Quantum Compiling with the Super-Kitaev Algorithm

Paul Pham

November 2nd, 2010

1 Introduction

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 usually local, 2-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.

How then, do we translate our quantum algorithms into something that can run on hardware? In analogy to classical computers, we describe this translation as *quantum compiling*. One of the central results of quantum computing, and the reason why it is interesting, is that we can actually perform this quantum compiling efficiently. This first result is called the Solovay-Kitaev theorem [2], which provides an efficient way to approximate any quantum gate. I presented this algorithm in an earlier talk, but I'll briefly review the results here. However, the constants for this method scale exponentially with the number of qubits and involve an intractably large preprocessing step.

The second result is more recent but much less well-known, and is called the Super-Kitaev algorithm (named by Aram Harrow). This new method improves upon the bounds for compiled circuit size over the normal Solovay-Kitaev algorithm, but comparison is more complicated. Super-Kitaev introduces some new tradeoffs to the comparison, such as ancillary qubits and classical postprocessing. In doing so, Kitaev also gives us some interesting modules which are worth studying for their own sake. In these notes, we will outline the Super-Kitaev algorithm at a high-level to develop some intuition about why it works, describe how we might implement it in practice.

2 Definitions

2.1 Approximation vs. Simulation

I'll use the terms ”approximation” and ”simulation” interchangeably, as in a quantum compiler will take an input circuit and produce as output a new circuit which approximates or simulates the original circuit to some degree of precision.

2.2 Standard Set

We use the following standard set $\mathcal{Q}$ from [1]

$$\mathcal{Q} = \{H, K, K^{-1}, \Lambda(\sigma^x), \Lambda^2(\sigma^x)\} \tag{1}$$

where these gates are defined as follows:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$K = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

Even though realistic implementations of a quantum computer may have a different set of hardware instructions, Kitaev's analysis assumes that the standard set can always be efficiently compiled down further into this hardware set.

Note that this may not be valid. For example, in order for ion traps to perform even a Hadamard gate to any precision requires a combination of σ^x and σ^z gates. However, we won't let this bother us for now.

2.3 Parameters

For these notes, we restrict ourselves to $d = 2^n$ for an n-qubit system (qudits) and consider the problem of compiling 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. 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 L/ϵ, which requires the following number of gates. We'll find this l parameter useful in describing the compiler overheads of both the Solovay-Kitaev and Super-Kitaev methods.

Why do we care about depth? This gives us a heuristic for how "parallelized" our circuit is. If we flatten our circuit into layers $\{l_i\}$, where each layer l_i is dependent only on inputs from the previous layer l_{i-1} and produces only outputs to the next layer l_{i+1}, then all the gates within the layer l_i can be performed 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.

Also, when running the compiler, there are several time resources to consider. The first is classical preprocessing time, denoted T_{pre}, which is a way of optimizing the generated quantum circuit by spending more classical resources upfront. The second is classical postprocessing time, denoted T_{post}, which likewise is meant to reduce the size of the quantum circuit but depends on quantum measurements and may be fed back into future quantum operations later.

A summary of these parameters are provided below:

d	dimension of quantum gates (matrix representation)
ϵ	the desired accuracy of the compiled circuit
L	size of input circuit to be compiled, in gates from $\mathcal{Q}$
d	depth of input circuit to be compiled
L'	size of compiled output circuit, in gates from $\mathcal{Q}$
d'	depth of compiled circuit to be compiled
n	$= O(\log L/\epsilon)$, shorthand parameter for asymptotic analysis
T	classical running time of the compiling algorithm
T_{pre}	classical preprocessing time, before the compiler is run
T_{post}	classical postprocessing time, after the compiler is run or during

2.4 Universal Instruction Sets and Compiled Sequences

More formally, suppose we have a universal instruction set $\mathcal{G}$ which contains some finite number of gates $g \in SU(d)$ and also their inverses $g^\dagger$. $\mathcal{G}$ is universal for $SU(d)$ in that it generates a dense subgroup. In other words, given an arbitrary quantum gate $U \in SU(d)$ and a desired accuracy ϵ we can find a product $S = g_1 \cdots g_m$ from $\mathcal{G}$ which approximates U to within ϵ using some distance function.

We can either use the matrix operator norm to define the distance function such as:

$$d(U,S) \equiv \parallel U - S \parallel \equiv \sup_{\parallel |\psi\rangle \parallel = 1} \parallel (U - S)\,|\psi\rangle \parallel < \epsilon \tag{2}$$

There is also a 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.

$$d(U,S) = \sqrt{\frac{d - \parallel \mathrm{tr}(U^\dagger S) \parallel}{d}} \tag{3}$$

3 Performance

Just to get this out of the way, in case you are wondering whether to be excited or not about Super-Kitaev.

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

where ν is a small positive constant. The remaining parameters are yet to be determined.

4 History

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 $T, L = (O(1/\epsilon))$ [5].

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 [6] which showed that $T, L = O(\log^c 1/\epsilon)$ for c between 3 and 4.

