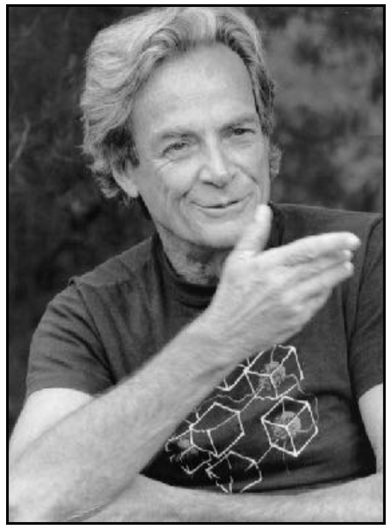
Simulating quantum mechanics with quantum computers

Andrew Childs
University of Maryland

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

UMIACS
University of Maryland
Institute for Advanced
Computer Studies

JOINT CENTER FOR
QUANTUM INFORMATION
AND COMPUTER SCIENCE



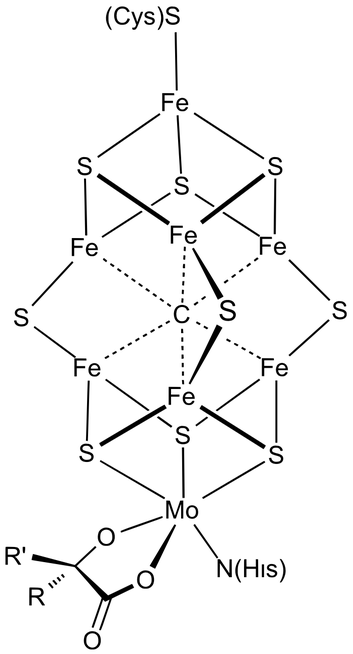
"… nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy."

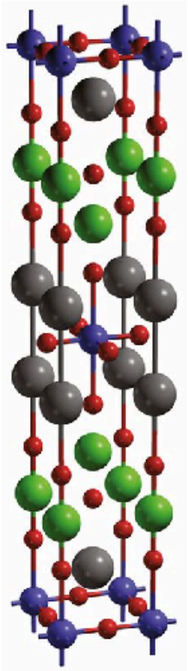
Richard Feynman
Simulating physics with computers (1981)

Why simulate quantum mechanics?

Computational chemistry/physics

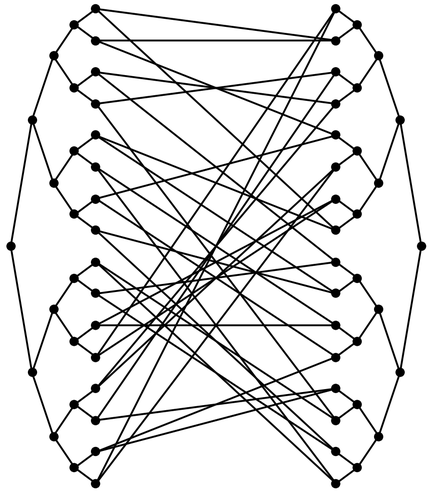
- chemical reactions (e.g., nitrogen fixation)
- properties of materials
- condensed matter physics

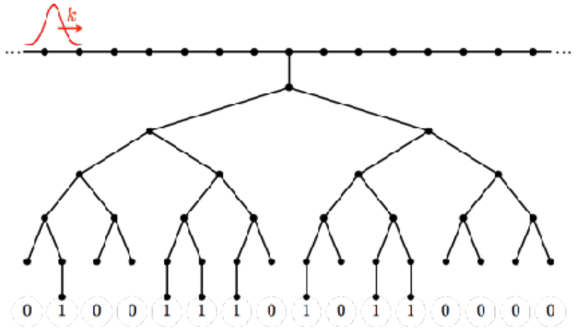




Implementing quantum algorithms

- continuous-time quantum walk (e.g., for formula evaluation, search, …)
- adiabatic quantum computation (e.g., for optimization or state generation)
- linear/differential equations





Simulating quantum mechanics with quantum computers

Algorithms

- Can we give an efficient algorithm?
- What is the best possible complexity as a function of various parameters?

Implementation

- What classically-hard simulations are easiest for a quantum computer?
- What simulation algorithm is best in practice for medium-scale problems?
- What optimizations can be applied to improve the implementation of algorithms?
- Are there resource tradeoffs (e.g., time vs. space)?
- Can we reliably do a classically-hard simulation without fault tolerance?
- How do the details of an experimental system (connectivity of qubits, timescales for different gates, etc.) interact with algorithmic issues?

Algorithms

Quantum dynamics

The dynamics of a quantum system are determined by its *Hamiltonian* H.

$$i\frac{\mathrm{d}}{\mathrm{d}t}|\psi(t)\rangle = H|\psi(t)\rangle \qquad \Rightarrow \qquad |\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

Quantum simulation problem: Given a description of the Hamiltonian H, an evolution time t, and an initial state $|\psi(0)\rangle$, produce the final state $|\psi(t)\rangle$ (to within some error tolerance ϵ)

A classical computer cannot even represent the state efficiently.

A quantum computer cannot produce a complete description of the state.

But given succinct descriptions of
- the initial state (suitable for a quantum computer to prepare it efficiently) and
- a final measurement (say, measurements of the individual qubits in some basis),

a quantum computer can efficiently answer questions that (apparently) a classical one cannot.

