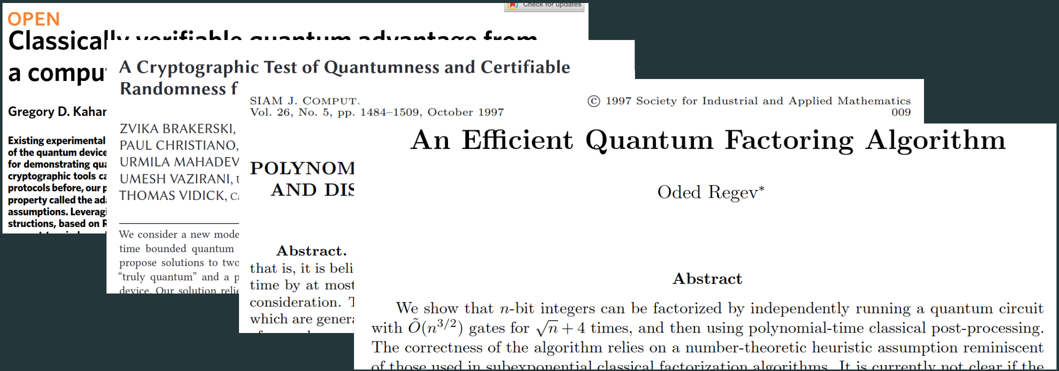
POLYNOMIAL-TIME ALGORITHMS FOR PRIME FACTORIZATION AND DISCRETE LOGARITHMS ON A QUANTUM COMPUTER*

PETER W. SHOR†

Abstract. A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis

Motivation

Arithmetic on quantum computers: why do we care?



Multiplication on quantum computers

Today's goal: implement the following unitaries

Multiplication on quantum computers

Today's goal: implement the following unitaries

$$\mathcal{U}_{q\times q} |x\rangle |y\rangle |0\rangle = |x\rangle |y\rangle |xy\rangle$$

Multiplication on quantum computers

Today's goal: implement the following unitaries

$$\mathcal{U}_{q\times q}\,|x\rangle\,|y\rangle\,|0\rangle = |x\rangle\,|y\rangle\,|xy\rangle$$

$$\mathcal{U}_{c\times q}(a)\,|x\rangle\,|0\rangle = |x\rangle\,|ax\rangle$$

Multiplication on quantum computers

Today's goal: implement the following unitaries

$$\mathcal{U}_{q\times q} |x\rangle |y\rangle |0\rangle = |x\rangle |y\rangle |xy\rangle$$

$$\mathcal{U}_{c\times q}(a) |x\rangle |0\rangle = |x\rangle |ax\rangle$$

... with as few gates and qubits as possible.

Multiplication on quantum computers

Today's goal: implement the following unitaries

$$\mathcal{U}_{q\times q} \left|x\right\rangle \left|y\right\rangle \left|w\right\rangle = \left|x\right\rangle \left|y\right\rangle \left|w + xy\right\rangle$$

$$\mathcal{U}_{c\times q}(a) \left|x\right\rangle \left|w\right\rangle = \left|x\right\rangle \left|w + ax\right\rangle$$

... **with as few gates and qubits as possible.**

Background: schoolbook multiplication

The “schoolbook” method: $xy = \sum_{ij}(2^i x_i)(2^j y_j) = \sum_{ij} 2^{i+j} x_i y_j$

				1	1	0	1
			$\times$	1	0	1	0
				1	0	1	0
		1	0	1	0		
+	1	0	1	0			
1	0	0	0	0	0	1	0

Background: schoolbook multiplication

The "schoolbook" method: $xy = \sum_{ij}(2^i x_i)(2^j y_j) = \sum_{ij} 2^{i+j} x_i y_j$

$$
\begin{array}{cccccccc}
 & & & & 1 & 1 & 0 & 1 \\
 & & & \times & 1 & 0 & 1 & 0 \\
\hline
 & & & & 1 & 0 & 1 & 0 \\
 & & 1 & 0 & 1 & 0 & & \\
+ & 1 & 0 & 1 & 0 & & & \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
$$

Running time: $\mathcal{O}(n^2)$ operations

Background: schoolbook multiplication

Given two n-bit numbers x and y, what if we use base $b = 2^{n/2}$?

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

Background: schoolbook multiplication

Given two n-bit numbers x and y, what if we use 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: schoolbook multiplication

Given two n-bit numbers x and y, what if we use base $b = 2^{n/2}$?

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

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

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

$$xy = x_1y_1b^2 + (x_0y_1 + x_1y_0)b + x_0y_0$$

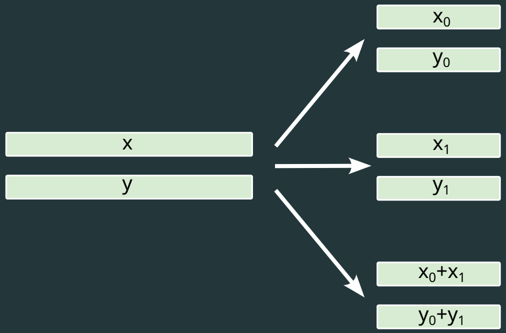
Observation: $x_0y_1 + x_1y_0 = (x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_0y_0$

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

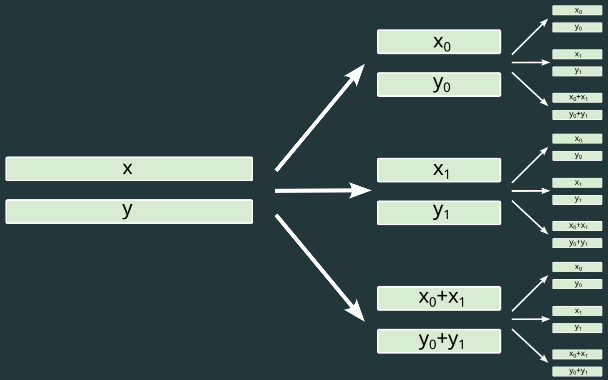
1. x_1y_1
2. x_0y_0
3. $(x_1 + x_0)(y_1 + y_0)$