In 2001, Aram completed his undergrad thesis arguing that it would be difficult to beat $c < 2$ for the above bounds using a successive approximation method.[3]

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 size and depth, but using ancillary qubits and having as yet unknown multiplicative constants.

In 2003, Aram, Ben Recht, and Ike 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 [4].

In 2005, Chris Dawson and Mike Nielsen published their pedagogical review paper which I used as the basis for my previous work on implementing a Solovay-Kitaev compiler [2].

5 Review of Solovay-Kitaev

Here I'll remind you of the basic pseudo-code for normal Solovay-Kitaev. Our development is taken from the excellent review paper [2].

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

This algorithm works by way of recursive, successive approximation.

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 the instruction set $\mathcal{G}$. 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.

And that's all I'm going to say about that.

6 The Super-Kitaev Algorithm

Our development of the Super-Kitaev algorithm follows closely the one in the book by Kitaev, Shen, and Vyalvi [1], although of course, it is simply called Theorem 13.5. Likewise, we will cite the theorem / lemma numbers from the book, which contains a self-contained description of all the parts needed for Super-Kitaev.

The basic steps of this method are the following:

1. Precompiling to get L' gates in $\mathcal{Q} \cup \Lambda(e^{i\phi})$. Here we see that there is an efficient decomposition of several important gates into the standard set and controlled-phase-shifts.
2. Implementing a controlled-phase-shift. This ends up being the hardest part of simulating a quantum circuit, and requires each of the remaining steps.
3. Use phase estimation of an addition operator to magic states in order to enact the desired phase shift. This requires both efficient creation of *magic states* and an efficient *quantum adder* circuit.
4. Create one magic state, and then make L' copies of it. Use one copy to simulate each $\Lambda(e^{i\phi})$ gate.

6.1 Precompiling to the Standard Set

In order to perform the precompilation step, we need to know that we can change any unitary in $SU(d)$ by tensor products of unitaries in $SU(2)$ and $SU(4)$, that is, single- and two-qubit gates.

To prove the universality of the standard set $\mathcal{Q}$ is beyond the scope of these notes. For our purposes, we assume that we can precompile any gate down to single-qubit rotations $U \in SU(2)$ and controlled-single-qubit rotations $\Lambda(U), U \in SU(2)$. (For example, it's known how to do that for the Toffoli gate, which is what we need to implement our quantum adder circuit). From there, how do we get down to the standard set?

Theorem 1 (Problem 8.1)**.** *Any single-qubit gate U can be simulated using $\mathcal{Q} \cup \{\Lambda(e^{i\phi})\}$.*

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

$$U = e^{i\phi}e^{i(\gamma/2)\sigma^z}e^{i(\beta/2)\sigma^x}e^{i(\alpha/2)\sigma^z} \tag{4}$$

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:

$$e^{i\phi} = \Lambda(e^{i\phi})\sigma^x\Lambda(e^{i\phi})\sigma^x \tag{5}$$

$$\sigma^x = H\Lambda(e^{i\pi})H \tag{6}$$

$$e^{i\phi\sigma^z} = \Lambda(e^{-i\phi}\sigma^x\Lambda(e^{i\phi})\sigma^x \tag{7}$$

$$e^{i\phi\sigma^x} = He^{i\phi\sigma^z}H \tag{8}$$

□

Theorem 2 (Problem 8.2)**.** *Any controlled operator $\Lambda(U)$ where $U \in SU(2)$ can be simulated using $\mathcal{Q} \cup \{\Lambda(e^{i\phi})$.*

Proof. 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.

$$V = A\sigma^x A^{-1} B\sigma^x B^{-1} \tag{9}$$

□

6.2 Controlled-Phase Shift

Now we are down to the standard set $\mathcal{Q}$, plus that pesky controlled-phase-shift gate $\Lambda e^{i\phi}$. This is in general a hard gate to implement, and one of the reasons why we need a quantum compiler in the first place!

We could just use normal Solovay-Kitaev and find some sequence of X and Z gates which implement $\Lambda e^{i\phi}$, but that wouldn't be any fun and we would have no reason to bring in all this extra machinery to do Super-Kitaev.

Note that in general we can't exactly implement $\Lambda e^{i\phi}$ but we can approximate it as $\Lambda e^{2\pi i l/2^n}$, where l is an integer. In general this is a special case of the following operator defined by Kitaev in the following lemma.

Theorem 3 (Lemma 13.4)**.** *The operator $\Upsilon_n(e^{2\pi i/2^n}) : |l\rangle \to e^{2\pi i l/2^n}|l\rangle$ for $0 \le l < 2^n$ can be realized with precision $\delta = 2^{-n}$ by an $O(n^2 \log n)$-size $O((\log n)^2)$-depth circuit over the standard basis using ancillas.*

Proof. To get started, we will need to introduce a magic state $|\psi\rangle$ with the integer parameters n and k such that

$$|\psi_{n,k}\rangle = \frac{1}{\sqrt{2^n} - 1} \sum_{j=0}^{2^n - 1} e^{2\pi i jk/2^n} |j\rangle \tag{10}$$

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

What is interesting to note about this state is that we can enact a phase shift simply by doing addition. In fact, addition is so useful in Super-Kitaev that we will later discuss a parallelized adder circuit which, however, is generic enough to be useful in other quantum algorithms.