Local and sparse Hamiltonians

Local Hamiltonians [Lloyd 96]

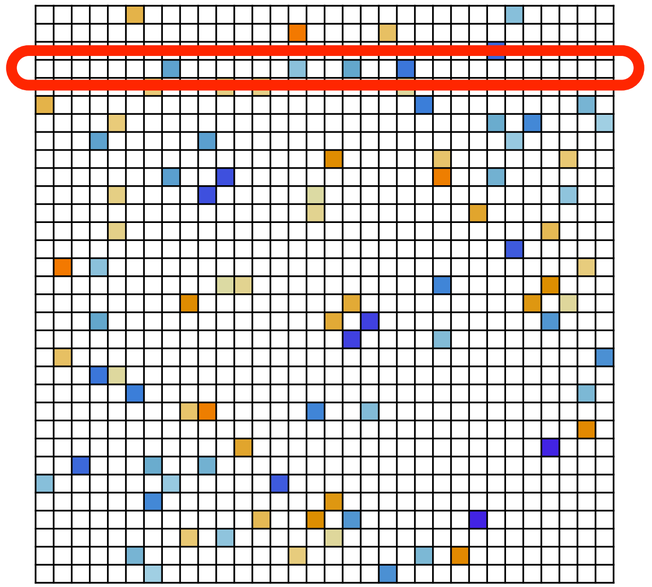
$H = \sum_{\ell=1}^{L} H_\ell$ where each H_ℓ acts on $k = O(1)$ qubits

Sparse Hamiltonians [Aharonov, Ta-Shma 03]

At most d nonzero entries per row, $d = \mathrm{poly}(\log N)$
(where H is $N \times N$)

In any given row, the location of the jth nonzero entry and its value can be computed efficiently

$H =$



Note: A k-local Hamiltonian with L terms is d-sparse with $d = 2^k L$

Product formula simulation

Suppose we want to simulate $H = \sum_{\ell=1}^{L} H_\ell$

Combine individual simulations with the Lie product formula. E.g., with two terms:

$$\lim_{r\to\infty} \left(e^{-iAt/r} e^{-iBt/r}\right)^r = e^{-i(A+B)t}$$

$$\left(e^{-iAt/r} e^{-iBt/r}\right)^r = e^{-i(A+B)t} + O(t^2/r)$$

To ensure error at most ϵ, take
$r = O\left((\|H\|t)^2/\epsilon\right)$

[Lloyd 96]

To get a better approximation, use higher-order formulas.

E.g., second order:

$$\left(e^{-iAt/2r} e^{-iBt} e^{-iAt/2r}\right)^r = e^{-i(A+B)t} + O(t^3/r^2)$$

Systematic expansions to arbitrary order are known [Suzuki 92]

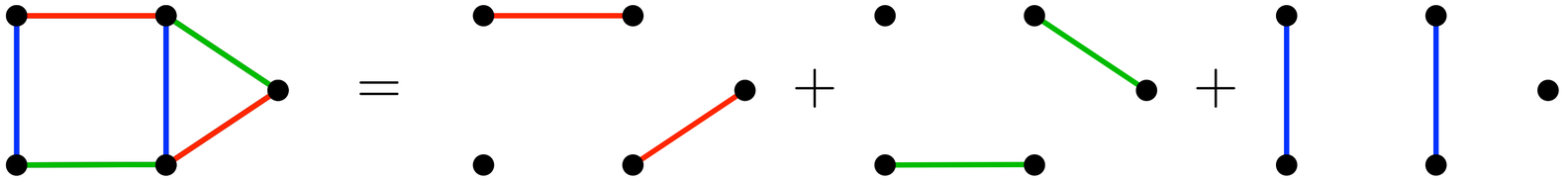
Using the $2k$th order expansion, the number of exponentials required for an approximation with error at most ϵ is at most

$$5^{2k} L^2 \|H\| t \left(\frac{L\|H\|t}{\epsilon}\right)^{1/2k}$$

[Berry, Ahokas, Cleve, Sanders 07]

Sparse Hamiltonians and coloring

Strategy: Color the edges of the graph of H. Then the simulation breaks into small pieces that are easy to handle.



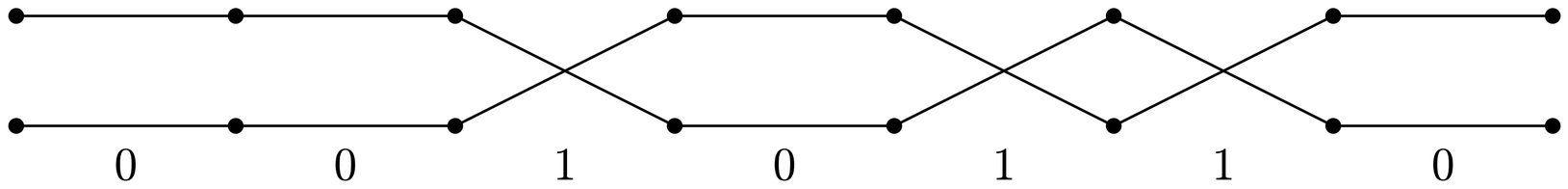
A sparse graph can be efficiently colored using only local information, so this gives efficient simulations.

Sometimes we can do better with different graph decompositions.

[Childs, Cleve, Deotto, Farhi, Gutmann, Spielman 03; Aharonov, Ta-Shma 03; Childs, Kothari 10]

Real-time simulation?

No fast-forwarding theorem: Simulating Hamiltonian dynamics for time t requires $\Omega(t)$ gates.



Complexity of kth order product formula simulation is $O(5^{2k}t^{1+1/2k})$.

Can we give an algorithm with complexity precisely $O(t)$?

Systems simulate their own dynamics in real time!

Hamiltonian simulation by quantum walk

Quantum walk corresponding to H

Alternately reflect about $\mathrm{span}\{|\psi_j\rangle\}_{j=1}^N$,

$$|\psi_j\rangle := |j\rangle \otimes \left(\nu \sum_{k=1}^{N} \sqrt{H_{jk}^*}|k\rangle + \nu_j |N+1\rangle \right),$$

and swap the two registers.

If H is sparse, this walk is easy to implement.

Spectral theorem: Each eigenvalue λ of H corresponds to two eigenvalues $\pm e^{\pm i \arcsin \lambda}$ of the walk operator (with eigenvectors closely related to those of H).

Simulation by phase estimation

$$\begin{aligned} |\lambda\rangle &\mapsto |\lambda\rangle|\widetilde{\arcsin \lambda}\rangle && \text{(phase estimation)} \\ &\mapsto e^{-i\lambda t}|\lambda\rangle|\widetilde{\arcsin \lambda}\rangle \\ &\mapsto e^{-i\lambda t}|\lambda\rangle && \text{(inverse phase est)} \end{aligned}$$

Theorem: $O(t/\sqrt{\epsilon})$ steps of this walk suffice to simulate H for time t with error at most ϵ.

