Fast quantum integer multiplication with almost no ancillas

Gregory D. Kahanamoku-Meyer

July 14, 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?

Check for updates

OPEN

Classically verifiable quantum advantage from a computational B

Gregory D. Kahanamoku-Meyer [1]

Existing experimental demonstrations of qua
of the quantum device requires exponentially
for demonstrating quantum computational a
cryptographic tools called trapdoor claw-free
protocols before, our protocol employs a surp
property called the adaptive hardcore bit, whi
assumptions. Leveraging the relaxed cryptog
structions, based on Rabin's function and the

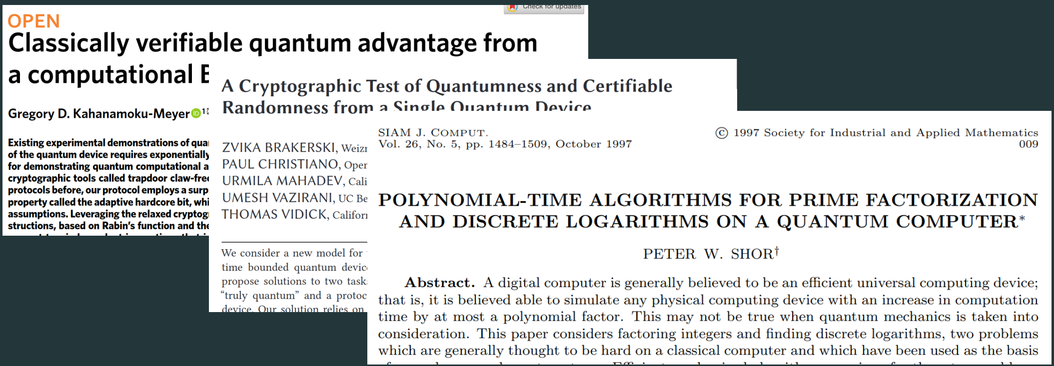
A Cryptographic Test of Quantumness and Certifiable Randomness from a Single Quantum Device

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 computational B

Gregory D. Kahanamoku-Meyer

Existing experimental demonstrations of qua of the quantum device requires exponentially for demonstrating quantum computational a cryptographic tools called trapdoor claw-free protocols before, our protocol employs a surp property called the adaptive hardcore bit, whi assumptions. Leveraging the relaxed cryptog structions, based on Rabin's function and the

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

ZVIKA BRAKERSKI, Weizı
PAUL CHRISTIANO, Oper
URMILA MAHADEV, Cali
UMESH VAZIRANI, UC Be
THOMAS VIDICK, Califor

We consider a new model for time bounded quantum devic propose solutions to two task "truly quantum" and a protoc device. Our solution relies on

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

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

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

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

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

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

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

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

Overview

1. Fast multiplication (few gates)

Overview

1. Fast multiplication (few gates)
2. Quantum multiplication (few qubits)

Overview

1. Fast multiplication (few gates)
2. Quantum multiplication (few qubits)
3. Fast quantum multiplication (few gates + qubits)

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

				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

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

Background: schoolbook multiplication

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

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

Background: schoolbook multiplication

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

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

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

Background: schoolbook multiplication

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

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

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 $n/2$:

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

Background: Karatsuba multiplication

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

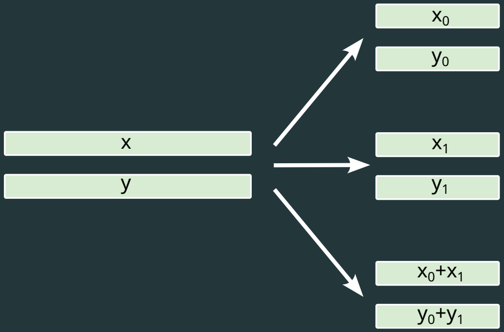
Observation: $x_0 y_1 + x_1 y_0 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$

Can compute xy with only three multiplications of size $n/2$:

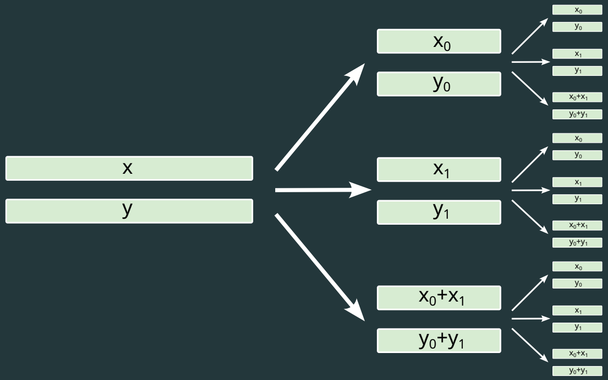
1. $x_1 y_1$
2. $x_0 y_0$
3. $(x_1 + x_0)(y_1 + y_0)$

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

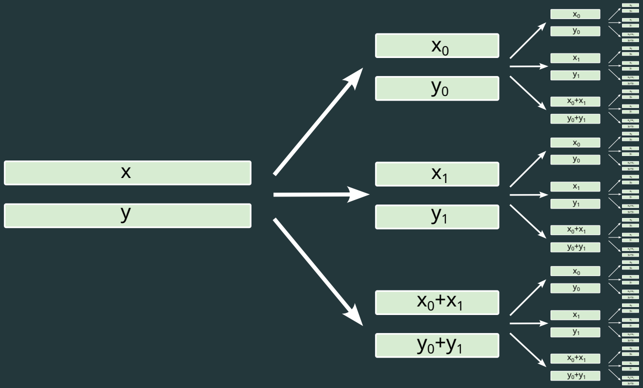
Background: Karatsuba multiplication



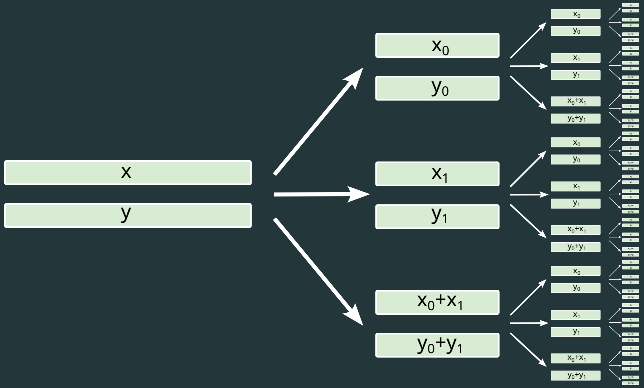
Background: Karatsuba multiplication



Background: Karatsuba multiplication

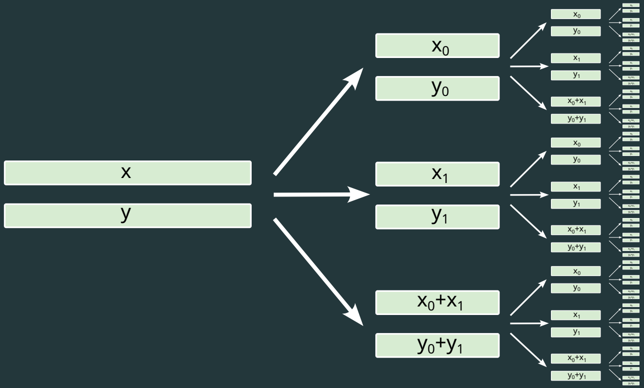


Background: Karatsuba multiplication



Depth: $d = \log_2 n$

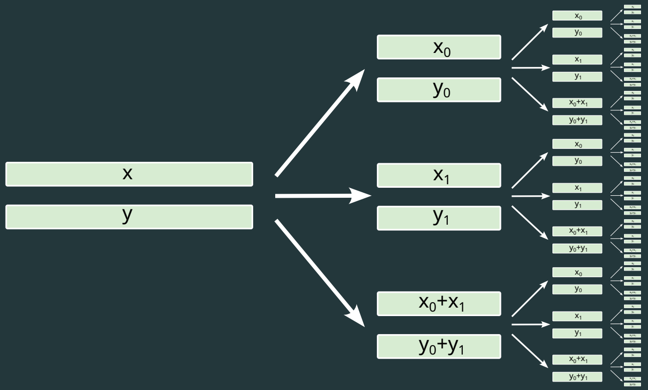
Background: Karatsuba multiplication



Depth: $d = \log_2 n$

Operations: 3^d

Background: Karatsuba multiplication

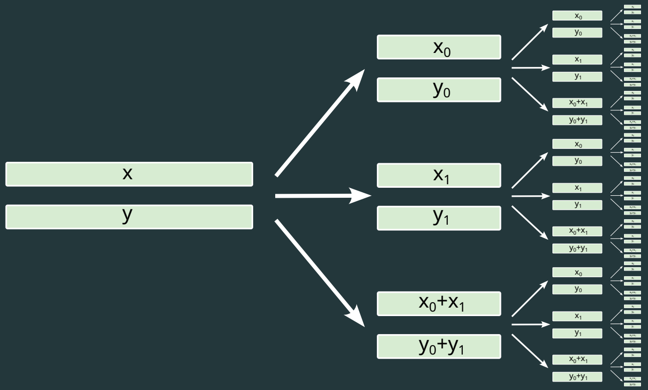


Depth: $d = \log_2 n$

Operations: 3^d

Recursion relation: $T(n) \sim 3T(n/2)$

Background: Karatsuba multiplication



Depth: $d = \log_2 n$

Operations: 3^d

Recursion relation: $T(n) \sim 3T(n/2)$

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

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

Background: Toom-Cook multiplication

Break the n bit numbers into k chunks of n/k bits.