Computational cost: $3(n/2)^2 = \frac{3}{4}n^2 = \mathcal{O}(n^2)$

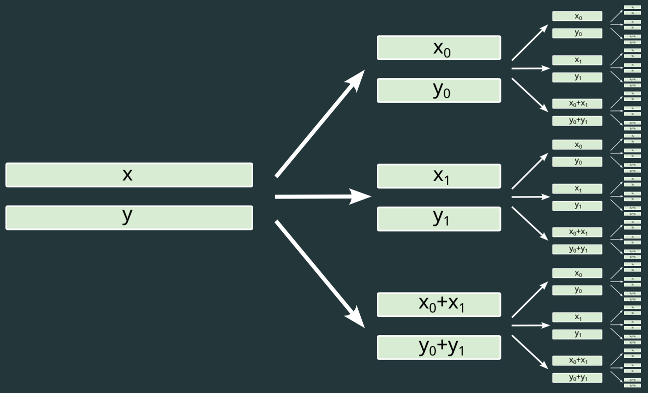
Background: Karatsuba multiplication



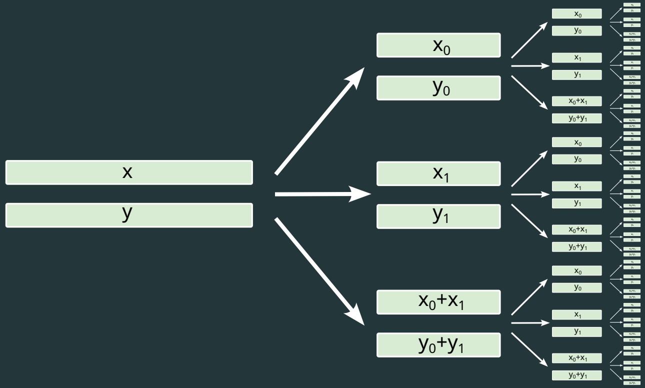
Background: Karatsuba multiplication



Background: Karatsuba multiplication

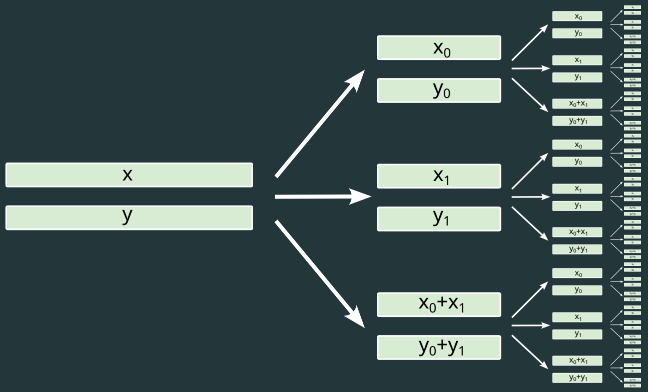


Background: Karatsuba multiplication



Depth: $d = \log_2 n$

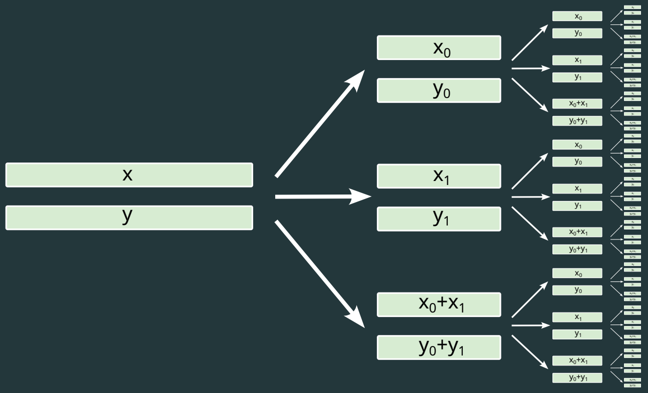
Background: Karatsuba multiplication



Depth: $d = \log_2 n$

Operations: 3^d

Background: Karatsuba multiplication



Depth: $d = \log_2 n$

Operations: 3^d

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

Background: Karatsuba multiplication

Question: why don't we always do this, classically?

Answer: the extra complexity isn't always worth it!

Background: Karatsuba multiplication

Question: why don't we always do this, classically?

Answer: the extra complexity isn't always worth it!

… but for large enough values, it is

Background: Karatsuba multiplication

Question: why don't we always do this, classically?

Answer: the extra complexity isn't always worth it!

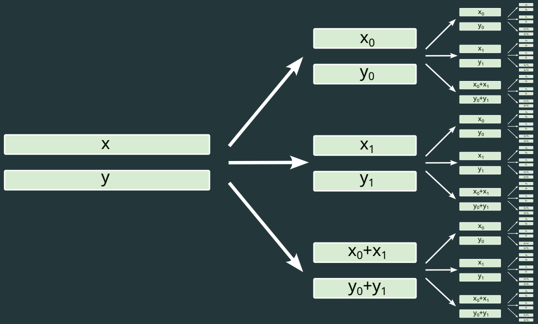
... but for large enough values, it is

GNU multiple-precision arithmetic library cutoff: 2176 bit numbers

Can these fast circuits be made quantum?

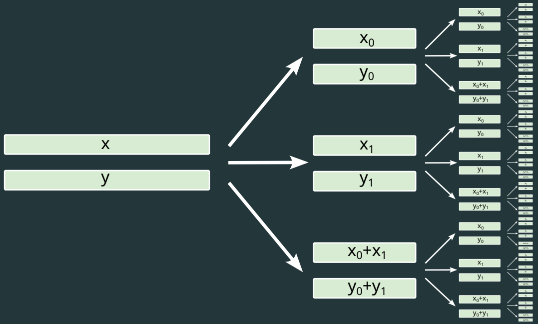
Can these fast circuits be made quantum?

Challenge: making recursive algorithms reversible



Can these fast circuits be made quantum?

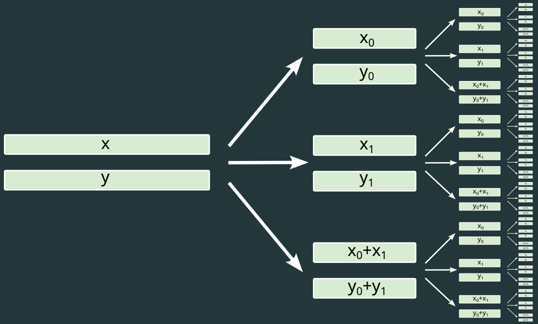
Challenge: making recursive algorithms reversible



Work	Qubits
Kowada et al. '06	$\mathcal{O}(n^{1.58\cdots})$
Parent et al. '18	$\mathcal{O}(n^{1.43\cdots})$
Gidney '19	$\mathcal{O}(n)$

Can these fast circuits be made quantum?

Challenge: making recursive algorithms reversible

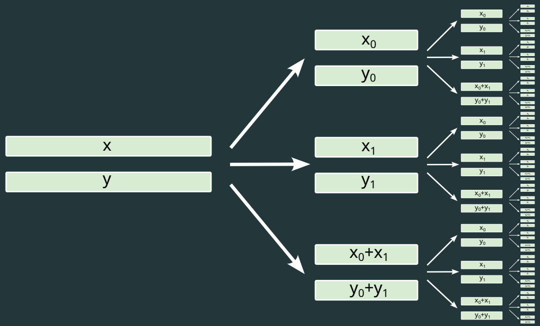


Work	Qubits
Kowada et al. '06	$\mathcal{O}(n^{1.58\cdots})$
Parent et al. '18	$\mathcal{O}(n^{1.43\cdots})$
Gidney '19	$\mathcal{O}(n)$

Gidney '19 requires over **12,000 ancilla qubits** for 2048-bit multiplication.

Can these fast circuits be made quantum?

Challenge: making recursive algorithms reversible



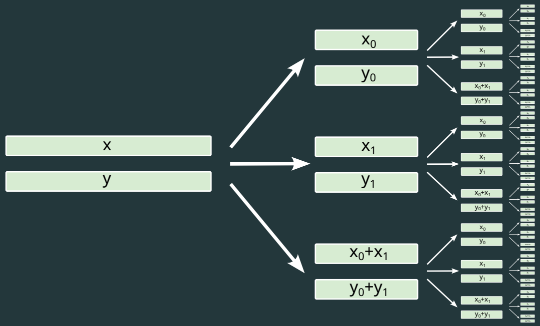
Work	Qubits
Kowada et al. '06	$\mathcal{O}(n^{1.58\cdots})$
Parent et al. '18	$\mathcal{O}(n^{1.43\cdots})$
Gidney '19	$\mathcal{O}(n)$

Gidney '19 requires over **12,000 ancilla qubits** for 2048-bit multiplication.

Is it possible to do better?

Can these fast circuits be made quantum?

Challenge: making recursive algorithms reversible



Work	Qubits
Kowada et al. '06	$\mathcal{O}(n^{1.58\cdots})$
Parent et al. '18	$\mathcal{O}(n^{1.43\cdots})$
Gidney '19	$\mathcal{O}(n)$

Gidney '19 requires over **12,000 ancilla qubits** for 2048-bit multiplication.

Is it possible to do better?

Result: Fast multiplication using 1 ancilla

A fundamentally quantum way of doing arithmetic

[Draper '04]: Arithmetic in Fourier space

$$|xy\rangle = \mathrm{QFT}^{-1} \sum_z \exp\left(\frac{2\pi i xyz}{2^n}\right) |z\rangle$$

A fundamentally quantum way of doing arithmetic

[Draper '04]: Arithmetic in Fourier space

$$|xy\rangle = \mathrm{QFT}^{-1} \sum_z \exp\left(\frac{2\pi i x y z}{2^n}\right) |z\rangle$$

How to implement $|x\rangle\,|y\rangle\,|0\rangle \rightarrow |x\rangle\,|y\rangle\,|xy\rangle$?