[Childs10; Berry, Childs 12]

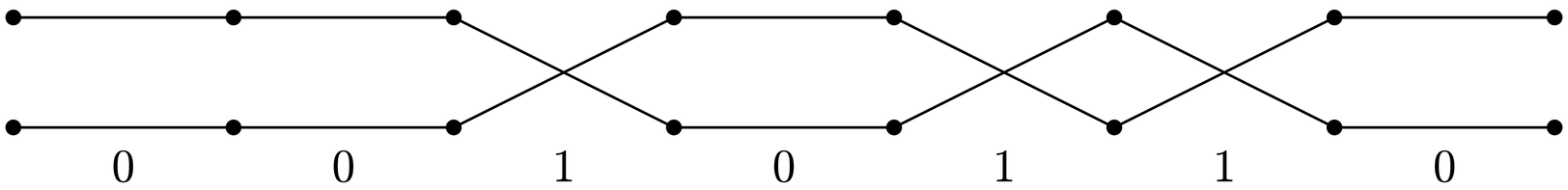
High-precision simulation?

Can we improve the dependence on ϵ?

Many approximate computations can be done with complexity $\mathrm{poly}(\log(1/\epsilon))$:

- computing π
- boosting a bounded-error subroutine
- Solovay-Kitaev circuit synthesis
- and more…

Lower bound (based on the *unbounded-error* query complexity of parity): $\Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$



Quantum walk simulation: $O(1/\sqrt{\epsilon})$

Product formulas ($2k$th order): $O(5^{2k}\epsilon^{-2k})$

Can we do better?

Hamiltonian simulation by linear combinations of unitaries

Main idea: Directly implement the series

$$e^{-iHt} = \sum_{k=0}^{\infty} \frac{(-iHt)^k}{k!}$$

$$\approx \sum_{k=0}^{K} \frac{(-iHt)^k}{k!}$$

Write $H = \sum_\ell \alpha_\ell H_\ell$ with H_ℓ unitary.

Then

$$\sum_{k=0}^{K} \sum_{\ell_1,\ldots,\ell_k} \frac{(-it)^k}{k!} \alpha_{\ell_1} \cdots \alpha_{\ell_k} H_{\ell_1} \cdots H_{\ell_k}$$

is a linear combination of unitaries.

LCU Lemma: Given the ability to perform unitaries V_j with unit complexity, one can perform the operation $U = \sum_j \beta_j V_j$ with complexity $O(\sum_j |\beta_j|)$. Furthermore, if U is (nearly) unitary then this implementation can be made (nearly) deterministic.

Main ideas:

- Implement U with some amplitude:

$$|0\rangle|\psi\rangle \mapsto \sin\theta|0\rangle U|\psi\rangle + \cos\theta|\Phi\rangle$$

- Boost the amplitude for success by *oblivious amplitude amplification*

Query complexity: $O\big(t \frac{\log(t/\epsilon)}{\log\log(t/\epsilon)}\big)$

[Berry, Childs, Cleve, Kothari, Somma 14 & 15]

Tradeoff between t and ϵ

Combining known lower bounds on the complexity of simulation as a function of t and ϵ gives

$$\Omega\left(t + \frac{\log \frac{1}{\epsilon}}{\log\log \frac{1}{\epsilon}}\right) \quad \text{vs. upper bound of} \quad O\left(t \frac{\log \frac{t}{\epsilon}}{\log\log \frac{t}{\epsilon}}\right)$$

Very recent work, using an alternative method for implementing a linear combination of quantum walk steps, gives an optimal tradeoff.

Main idea: Encode the eigenvalues of H in a two-dimensional subspace; use a carefully-chosen sequence of single-qubit rotations to manipulate those eigenvalues.

To compute the rotation angles, we must find the roots of a high-degree polynomial to high precision. This can be done in polynomial time (classically), but it's expensive in practice.

[Low, Chuang 16]

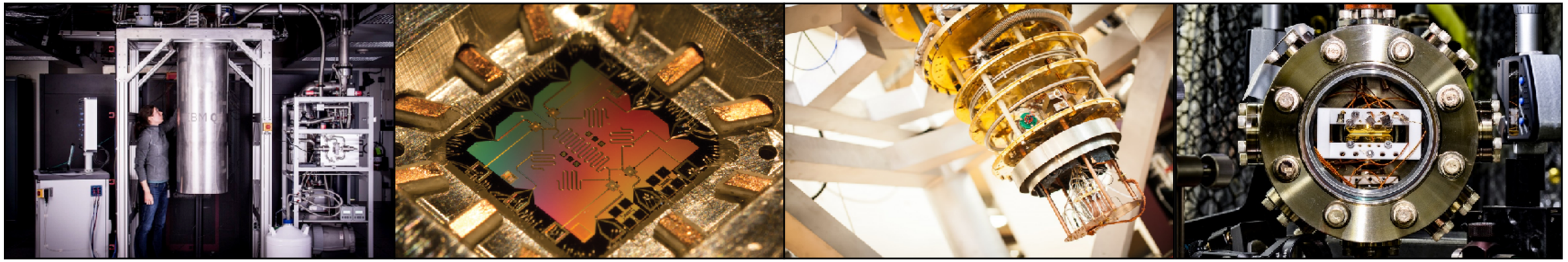
Algorithm comparison

Algorithm	Query complexity	Gate complexity
Product formula, 1st order	$O(d^4t^2/\epsilon)$	$O(d^4t^2/\epsilon)$
Product formula, (2k)th order	$O\left(5^{2k}d^3t(\frac{dt}{\epsilon})^{1/2k}\right)$	$O\left(5^{2k}d^3t(\frac{dt}{\epsilon})^{1/2k}\right)$
Quantum walk	$O(dt/\sqrt{\epsilon})$	$O(dt/\sqrt{\epsilon})$
Fractional-query simulation	$O\left(d^2t\frac{\log(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(d^2t\frac{\log^2(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$
Taylor series	$O\left(d^2t\frac{\log(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(d^2t\frac{\log^2(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$
Linear combination of q. walk steps	$O\left(dt\frac{\log(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(dt\frac{\log^{3.5}(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$
Quantum signal processing	$O\left(dt+\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$	$O\left(dt+\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$

OPTIMAL!