Background: Toom-Cook multiplication

Break the n bit numbers into k chunks of n/k bits.

Perform n bit multiply via $2k - 1$ multiplications of n/k bit numbers (compare: k^2).

Background: Toom-Cook multiplication

Break the n bit numbers into k chunks of n/k bits.

Perform n bit multiply via $2k-1$ multiplications of n/k bit numbers (compare: k^2).

Algorithm	Gate count
Schoolbook	$\mathcal{O}(n^2)$
$k=2$	$\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.58\cdots})$
$k=3$	$\mathcal{O}(n^{\log_3 5}) = \mathcal{O}(n^{1.46\cdots})$
$k=4$	$\mathcal{O}(n^{\log_4 7}) = \mathcal{O}(n^{1.40\cdots})$

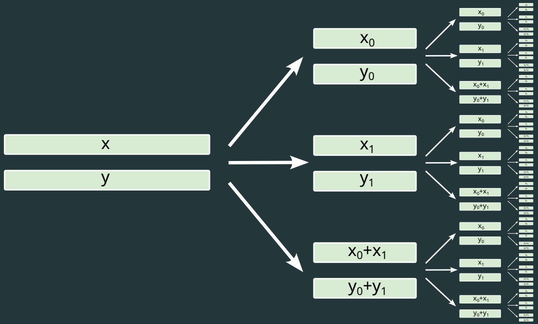
Summary: fast multiplication

- “Standard” multiplication requires time $\mathcal{O}(n^2)$ operations
- Faster algorithms exist, but have large constant factors

Can these fast circuits be made quantum?

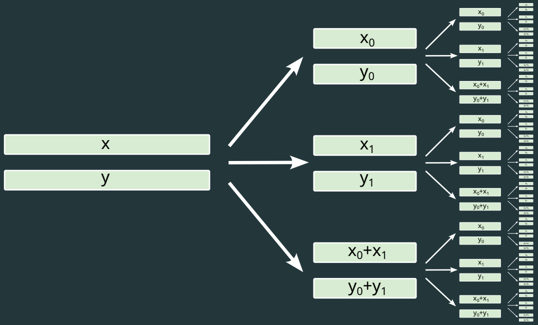
Can these fast circuits be made quantum?

Challenge: making recursive algorithms reversible



Can these fast circuits be made quantum?

