Fast quantum integer multiplication without ancillas

arXiv:2403.18006

Gregory D. Kahanamoku-Meyer

May 16, 2024

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.

Results (spoilers)

Asymptotic costs for quantum multiplication:

Results (spoilers)

Asymptotic costs for quantum multiplication:

Schoolbook (via Cuccaro ’04):

- Gates: $\mathcal{O}(n^2)$
- Ancillas: 1

Results (spoilers)

Asymptotic costs for quantum multiplication:

Schoolbook (via Cuccaro ’04):

- Gates: $\mathcal{O}(n^2)$
- Ancillas: 1

Karatsuba (Gidney ’19):

- Gates: $\mathcal{O}(n^{1.58\cdots})$
- Ancillas: $\mathcal{O}(n)$

Results (spoilers)

Asymptotic costs for quantum multiplication:

Schoolbook (via Cuccaro '04):

- Gates: $\mathcal{O}(n^2)$
- Ancillas: 1

Karatsuba (Gidney '19):

- Gates: $\mathcal{O}(n^{1.58\cdots})$
- Ancillas: $\mathcal{O}(n)$

This work:

- Gates: $\mathcal{O}(n^{1+\epsilon})$ for any $\epsilon > 0$
- Ancillas: 0

Results (spoilers): in practice

Cost to multiply a 2048-bit quantum register by a 2048-bit classical value:

[1] Gidney '19, "Windowed quantum arithmetic"

Algorithm	Asymptotic scaling	Gate count (millions)			Ancillas
		Toffoli	CR_ϕ	H,X,CNOT	
Schoolbook [1]	$\mathcal{O}(n^2)$	6.4	—	38	1*
Karatsuba [1]	$\mathcal{O}(n^{1.58})$	5.6	—	34	12730
Windowed [1]	$\mathcal{O}(n^2/\log^2 n)$	1.8	—	2.5	4106

Results (spoilers): in practice

Cost to multiply a 2048-bit quantum register by a 2048-bit classical value:

[1] Gidney '19, "Windowed quantum arithmetic"

Algorithm	Asymptotic scaling	Gate count (millions)			Ancillas
		Toffoli	CR_ϕ	H,X,CNOT	
Schoolbook [1]	$\mathcal{O}(n^2)$	6.4	—	38	1*
Karatsuba [1]	$\mathcal{O}(n^{1.58})$	5.6	—	34	12730
Windowed [1]	$\mathcal{O}(n^2/\log^2 n)$	1.8	—	2.5	4106
This work (standard QFT)	$\mathcal{O}(n^{1.29})$**	**0.6**	**0.3**	**1.9**	**79**
This work (phase gradient QFT)	$\mathcal{O}(n^{1.29})$**	**0.9**	**0.1**	**3.2**	**80**

Plan

1. Background and core algorithm (slides)

Plan

1. Background and core algorithm (slides)
2. Practical considerations and optimizations (choose your own adventure)

Background: schoolbook multiplication

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

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

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 $b = 2^{n/2}$?

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

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

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

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

Background: Karatsuba multiplication

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

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

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. $(x_1 + x_0)(y_1 + y_0)$

Background: Karatsuba multiplication

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

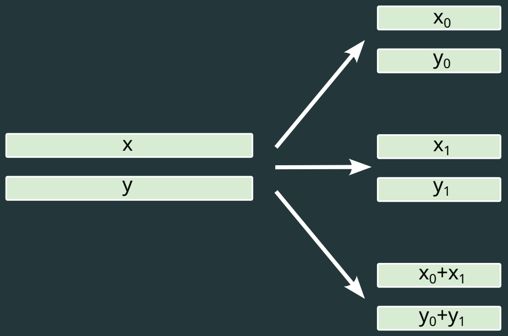
Observation: $x_0 y_1 + x_1 y_0 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$

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

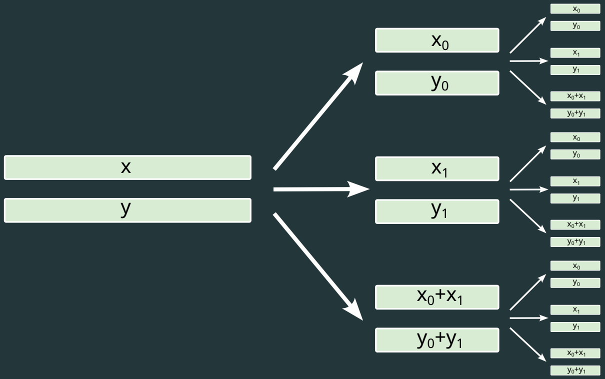
1. $x_1 y_1$
2. $x_0 y_0$
3. $(x_1 + x_0)(y_1 + y_0)$