A fundamentally quantum way of doing arithmetic

[Draper '04]: Arithmetic in Fourier space

$$|xy\rangle = \mathrm{QFT}^{-1} \sum_z \exp\left(\frac{2\pi i xyz}{2^n}\right) |z\rangle$$

How to implement $|x\rangle\,|y\rangle\,|0\rangle \rightarrow |x\rangle\,|y\rangle\,|xy\rangle$?

1) Generate $|x\rangle\,|y\rangle \sum_z |z\rangle$

A fundamentally quantum way of doing arithmetic

[Draper '04]: Arithmetic in Fourier space

$$|xy\rangle = \mathrm{QFT}^{-1} \sum_z \exp\left(\frac{2\pi i x y z}{2^n}\right) |z\rangle$$

How to implement $|x\rangle\,|y\rangle\,|0\rangle \to |x\rangle\,|y\rangle\,|xy\rangle$?

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

A fundamentally quantum way of doing arithmetic

[Draper '04]: Arithmetic in Fourier space

$$|xy\rangle = \mathrm{QFT}^{-1} \sum_z \exp\left(\frac{2\pi i x y z}{2^n}\right) |z\rangle$$

How to implement $|x\rangle\,|y\rangle\,|0\rangle \to |x\rangle\,|y\rangle\,|xy\rangle$?

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

A fundamentally quantum way of doing arithmetic

How do we apply $\exp\left(\frac{2\pi ixyz}{2^n}\right)$?

A fundamentally quantum way of doing arithmetic

How do we apply $\exp\left(\frac{2\pi i xyz}{2^n}\right)$?

$$xy = \sum_{i,j} 2^i 2^j x_i y_j$$

A fundamentally quantum way of doing arithmetic

How do we apply $\exp\left(\frac{2\pi i xyz}{2^n}\right)$?

$$xyz = \sum_{i,j,k} 2^i 2^j 2^k x_i y_j z_k$$

$$\exp\left(\frac{2\pi i xyz}{2^n}\right) = \prod_{i,j,k} \exp\left(\frac{2\pi i 2^{i+j+k}}{2^n} x_i y_j z_k\right)$$

A fundamentally quantum way of doing arithmetic

How do we apply $\exp\left(\frac{2\pi i xyz}{2^n}\right)$?

$$xyz = \sum_{i,j,k} 2^i 2^j 2^k x_i y_j z_k$$

$$\exp\left(\frac{2\pi i xyz}{2^n}\right) = \prod_{i,j,k} \exp\left(\frac{2\pi i 2^{i+j+k}}{2^n} x_i y_j z_k\right)$$

x_i, y_j, z_k are binary values—apply phase only if they all are equal to 1!

A fundamentally quantum way of doing arithmetic

How do we apply $\exp\left(\frac{2\pi i xyz}{2^n}\right)$?

$$xyz = \sum_{i,j,k} 2^i 2^j 2^k x_i y_j z_k$$

$$\exp\left(\frac{2\pi i xyz}{2^n}\right) = \prod_{i,j,k} \exp\left(\frac{2\pi i 2^{i+j+k}}{2^n} x_i y_j z_k\right)$$

x_i, y_j, z_k are binary values—apply phase only if they all are equal to 1!

A series of CCR_ϕ gates between the bits of $|x\rangle$, $|y\rangle$, and $|z\rangle$!

A fundamentally quantum way of doing arithmetic

$$\exp\left(\frac{2\pi i xyz}{2^n}\right) = \prod_{i,j,k} \exp\left(\frac{2\pi i 2^{i+j+k}}{2^n} x_i y_j z_k\right)$$

The downside:

A fundamentally quantum way of doing arithmetic

$$\exp\left(\frac{2\pi i xyz}{2^n}\right) = \prod_{i,j,k} \exp\left(\frac{2\pi i 2^{i+j+k}}{2^n} x_i y_j z_k\right)$$

The downside: For n-bit numbers, this requires n^3 gates!

A fundamentally quantum way of doing arithmetic

$$\exp\left(\frac{2\pi i xyz}{2^n}\right) = \prod_{i,j,k} \exp\left(\frac{2\pi i 2^{i+j+k}}{2^n} x_i y_j z_k\right)$$

The downside: For n-bit numbers, this requires n^3 gates!

A modest improvement: classical-quantum multiplication $\mathcal{U}(a)\left|x\right\rangle\left|0\right\rangle = \left|x\right\rangle\left|ax\right\rangle$

A fundamentally quantum way of doing arithmetic

$$\exp\left(\frac{2\pi i xyz}{2^n}\right) = \prod_{i,j,k} \exp\left(\frac{2\pi i 2^{i+j+k}}{2^n} x_i y_j z_k\right)$$

The downside: For n-bit numbers, this requires n^3 gates!

A modest improvement: classical-quantum multiplication $\mathcal{U}(a)\left|x\right\rangle\left|0\right\rangle = \left|x\right\rangle\left|ax\right\rangle$

$$\exp\left(\frac{2\pi i axz}{2^n}\right) = \prod_{i,j} \exp\left(\frac{2\pi i a 2^{i+j}}{2^n} x_i z_j\right)$$

Here: $\mathcal{O}(n^2)$ controlled phase rotations (matches Schoolbook algorithm)

Fast quantum multiplication

Main question: Can we combine fast multiplication with Fourier arithmetic to get the benefits of both?

Fast classical-quantum multiplication

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

Fast classical-quantum multiplication

Goal: Apply phase $\exp\left(\frac{2\pi i a}{2^n} xz\right)$; x and z are quantum

Fast classical-quantum multiplication

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

Fast classical-quantum multiplication

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

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

Fast classical-quantum multiplication

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

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

Previously:

$$\exp(i\phi xz) = \prod_{i,j} \exp\left(i\phi 2^{i+j} x_i z_j\right)$$

Fast classical-quantum multiplication

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

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

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

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

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

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

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$(\phi)\,|x\rangle\,|z\rangle = \exp{(i\phi xz)}\,|x\rangle\,|z\rangle$

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

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

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

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(ϕ) $|x\rangle |z\rangle = \exp(i\phi xz) |x\rangle |z\rangle$

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

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 $\text{PhaseProduct}(\phi)\left|x\right\rangle\left|z\right\rangle = \exp\left(i\phi xz\right)\left|x\right\rangle\left|z\right\rangle$

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

Plugging in reordered Karatsuba:

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

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

Fast classical-quantum multiplication

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

We want to split the phase ϕxz into the sum of many phases, which are easy to implement.

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

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

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

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

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

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

Use quantum addition circuits.

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

Total number of ancillas:

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

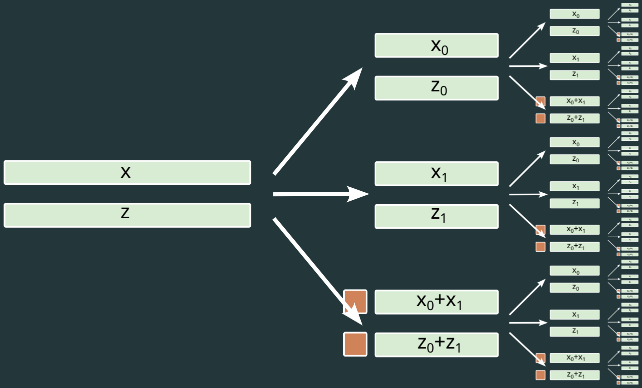
Use quantum addition circuits.

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

Total number of ancillas: $\mathcal{O}(\log n)$

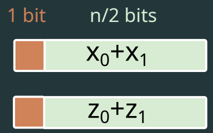
How many qubits do we need?

Total number of ancillas: $\mathcal{O}(\log n)$



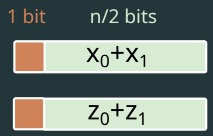
How many qubits do we need?

Total number of ancillas: $\mathcal{O}(\log n)$



How many qubits do we need?

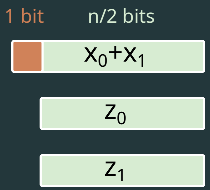
Total number of ancillas: 2



Idea: "Shave off" the high bit before recursing

How many qubits do we need?

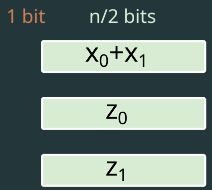
Total number of ancillas: 1



Idea: "Shave off" the high bit before recursing

How many qubits do we need?

