Quantum Compiling with the Super-Kitaev Algorithm

Paul Pham

February 11, 2011

1 Abstract

Quantum compilers will be needed to implement algorithms on an 80-qubit quantum computer being constructed in the next five years. Like digital computers, quantum computers need compilers to approximate high-level descriptions of an algorithm using a low-level, universal, machine-dependent instruction set. This work contributes a numerical comparison of the resources needed to run two separate quantum compiling algorithms, along with the underlying open source code. The first is the well-known Solovay-Kitaev theorem which showed that efficient quantum compiling was possible in theory, but with some large performance overheads in practice. The other is a lesser-known result known as Super-Kitaev, which optimizes the compiled circuit depth using parallelization at the expense of more ancilla qubits and a larger overall circuit size. Finally, we discuss some implications for future quantum architectures and also analog computers (for when they come back into fashion).

2 Introduction

Quantum computers can do some pretty amazing things in theory: they can break the RSA cryptosystem using Shor's factoring algorithm, ensure perfectly private communication, do unstructured search, solve random walks, and simulate quantum physical systems much more efficiently than classical computers. But how do we bridge the gap between algorithms that run on paper and those that will soon run on actual machines being built in laboratories as we speak? One important step in that direction is *quantum compiling*, the approximation of a high-level quantum algorithm description to a sequence of low-level, universal quantum gates that depend on our hardware, the "assembly language" of quantum computing.

When describing a quantum algorithm, we use a high-level formalism of gates operating on qubits in the quantum circuit model. We can treat gates operating on an n-qubit quantum computer as unitary matrices of dimension $2^n \times 2^n$ with unit determinant. However, in experimental settings, we can only perform some gates efficiently, and these are local two-qubit or single-qubit operations. Moreover, most of our results for fault-tolerant quantum computing in the presence of noise stipulates that we have a finite number of universal gates we can perform with some limited precision.

Two quantum compiling algorithms are currently known. The first algorithm, by Solovay and Kitaev [3], is one of the central results of quantum computing which states that we can approximate quantum gates efficiently without losing any performance gains over classical computers. The second result, which is more recent but much less well-known, improves Solovay-Kitaev by making a time-space tradeoff and using parallelism [1]. Therefore, we call it Super-Kitaev (a name originated by Aram Harrow).

This work contributes the calculation of physical resources needed to run these two quantum compiling algorithms, open source code to duplicate these results available at `http://quantum-compiler.org`, and much needed publicity for Super-Kitaev.

The rest of this report is organized as follows. First things first, Section 3 defines terms and parameters so that we can discuss quantum compilers with some precision as well as giving asymptotic bounds for specific algorithms. Then Section 4 gives a brief history of quantum compiling. The next two sections describe the two compiling algorithms and how to measure their relative performance. Section 5 reviews the original Solovay-Kitaev result and Section 6 describes Super-Kitaev along with its most resource-intensive modules. Section 7 describe our methods for the performance comparisons, which are given in Section 8. Finally, we make some comments about these results and suggest future directions for extending this work. Ready? Let's go.

3 Preliminaries

3.1 Some Special Operator Notation

Borrowing the notation in [1], we can define two "meta-operators" which takes some unitary U as a parameter. The first describes a controlled-U operation where the control is some single qubit.

$$\Lambda(U) = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U$$

The second describes a registered-U operation, which can be thought of as controlled on some multi-qubit register $|p\rangle$ encoding an m-qubit number p to apply U a certain number of times to a target register.

$$\Upsilon_m(U) : |p\rangle \otimes |\psi\rangle \rightarrow |p\rangle \otimes U^p |\psi\rangle$$

3.2 A Universal Set of Gates

We use the following universal standard set of gates $\mathcal{G}$. The single-qubit rotations about the $x-$ and $z-$ axes (of which X and Z are special cases) are known to be efficient on ion trap implementations, but the set of realizable angles θ is finite.

$$\mathcal{G} = \{H, K, K^{-1}, X, Z, \Lambda(\sigma_x), \Lambda^2(\sigma_x)\}$$

$\Lambda(\sigma_x)$ and $\Lambda^2(\sigma_x)$ are CNOT and Toffoli, respectively. Any current or future physical implementations of a quantum computer will need to efficiently implement this set or an equivalent one. Without proving the universality of $\mathcal{G}$, we note that all known quantum algorithms reduce to it.

3.3 Parameters

The problem of quantum compiling is to translate an entire circuit C of L gates with depth d to a new, compiled circuit C' of size L' and depth d' which approximates C within some error ϵ using some distance measure.

In our code, we use the trace measure introduced by Austin Fowler which disregards the global phase factor, so that we don't waste time trying to approximate the unmeasurable phase of our target gate. Here, l refers to the dimensionality of our system (for $n = 2^l$ qubits).

$$d(U, \tilde{U}) = \sqrt{\frac{l - \parallel \mathrm{tr}(U^\dagger \tilde{U}) \parallel}{l}} \qquad (1)$$

	Solovay-Kitaev	Super-Kitaev
L'	$O(Ln^{3+\nu})$	$O(Ln + n^2 \log n)$
d'	$O(dn^{3+\nu})$	$O(d \log n + (\log n)^2))$

We will be somewhat sloppy and use the terms "error", "precision", and "accuracy" interchangeably when approximating gates. There is some overhead in the compiled circuit, so in general C' is larger (that is, $L' > L$ and $d' > d$). It's also known that in order to approximate a circuit with L gates to a total precision of ϵ requires each gate to be approximated to a precision of $n = O(\log(L/\epsilon)$. We'll call the classical preprocessing time to produce C' as T.