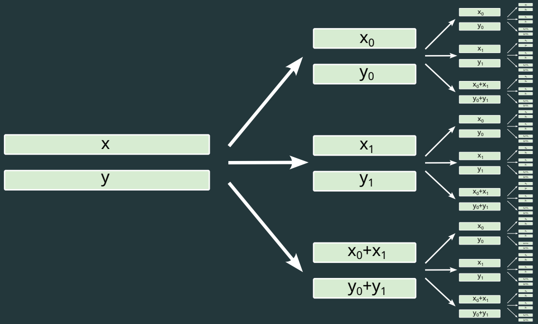
Challenge: making recursive algorithms reversible



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

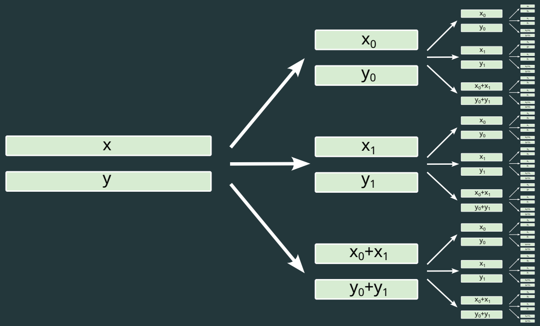


Work	Qubits
Kowada et al. ’06	$\mathcal{O}(n^{1.58\cdots})$
Parent et al. ’18	$\mathcal{O}(n^{1.43\cdots})$
Gidney ’19	$\mathcal{O}(n)$

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

Can these fast circuits be made quantum?

Challenge: making recursive algorithms reversible



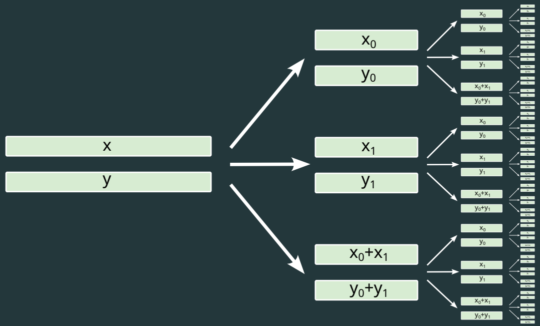
Work	Qubits
Kowada et al. ’06	$\mathcal{O}(n^{1.58\cdots})$
Parent et al. ’18	$\mathcal{O}(n^{1.43\cdots})$
Gidney ’19	$\mathcal{O}(n)$

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

Quantum Fourier transform:

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

A fundamentally quantum way of doing arithmetic

Quantum Fourier transform:

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

[Draper '04]: Arithmetic in Fourier space

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

A fundamentally quantum way of doing arithmetic

Quantum Fourier transform:

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

[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

Quantum Fourier transform:

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

[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 $|x\rangle\,|y\rangle\,|0\rangle \rightarrow |x\rangle\,|y\rangle\,|xy\rangle$?

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

A fundamentally quantum way of doing arithmetic

Quantum Fourier transform:

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

[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 $|x\rangle\,|y\rangle\,|0\rangle \rightarrow |x\rangle\,|y\rangle\,|xy\rangle$?

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

Quantum Fourier transform:

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

[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 $\exp\left(\frac{2\pi i x y z}{2^n}\right)$?

A fundamentally quantum way of doing arithmetic

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

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

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

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

The downside:

A fundamentally quantum way of doing arithmetic

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

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

A fundamentally quantum way of doing arithmetic

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

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

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

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

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

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

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

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.

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

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

Plugging in Karatsuba:

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

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:

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

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

Fast classical-quantum multiplication

Goal: Implement PhaseProduct(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.

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

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

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

Plugging in reordered Karatsuba:

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

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

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

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

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

Complexity: $T(n) = 3T(n/2)$

Fast classical-quantum multiplication

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

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

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

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

How many qubits do we need?

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

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

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

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.

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. $|x_1\rangle\,|x_0\rangle \to |x_1\rangle\,|x_0 + x_1\rangle$

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:

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: 1

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

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) $|x\rangle\,|y\rangle\,|z\rangle = \exp(i\phi xyz)\,|x\rangle\,|y\rangle\,|z\rangle$

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! $xyz = x(yz)$.

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 it's $\mathcal{O}(n^2)$

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 it's $\mathcal{O}(n^2)$

Doesn't work in the phase!!

Detour: Generalizing Karatsuba's method

"Triple schoolbook" method uses $\mathcal{O}(n^3)$ operations—8 multiplications of size $n/2$.

Detour: Generalizing Karatsuba's method

"Triple schoolbook" method uses $\mathcal{O}(n^3)$ operations—8 multiplications of size $n/2$.

Is there a decomposition like Karatsuba for the triple product?