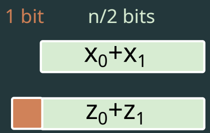
Total number of ancillas: 1



Idea: “Shave off” the high bit before recursing

How many qubits do we need?

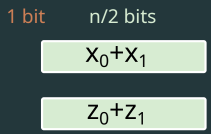
Total number of ancillas: 1



Idea: “Shave off” the high bit before recursing

How many qubits do we need?

Total number of ancillas: 1



Idea: “Shave off” the high bit before recursing

Making it go faster

So far: $\mathcal{O}(n^{1.58})$ gates using 1 ancilla

Making it go faster

So far: $\mathcal{O}(n^{1.58})$ **gates** using **1 ancilla**

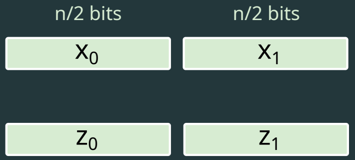
Can we make it go faster?

Background: Toom-Cook multiplication

Let $b = 2^{n/2}$.

$$x = x_1 b + x_0$$

$$z = z_1 b + z_0$$

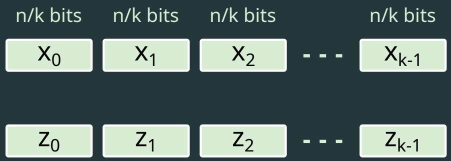


Background: Toom-Cook multiplication

Let $b = 2^{n/k}$.

$$x = \sum_{i=0}^{k-1} x_i b^i$$

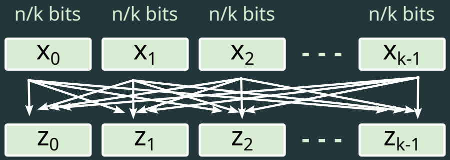
$$z = \sum_{i=0}^{k-1} z_i b^i$$



Background: Toom-Cook multiplication

Let $b = 2^{n/k}$.

$$xz = \left(\sum_{i=0}^{k-1} x_i b^i\right)\left(\sum_{i=0}^{k-1} z_i b^i\right)$$



Schoolbook: k^2 multiplications of size n/k

Background: Toom-Cook multiplication

$$x(b) = \sum_{i=0}^{k-1} x_i b^i$$

$$z(b) = \sum_{i=0}^{k-1} z_i b^i$$

Background: Toom-Cook multiplication

$$x(b) = \sum_{i=0}^{k-1} x_i b^i$$

$$z(b) = \sum_{i=0}^{k-1} z_i b^i$$

$$p(b) = x(b)z(b)$$

Background: Toom-Cook multiplication

$$x(b) = \sum_{i=0}^{k-1} x_i b^i$$

$$z(b) = \sum_{i=0}^{k-1} z_i b^i$$

$$p(b) = x(b)z(b)$$

$$p(2^{n/k}) = x(2^{n/k})z(2^{n/k})$$

Background: Toom-Cook multiplication

$$x(b) = \sum_{i=0}^{k-1} x_i b^i$$

$$z(b) = \sum_{i=0}^{k-1} z_i b^i$$

$$p(b) = x(b)z(b)$$

$$p(2^{n/k}) = x(2^{n/k})z(2^{n/k})$$

Facts:

- For any point w, $p(w) = x(w)z(w)$

Background: Toom-Cook multiplication

$$x(b) = \sum_{i=0}^{k-1} x_i b^i$$

$$z(b) = \sum_{i=0}^{k-1} z_i b^i$$

$$p(b) = x(b)z(b)$$

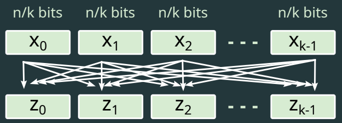
$$p(2^{n/k}) = x(2^{n/k})z(2^{n/k})$$

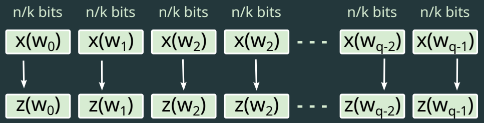
Facts:

- For any point w, $p(w) = x(w)z(w)$
- $p(b)$ has degree $2(k-1) \Rightarrow$ uniquely determined by $q = 2(k-1)+1$ points w_ℓ!

Background: Toom-Cook multiplication

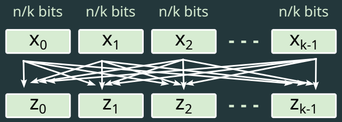
1. Compute $x(w_\ell)$, $z(w_\ell)$ at q points w_ℓ

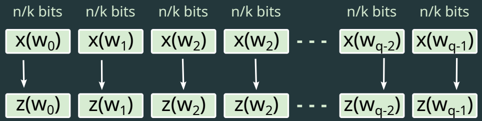




Only $2k - 1$ multiplications of size n/k!

1. Compute $x(w_\ell)$, $z(w_\ell)$ at q points w_ℓ
2. Pointwise multiply

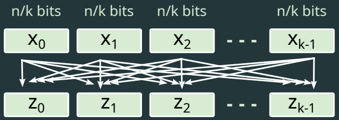


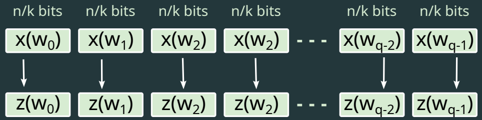


Only $2k - 1$ multiplications of size n/k!

Background: Toom-Cook multiplication

1. Compute $x(w_\ell)$, $z(w_\ell)$ at q points w_ℓ
2. Pointwise multiply
3. Interpolate $p(b)$

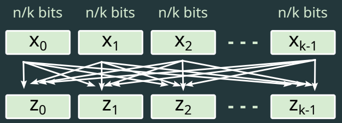


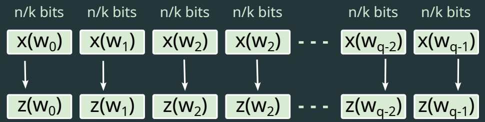


Only $2k - 1$ multiplications of size n/k!

Background: Toom-Cook multiplication

1. Compute $x(w_\ell)$, $z(w_\ell)$ at q points w_ℓ
2. Pointwise multiply
3. Interpolate $p(b)$
4. Evaluate $p(2^{n/k})$





Only $2k - 1$ multiplications of size n/k!

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with $k = 2$

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with $k = 2$

$$x(b) = x_1 b + x_0$$
$$z(b) = z_1 b + z_0$$

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with $k = 2$

$$x(b) = x_1 b + x_0$$
$$z(b) = z_1 b + z_0$$

$p(b) = x(b)z(b)$ has degree 2

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with $k = 2$

Let $w \in \{0, \infty, 1\}$

$$x(b) = x_1 b + x_0$$
$$z(b) = z_1 b + z_0$$

$p(b) = x(b)z(b)$ has degree 2

$$p(b) = p(\infty)b^2 + [p(1) - p(\infty) - p(0)]\, b + p(0)$$

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with $k = 2$

Let $w \in \{0, \infty, 1\}$

$$x(b) = x_1 b + x_0$$
$$z(b) = z_1 b + z_0$$

$p(b) = x(b)z(b)$ has degree 2

$$p(b) = x(\infty)z(\infty)b^2 + [x(1)z(1) - x(\infty)z(\infty) - x(0)z(0)]\, b + x(0)z(0)$$

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with $k = 2$

$$x(b) = x_1 b + x_0$$

$$z(b) = z_1 b + z_0$$

$p(b) = x(b)z(b)$ has degree 2

Let $w \in \{0, \infty, 1\}$

$$x(0) = x_0$$

$$x(\infty) = x_1$$

$$x(1) = x_0 + x_1$$

$$p(b) = x(\infty)z(\infty)b^2 + [x(1)z(1) - x(\infty)z(\infty) - x(0)z(0)]\, b + x(0)z(0)$$

