Quantum Circuits for the Schur Transform

Oliver Thomas

Quantum Engineering CDT
University of Bristol

April 25, 2018

1 Introduction

There are many uses of the uses of the Schur transform, compression of local properties to a global system. in boson sampling experiments preparing highly symmetrerised states, [1]. We look at existing algorithms for implementing the Schur transform and aim to produce a minimal gate quantum circuit which in principle could be used to implement the Schur transform on a small number of qubits in the near future [2].

There are two distinct ways of performing the Schur transform on n qubits, it can either be built up from coupling all n qubits together in a single iteration which we call the spatial multiplexed approach. The other approach is performing Clebsch-Gordan (CG) transforms on the n qubits one at a time which we call the temporal multiplexed approach.

This report is structured as follows, the Schur transform is introduced and we construct a quantum circuit using the Givens rotation method [3]. We comment on the scalability of this method and compare it to the streaming procedure suggested in [4].

2 Schur transform

The Schur transform maps the computation basis states to a Schur basis, in the qubit case this corresponds to performing a Clebsch-Gordan transform. There are different ways of defining a Schur basis but here we choose to use angular momentum, defined in terms of a labeling of J (Total angular momentum values), M (z-axis projection of the angular momentum) and multiplicity values which we refer to as P.

The Schur basis vectors for two qubits are,

$$
\begin{aligned}
|J=1, M=+1\rangle &= |00\rangle \\
|J=1, M=\ \ 0\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \quad (1a) \\
|J=1, M=-1\rangle &= |11\rangle \\
|J=0, M=\ \ 0\rangle &= \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \quad (1b)
\end{aligned}
$$

We have suppressed the multiplicity label here for simplicity as there are no multiplicities in the two qubit case. The matrix for the transform which takes the computational basis to the spin basis is,

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix}
\begin{bmatrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{bmatrix}
=
\begin{bmatrix} |00\rangle \\ \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \\ |11\rangle \\ \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \end{bmatrix}
=
\begin{bmatrix} |J=1, M=1\rangle \\ |J=1, M=0\rangle \\ |J=1, M=-1\rangle \\ |J=0, M=0\rangle \end{bmatrix}
\quad (2)
$$

This transform takes the computational basis state, $|01\rangle$ to a superposition of the singlet and triplet states, $\frac{1}{\sqrt{2}}(|J=1, M=0\rangle + |J=0, M=0\rangle)$.

The Schur basis vectors can be calculated for any n qubit couplings using the Glebsch-Gordan coefficients, here we have calculated up to 4 qubits. For more than two qubits multiplicities are introduced, multiplicities keep track of which subspace the system is in. The multiplicities, P are defined as $J' - J$ the new J value minus the previous J value, the number of combinations for a given amount of 1s in a P string is the number of multiplicities for that J value.

The transform for three qubits contains 8 terms, the allowed J values are J=3/2 which contains 4 states and J=1/2 which contains 2 states. The J=1/2 subspace contains 2 multiplicities corresponding to the two possible routes to that coupling, either J=1 to J=1/2 (couple down) or J=0 to J=1/2 (couple up). The basis vectors are given in the appendix for completeness Eq. 5c. There are multiple ways of writing the spin basis vectors, the traditional C-G coefficients and also what we refer to as the phase encoding Eq. 9. The phase encoded transform matrix will have a different decomposition as the shape of the matrix is different to the regular encoding.

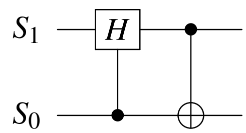
The transform for four qubits contains 16 terms, J=2 contains 5 terms, J=1 contains 3 terms with 3 multiplicities and the J=0 contains 1 term with 2 multiplicities. The equations are given in Eq. 10f. In the J=1 subspace there are 3 acceptable bit strings containing a single one, 0001, 0010 and 0100. This means there are 3 multiplicities present.

2.1 Schur Transform circuits

After building the unitary matrices from the Schur basis vectors using the CG coefficients we now calculate the gate cost of implementing the unitaries for 2, 3 and 4 qubits. See online [5] for Fortran code which implements the Givens rotation method to give the $C^{n-1}U$ decomposition for each of the Schur transform unitaries.

As two-qubit (entangling) gates are much more expensive to perform compared to single qubit gates, the cost of the circuits discussed here will all be given in terms of the number of two-qubit gates. The decomposition scheme for the n-qubit case is bounded by $2^{n-1}(2^n-1)$ $C^{n-1}U$ gates [3], where $C^{n-1}U$ means a unitary acting on 1 qubit controlled on the other n-1 qubits. It is possible to calculate a optimal gate sequence however current classical algorithms are computationally expensive [**?**].When n is greater than 2, it is possible to reduce higher order control gates to single control gates, $C^nU \sim 5C^{n-1}V$ where $U\&V$ are unitaries [6].