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

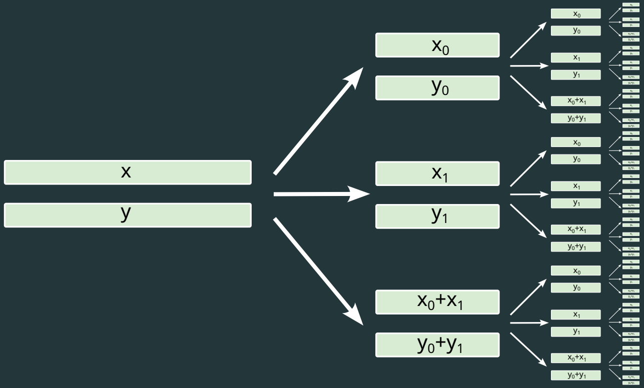
Background: Karatsuba multiplication



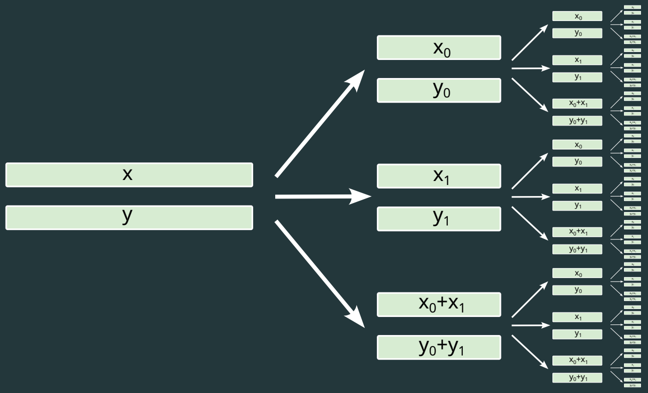
Background: Karatsuba multiplication



Background: Karatsuba multiplication

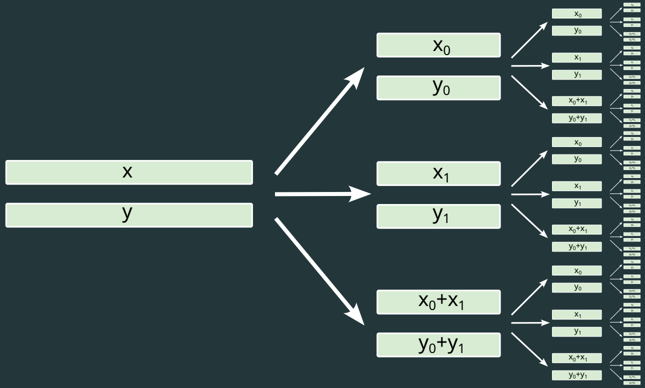


Background: Karatsuba multiplication



Depth: $d = \log_2 n$

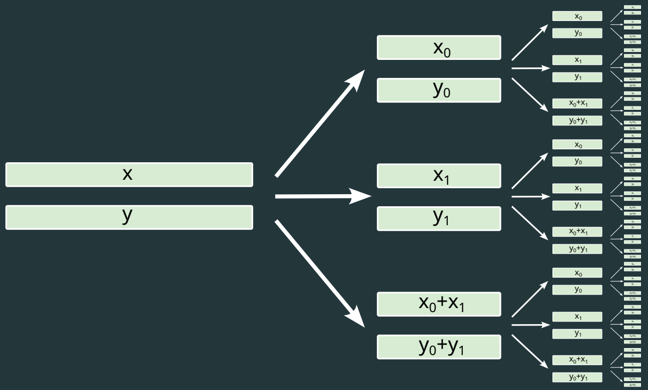
Background: Karatsuba multiplication



Depth: $d = \log_2 n$

Operations: 3^d

Background: Karatsuba multiplication



Depth: $d = \log_2 n$

Operations: 3^d

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

A fundamentally quantum way of doing arithmetic

[Draper '04]: Arithmetic in Fourier space

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

A fundamentally quantum way of doing arithmetic

[Draper '04]: Arithmetic in Fourier space

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

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

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

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

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

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

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

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

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

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

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

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.