Detour: Generalizing Karatsuba's method

"Triple schoolbook" method uses $\mathcal{O}(n^3)$ operations—8 multiplications of size $n/2$.

Is there a decomposition like Karatsuba for the triple product? Yes!

$$
\begin{aligned}
xyz =& (2^{3n/2} - 2^{n/2})x_1y_1z_1 \\
&+ \frac{1}{2}(2^n + 2^{n/2})(x_0 + x_1)(y_0 + y_1)(z_0 + z_1) \\
&+ \frac{1}{2}(2^n - 2^{n/2})(x_0 - x_1)(y_0 - y_1)(z_0 - z_1) \\
&+ (1 - 2^n)x_0y_0z_0
\end{aligned}
$$

Detour: Generalizing Karatsuba's method

"Triple schoolbook" method uses $\mathcal{O}(n^3)$ operations—8 multiplications of size $n/2$.

Is there a decomposition like Karatsuba for the triple product? Yes!

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

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

Detour: Generalizing Karatsuba's method

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

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

Detour: Generalizing Karatsuba's method

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

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

Detour: Generalizing Karatsuba's method

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

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

Result (modified Toom-Cook): k pieces $\rightarrow \mathcal{O}(n^{\log_k(3k-2)})$ runtime

Detour: Generalizing Karatsuba's method

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

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

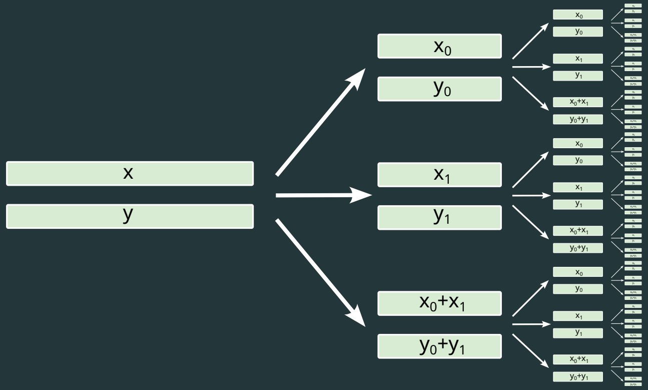
Result (modified Toom-Cook): k pieces $\rightarrow \mathcal{O}(n^{\log_k(3k-2)})$ runtime

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

These runtimes are achieved with 2 ancilla qubits.

Depth considerations

Parallelization is natural—achieving depth $\mathcal{O}(n)$ is easy!



Depth considerations

Parallelization is natural—achieving depth $\mathcal{O}(n)$ is easy!

But $\mathcal{O}(n^{1.58})/n = \mathcal{O}(n^{0.58})$ is our lower bound: can we get there?

Depth considerations

Parallelization is natural—achieving depth $\mathcal{O}(n)$ is easy!

But $\mathcal{O}(n^{1.58})/n = \mathcal{O}(n^{0.58})$ is our lower bound: can we get there?

Claim: Can implement PhaseProduct and PhaseTripleProduct in sub-linear time, using $\mathcal{O}(n)$ ancillas!

Suprisingly, multiplication itself is bottlenecked by QFT

Modular arithmetic

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:

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

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

Summary

Classical-quantum

1 ancilla qubit

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

Quantum-quantum

2 ancilla qubits

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

Summary

Classical-quantum

1 ancilla qubit

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

Quantum-quantum

2 ancilla qubits

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

Implications:

Shor's algorithm: $\mathcal{O}(n^{2.46})$ 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	$\mathcal{O}(n^{1.40\cdots})$

Quantum-quantum

2 ancilla qubits

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

Implications:

Shor's algorithm: $\mathcal{O}(n^{2.46})$ gates using $2n + \mathcal{O}(\log n)$ qubits

Exact QFT in $\mathcal{O}(n^{1.46})$ gates using 1 ancilla

Open Questions

- Can multiplication $\mathrm{mod} N$ 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?
- **How well can we optimize explicit circuits?**

Open Questions

- Can multiplication $\bmod N$ 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?
- **How well can we optimize explicit circuits?**

Thank you!

Greg Kahanamoku-Meyer — gkm@berkeley.edu

Backup

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 $|x_1\rangle |x_0\rangle |z_1\rangle |z_0\rangle$.