Circuit depth is a heuristic for how parallelizable our circuit is. For example, in an ion trap, if we had multiple lasers, we could "flatten" our circuit into layers with bounded fan-in and fan-out and operate on multiple ions in parallel. All other things being equal, a circuit with low depth will complete faster than one with high depth, although in practice we can only execute fixed-width circuits.

3.4 Quantum Coprocessor Model

All experimental implementations of quantum computers treat them as an auxiliary device controlled by a classical computer. This is the way quantum computers will function for the foreseeable future, and many quantum algorithms can actually be split into classical and quantum parts to reflect this distinction. For example, Solovay-Kitaev is a completely classical algorithm which is run before a quantum algorithm to yield a deterministic set of gates. Super-Kitaev also contains classical postprocessing as part of its parallelized phase-estimation. Since classical computers are well understood and pretty fast, we will neglect the performance of these classical parts if they are polynomial in time. However, we will discuss one aspect of classical overhead later since it appears to be intractable.

3.5 Asymptotic Bounds

The Solovay-Kitaev and Super-Kitaev algorithms compile circuits with a size and depth which depend on the desired precision via the parameter n as shown below. We'll see these asymptotic bounds reflected later in the actual numerical results in Section 8. The version of Solovay-Kitaev in Section 5 has a larger exponent for the logarithm but is easier to understand.

where ν is a small positive constant. The improve Solovay-Kitaev with the improved $3+\nu$ exponent and Super-Kitaev are both described in [1].

4 Related Work

In 1995, Seth Lloyd found that almost any two distinct single-qubit rotations are universal for approximating an arbitrary single-qubit rotation, but that this approximation could be exponentially long in both time and length $T, L = (O(1/\epsilon))$ [7].

The theorem which is now called Solovay-Kitaev was discovered by Solovay in 1995 in an unpublished manuscript and independently later discovered by Kitaev in 1997 [8] which showed that $T, L = O(\log^c 1/\epsilon)$ for c between 3 and 4.

In 2001, Aram Harrow completed his undergrad thesis arguing that it would be difficult to beat $c < 2$ for the above bounds using a successive approximation method[5].

In 2002, Kitaev, Shen, and Vyalyi published their book which contains an application of parallelized phase estimation towards simulating a quantum circuit (what we are calling Super-Kitaev) [1]. That is, an alternative quantum compiler to Solovay-Kitaev which has asymptotically better circuit depth and $T = O(1)$ but using ancillary qubits and increased circuit size.

In 2003, Harrow, Recht, and Chuang demonstrated that a certain universal set could be used to saturate the lower bound $L = O(\log 1/\epsilon)$ but it remains an open problem whether any efficient algorithm exists which can do this in tractable T [6].

In 2005, Dawson and Nielsen published their pedagogical review paper of Solovay-Kitaev [3].

In 2010, Burrello, Mussardo, and Wan discovered a quantum compiler for topological quantum computing which saturates the lower bound ($c = 1$) for non-Abelian anyons [2].

5 Review of Solovay-Kitaev

The essential structure of Solovay-Kitaev is recursive, successive approximation. Our pseudo-code and explanation follows the excellent exposition in [3] and [5]

1: FUNCTION $\tilde{U}_n \leftarrow$ SOLOVAY-KITAEV(U, n)
2: **if** $n == 0$ **then**
3: $\quad \tilde{U}_n \leftarrow$ BASIC-APPROX(U)
4: **else**
5: $\quad \tilde{U}_{n-1} \leftarrow$ SOLOVAY-KITAEV$(U, n-1)$
6: $\quad A, B \leftarrow$ FACTOR$(U\tilde{U}_{n-1}^\dagger)$
7: $\quad \tilde{A}_{n-1} \leftarrow$ SOLOVAY-KITAEV$(A, n-1)$
8: $\quad \tilde{B}_{n-1} \leftarrow$ SOLOVAY-KITAEV$(B, n-1)$
9: $\quad \tilde{U}_n \leftarrow \tilde{A}_{n-1}\tilde{B}_{n-1}\tilde{A}_{n-1}^\dagger\tilde{B}_{n-1}^\dagger\tilde{U}_{n-1}$
10: **end if**return $\tilde{U}_n$

The BASIC-APPROX function above does a lookup (via some kd-tree search maneuvers through higher-dimensional vector spaces) using the results of precompiled sequences from $\mathcal{G}$. This can be done offline and reused across multiple runs of the compiler, assuming $\mathcal{G}$ for your quantum computer doesn't change.

The FACTOR function performs a balanced group commutator decomposition, $U = ABA^\dagger B^\dagger$, and then recursively approximates the A and B operators using Solovay-Kitaev. Intuitively, when they are multiplied together again, along with their inverses, their errors (which go like ϵ) are symmetric and cancel out in such a way that the resulting product U has errors which go like ϵ^2. In this manner, we can eventually sharpen our desired error down to any value.

Each level n of recursion solves the problem of compiling U_n by combining five gates compiled at the lower $n-1$ level. Therefore, the compiled circuit size is upper-bounded by 5^n, and this is in general the same as the circuit depth (since not all gates in the $\mathcal{G}$ commute).

6 The Main Algorithm

The Super-Kitaev algorithm takes a radically different approach than Solovay-Kitaev by identifying that most gates can be easily reduced to $\mathcal{G}$ in constant time with the exception of the "targetless" controlled phase operator, which will take some sweat to approximate.

$$\Lambda(e^{i\phi}) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$$

Thankfully [1] is a self-contained reference for enacting $\Lambda(e^{i\phi})$ with several modules which are useful in their own right: parallelized phase estimation procedure, parallel iteration of finite automata, and a logarithmic depth quantum arithmetic operations such as addition (and subtraction) described in 6.8, multiplication described in 6.9, and division described in 6.10.