Implementation

Toward practical quantum speedup



IBM | Google/UCSB | Delft | Maryland

Important early goal: demonstrate quantum computational advantage

… but can we find a *practical* application of near-term devices?

Challenges

- Improve experimental systems
- Improve algorithms and their implementation, making the best use of available hardware

Our goal: Produce concrete resource estimates for the simplest possible practical application of quantum computers

What to simulate?

~~Quantum chemistry?~~ Spin systems!

Heisenberg model on a ring: $H = \sum_{j=1}^{n} \left(\vec{\sigma}_j \cdot \vec{\sigma}_{j+1} + h_j \sigma_j^z \right)$ $h_j \in [-h, h]$ uniformly random

This provides a model of *self-thermalization* and *many-body localization*.

The transition between thermalized and localized phases (as a function of h) is poorly understood. Most extensive numerical study: fewer than 25 spins. [Luitz, Laflorencie, Alet 15]

Could explore the transition by preparing a simple initial state, evolving, and performing a simple final measurement. Focus on the cost of simulating dynamics.

For concreteness: $h = 1, \quad t = n, \quad \epsilon = 10^{-3}, \quad 20 \leq n \leq 100$

Algorithms

Algorithm	Gate complexity (t, ϵ)	Gate complexity (n)
Product formula (PF), 1st order	$O(t^2/\epsilon)$	$O(n^5)$
Product formula (PF), (2k)th order	$O(5^{2k} t^{1+1/2k}/\epsilon^{1/2k})$	$O(5^{2k} n^{3+1/k})$
Quantum walk	$O(t/\sqrt{\epsilon})$	$O(n^4 \log n)$
Fractional-query simulation	$O\big(t \frac{\log^2(t/\epsilon)}{\log\log(t/\epsilon)}\big)$	$O\big(n^4 \frac{\log^2 n}{\log\log n}\big)$
Taylor series (TS)	$O\big(t \frac{\log^2(t/\epsilon)}{\log\log(t/\epsilon)}\big)$	$O\big(n^3 \frac{\log^2 n}{\log\log n}\big)$
Linear combination of q. walk steps	$O\big(t \frac{\log^{3.5}(t/\epsilon)}{\log\log(t/\epsilon)}\big)$	$O\big(n^4 \frac{\log^2 n}{\log\log n}\big)$
Quantum signal processing (QSP)	$O(t + \log(1/\epsilon))$	$O(n^3)$
Segmented QSP, q iterates per segment	$O(t^{1+2/q}/\epsilon^{2/q})$	$O\big(n^{3+4/q}\big)$

Algorithms

Algorithm	Gate complexity (t, ϵ)	Gate complexity (n)
Product formula (PF), 1st order	$O(t^2/\epsilon)$	$O(n^5)$
Product formula (PF), (2k)th order	$O(5^{2k}t^{1+1/2k}/\epsilon^{1/2k})$	$O(5^{2k}n^{3+1/k})$
Quantum walk	$O(t/\sqrt{\epsilon})$	$O(n^4 \log n)$
Fractional-query simulation	$O\big(t\frac{\log^2(t/\epsilon)}{\log\log(t/\epsilon)}\big)$	$O\big(n^4\frac{\log^2 n}{\log\log n}\big)$
Taylor series (TS)	$O\big(t\frac{\log^2(t/\epsilon)}{\log\log(t/\epsilon)}\big)$	$O\big(n^3\frac{\log^2 n}{\log\log n}\big)$
Linear combination of q. walk steps	$O\big(t\frac{\log^{3.5}(t/\epsilon)}{\log\log(t/\epsilon)}\big)$	$O\big(n^4\frac{\log^2 n}{\log\log n}\big)$
Quantum signal processing (QSP)	$O(t + \log(1/\epsilon))$	$O(n^3)$
Segmented QSP, q iterates per segment	$O(t^{1+2/q}/\epsilon^{2/q})$	$O\big(n^{3+4/q}\big)$

Circuit synthesis

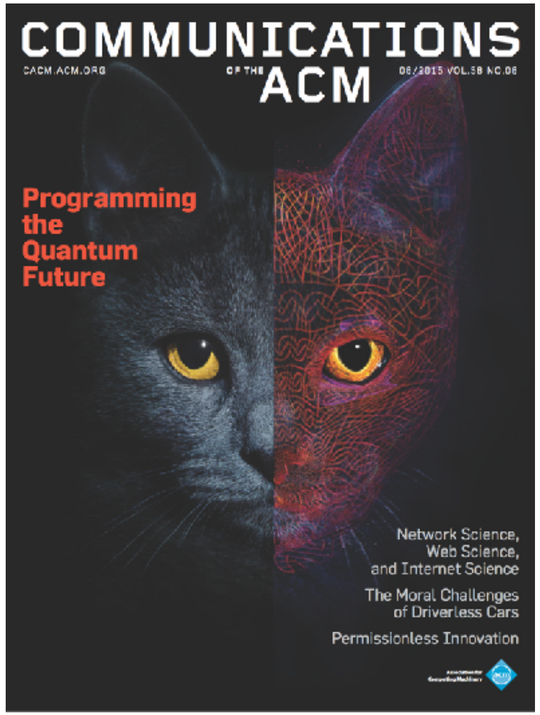
We implemented each of these algorithms using Quipper, a quantum circuit description language that facilitates concrete resource counts

Gate sets:

- Clifford+R_z
- Clifford+T

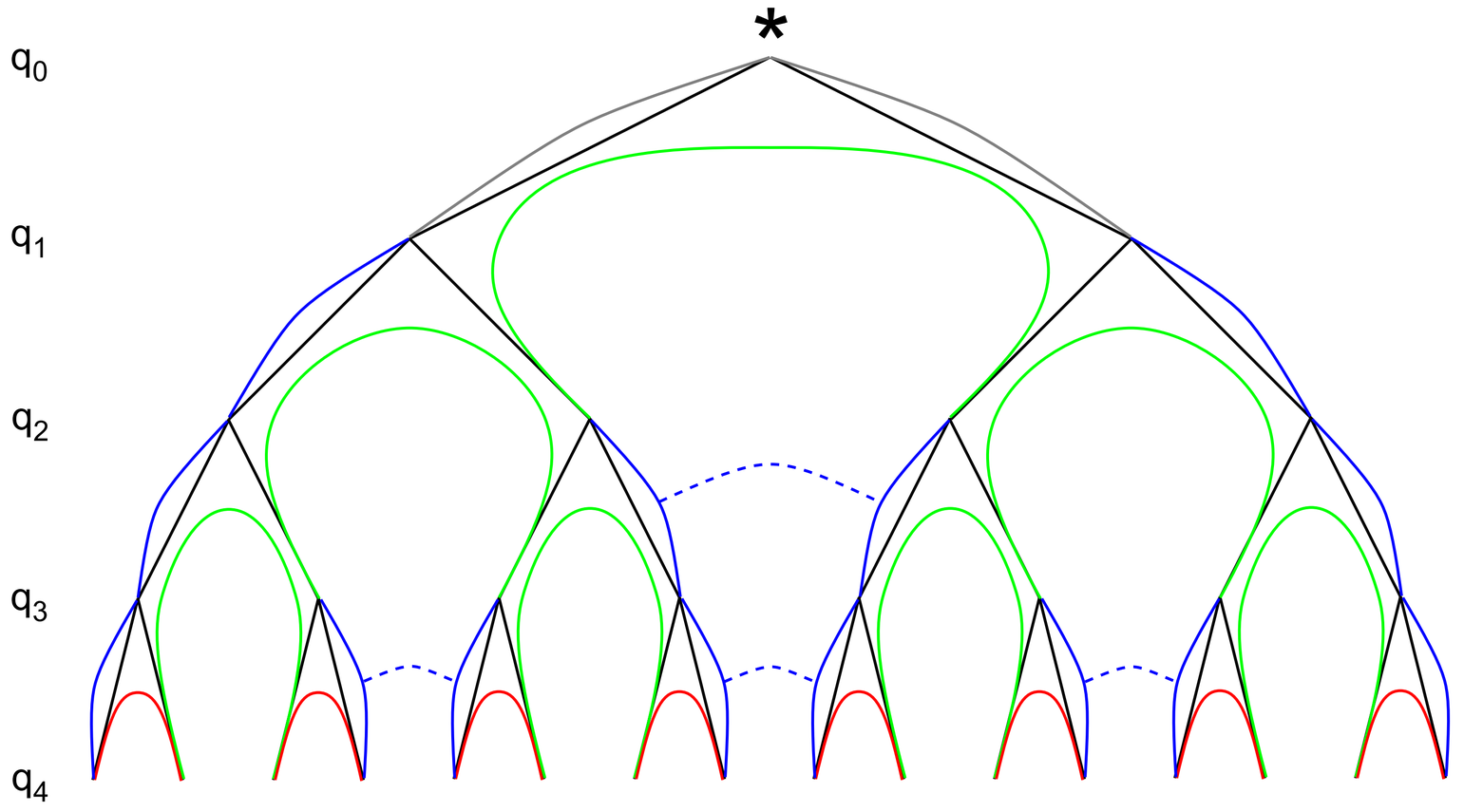
Quipper can produce Clifford+T circuits using recently-developed optimal synthesis algorithms [Kliuchnikov, Maslov, Mosca 13; Ross, Selinger 16]

We verified correctness using simulations of subroutines and small instances.