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with k = 2

$$x(b) = x_1 b + x_0$$

$$z(b) = z_1 b + z_0$$

$p(b) = x(b)z(b)$ has degree 2

Let $w \in \{0, \infty, 1\}$

$$x(0) = x_0$$

$$x(\infty) = x_1$$

$$x(1) = x_0 + x_1$$

$$p(b) = x_1 z_1 b^2 + \left[(x_0 + x_1)(z_0 + z_1) - x_1 z_1 - x_0 z_0\right] b + x_0 z_0$$

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with k = 2

$$x(b) = x_1 b + x_0$$
$$z(b) = z_1 b + z_0$$

$p(b) = x(b)z(b)$ has degree 2

Let $w \in \{0, \infty, 1\}$

$$x(0) = x_0$$
$$x(\infty) = x_1$$
$$x(1) = x_0 + x_1$$

$$p(2^{n/2}) = x_1 z_1 2^n + \left[(x_0 + x_1)(z_0 + z_1) - x_1 z_1 - x_0 z_0\right] 2^{n/2} + x_0 z_0$$

Karatsuba in the language of Toom-Cook

Karatsuba is Toom-Cook with k = 2

$$x(b) = x_1 b + x_0$$
$$z(b) = z_1 b + z_0$$

$p(b) = x(b)z(b)$ has degree 2

Let $w \in \{0, \infty, 1\}$

$$x(0) = x_0$$
$$x(\infty) = x_1$$
$$x(1) = x_0 + x_1$$

$$xz = x_1 z_1 2^n + \left[(x_0 + x_1)(z_0 + z_1) - x_1 z_1 - x_0 z_0\right] 2^{n/2} + x_0 z_0$$

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

Complexity vs. k

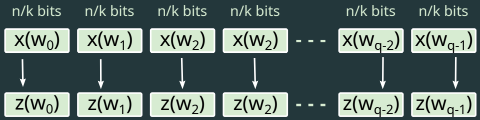
Toom-Cook has asymptotic complexity $\mathcal{O}(n^{log_k(2k-1)})$

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

These are the gate counts for our classical-quantum multiplication!

Overhead moves to classical precomputation

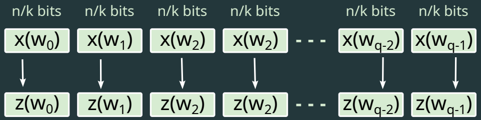
1. Compute $x(w_\ell), z(w_\ell)$ at q points w_ℓ
2. Pointwise multiply
3. Interpolate $p(b)$
4. Evaluate $p(2^{n/k})$



$$\phi xz = \sum_{\ell=0}^{2k-2} \phi_\ell x(w_\ell) z(w_\ell) \tag{1}$$

Overhead moves to classical precomputation

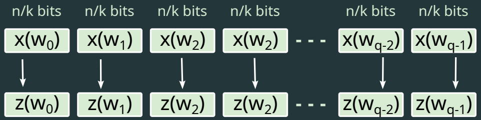
1. Compute $x(w_\ell)$, $z(w_\ell)$ at q points w_ℓ
2. Pointwise multiply
3. ~~Interpolate $p(b)$~~
4. ~~Evaluate $p(2^{n/k})$~~



$$\phi xz = \sum_{\ell=0}^{2k-2} \phi_\ell x(w_\ell) z(w_\ell) \tag{1}$$

Overhead moves to classical precomputation

1. Compute $x(w_\ell), z(w_\ell)$ at q points w_ℓ
2. Pointwise multiply
3. ~~Interpolate $p(b)$~~
4. ~~Evaluate $p(2^{n/k})$~~



$$\phi xz = \sum_{\ell=0}^{2k-2} \phi_\ell x(w_\ell) z(w_\ell) \tag{1}$$

Much of the overhead has moved to classical precomputation!

Fast quantum-quantum multiplication

Goal: $\mathcal{U} |x\rangle |y\rangle |0\rangle = |x\rangle |y\rangle |xy\rangle$

Fast quantum-quantum multiplication

Goal: Apply phase $\exp\left(\frac{2\pi i}{2^n}xyz\right)$; x, y, and z are quantum

Fast quantum-quantum multiplication

Goal: Implement $\text{PhaseTripleProduct}(\phi) \left|x\right\rangle \left|y\right\rangle \left|z\right\rangle = \exp\left(i\phi xyz\right) \left|x\right\rangle \left|y\right\rangle \left|z\right\rangle$

Fast quantum-quantum multiplication

Goal: Implement PhaseTripleProduct(ϕ) $|x\rangle |y\rangle |z\rangle = \exp(i\phi xyz) |x\rangle |y\rangle |z\rangle$

Previously:

$$\exp(i\phi xyz) = \prod_{i,j,k} \exp\left(i\phi 2^{i+j+k} x_i y_j z_k\right)$$

(n^3 doubly-controlled phase rotations)

Fast quantum-quantum multiplication

Goal: Implement PhaseTripleProduct(ϕ) $|x\rangle |y\rangle |z\rangle = \exp(i\phi xyz) |x\rangle |y\rangle |z\rangle$

Previously:

$$\exp(i\phi xyz) = \prod_{i,j,k} \exp\left(i\phi 2^{i+j+k} x_i y_j z_k\right)$$

(n^3 doubly-controlled phase rotations)

Question: How would you classically compute a triple product like *xyz*?

Fast quantum-quantum multiplication

Goal: Implement PhaseTripleProduct(ϕ) $|x\rangle\,|y\rangle\,|z\rangle = \exp\left(i\phi xyz\right)|x\rangle\,|y\rangle\,|z\rangle$

Previously:

$$\exp\left(i\phi xyz\right) = \prod_{i,j,k} \exp\left(i\phi 2^{i+j+k} x_i y_j z_k\right)$$

(n^3 doubly-controlled phase rotations)

Question: How would you classically compute a triple product like xyz?

Answer: Use parentheses! $xyz = x(yz)$.

Fast quantum-quantum multiplication

Goal: Implement PhaseTripleProduct(ϕ) $|x\rangle |y\rangle |z\rangle = \exp(i\phi xyz) |x\rangle |y\rangle |z\rangle$

Previously:

$$\exp(i\phi xyz) = \prod_{i,j,k} \exp\left(i\phi 2^{i+j+k} x_i y_j z_k\right)$$

(n^3 doubly-controlled phase rotations)

Question: How would you classically compute a triple product like xyz?

Answer: Use parentheses! $xyz = x(yz)$. Then asymptotic cost is the same

Fast quantum-quantum multiplication

Goal: Implement PhaseTripleProduct(ϕ) $|x\rangle\,|y\rangle\,|z\rangle = \exp(i\phi xyz)\,|x\rangle\,|y\rangle\,|z\rangle$

Previously:

$$\exp(i\phi xyz) = \prod_{i,j,k} \exp\left(i\phi 2^{i+j+k} x_i y_j z_k\right) \qquad (n^3 \text{ doubly-controlled phase rotations})$$

Question: How would you classically compute a triple product like xyz?

Answer: Use parentheses! $xyz = x(yz)$. Then asymptotic cost is the same

Doesn't work in the phase!!

Generalizing Toom-Cook

Goal: Compute xyz “all at once”

Generalizing Toom-Cook

Goal: Compute xyz "all at once"

Before

$$p(b) = x(b)z(b)$$

$p(b)$ has degree $q = 2k - 1$

Generalizing Toom-Cook

Goal: Compute xyz "all at once"

Before

$$p(b) = x(b)z(b)$$

$p(b)$ has degree $q = 2k - 1$

Now

$$p(b) = x(b)y(b)z(b)$$

$p(b)$ has degree $q = 3k - 2$

Example: Generalizing Karatsuba's method

For $k = 2$, we have $q = 4$. Using $w_i \in \{0, \infty, 1, -1\}$:

Example: Generalizing Karatsuba's method

For $k = 2$, we have $q = 4$. Using $w_i \in \{0, \infty, 1, -1\}$:

$$
\begin{aligned}
xyz = & (2^{3n/2} - 2^{n/2}) x_1 y_1 z_1 \\
& + \frac{1}{2}(2^n + 2^{n/2})(x_0 + x_1)(y_0 + y_1)(z_0 + z_1) \\
& + \frac{1}{2}(2^n - 2^{n/2})(x_0 - x_1)(y_0 - y_1)(z_0 - z_1) \\
& + (1 - 2^n) x_0 y_0 z_0
\end{aligned}
$$

Example: Generalizing Karatsuba's method

For $k = 2$, we have $q = 4$. Using $w_i \in \{0, \infty, 1, -1\}$:

$$\begin{aligned} xyz =& (2^{3n/2} - 2^{n/2}) x_1 y_1 z_1 \\ &+ \frac{1}{2}(2^n + 2^{n/2})(x_0 + x_1)(y_0 + y_1)(z_0 + z_1) \\ &+ \frac{1}{2}(2^n - 2^{n/2})(x_0 - x_1)(y_0 - y_1)(z_0 - z_1) \\ &+ (1 - 2^n) x_0 y_0 z_0 \end{aligned}$$

Only 4 multiplications of length $n/2$ instead of 8!