It is now useful to discuss certain states with magical properties. Their superposition is easy to form,

they are the eigenstates of the afore-mentioned addition operator. Used with phase estimation, this would allow us to enact a random phase shift simply by adding numbers, which is better than not being able to shift phases at all (or only being able to do it with a complicated operator). Moreover, it turns out we can copy these states also using simple addition. These ideas make Super-Kitaev a beautiful result regardless of the outcome of our later numerical comparisons.

6.1 Magic States

Suppose you had some magic state $|\psi\rangle$, given to you from some all-powerful being, parameterized by the integers n and k such that

$$|\psi_{n,k}\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{-2\pi i jk/2^n} |j\rangle$$

where $0 < k < 2^n$, k is odd.

Note that the magic states $|\psi\rangle$ look suspiciously like a QFT state of the computational basis. However that doesn't help us here, because QFT in general requires $O(n^2)$-depth and we need a logarithmic-depth compiler. This is intuitively why these states are hard to create. If we were able to construct them easily, we would also have a quicker way of implementing QFT.

However, what we can do is create a superposition over all odd $k = (2s-1)$ by starting in the state $|0\rangle^{\otimes n}$, applying a Hadamard to the most significant qubit to get an equal mix of $|0\rangle$ and $|1\rangle$, then put a negative sign on the $|1\rangle$ component by applying σ^z to the same qubit.

$$|\eta\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|2^{n-1}\rangle = \frac{1}{\sqrt{2^{n-1}}} \sum_{s=1}^{2^{n-1}} |\psi_{n,2s-1}\rangle$$

More obviously relevant to our overall goal of approximating $\Lambda(e^{i\phi})$, we can enact a phase shift simply by doing addition, which is a result of these states being eigenvectors of the modular addition operator defined

$$A|\psi_{n,k}\rangle = e^{2\pi i\phi_k}|\psi_{n,k}\rangle$$

$$A|j\rangle \to |j+1 \mod 2^n\rangle$$

where finding the eigenvalue $e^{2\pi i\phi_k}$ corresponds to finding the phase $\phi_k = k/2^n$.

Repeated application of A (say p times) would result in a phase added to the eigenstate equal to a multiple of $e^{2\pi ip/2^n}$

$$A^p|\psi_{n,k}\rangle = e^{2\pi i\phi_k/2^n}|\psi_{n,k}\rangle \tag{2}$$

This explains why we don't find even k interesting, since then we would not get a cyclic distribution of 2^n different phases, since only odd k are coprime with 2^n. The exception is $k = 0$, since this is the equal superposition of computational basis states, which we can also efficiently create. This will be a useful starting point later on to create magic states for odd k.

$$|\psi_{n,0}\rangle = H^{\otimes n}|0^n\rangle$$

Suppose we have a certain state $\psi_{n,k}$ but we want to get enact a phase shift $e^{2\pi il/2^n}$. We can do this by solving $p = p(s,l)$ in this equation:

$$(2s-1)p \equiv l(\mod 2^n) \tag{3}$$

Stipulating k to be odd guarantees that there is a unique solution p.

We then applying A^p as follows:

$$\Upsilon_n(A)|p,\psi_{n,k}\rangle \to e^{2\pi il/2^n}|p,\psi_{n,k}\rangle \tag{4}$$

But p is in general hard to solve for, unless $k = 1$, in which case $p = l$. Therefore, we will later see that $|\psi_{n,1}\rangle$ is a desirable state to have.

Finally, to copy the state $|\psi_{n,k}\rangle$ it suffices to apply the following operator which only uses subtraction (addition with one addend and the outcome negated in two's complement representation).

$$|\psi_{n,k}\rangle^{\otimes m} = W^{-1}\left(|\psi_{n,0}\rangle^{\otimes}(m-1)\otimes|\psi_{n,k}\rangle\right)$$

where W is defined by

$$W : |x_1,\ldots,x_{m-1},x_m\rangle \to |x_1,\ldots,x_{m-1},x_1+\ldots+x_m\rangle \tag{5}$$

Armed with these properties, we're now ready to enact arbitrary phase shifts.

6.2 Super-Kitaev Steps

Given a circuit C to compile,

1. Precompile C into gates from $\mathcal{G} \cup \{\Lambda(e^{2\pi i l/2^n})\}$ using the results from Section 6.3 in $O(1)$ time, depth, and size. Now we are done with the single-qubit gates and CNOT, and we have computed the values $\{l_1, \ldots, l_m\}$ that allow us to approximate our desired m $\Lambda(e^{i\phi})$ gates as $\phi \approx l/2^n$ to within precision 2^{-n}.

2. Create the state magic state $|\psi_{n,0}\rangle$ with n Hadamards.

3. Turn it into $|\psi_{n,1}\rangle = \Upsilon(e^{-2\pi i/2^n})\,|\psi_{n,0}\rangle$ using the registered phase shift procedure in Section 6.4 This is done with a circuit of size $O(n^2 \log n)$ and $O((\log n)^2)$ depth.

4. Make m copies of the state $|\psi_{n,1}\rangle$ out of one copy by applying the addition operation A.

5. Simulate each $\Lambda(e^{2\pi i l/2^n})$ using one copy each of $|\psi_{n,1}\rangle$, to which we can add our values l using $\Upsilon(A)$. This takes size $O(mn)$ and depth $O(\log n)$, since we can enact all these phase shifts in parallel.

Now for the resource calculations of these individual steps and their substeps.