The two qubit Schur transform can be implemented in a circuit which only contains two gates as Fig. 1,



(3)

Figure 1: Schur transform for 2 qubits

For 3 qubits this upper bound is 28 C^2U gates.This means the maximum two-qubit gates needed would be 140 CU gates. The decomposition of the 3 qubit CG transform was performed using the Givens rotation method for unitary decomposition into a gate sequence. The matrix Eq. 6 can be expressed as a product of 19 C^2U gates (control-control-unitaries) which is $\sim$80 CU gates.

The decomposition scheme for the 4 qubit case the upper bound is 120 C^3U gates which could be up to $\sim$ 3000 CU gates. The 4 qubit Schur transform requires 72 C^3U gates which is equal to $\sim$ 1800CU gates. This suggests a different approach was needed.

The 2 qubit case 1 out of 2 gates are CNOTs, 3 qubits, 14 out of 19 are C^2NOTs and 4 qubits, 50 out of 72 are C^3-NOT gates.

3 Streaming Scheme

Entanglement purification is an example of a process which gains an advantage when using a streaming scheme [7]. Rather than wait for all of the entangled resource to be purified, it is possible to optimally perform purification in a streaming setup, send some of the purified resource while purifying the rest continually. The streaming scheme for the Schur transform can also be thought of in this way. However, it can still be useful to only purify part of the resource, it is not useful to only be able to perform a partial Schur transform which suggests there may be other equivalent or better methods compared to streaming for implementing the Schur transform.

We look at whether it is possible to achieve decrease in time by instead of coupling 1 qubit in at a time to the J & M registers, to pairing each of the couplings up similar to a binary tree. To investigate the problem we look at how the complexity of performing the Clebsch-Gordan transform scales moving from coupling a single qubit to an arbitrary $J\&M$ to coupling arbitrary $J\&M$ registers together. The pairing approach has a symmetry in the max range of $J\&M$ values will be the same within each pairing which will help simplify the problem compared to any arbitrary $J\&M$ values.

For the streaming scheme Fig. 2 the U_{CG} block can be chosen so that it contains all of the gates for up to the n-th qubit meaning the same block can be repeated. Where the U_{CG} is the Clebsch-Gordan transform between the J & M registers and the k-th qubit $|i_k\rangle$.

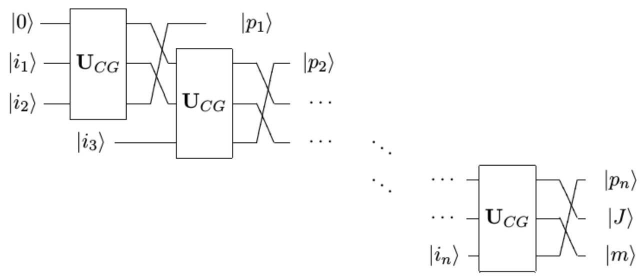


Figure 2: Streaming structure where the Schur transform is built up from consecutive Clebsch-Gordan transforms [4].

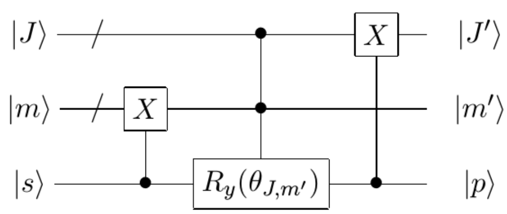


Figure 3: U_{CG} block, Qadder, controlled rotation, Qadder [4].

The controlled Rotation matrix [4], $R_y(\theta_{J,m'})$ Fig. 3 is the Clebsch-Gordan coefficients for coupling 1 qubit sequentially,

$$R_y(\theta_{J,m'}) = \begin{bmatrix} \cos(\theta_{J,m'}) & -\sin(\theta_{J,m'}) \\ \sin(\theta_{J,m'}) & \cos(\theta_{J,m'}) \end{bmatrix} = \frac{1}{\sqrt{2J+1}} \begin{bmatrix} \sqrt{J+\frac{1}{2}+m'} & -\sqrt{J+\frac{1}{2}-m'} \\ \sqrt{J+\frac{1}{2}-m'} & \sqrt{J+\frac{1}{2}+m'} \end{bmatrix} \tag{4}$$

Where primed variables means after the angular momentum addition so J is the total J that the spin is coupling to, the system will have J' total angular momentum after the coupling. m is the z component of the system before and m' is the total z component after the coupling.