Example: Generalizing Karatsuba's method

Only 4 multiplications of length $n/2$, instead of 8!

Recursion relation: $T(n) \sim 4T(n/2)$

Example: Generalizing Karatsuba's method

Only 4 multiplications of length $n/2$, instead of 8!

Recursion relation: $T(n) \sim 4T(n/2)$ thus: $T(n) = \mathcal{O}(n^2)$

Example: Generalizing Karatsuba's method

Only 4 multiplications of length $n/2$, instead of 8!

Recursion relation: $T(n) \sim 4T(n/2)$ thus: $T(n) = \mathcal{O}(n^2)$

As before: $k > 2$ is faster.

Example: Generalizing Karatsuba's method

Only 4 multiplications of length $n/2$, instead of 8!

Recursion relation: $T(n) \sim 4T(n/2)$ thus: $T(n) = \mathcal{O}(n^2)$

As before: $k > 2$ is faster.

k	Gates $\mathcal{O}(n^{\log_k(3k-2)})$
1*	$\mathcal{O}(n^3)$
2	$\mathcal{O}(n^2)$
3	$\mathcal{O}(n^{1.77\cdots})$
4	$\mathcal{O}(n^{1.66\cdots})$
5	$\mathcal{O}(n^{1.59\cdots})$
6	$\mathcal{O}(n^{1.55\cdots})$
$\vdots$	$\vdots$

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively
- Sandwiched by QFTs, this implements multiplication

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively
- Sandwiched by QFTs, this implements multiplication

Next up:

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively
- Sandwiched by QFTs, this implements multiplication

Next up:

- Sub-quadratic-time exact QFT with 1 ancilla

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively
- Sandwiched by QFTs, this implements multiplication

Next up:

- Sub-quadratic-time exact QFT with 1 ancilla
- Depth

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively
- Sandwiched by QFTs, this implements multiplication

Next up:

- Sub-quadratic-time exact QFT with 1 ancilla
- Depth
- Modular multiplication

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively
- Sandwiched by QFTs, this implements multiplication

Next up:

- Sub-quadratic-time exact QFT with 1 ancilla
- Depth
- Modular multiplication
- Applications

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively
- Sandwiched by QFTs, this implements multiplication

Next up:

- Sub-quadratic-time exact QFT with 1 ancilla
- Depth
- Modular multiplication
- Applications
    - Shor's algorithm

Summary so far

- Circuits for phase rotations ϕxz or ϕxyz in sub-quadratic time, using 1 or 2 ancillas respectively
- Sandwiched by QFTs, this implements multiplication

Next up:

- Sub-quadratic-time exact QFT with 1 ancilla
- Depth
- Modular multiplication
- Applications
  - Shor's algorithm
  - Efficiently-verifiable quantum advantage

Fast exact quantum Fourier transform

[Cleve and Watrous 2000]: QFT can be defined recursively.

Fast exact quantum Fourier transform

[Cleve and Watrous 2000]: QFT can be defined recursively.

For any $m < n$, we may implement QFT_{2^n}:

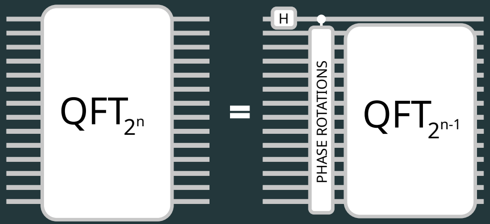
1. Apply QFT_{2^m} on first m qubits
2. Apply phase rotation $2\pi xz/2^n$
   - $|x\rangle$ is value of first m qubits
   - $|z\rangle$ is value of final $n-m$ qubits
3. Apply $\mathrm{QFT}_{2^{n-m}}$ on final $n-m$ qubits

Fast exact quantum Fourier transform

[Cleve and Watrous 2000]: QFT can be defined recursively.

For any $m < n$, we may implement QFT_{2^n}:

1. Apply QFT_{2^m} on first m qubits
2. Apply phase rotation $2\pi xz/2^n$
    - $|x\rangle$ is value of first m qubits
    - $|z\rangle$ is value of final $n - m$ qubits
3. Apply $\text{QFT}_{2^{n-m}}$ on final $n - m$ qubits

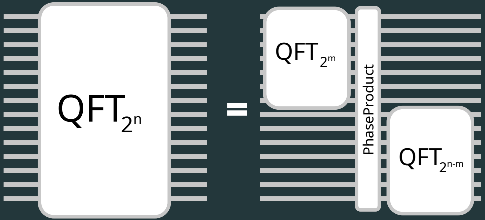


Fast exact quantum Fourier transform

[Cleve and Watrous 2000]: QFT can be defined recursively.

For any $m < n$, we may implement QFT_{2^n}:

1. Apply QFT_{2^m} on first m qubits
2. Apply phase rotation $2\pi xz/2^n$
    - $|x\rangle$ is value of first m qubits
    - $|z\rangle$ is value of final $n-m$ qubits
3. Apply $\mathrm{QFT}_{2^{n-m}}$ on final $n-m$ qubits

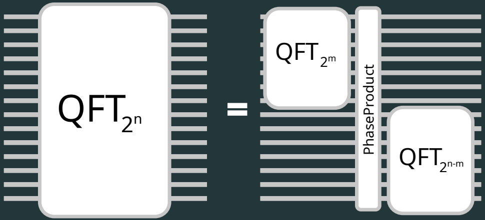


Fast exact quantum Fourier transform

[Cleve and Watrous 2000]: QFT can be defined recursively.

For any $m < n$, we may implement QFT_{2^n}:

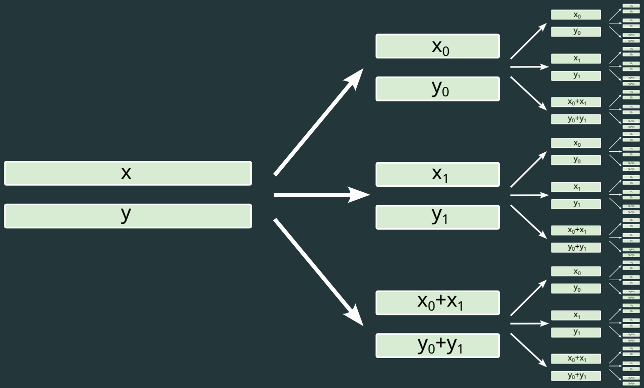
1. Apply QFT_{2^m} on first m qubits
2. Apply phase rotation $2\pi xz/2^n$
    - $|x\rangle$ is value of first m qubits
    - $|z\rangle$ is value of final $n - m$ qubits
3. Apply $\mathrm{QFT}_{2^{n-m}}$ on final $n - m$ qubits



Immediately gives us sub-quadratic exact QFT using only 1 ancilla.