6.3 Precompiling to the Standard Set

This is the completely classical preprocessing step for Super-Kitaev which assumes we can express any n-qubit gate as a tensor product of single- and two-qubit gates. From there, how do we get down to our universal set?

We use the Euler angle decomposition into X and Z rotations, we can express any unitary U as follows, with real angles ϕ, γ, β, α:

$$U = e^{i\phi} e^{i(\gamma/2)\sigma^z} e^{i(\beta/2)\sigma^x} e^{i(\alpha/2)\sigma^z}$$

This involves solving four equations in four variables and can be done in constant time.

Now how do we implement each of the elements of the Euler angle decomposition from the $\mathcal{Q}$? Using these identities:

$$e^{i\phi} = \Lambda(e^{i\phi})\sigma^x \Lambda(e^{i\phi})\sigma^x$$

$$\sigma^x = H\Lambda(e^{i\pi})H$$

$$e^{i\phi\sigma^z} = \Lambda(e^{-i\phi}\sigma^x \Lambda(e^{i\phi})\sigma^x$$

$$e^{i\phi\sigma^x} = He^{i\phi\sigma^z}H$$

In addition, any controlled operator $\Lambda(U)$ where $U \in SU(2)$ can be simulated using $\mathcal{Q} \cup \{\Lambda(e^{i\phi})$.

We can represent the controlled operator as $\Lambda(U) = \Lambda(e^{i\phi})\Lambda(V)$. $\Lambda(e^{i\phi})$ is already in our set, so it remains to implement $\Lambda(V)$, where $V \in SU(2)$. Geometrically, we can decompose any three-dimensional rotation into two 180° rotations about appropriate axes. Using the homomorphism between $SO(3)$ and $SU(2)$ and the fact that σ^x corresponds to a 180° rotation about the x-axis.

$$V = A\sigma^x A^{-1} B\sigma^x B^{-1}$$

6.4 Registered-Phase Shift

Registered phase shifting is the only use of phase estimation in Super-Kitaev but it is the most resource-intensive step. We would like to enact the operator $\Upsilon(e^{-2\pi i/2^n})$ on $|\psi_{n,0}\rangle$, which is easy to create, to get $|\psi_{n,1}\rangle$, which we can later use to enact $\Lambda(e^{2\pi i l/2^n})$.

1. Create the state $|\eta\rangle$ as described in 6.1.

2. Measure k by determining the phase $\phi_k = k/2^n$ with precision $\delta = 2^{-n}$ by using phase estimation. One of the n-qubit registers now contains $|\psi_{n,k}\rangle$ for some fixed k. This is done by parallelized phase estimation described in 6.5. This is done by a circuit of size $O(n^2)$ and depth $O(\log n)$.

3. Solve for p in equation Equation 3 using modular division, described in 6.10.

4. Apply A^p to the n-qubit register containing $|\psi\rangle$, which enacts the desired phase shift.

6.5 Parallelized Phase Estimation

One of the key components of the registered phase shifting procedure described in the previous section is the ability to "pick" a random eigenstate $|\psi_k\rangle$ of a unitary operator U and measure its corresponding eigenvalue (phase) ϕ_k with some degree of precision $\delta = 2^{-n}$ and error probability $\epsilon = 2^{-l}$. As n increases, the phases generally become closer together, which is why we need exponential precision to distinguish between them. Of course, this exactly describes the phase estimation procedure, a key technique in many quantum algorithms developed by Kitaev in his derivation of Shor's factoring result [**?**].

$$U\,|\psi_k\rangle = e^{2\pi i\phi_k}\,|\psi_k\rangle$$

Phase estimation holds some superposition of eigenstates $\sum_i \alpha_i |\psi_i\rangle$ in an n-qubit target register, to which it applies repeated measuring operators $\Lambda(U^{2^k})$ controlled on some t-qubit register, which holds an approximation $p\tilde{h}i$ to the real phase ϕ. The unitary U is applied in successive powers of two to get power-of-two multiples of the phase for increased precision. The error probability of approximating the phase to within a given precision is given by the following:

$$\Pr\left[|\phi - p\tilde{h}i| \geq \delta\right] \leq \epsilon$$

The parameter $t = t(\delta, \epsilon)$ encodes the dependence of the number of $\Lambda(U)$ measuring operators as a function of our desired δ and ϵ. It varies according to the exact phase estimation procedure used.

The popular version of phase estimation presented in [**?**, **?**, nc00] requires t repeated controlled applications of some unitary U (and its successive powers as U^{2^k}, $0 < k < 2^t$) to a target state which holds some superposition of its eigenvectors, controlled by t bits which will hold the approximation to a corresponding eigenvalue (phase). This version requires applying an inverse quantum Fourier transform (QFT), which is already high-depth and way more inefficient than our desired quantum compiler.

To achieve our desired low-depth, we can "parallelize" the application of $\Lambda(U)$ by interpreting the $t = (n+2)s$ control bits as an n-bit number q and apply $\Lambda(A^q)$ only once. In the Super-Kitaev procedure, A is the addition operator on an n qubit target register containing $\psi_{n,k}$, so we can only effectively add the lowest n bits of q. Furthermore, the eigenvalues of A are rational with a fixed denominator, $\phi_k = k/2^n$. To avoid the inverse QFT, we can do a classical postprocessing step, which we'll mostly skip over for fairly good reasons, and then we'll do a detailed resource count of parallelized phase estimation.

6.6 Classical Postprocessing

It is now the point to mention that Kitaev's phase estimation procedure contains a post processing step which is completely classical in character, in that they involve a measurement. If this measurement is projective and the outcomes are completely classical, the remaining steps can be done on our classical computer (recall our quantum coprocessor model), and the results fed back into our quantum subroutine, registered phase shifting. Therefore, as long as we can perform these classical algorithms in polynomial time (which we can), we don't really care about the equivalent circuit size and depth.