Note that this magic state is an eigenstate of the addition operator defined by $A : |j\rangle \to |(j+1) mod 2^n\rangle$

$$A|\psi_{n,k}\rangle = e^{2\pi i\phi_k/2^n}|\psi_{n,k}\rangle \tag{11}$$

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 $e^{2\pi i p/2^n}$

$$A^p|\psi_{n,k}\rangle = e^{2\pi i\phi k/2^n}|\psi_{n,k}\rangle \tag{12}$$

To realize our desired operator $\Upsilon_n e^{2\pi i/2^n}$ it suffices to use the value of l to compute p and then apply the operator:

$$\Upsilon_n(X) : |p, j\rangle \to |p, (j + p) \mod 2^n\rangle \tag{13}$$

for $p, j \in 0, \dots, 2^n - 1$. That is, two n-qubit registers contain the values p and j.

Ideally, we would just create $|\psi_{n,k}\rangle$ for some particular k ($k = 1$ would be ideal). Then we would know exactly which $|\psi_{n,k}\rangle$ we had, and how many times p to apply the A operator to enact the desired phase shift, where p is the solution to the following equation:

$$kp \equiv l(\mod 2^n) \tag{14}$$

Note that we stipulate k to be odd so that there is a unique solution p.

Recall now that the magic states $|\psi\rangle$ look suspiciously like a Quantum Fourier Transform state of the computational basis. However that doesn't help us here, because QFT is way too complicated to use as our basic 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 in the following way:

$$|\eta\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}\left|2^{n-1}\right\rangle = \frac{1}{\sqrt{2^{n-1}}}\sum_{s=1}^{2^{n-1}}|\psi_{n,2s-1}\rangle \tag{15}$$

Now then here are the required steps.

1. Create the state $|\eta\rangle$ 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.
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 a circuit of size $O(n^2)$ and depth $O(\log n)$.
3. Solve for p as a function of k and l satisfying Equation 14
4. Apply X^p to the n-qubit register containing $|\psi\rangle$, which enacts the desired phase shift.
5. Reverse the first three steps to uncompute p

Note that we skimmed over the solution to Equation 14 even though this is the most resource-intensive part of the algorithm. Rest assured, it reduces to division (or computing the inverse of $2s-1 \mod 2^n$) which Kitaev provides an efficient solution to as Problem 2.14a in his book.

□

6.3 The Main Algorithm

Now assuming we have parallelized phase estimation as a black box, here are the steps to the Super-Kitaev algorithm.

Theorem 4 (Theorem 13.5)**.** *Any circuit C of size L and depth d over a fixed finite basis can be simulated with precision δ by an $O(Ln + n^2\log n)$-size $O(d\log n + (\log n)^2)$-depth circuit C' over the standard basis, where $n = O(\log(L/\delta))$.*

Proof. For the steps below, we note that we can avoid solving the equation $kp \equiv l(mod 2^n)$ if we set $k = 1$. This is equivalent to realizing $\Upsilon_n(e^{2\pi i/2^n})$ by applying $\Upsilon(A)$ from Section 6.2 to the state $|\psi_{n,1}\rangle$. So we need to create one copy of this magic state to simulate each $\Lambda(e^{i\phi})$ gate, possibly up to L' of them.

1. Precompile the circuit into gates from $\mathcal{Q} \cup \{\Lambda(e^{i\phi})\}$ using the results from Section 6.1.
2. Create the state magic state $|\psi_{n,0}\rangle = H^{\otimes n}|0^n\rangle$
3. Turn it into $|\psi_{n,1}\rangle = \Upsilon(e^{-2\pi i/2^n})|\psi_{n,0}\rangle$ using the procedure in Section 6.2 This is done with a circuit of size $O(n^2\log n)$ and $O((\log n)^2)$ depth.
4. Make L' copies of the state $|\psi_{n,1}\rangle$ out of one copy by apply the addition operation below.
5. Simulate the circuit C' using one copy of $|\psi_{n,1}\rangle$ per gate, this takes size $O(n)$ and depth $O(\log n)$.
6. Reverse the first three steps.

To copy the state $|\psi_{n,k}\rangle$ it suffices to apply the following operator:

$$|\psi_{n,k}\rangle^{\otimes m} = W^{-1}\left(|\psi_{n,0}\rangle^{\otimes}(m-1)\otimes|\psi_{n,k}\rangle\right) \tag{16}$$

where W is defined by

$$W : |x_1,\ldots,x_{m-1},x_m\rangle \to |x_1,\ldots,x_{m-1},x_1+\ldots+x_m\rangle \tag{17}$$

□

References

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

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

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

[4] 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].

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

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