Depth considerations

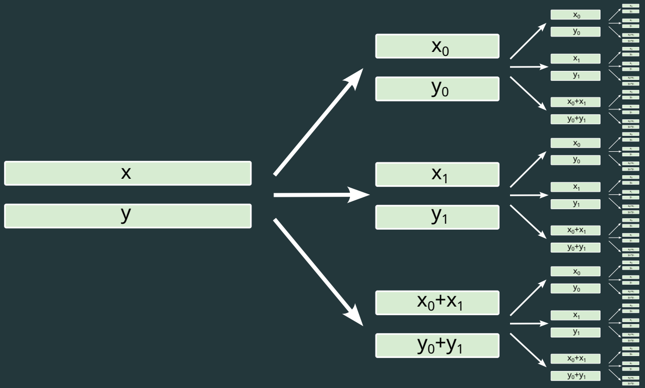
Parallelization is natural.



Depth considerations

Parallelization is natural.

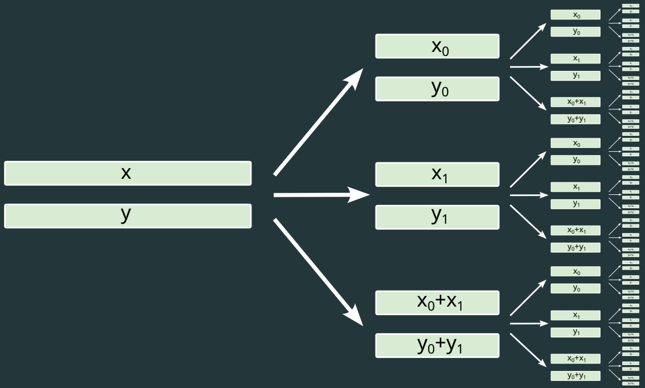
We have k sub-registers to work with—can do k sub-products in parallel.



Depth considerations

Parallelization is natural.

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

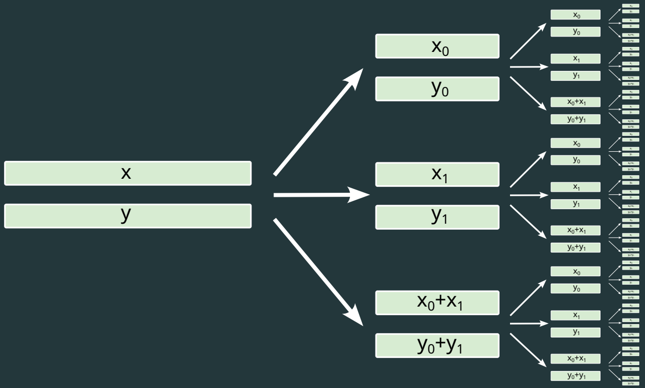


Depth: PhaseProduct in $\mathcal{O}(n^{\log_k 2})$ and PhaseTripleProduct in $\mathcal{O}(n^{\log_k 3})$ using a few more ancillas

Depth considerations

Parallelization is natural.

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



Challenge for multiply: How to do the QFT in sublinear depth with even $\mathcal{O}(n)$ ancillas?

So far: have been using phase

$$\exp\left(2\pi i \frac{xyz}{2^n}\right)$$

Modular arithmetic

So far: have been using phase

$$\exp\left(2\pi i \frac{xyz}{2^n}\right)$$

(denominator matches order of QFT)

Modular arithmetic

So far: have been using phase

$$\exp\left(2\pi i \frac{xyz}{2^n}\right)$$

(denominator matches order of QFT)

Observation:

$$\exp\left(2\pi i \frac{xyz}{N}\right) = \exp\left(2\pi i \frac{(xy \bmod N)z}{N}\right)$$

Modular arithmetic

Goal: only use n bits for output modulo N

Observation:

$$\exp\left(2\pi i \frac{xyz}{N}\right) = \exp\left(2\pi i \frac{(xy \bmod N)z}{N}\right)$$

Define

$$w = \frac{xy \bmod N}{N}$$

Modular arithmetic

Goal: only use n bits for output modulo N

Observation:

$$\exp\left(2\pi i \frac{xyz}{N}\right) = \exp\left(2\pi i \frac{(xy \bmod N)z}{N}\right)$$

Define

$$w = \frac{xy \bmod N}{N}$$

Now, multiplication:

$$|x\rangle\,|0\rangle \rightarrow |x\rangle\,|w\rangle$$

Modular arithmetic

Goal: only use n bits for output modulo N

Observation:

$$\exp\left(2\pi i \frac{xyz}{N}\right) = \exp\left(2\pi i \frac{(xy \bmod N)z}{N}\right)$$

Define

$$w = \frac{xy \bmod N}{N}$$

Now, multiplication:

$$|x\rangle\,|0\rangle \rightarrow |x\rangle\,|w\rangle$$

Output register requires $n + \mathcal{O}(\log(1/\epsilon))$ qubits

Application: Shor's algorithm

For Shor's algorithm: $\mathcal{O}(n)$ modular classical-quantum multiplications

Application: Shor's algorithm

For Shor's algorithm: $\mathcal{O}(n)$ modular classical-quantum multiplications

Using phase modulo and $k = 4$ multiplier:

Gates: $\mathcal{O}(n^{2.4})$
Total qubits: $2n + \mathcal{O}(\log(n/\epsilon))$

(Here ϵ is error across the whole algorithm)

Application: Shor's algorithm

Cost estimates for one 2048-bit classical-quantum multiplication: (here not modular)

Algorithm	Complexity	Gate count (millions)			Ancilla qubits
		Toffoli	CR_ϕ	Other	
This work	$\mathcal{O}(n^{1.4})$	**0.6**	**0.9**	**2.1**	**50**
Karatsuba [1]	$\mathcal{O}(n^{1.58})$	5.6	—	34	12730
Windowed [1]	$\mathcal{O}(n^2)$	1.8	—	2.5	4106
Schoolbook [1]	$\mathcal{O}(n^2)$	6.4	—	38	2048*

(Note: $\sim$ 15% of the CR_ϕ come from approximate QFTs with $\epsilon = 10^{-12}$)

[1] C. Gidney, "Windowed quantum arithmetic." (arXiv:1905.07682)

Application: Shor's algorithm

Cost estimates for one 2048-bit classical-quantum multiplication: (here not modular)

Algorithm	Complexity	Gate count (millions) Toffoli	CR_ϕ	Other	Ancilla qubits
This work	$\mathcal{O}(n^{1.4})$	**0.6**	**0.9**	**2.1**	**50**
Karatsuba [1]	$\mathcal{O}(n^{1.58})$	5.6	—	34	12730
Windowed [1]	$\mathcal{O}(n^2)$	1.8	—	2.5	4106
Schoolbook [1]	$\mathcal{O}(n^2)$	6.4	—	38	2048*

(Note: $\sim 15\%$ of the CR_ϕ come from approximate QFTs with $\epsilon = 10^{-12}$)

Open q.: Can we use windowing with our construction?

[1] C. Gidney, "Windowed quantum arithmetic." (arXiv:1905.07682)

Application: efficiently-verifiable quantum advantage

Protocol for a "proof of quantumness" requires evaluating $f(x) = x^2 \mod N$

Application: efficiently-verifiable quantum advantage

Protocol for a "proof of quantumness" requires evaluating $f(x) = x^2 \bmod N$

Cost estimates for protocol with 1024-bit N:

Algorithm	Gate count (millions)			Total qubits
	Toffoli	C^*R_ϕ	Other	
Gate optimized	**0.7**	**0.9**	**0.7**	**2400**
Balanced	**0.9**	**1.0**	**0.9**	**2070**
Qubit optimized	**2.2**	**2.0**	**2.2**	**1560**
"Digital" Karatsuba [2]	1.6	—	1.6	6801
"Digital" Schoolbook [2]	3.5	—	2.9	4097
Prev. Fourier 1 [2]	—	539	—	1025
Prev. Fourier 2 [2]	—	35	—	2062

[2] GDKM, Choi, Vazirani, Yao. "Efficiently-verifiable quantum advantage from a computational Bell test." (arXiv:2104.00687)

Summary

Classical-quantum

1 ancilla qubit

k	Gates
2	$\mathcal{O}(n^{1.58\cdots})$
3	$\mathcal{O}(n^{1.46\cdots})$
4	$\mathcal{O}(n^{1.40\cdots})$
$\vdots$	$\vdots$