The steps of classical postprocessing, which will determine some of the parameters in the earlier, quantum part of phase estimation are as follows.

1. Estimate the phase and its power-of-two multiples $2^j\phi_k$ to some constant, modest precision δ'', where $0 \leq j < (n+2)$. For each j, we apply a series of s measuring operators targeting the state $|\nu\rangle$ controlled on s qubits in the state $(|0\rangle + |1\rangle)/\sqrt{2}$, essentially encoding the $2^j\phi_k$ as a bias in a coin, and flipping the coin s times in a Bernoulli trial, counting the number of 1 outcomes, and using that fraction to approximate the real $2^j\phi_k$.

2. Sharpen our estimate to exponential precision $1/2^(n+2)$ using the $(n+2)$ estimates, each for different bits in the binary expansion of phi_k. Multiplication by successive powers-of-two shift these bits up to a fixed position behind the zero in a binary fraction representation, where we can use a finite-automata and a constant number of bits to refine our $O(n)$-length running approximation.

Three things are worth mentioning about the interrelation of the parameters between these two steps. Since our phases all have a denominator of 2^n, there is no need to run the continued fractions algorithm on multiple convergents, as is the case with period-finding in Shor's factoring algorithm. Furthermore, the phases are $1/2^n$ apart, therefore it suffices to approximate the phases to within $1/2^{n+2}$ in order to break ties, which is where our range for j comes from above.

The number of trials s comes from the Chernoff bound:

$$\Pr\left[|s^{-1}\sum_{r=1}^{s} v_r - p_*| \geq \delta''\right] \leq 2e^{-2\delta'^2 s}$$

Setting this equal to the desired error probabiliy ϵ we get

$$s = \frac{1}{2\delta''^2}\ln\frac{1}{\epsilon}$$

We are actually estimating the values $\cos(2\pi\cdot 2^j\phi_k)$ and $\sin(2\pi\cdot 2^j\phi_k)$, so if we wish to know $2^j\phi_k$ with precision δ'', we actually need to determine the $\cos(\cdot)$

δ''	δ'	$1/(2\delta''2)$
1/16	0.0019525	131, 160
1/8	0.0078023	8, 213
1/4	0.0310880	517

and sin(·) values with a different precision δ', lower-bounding it with the steepest part of the cosine and sine curves.

$$\delta' = 1 + cos(\pi - \delta'')$$

The factor $\frac{1}{2\delta''^2}$ depends on the constant precision with which we determine our $2^j\phi_k$ values. Since classical time is cheap and quantum gates are expensive, it makes sense to minimize the number of trials s. The following table shows the corresponding values of $1/(2\delta''^2)$ and δ' as a function of various choices for δ'.

By making our δ' exponentially worse (doubling it) we are only increasing the range of j a linear amount (by one). In general, for $\delta' = 1/2^l$, we get a final estimate for $\phi = 2^{m-3}$

However, projective measurements are irreversible, and we may wish to uncompute the phase estimation procedure to restore our ancilla to their initialized state and reuse them later on. In that case, the post-processing can actually be done on a quantum computer using reversible gates and without projective measurements. That's why the authors of [1] go to some care to show that all the classical postprocessing steps can be done in polynomial-size and logarithmic-depth circuit. However, to simplify our analysis, we assume the case in the previous paragraph, and accept the loss of n ancilla qubits, After all, we only run phase estimation once to get our initial $|\psi_{n,1}\rangle$ state.

6.7 Steps in the Parallelized Phase Estimation Algorithm

The main steps in parallelized phase estimation as applied to Super-Kitaev are:

1. Begin with a t-qubit ancilla register initialized to $|0\rangle^{\otimes t}$.

2. Place the t-qubit register into an equal superposition by applying n Hadamard gates.

3. Treat t as $2s$ groups of bits, each encoding an n-bit number. Sum them up out-of-place, retaining only the lowest n-bits, to get the superposition of all n-bit numbers, $1/(\sqrt{2^n})\sum_{i=0}^{2^n-1}|q_i\rangle$. Call this register $|q_i\rangle$.

4. Reverse the first step by applying another n Hadamards.

5. Apply the gate $\Upsilon(A)$ to the target $|\nu\rangle$ controlled on $|q_i\rangle$, which is equivalent to adding all q_i in superposition (in place).

6. Measure the register $|q_i\rangle$ in the computational basis to get a particular value q and collapse the register to $|q\rangle$. All $t = (n-2)s$ now contain classical 0 or 1 as outcomes of $(n+2)s$ Bernoulli trials.

7. Read out these outcomes into our classical controller and perform the postprocessing described above to get an approximation of ϕ with precision δ.

6.8 Quantum Adding

Since adding is such a common operation in Super-Kitaev and many other quantum algorithms, it's important for us to make sure we can do these efficiently. The original development of Super-Kitaev in [1] gives us two ways to add efficiently. The first way reduces the sum of three numbers (a, b, c) to the sum of two encoded numbers (u, v) in constant depth, simply by encoding the bitwise sum $a_i + b_i + c_i$ as a 2-bit element from $\{0, 1, 2, 3\}$. We arbitrarily choose u_i to be the low-order bit and v_i to be high-order. This can be used to parallelize the sum of m numbers in a $\log_{3/2} m$-depth tree, with $m - 2$ encoded addition operations total.