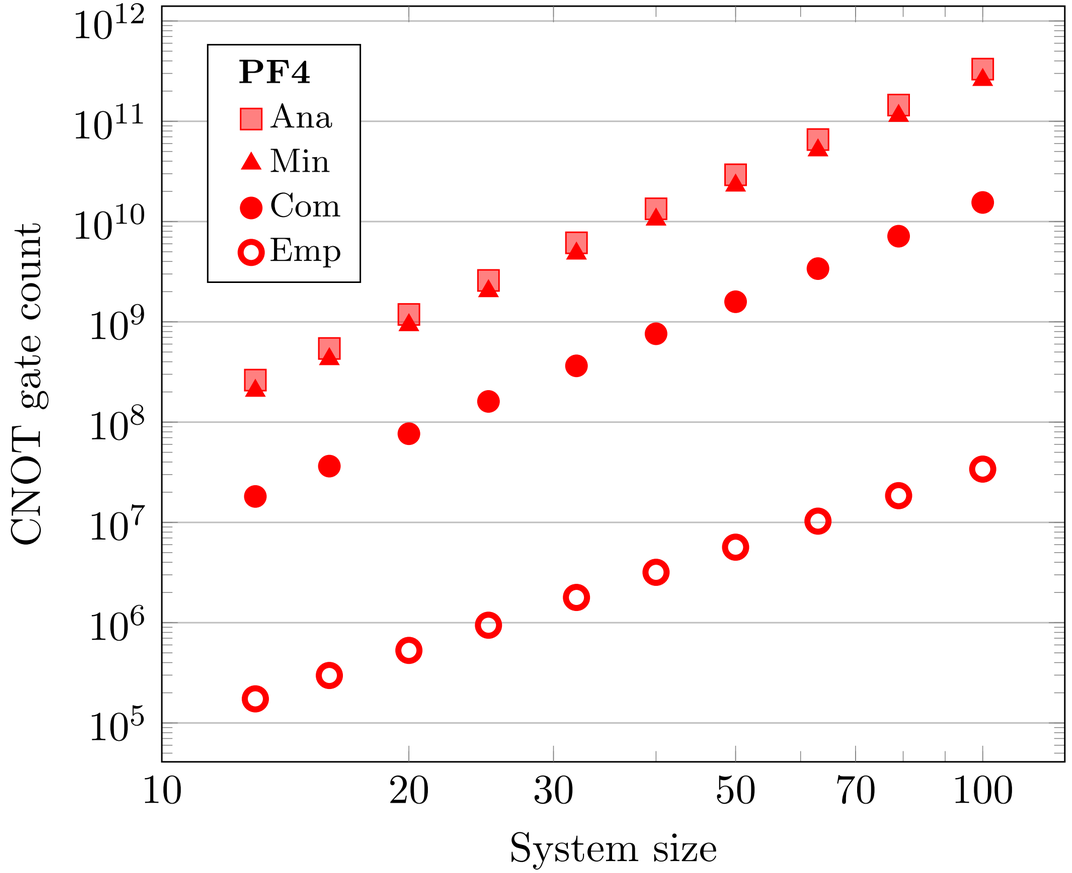


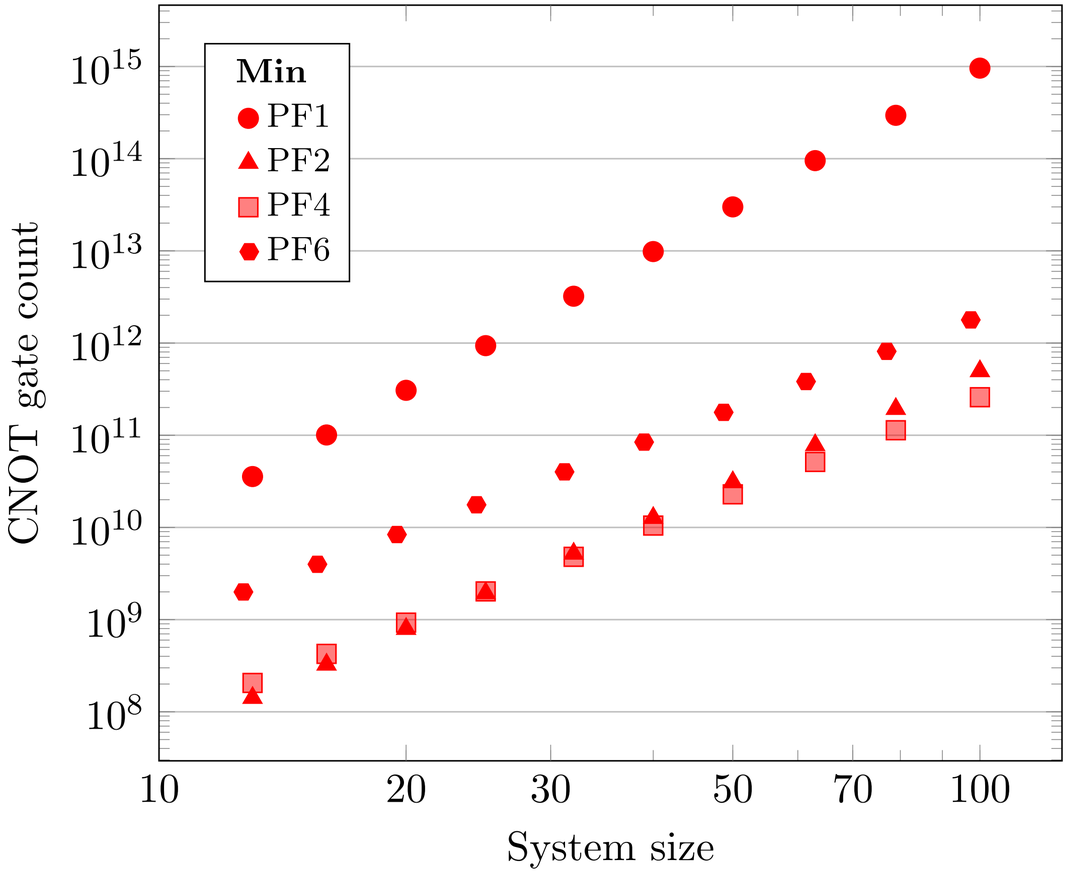

```
multiplexor :: [Double] -> [Qubit] -> Qubit -> Circ ([Qubit], Qubit)
multiplexor as controls target = case controls of
  -- No controls.
  [] -> do
    let angle = as !! 0
    expYt (- angle) target
    return ([], target)

  -- One control.
  [q0] -> do
    let (as0, as1) = split_angles as
    ([], target) <- multiplexor as0 [] target
    target <- qnot target `controlled` q0
    ([], target) <- multiplexor as1 [] target
    target <- qnot target `controlled` q0
    return ([q0], target)

  -- Two controls.
  [q0,q1] -> do
    let (as0, as1) =  split_angles as
    ([q1], target) <- multiplexor as0 [q1] target
    target <- qnot target `controlled` q0
    ([q1], target) <- multiplexor as1 [q1] target
    target <- qnot target `controlled` q0
    return ([q0,q1], target)

  -- Three controls.
  [q0,q1,q2] -> do
    let (as0, as1, as2, as3) = split_angles_3 as
    ([q2], target) <- multiplexor as0 [q2] target
    target <- qnot target `controlled` q1
    ([q2], target) <- multiplexor as1 [q2] target
    target <- qnot target `controlled` q0
    ([q2], target) <- multiplexor as3 [q2] target
    target <- qnot target `controlled` q1
    ([q2], target) <- multiplexor as2 [q2] target
    target <- qnot target `controlled` q0
    return ([q0,q1,q2], target)

  -- Four or more controls.
  qs -> do
    let (as0, as1) =  split_angles as
    let (qhead:qtail) = qs
    (qtail, target) <- multiplexor as0 qtail target
    target <- qnot target `controlled` qhead
    (qtail, target) <- multiplexor as1 qtail target
    target <- qnot target `controlled` qhead
    return (qs, target)

  where
    -- Compute angles for recursive decomposition of a multiplexor.
    split_angles :: [Double] -> ([Double], [Double])
    split_angles l =
      let (l1, l2) = splitIn2 l in
        let p w x  = (w + x) / 2 in
          let m w x = (w - x) / 2 in
            (zipWith p l1 l2, zipWith m l1 l2)

    -- Compute the angles for recursive decomposition of a multiplexor
    -- with three controls, saving 2 CNOT gates, as in the
    -- optimization in Fig. 2 of Shende et.al.
    split_angles_3 :: [Double] -> ([Double],[Double],[Double],[Double])
    split_angles_3 l =
      let (l1, l2, l3, l4) = splitIn4 l in
        let pp w x y z = (w + x + y + z) / 4 in
          let pm w x y z = (w + x - y - z) / 4 in
            let mp w x y z = (w - x - y + z) / 4 in
              let mm w x y z = (w - x + y - z) / 4 in
                let lpp = zipWith4 pp l1 l2 l3 l4 in
                  let lpm = zipWith4 pm l1 l2 l3 l4 in
                    let lmp = zipWith4 mp l1 l2 l3 l4 in
                      let lmm = zipWith4 mm l1 l2 l3 l4 in
                        (lpp, lmm, lpm, lmp)
```