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.

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

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:

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

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

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

Re-ordering Karatsuba:

$$xz = (2^n - 2^{n/2})x_1z_1 + 2^{n/2}(x_0 + x_1)(z_0 + z_1) + (1 - 2^{n/2})x_0z_0$$

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 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:

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

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

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

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

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

Use quantum addition circuits.

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

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

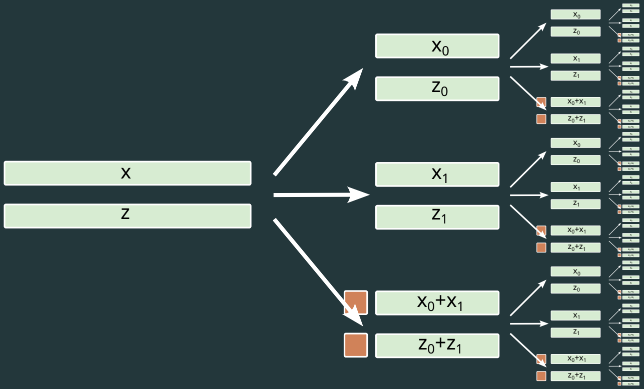
What about $\exp(i\phi_3 (x_0 + x_1)(z_0 + z_1))$?

Use quantum addition circuits.

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

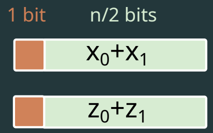
How many qubits do we need?

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



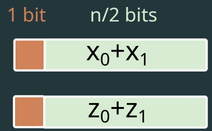
How many qubits do we need?

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



How many qubits do we need?

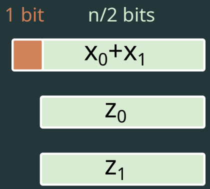
Total number of ancillas: 2



Idea: “Shave off” the high bit before recursing

How many qubits do we need?

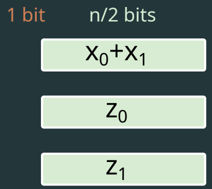
Total number of ancillas: 1



Idea: “Shave off” the high bit before recursing

How many qubits do we need?

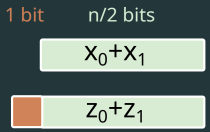
Total number of ancillas: 1



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

1 bit	n/2 bits
	x_0+x_1
	z_0+z_1

Idea: “Shave off” the high bit before recursing

How many qubits do we need?

Total number of ancillas: 0

1 bit   n/2 bits

x_0+x_1

z_0+z_1

Idea: “Shave off” the high bit before recursing

Trick: Using dirty qubits, can reduce to zero!

Making it go faster

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

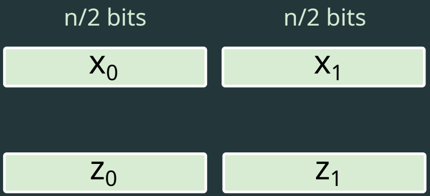
Making it go faster

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

Can we make it go faster?

Generalizing: Toom-Cook multiplication

n/2 bits	n/2 bits
x_0	x_1
z_0	z_1



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

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

Generalizing: Toom-Cook multiplication

n/2 bits	n/2 bits
x_0	x_1
z_0	z_1

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

$$x = x_1 b + x_0$$

$$z = z_1 b + z_0$$

$$\phi xz = \phi_1 x_1 z_1 + \phi_2 (x_0 + x_1)(z_0 + z_1) + \phi_3 x_0 z_0$$

Generalizing: Toom-Cook multiplication

n/k bits	n/k bits	n/k bits		n/k bits
x_0	x_1	x_2	- - -	x_{k-1}
z_0	z_1	z_2	- - -	z_{k-1}

Generalizing: Toom-Cook multiplication

n/k bits	n/k bits	n/k bits		n/k bits
x_0	x_1	x_2	- - -	x_{k-1}
z_0	z_1	z_2	- - -	z_{k-1}

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

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

Generalizing: Toom-Cook multiplication

n/k bits	n/k bits	n/k bits		n/k bits
X_0	X_1	X_2	- - -	X_{k-1}
Z_0	Z_1	Z_2	- - -	Z_{k-1}

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

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

$$\phi xz = \sum_{\ell=1}^{2k-1} \phi_\ell \left(\sum_i^k w_\ell^i x_i\right) \left(\sum_i^k w_\ell^i z_i\right)$$

Generalizing: Toom-Cook multiplication

$$\phi xz = \sum_{\ell=1}^{2k-1} \phi_\ell \left(\sum_i w_\ell^i x_i\right) \left(\sum_i w_\ell^i z_i\right)$$

We get to choose the $2k-1$ values w_ℓ! (The ϕ_ℓ depend on our choices).