The encoded adding procedure is out-of-place and requires two ancillae (one to hold the output bit u_i or v_i). In order to apply Bennett's uncomputing trick to perform this operation in-place, we round the number of ancillae up to three to match the number of inputs, and we use 3 SWAP gates (consisting of 3 CNOTs each). Because v_i is high-order, it always gets shifted up one bit: $v_0 = 0$ and v_n the high-order carry bit is discarded for sums modulo 2^n.

The resource counts of Super-Kitaev encoded adding in-place of m n-bit numbers are shown below.

Resource	ENC-ADD
Depth	9
Toffoli Gates	$12(n-1)+6$
CNOT Gates	$21(n-1)+15$
NOT Gates	$10(n-1)$
Ancillae	$3n$

We can combine the final two encoded numbers into a normal $(O(\log m)+n)$-bit number using the second way of adding. This method uses parallelized iteration of general finite automata (FA) to achieve $O(\log n)$-depth circuits for $O(n)$ inputs. Many functions, most importantly binary addition and sharpening estimated phase values to exponential precision, can ingeniously be recast in this parallelized FA setting. First, n copies are created of the FA transition function table. Each copy is then narrowed by its corresponding input, composing adjacent function tables in a binary tree hierarchy, and applying these composed functions to "skip ahead" in the FA and compute intermediate states in parallel.

However, while achieving good asymptotic bounds, this approach has some disadvantages when applied to quantum adding: an overly-general narrowing procedure, and the inability to add in place without resorting to Bennett's uncomputing trick. For these reasons, we use the quantum carry-lookahead adder (QCLA) presented in [4]. That reference also contains the resource counts for in-place addition of two n-bit numbers, which we denote QCLA(n). QCLA follows the spirit of parallel iteration using classical carry-lookahead techniques, in essence "hardcoding" the narrowing and composing steps above for the carry bits generated and propagated in binary addition.

By combining Super-Kitaev's encoded adding with the QCLA, we can now add m numbers of n-bits each with a circuit of size $O(mn)$ and depth $O(\log m + \log n)$.

We denote the final resource counts of adding as the following:

$$\text{ADD}(m,n) \quad = \quad \text{QCLA}(n) + \text{ENC-ADD}(m,n)$$

6.9 Quantum Multiplication

Now that we know how to add quantum numbers efficiently, we can use this to implement a similarly low-depth multiplication circuit.

Figure 1: Piecewise definition of $c(i)$ shifted multiples for multiplication

Consider the inputs to our quantum multiplier as two n-bit numbers, a and b. Our grade school multiplication algorithm simply sums up n copies of a, where $c(i)$ denotes a shifted up by i bits and multiplied by bit b_i.

There are n such shifted multiples $c(i)$. By summing them up and retaining only the low-order n-bits, we are reducing multiplication of two n-bit numbers, modulo 2^n, to the addition of n n-bit numbers, which we already know how to do in parallel. In fact, we can do it in $\log(n)+\log(n) = O(\log n)$ depth. To hold the shifted multiples, we will need $n(n+1)/2$ ancillae, since each bit shift leaves a low-order zero bit that we don't need to include in our sum.

We define the resource counts of in-place multiplication as follows.

$$MULT-2(n) = ADD(n,n) + n(n+1)/2\text{ancillae}$$

6.10 Quantum Divider

The most resource-intensive step of the registered phase shift operation is solving $p = p(s,l)$ in Equation 3 for a given resulting eigenvector $|\psi_{2s-1}\rangle$ after phase estimation and a desired phase shift $exp(2\pi il/2^n)$. This involves finding a modular inverse, which we can convert to a product of sums (addition and multiplication, which we already know how to do efficiently).

To find $y = (2s-1)^{-1} \mod 2^n$, it should have the property $y \cdot (2s-1) \equiv 1 \mod 2^n$ or equivalently $-y \cdot (2s-1) \equiv -1 \mod 2^n$. As a guess, we will choose the $-y$ version and have it be a fraction with $(2s-1)$ in the denominator and something in the numerator which is 1 less than a multiple of 2^n, such as $2^n s^n - 1$. We choose this to resemble the closed form of a geometric series, as shown below.

$$-y = \frac{2^n s^n - 1}{2s-1} = \frac{(2s)^n - 1}{(2s)-1} = \frac{1-(2s)^n}{1-(2s)} = \sum_{i=0}^{n} (2s)^i$$

We use the following observation to convert this geometric series into a product of sums involving power-of-two exponents, which makes for easy multiplication via bitshifting.

$$\sum_{k=0}^{2^l-1} z^k = (1+z)(1+z^2)(1+z^4)\cdots(1+z^{2^{l-1}})$$

In general a geometric series with $m = 2^l$ terms involves m sums and m products. Using the above formula, we have $\log m$ products and $\log m$ sums. Both are necessary to achieve our desired logarithmic depth for the overall registered phase shifting procedure.

Substituting, we get:

$$-y = \prod_{r=0}^{\lceil \log_n \rceil - 1} (1 + (2s)^{2^r})$$

Then the final solution of p in the first equation above is:

$$p = -yl \mod 2^n = -l \prod_{r=1}^{\lceil \log n \rceil - 1} (1 + (2s)^{2^r})$$

The resource counts for this modular division is equivalent to:

$$\text{DIV-MOD}(n) = \lceil \log n \rceil \cdot \text{MULT-2}(n) + (\lceil \log n \rceil - 1) \cdot \text{ADD}(2, n) \quad (6)$$

7 Methods

The code simulating Solovay-Kitaev and Super-Kitaev were written using Python 2.5 and numpy 1.4.1. Resource requirements were measured by running this code on a Linux machine with an Intel Pentium 3.2 GHz dual core CPU and 1 GB of RAM.