Quantum-quantum

2 ancilla qubits

k	Gates
2	$\mathcal{O}(n^{2})$
3	$\mathcal{O}(n^{1.77\cdots})$
4	$\mathcal{O}(n^{1.66\cdots})$
$\vdots$	$\vdots$

Summary

Classical-quantum

1 ancilla qubit

k	Gates
2	$\mathcal{O}(n^{1.58\cdots})$
3	$\mathcal{O}(n^{1.46\cdots})$
4	$\mathcal{O}(n^{1.40\cdots})$
$\vdots$	$\vdots$

Quantum-quantum

2 ancilla qubits

k	Gates
2	$\mathcal{O}(n^{2})$
3	$\mathcal{O}(n^{1.77\cdots})$
4	$\mathcal{O}(n^{1.66\cdots})$
$\vdots$	$\vdots$

Implications:

Shor's algorithm: $\mathcal{O}(n^{2.4})$ gates using
$2n + \mathcal{O}(\log n)$ qubits

Summary

Classical-quantum

1 ancilla qubit

k	Gates
2	$\mathcal{O}(n^{1.58\cdots})$
3	$\mathcal{O}(n^{1.46\cdots})$
4	$\mathcal{O}(n^{1.40\cdots})$
$\vdots$	$\vdots$

Quantum-quantum

2 ancilla qubits

k	Gates
2	$\mathcal{O}(n^{2})$
3	$\mathcal{O}(n^{1.77\cdots})$
4	$\mathcal{O}(n^{1.66\cdots})$
$\vdots$	$\vdots$

Implications:

Shor's algorithm: $\mathcal{O}(n^{2.4})$ gates using
$2n + \mathcal{O}(\log n)$ qubits

Exact QFT in $\mathcal{O}(n^{1.4})$ gates using 1 ancilla

Summary

Classical-quantum

1 ancilla qubit

k	Gates
2	$\mathcal{O}(n^{1.58\cdots})$
3	$\mathcal{O}(n^{1.46\cdots})$
4	$\mathcal{O}(n^{1.40\cdots})$
$\vdots$	$\vdots$

Quantum-quantum

2 ancilla qubits

k	Gates
2	$\mathcal{O}(n^{2})$
3	$\mathcal{O}(n^{1.77\cdots})$
4	$\mathcal{O}(n^{1.66\cdots})$
$\vdots$	$\vdots$

Implications:

Shor's algorithm: $\mathcal{O}(n^{2.4})$ gates using $2n + \mathcal{O}(\log n)$ qubits

Exact QFT in $\mathcal{O}(n^{1.4})$ gates using 1 ancilla

In practice:

Low overheads—circuits are useful at practical sizes

Summary

Classical-quantum

1 ancilla qubit

k	Gates
2	$\mathcal{O}(n^{1.58\cdots})$
3	$\mathcal{O}(n^{1.46\cdots})$
4	$\mathcal{O}(n^{1.40\cdots})$
$\vdots$	$\vdots$

Quantum-quantum

2 ancilla qubits

k	Gates
2	$\mathcal{O}(n^{2})$
3	$\mathcal{O}(n^{1.77\cdots})$
4	$\mathcal{O}(n^{1.66\cdots})$
$\vdots$	$\vdots$

Implications:

Shor's algorithm: $\mathcal{O}(n^{2.4})$ gates using $2n + \mathcal{O}(\log n)$ qubits

Exact QFT in $\mathcal{O}(n^{1.4})$ gates using 1 ancilla

In practice:

Low overheads—circuits are useful at practical sizes

Low crossover—in some cases, already faster for 20 bit inputs!

Open Questions

- Can multiplication $\bmod N$ be performed with $\mathcal{O}(1)$ ancillas?

Open Questions

- Can multiplication $\mathrm{mod} N$ be performed with $\mathcal{O}(1)$ ancillas?
- Can QFT be done in sub-linear depth without needing a lot of ancillas?

Open Questions

- Can multiplication $\mathrm{mod} N$ be performed with $\mathcal{O}(1)$ ancillas?
- Can QFT be done in sub-linear depth without needing a lot of ancillas?
- Can we do any of these things with *zero* ancillas?

Open Questions

- Can multiplication $\mathrm{mod} N$ be performed with $\mathcal{O}(1)$ ancillas?
- Can QFT be done in sub-linear depth without needing a lot of ancillas?
- Can we do any of these things with *zero* ancillas?
- Can this technique be applied to e.g. the $\mathcal{O}(n \log n \log \log n)$ Schonhage-Strassen algorithm?

Open Questions

- Can multiplication $\mathrm{mod}\, N$ be performed with $\mathcal{O}(1)$ ancillas?
- Can QFT be done in sub-linear depth without needing a lot of ancillas?
- Can we do any of these things with *zero* ancillas?
- Can this technique be applied to e.g. the $\mathcal{O}(n \log n \log \log n)$ Schonhage-Strassen algorithm?
- **How well can we optimize explicit circuits (especially the base case)?**

Thank you!

Greg Kahanamoku-Meyer — gkm@berkeley.edu

Backup

What about all the arbitrary rotation gates?

In error-corrected setting, arbitrary rotation gates need to be synthesized.

What about all the arbitrary rotation gates?

In error-corrected setting, arbitrary rotation gates need to be synthesized.

Idea: “convert” some rotation gates into e.g. Toffolis; easier to synthesize

What about all the arbitrary rotation gates?

In error-corrected setting, arbitrary rotation gates need to be synthesized.

Idea: "convert" some rotation gates into e.g. Toffolis; easier to synthesize

All rotations are in the **base case**: 32-bit (say) PhaseProduct $\phi x' z'$

What about all the arbitrary rotation gates?

In error-corrected setting, arbitrary rotation gates need to be synthesized.

Idea: "convert" some rotation gates into e.g. Toffolis; easier to synthesize

All rotations are in the **base case**: 32-bit (say) PhaseProduct $\phi x'z'$

Direct (schoolbook)
Apply $32^2 = 1024$ CR_ϕ gates

CR_ϕ **optimized**

1. Compute $|x'z'\rangle$ via a regular digital multiplier circuit
2. Apply phase rotations on the output
3. Uncompute $|x'z'\rangle$

What about all the arbitrary rotation gates?

In error-corrected setting, arbitrary rotation gates need to be synthesized.

Idea: "convert" some rotation gates into e.g. Toffolis; easier to synthesize

All rotations are in the **base case**: 32-bit (say) PhaseProduct $\phi x'z'$

Direct (schoolbook)
Apply $32^2 = 1024$ CR_ϕ gates

CR_ϕ **optimized**

1. Compute $|x'z'\rangle$ via a regular digital multiplier circuit
2. Apply phase rotations on the output
3. Uncompute $|x'z'\rangle$

$1024\ CR_\phi \rightarrow 64\ R_\phi$ plus ~ 2048 Toffoli

Fast classical-quantum multiplication: algorithm

PhaseProduct($\phi, |x\rangle, |z\rangle$)

Input: Quantum state $|x\rangle |z\rangle$, classical value ϕ

Output: Quantum state $\exp(i\phi xz) |x\rangle |z\rangle$

1. Split $|x\rangle$ and $|z\rangle$ in half, as $|x_1\rangle |x_0\rangle$ and $|z_1\rangle |z_0\rangle$
2. Apply PhaseProduct($(2^n - 2^{n/2})\phi, |x_1\rangle, |z_1\rangle$)
3. Apply PhaseProduct($(1 - 2^{n/2})\phi, |x_0\rangle, |z_0\rangle$)
4. Add $|x_1\rangle$ to $|x_0\rangle$, and $|z_1\rangle$ to $|z_0\rangle$. Registers now hold $|x_1\rangle |x_0 + x_1\rangle |z_1\rangle |z_0 + z_1\rangle$.
5. Apply PhaseProduct($2^{n/2}\phi, |x_0 + x_1\rangle, |z_0 + z_1\rangle$).
6. Subtract $|x_1\rangle$, $|z_1\rangle$ to return to registers to $|x_1\rangle |x_0\rangle |z_1\rangle |z_0\rangle$.