Generalizing: Toom-Cook multiplication

$$\phi xz = \sum_{\ell=1}^{2k-1} \phi_\ell \left(\sum_i w_\ell^i x_i\right) \left(\sum_i w_\ell^i z_i\right)$$

We get to choose the $2k-1$ values w_ℓ! (The ϕ_ℓ depend on our choices).

Let $k = 2$.

$w_1 = 0$

$$\sum_{i=0}^{k-1} w_1^i x_i = x_0 \tag{1}$$

Generalizing: Toom-Cook multiplication

$$\phi xz = \sum_{\ell=1}^{2k-1} \phi_\ell \left(\sum_i w_\ell^i x_i\right) \left(\sum_i w_\ell^i z_i\right)$$

We get to choose the $2k-1$ values w_ℓ! (The ϕ_ℓ depend on our choices).

Let $k = 2$.

$w_2 = 1$

$$\sum_{i=0}^{k-1} w_2^i x_i = x_0 + x_1 \tag{1}$$

Generalizing: Toom-Cook multiplication

$$\phi x z = \sum_{\ell=1}^{2k-1} \phi_\ell \left(\sum_i w_\ell^i x_i\right) \left(\sum_i w_\ell^i z_i\right)$$

We get to choose the $2k-1$ values w_ℓ! (The ϕ_ℓ depend on our choices).

Let $k = 2$.

$w_3 = \infty$

$$1/w_3 \sum_{i=0}^{k-1} w_3^i x_i = x_1 \tag{1}$$

Generalizing: Toom-Cook multiplication

$$\phi xz = \sum_{\ell=1}^{2k-1} \phi_\ell \left(\sum_i w_\ell^i x_i\right) \left(\sum_i w_\ell^i z_i\right)$$

We get to choose the $2k-1$ values w_ℓ! (The ϕ_ℓ depend on our choices).

Let $k = 2$.

$w_1 = 0$, $w_2 = 1$, $w_3 = \infty$

$$\phi xz = \phi_1 x_0 z_0 + \phi_2 (x_0 + x_1)(z_0 + z_1) + \phi_3 x_1 z_1 \tag{1}$$

Complexity vs. k

This strategy yields asymptotic complexity $\mathcal{O}(n^{\log_k(2k-1)})$

Complexity vs. k

This strategy yields asymptotic complexity $\mathcal{O}(n^{log_k(2k-1)})$

Algorithm	Gate count
Schoolbook	$\mathcal{O}(n^2)$
$k = 2$	$\mathcal{O}(n^{1.58\cdots})$
$k = 5$	$\mathcal{O}(n^{1.37\cdots})$
$k = 8$	$\mathcal{O}(n^{1.30\cdots})$
$\vdots$	$\vdots$

Complexity vs. k

This strategy yields asymptotic complexity $\mathcal{O}(n^{\log_k(2k-1)})$

Algorithm	Gate count
Schoolbook	$\mathcal{O}(n^2)$
$k = 2$	$\mathcal{O}(n^{1.58\cdots})$
$k = 5$	$\mathcal{O}(n^{1.37\cdots})$
$k = 8$	$\mathcal{O}(n^{1.30\cdots})$
$\vdots$	$\vdots$

Note: For quantum-quantum mult., get $\mathcal{O}(n^{\log_k(3k-2)})$

Thank you!

arXiv:2403.18006

Greg Kahanamoku-Meyer — gkm@mit.edu — https://gregkm.me/

I will be at Google QSS, including Friday resource estimation workshop!