We calculated resource usage by the two algorithms in approximating a two-qubit controlled-rotation gate ($\Lambda(R_k) \in SU(4)$) used in the quantum Fourier transform that is central to many important quantum algorithms. Due to space and time limitations, we were unable to obtain the actual compiled universal gate sequences needed to implement this beyond modest precision. Algorithm-dependent constraints are provided below with an indication of why they are beyond the scope of this work. Therefore, the estimates provided are upper bounds indicating the worst-case resource usage, which are nevertheless more concrete than previously known asymptotic bounds. In actual practice, depending on the gate to be compiled, the performance may be better.

Errors in individual gates in a circuit sum up according to the following formula, where U_i indicates our ideal gate, $\tilde{U}_i$ indicates our compiled, approximated gate, and $\epsilon_i = ||U_i - \tilde{U}_i||$.

$$||U_1 \cdots U_n - \tilde{U_1} \cdots \tilde{U_n}|| \leq \sum_{i=1}^{n} \epsilon_i$$

Therefore, for a circuit like the QFT on t-qubits with size $O(t^2)$ gates, we would like our error for individual gates to scale like $O(1/t^3)$. For Shor's factoring algorithm on an L-bit number, the reduction to order-finding applies the inverse QFT on $t = 2L + O(1)$ assuming some constant overall error precision. For the commonly used RSA key size of $L = 2048 = 2^1 1$ bits, $t \approx 4096 = 2^1 2$, and we would like an individual $\Lambda(R_k)$ gate in $\text{QFT}^\dagger$ to have error on the order of $1/(2^3 6)$. Accordingly, we compare resources on error precisions from $1/2$ to $1/2^4 0$, expressed as $1 \leq \log_2(1/\epsilon) \leq 40$. Of course, we are neglecting the enormous overhead of error correction, the threshold theorem for single qubit error rates [8], and the implementation error of our universal gate set $\mathcal{G}$.

In addition to the modifications to the original Super-Kitaev procedure described in the previous sections, we make several assumptions to simplify our analysis. We treat CNOT and single-qubit gates as taking equal time, when in actual implementations (for example, ion traps) the CNOT operation is much longer. The Toffoli gate, which can be implemented using 6 CNOTs and 10 single-qubit gates, is counted as 16 gates.

For Solovay-Kitaev, we used a maximum sequence length of $l_0 = 6$ in the preprocessing step for $SU(4)$, an initial $\epsilon_0 = 1/64$ and a $c_{approx} = 4\sqrt{(2)}$, which provided a converging value for the number of recursion levels required to achieve a given error ϵ. Generating sequences with longer length would require terabytes of storage and days of computing time.

For Super-Kitaev, we ignore the precompiling step since for the $\Lambda(R_k)$ gate it generates at most 8 $\Lambda(e^{i\phi})$ gates to compile. The primary overhead of Super-Kitaev is in generating the initial magic state $|\psi_{n,1}\rangle$ which is amortized over all $\Lambda(e^{i\phi})$ gates, therefore a per-gate comparison of resource usage would only improve Super-Kitaev's standing relative to Solovay-Kitaev. We assume that we are only running a single circuit on our quantum computer, therefore some ancillae qubits are not uncomputed: the n qubits used to store the initial $|\psi_{n,1}\rangle$ state and the t qubits which are measured to do classical postprocessing are left in

a non-$|0\rangle$ state. If we wish to run multiple-circuits using Super-Kitaev, we would at least double the size of our circuit to uncompute this part as well as adding the gates needed to perform the classical postprocessing on our quantum computer (since the state $|\psi_{n,0}\rangle$ is in superposition).

8 Results

Our main result is the comparison of compiled circuit sizes and depths between normal Solovay-Kitaev and Super-Kitaev as shown in the Figure 8. In the graph legend, "Super" referes to Super-Kitaev and "Normal" refers to Solovay-Kitaev. Solovay-Kitaev in the worst case has identical The discrete steps in Solovay-Kitaev's circuit size and depth are due to the levels of recursion, which are conservatively chosen to meet the desired precision. The initial step reflects the fact that even before successive approximation, our initial generated sequences (basic approximations in Section 5) already meet large desired precisions. As expected, Super-Kitaev's depth compares very favorably with Solovay-Kitaev. The graphs are roughly linear in a log-log plot, with a non-linearity for low precisions due to irregularities in the adder circuits for small n.

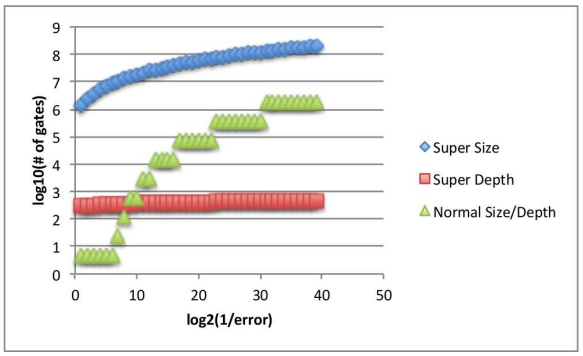

Figure 2: Compiled circuit size and depth

The tradeoff between Solovay-Kitaev and Super-Kitaev can most clearly be seen in the use of space. Solovay-Kitaev uses no ancillae at run-time, but requires a classical preprocessing step to generate basic approximations (precompiled sequences from the universal set). Although we did not generate sequences beyond $l_0 = 9$ for $SU(4)$, the curve indicates it would soon require terabytes of storage, even if a more efficient encoding were used (e.g. programming in C instead of Python). Super-Kitaev, on the other hand, requires no preprocessing space, but has a (currently intractable) requirement for ancilla qubits are runtime which tracks the circuit size as $O(n^2 \log n)$.