$$\mathrm{select}(V) = \sum_j |j\rangle\langle j| \otimes V_j$$

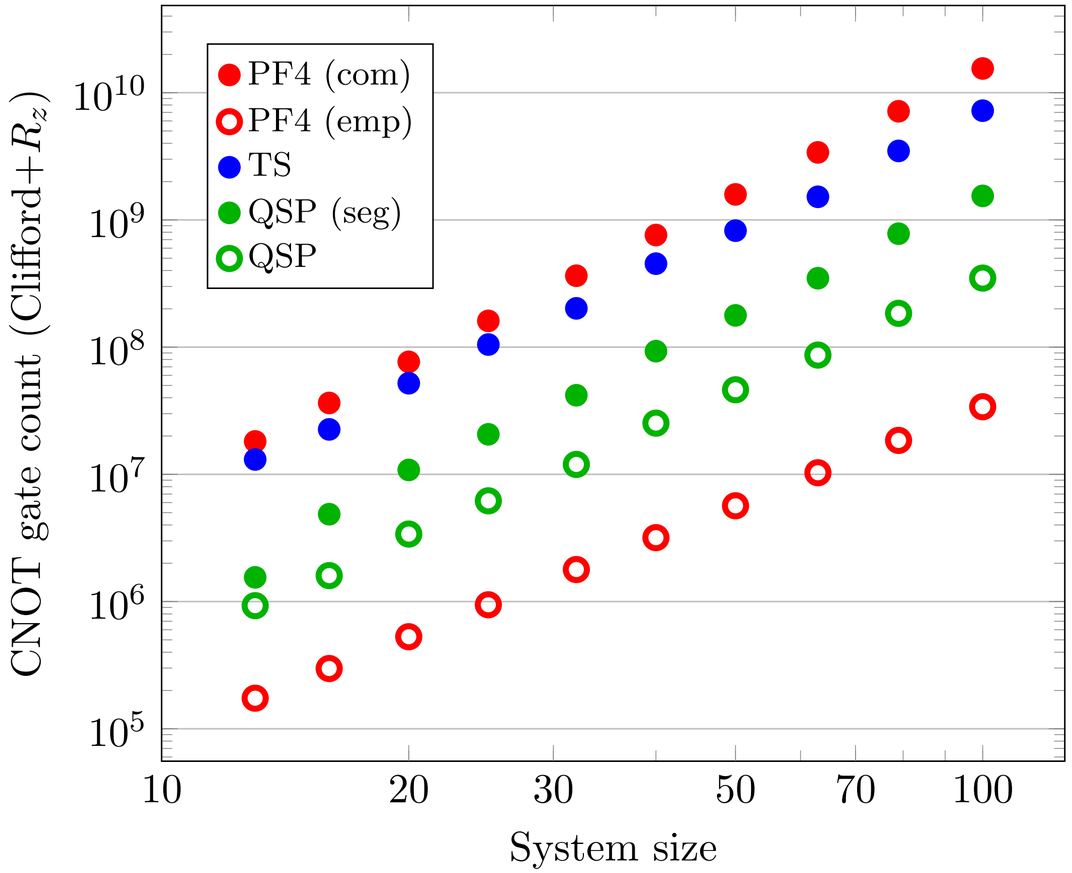


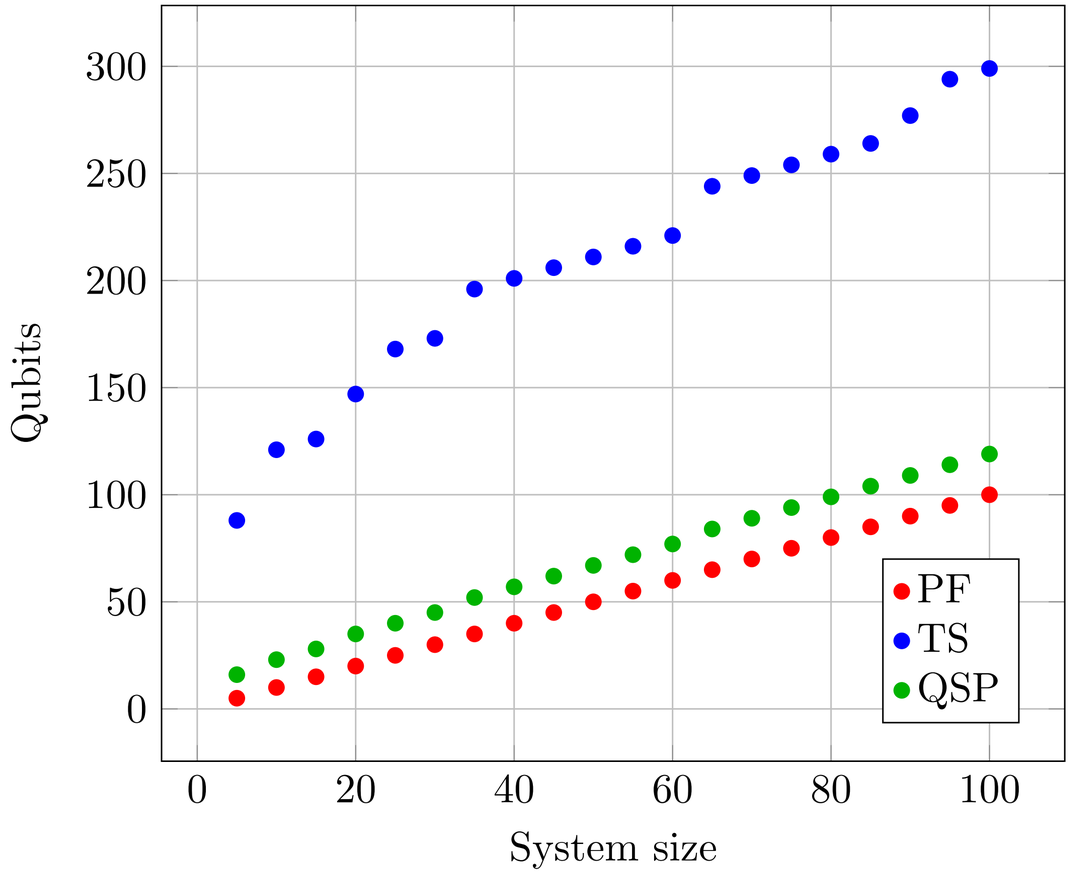
Product formula comparisons



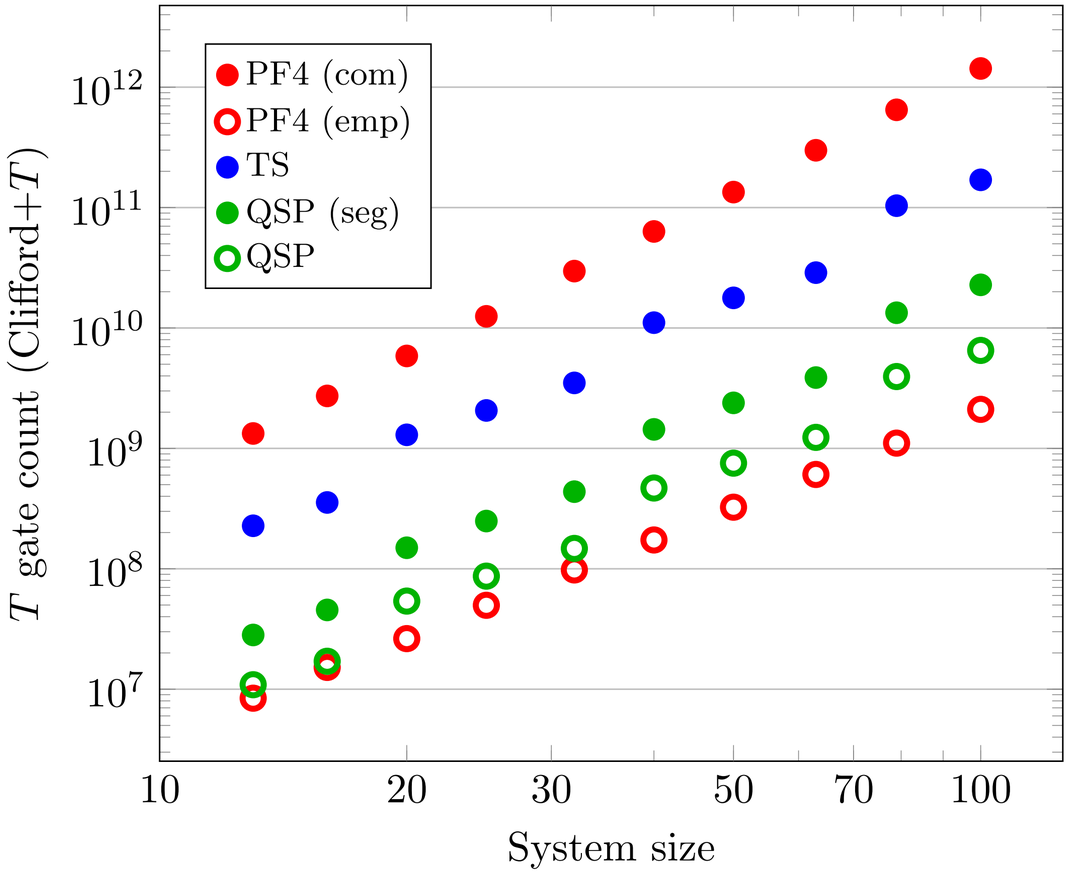


Resource estimates





Resource estimates



Simulating a 50-qubit system (PF4, empirical):

- 50 qubits
- 3.2×10^8 T gates

Factoring a 1024-bit number [Kutin 06]:

- 3132 qubits
- 5.7×10^9 T gates

Simulating FeMoco [Reiher et al. 16]:

- 111 qubits
- 1.0×10^{14} T gates

Outlook

Super-classical quantum simulation without invoking fault tolerance?

- Improved error bounds (e.g., empirical error bound for QSP algorithm?)
- Optimized implementations
- Alternative target systems
- New simulation algorithms
- Experiments!

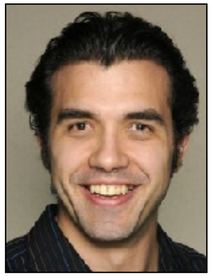
Resource estimates for more practical models

- Architectural constraints, parallelism
- Fault-tolerant implementations

Better provable bounds for simulation algorithms

- Product formula error bounds beyond the triangle inequality
- Efficient synthesis of the QSP circuit

Algorithms


Dominic Berry


Richard Cleve


Robin Kothari

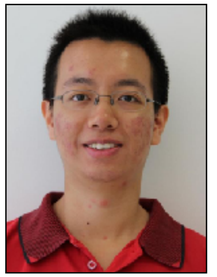

Rolando Somma

Implementation


Dmitri Maslov


Yunseong Nam


Neil Julien Ross


Yuan Su

CIFAR
CANADIAN INSTITUTE FOR ADVANCED RESEARCH

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Product formula error extrapolation (4th order)