Thank you!

arXiv:2403.18006

Greg Kahanamoku-Meyer — gkm@mit.edu — https://gregkm.me/

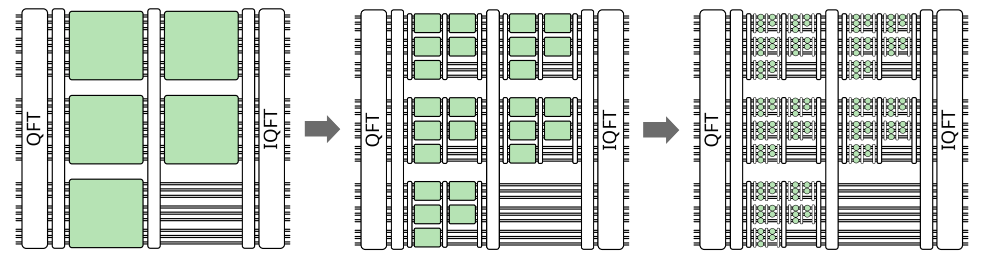
I will be at Google QSS, including Friday resource estimation workshop!

Further discussion:

- Circuit depth (new result: log-depth approx. QFT with few ancillas)
- Optimizing the base case, and implementing arbitrary phase rotations
- Optimizing choice of k
- Optimizing choice of w_ℓ and computation of linear combinations
- Modular arithmetic
- Dirty qubit construction

Backup

Circuit structure (depth and locality)



What about all the arbitrary rotation gates?

Under restricted gate sets, arbitrary rotation gates need to be synthesized.

What about all the arbitrary rotation gates?

Under restricted gate sets, arbitrary rotation gates need to be synthesized.

Idea: trade off rotation gates for other easier to synthesize gates

What about all the arbitrary rotation gates?

Under restricted gate sets, arbitrary rotation gates need to be synthesized.

Idea: trade off rotation gates for other easier to synthesize gates

Optimize the **base case**: 32-bit (say) PhaseProduct $\phi x' z'$

What about all the arbitrary rotation gates?

Under restricted gate sets, arbitrary rotation gates need to be synthesized.

Idea: trade off rotation gates for other easier to synthesize gates

Optimize the **base case**: 32-bit (say) PhaseProduct $\phi x' z'$

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

What about all the arbitrary rotation gates?

Under restricted gate sets, arbitrary rotation gates need to be synthesized.

Idea: trade off rotation gates for other easier to synthesize gates

Optimize the **base case**: 32-bit (say) PhaseProduct $\phi x'z'$

Direct (schoolbook)

Apply $32^2 = 1024$ CR_ϕ gates

CR_ϕ **optimized**

1. Compute $|x'z'\rangle$ directly on 64 ancillas
2. Apply phase rotations on the output
3. Uncompute $|x'z'\rangle$

What about all the arbitrary rotation gates?

Under restricted gate sets, arbitrary rotation gates need to be synthesized.

Idea: trade off rotation gates for other easier to synthesize gates

Optimize the **base case**: 32-bit (say) PhaseProduct $\phi x'z'$

Direct (schoolbook)

Apply $32^2 = 1024$ CR_ϕ gates

CR_ϕ **optimized**

1. Compute $|x'z'\rangle$ directly on 64 ancillas
2. Apply phase rotations on the output
3. Uncompute $|x'z'\rangle$

1024 $CR_\phi \to 64$ R_ϕ plus ~ 2048 Toffoli

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.

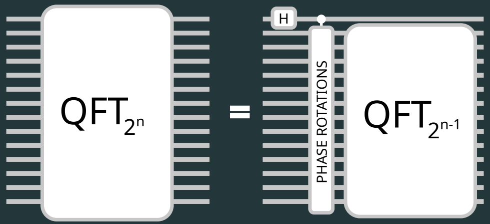
For any $m < n$, we may implement QFT_{2^n}:

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

[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

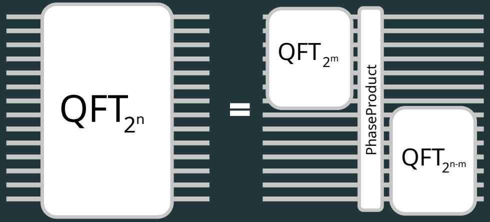


Fast exact quantum Fourier transform

[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

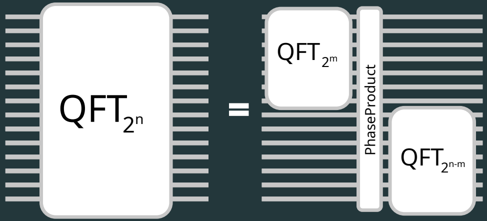


Fast exact quantum Fourier transform

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

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

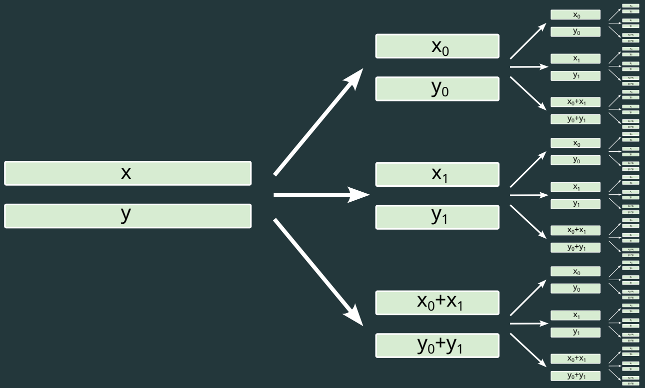
1. Apply QFT_{2^m} on first m qubits
2. Apply phase rotation $2\pi xz/2^n$
   - $|x\rangle$ is value of first m qubits
   - $|z\rangle$ is value of final $n - m$ qubits
3. Apply $\mathrm{QFT}_{2^{n-m}}$ on final $n - m$ qubits



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

Depth considerations

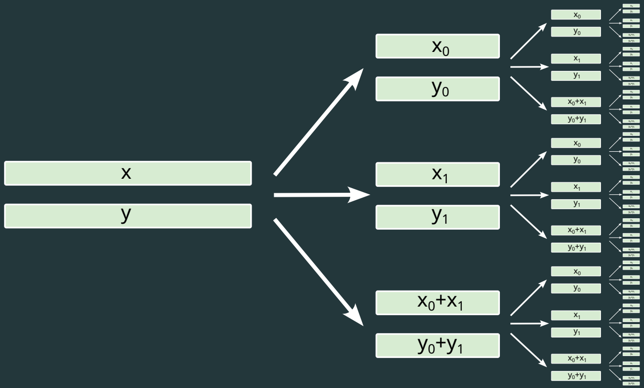
Parallelization is natural.



Depth considerations

Parallelization is natural.

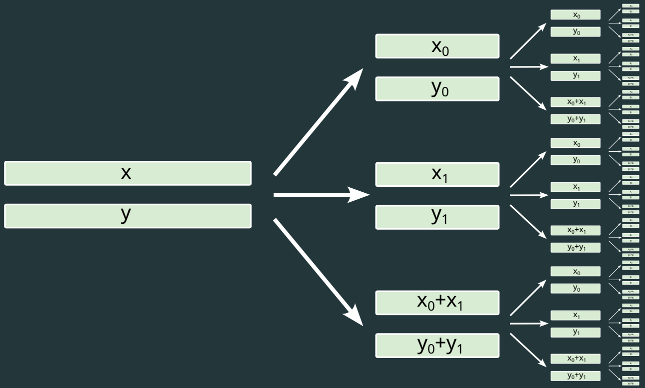
We have k sub-registers to work with—can do k sub-products in parallel.



Depth considerations

Parallelization is natural.

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

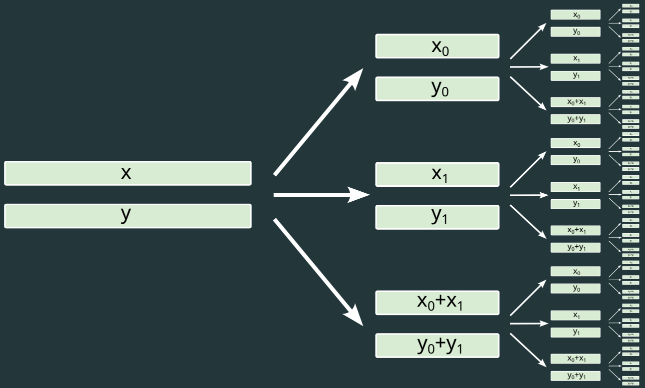


Depth: PhaseProduct in $\mathcal{O}(n^{\log_k 2})$ and PhaseTripleProduct in $\mathcal{O}(n^{\log_k 3})$ using a few more ancillas

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)

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

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

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

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 $\mathrm{k} = 2$

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

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

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

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

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

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

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

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

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

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

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

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

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

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