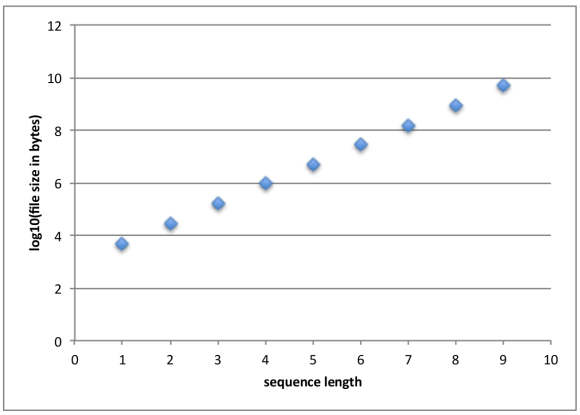

Figure 3: Solovay-Kitaev preprocessing file sizes

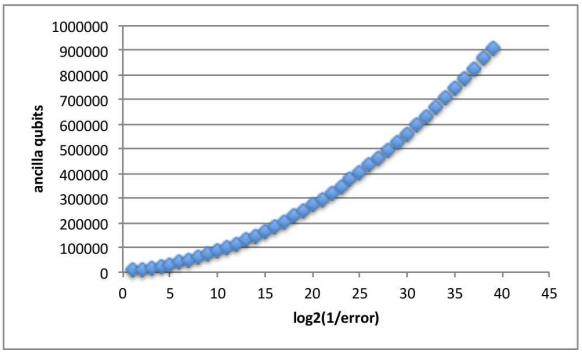

Figure 4: Super-Kitaev ancilla

9 Conclusion and Future Directions

In summary, we have seen the expected asymptotic bounds of both the Solovay-Kitaev and Super-Kitaev quantum compiling algorithms reflected in numerical resource comparisons. Solovay-Kitaev requires a large classical preprocessing overhead but produces a more tractable number of compiled gates and zero

ancillae, albeit at larger circuit depth. This seems to be a more reasonable choice for early experiments in running algorithms on an 80-qubit ion-trap quantum computer, where physical trapping constraints make large numbers of qubits problematic but we are potentially willing to wait a long time for the computation to complete. On the other hand, the situation may change in the future when quantum computers become more mature, scalable, and parallel; when we become more ambitious in our algorithm input sizes; and when the performance bottleneck becomes circuit depth. In that case, Super-Kitaev may be preferrable. Quantum computer engineers of the future will be able to use this work to choose the most suitable quantum compiling technique currently available as well as compare it to future compiling algorithms.

Moreover, the development of a compiler is intertwined with the development of the underlying architecture. Just as classical programming languages hint at what features would be desirable to move out of software (and compile-time) and into hardware (and run-time), Super-Kitaev provides some suggestions to future quantum architectures. Hardware which seeks to take advantage of this low-depth compiling would need to have a "phase factory" for enacting the $\Lambda(e^{i\phi})$ gate, which would included an efficient parallelized phase estimation routine. These are novel resources which are currently not being considered in related literature.

Furthermore, quantum compiling may provide some hints to how we can program their analog cousins. Although analog computers have fallen out of favor as a primary computing model due to noise problems, they are making a resurgence as a complementary "coprocessor" to digital computers, much like the imagined future role of quantum computers. Like quantum computers, analog computing uses existing physical properties to perform useful computation, such as solving differential equations using electric current and potential. Multiple AC signals and a single DC signal can coexist in superposition on a wire, and complex impedance from resistors, capacitors, and inductors can differentiate between components of different frequencies.

$$V = I(R + \frac{1}{j\omega_1 C} + j\omega_2 L + \ldots)$$

While it is unlikely that such a model would be more powerful in general from a computational complexity point of view, we may be able to use them as a practical aid in solving specific engineering problems, such as encoding the periods of cyclic integer groups in corresponding periods of sinusoidal signals. Such an analog computing model might be able to be programmed via recursive successive approximation like Solovay-Kitaev or using the techniques of Super-Kitaev.

10 Acknowledgements

The author would like to gratefully acknowledge the help of his advisors Dave Bacon and Mark Oskin, as well as Aram Harrow for the introduction to Super-Kitaev and related expertise.

References

[1] M.N. Vyalyi A. Yu. Kitaev, A.H. Shen: *Classical and Quantum Computation*. American Mathematical Society, Providence, Rhode Island, 2002.

[2] Michele Burrello, Giuseppe Mussardo, and Xin Wan: Topological Quantum Gate Construction by Iterative Pseudogroup Hashing. p. 20, September 2010.

[3] Christopher M. Dawson and Michael A. Nielsen: The Solovay-Kitaev algorithm. p. 15, May 2005. [arXiv:quant-ph/0505030].

[4] Thomas G. Draper, Samuel A. Kutin, Eric M. Rains, and Krysta M. Svore: A logarithmic-depth quantum carry-lookahead adder. p. 21, June 2004. [arXiv:quant-ph/0406142].

[5] Aram Harrow: Quantum compiling, May 2001.

[6] Aram W. Harrow, Benjamin Recht, and Isaac L. Chuang: Efficient discrete approximations of quantum gates. *Journal of Mathematical Physics*, 43(9):4445, November 2002. [doi:10.1063/1.1495899, arXiv:quant-ph/0111031].

[7] Seth Lloyd: Almost Any Quantum Logic Gate is Universal. *Physical Review Letters*, 75(2):346–349, July 1995. [doi:10.1103/PhysRevLett.75.346].

[8] Michael A. Nielsen and Isaac L. Chuang: *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, U.K., 2000.