To build this circuit the rotation matrix, $R_y(\theta_{J,m'})$ needs to be calculated using Eq. 4, values here Eq. 11e, and a function to update the $|m\rangle$ and $|J\rangle$ registers is needed. Updating the registers can be implemented relatively simply using the coherent (meaning the registers are allowed to be in superpositions) equivalent of the digital adders and subtracters. The complexity now has been reduced to implementing the Clebsch-Gordan transform, U_{CG}.

4 General circuit for the Quantum Schur transform Fig. 4

In the section above we have investigated the cost of implementing directly the Clebsch-Gordan unitary matrix using a unitary gate decomposition scheme. We note that the majority, over half of the gates are CNOT or higher dimensional equivalents.

We now compare directly decomposing the Schur transform unitary to the streaming scheme above, the streaming method has the advantage of being modular and relatively easy to calculate. Calculating the circuits (shown in Fig. 5) was done by calculating the truth tables for each step, the quantum adder which can be implemented reversibly, [8] and other possibly more optimal schemes for very large registers which makes use of the quantum Fourier transform (QFT) [9] which are outside the scope of this work. The streaming scheme uses temporal multiplexing to perform the Schur transform in polynomial time if a recursive streaming scheme is used [10].

We note that it is possible to reduce the gate count of the general streaming scheme by changing the encoding. Here we remove the intermediate values and hence can reduce both the number of qubits required and the gate count. This scheme works by coupling in two qubits at a time as opposed to one (Fig. 6, Fig. 7). This spatial multiplexing approach only stores the final output values of the J & M registers whereas the streaming scheme stores all intermediate values. This reduces the number of qubits needed.

The circuit adds the value of the spin to be added $|S\rangle$ using the encoding $|S\rangle : |0\rangle \mapsto Spin = +\frac{1}{2}, |1\rangle \mapsto Spin = -\frac{1}{2}$ to the M register to calculate the M' register value. This is done by implementing the quantum reversible equivalent to the digital adder. The CG transform gates are performed to generate P then the result is added to the J register to give the J' register value.

5 Discussion

The majority of the gates are CNOT gates. This is mainly due to the re-ordering of the basis and is similar to the quantum Fourier transform (QFT). The QFT produces the output in reverse qubit order the actual number of gates required to do the transform is massively reduced. The overhead calculated here is due to the rearranging of the basis. This means that depending on what the transform is used the transform could be computed with less gates. For example, if the transform was only used to check if the state was in a particular J block but didn't need to know the specific M value the order afterwards wouldn't be as important reducing the CNOTs needed.

There is a lot of freedom in the choice of basis used. We have chosen to use Two's complement encoding for the registers. The encoding for the multiplicities remains consistent throughout although it is more complex. For the general circuits the encoding remains consistent throughout. In order to achieve the optimal number of gates for the Schur transform on a specific number of qubits the encoding is changed on a case by case basis.

For large qubit coupling numbers the R matrix in the UCG tends to a Hadamard gate for the $|M| < |J|$ values of M. This means that depending on application of Schur transform and input states, it could be reasonable to approximate the transform when acting on highly symmetric states to just doing Hadamard gates.

6 Conclusion

References

[1] Alexandra E Moylett and Peter S Turner. Quantum simulation of partially distinguishable boson sampling. *arXiv:1803.03657*, 2018.

[2] William M Kirby and Frederick W Strauch. A practical quantum algorithm for the schur transform. *arXiv:1709.07119*, 2017.

[3] Chi-Kwong Li, Rebecca Roberts, and Xiaoyan Yin. Decomposition of unitary matrices and quantum gates. *International Journal of Quantum Information*, 11(01):1350015, 2013.

[4] Dave Bacon, Isaac L Chuang, and Aram W Harrow. Efficient quantum circuits for schur and clebsch-gordan transforms. *Physical review letters*, 97(17):170502, 2006.

[5] https://github.com/ot561/schurtransform/blob/master/matrixmul.f90.

[6] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical review A*, 52(5):3457, 1995.

[7] Robin Blume-Kohout, Sarah Croke, and Daniel Gottesman. Streaming universal distortion-free entanglement concentration. *IEEE Transactions on Information Theory*, 60(1):334–350, 2014.

[8] Thomas G Draper, Samuel A Kutin, Eric M Rains, and Krysta M Svore. A logarithmic-depth quantum carry-lookahead adder. *arXiv preprint quant-ph/0406142*, 2004.

[9] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000.

[10] Dave Bacon, Isaac L Chuang, and Aram W Harrow. The quantum schur and clebsch-gordan transforms: I. efficient qudit circuits. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1235–1244. Society for Industrial and Applied Mathematics, 2007.

A Appendix: Maths

A.1 3 Qubit transformation

This is the J=3/2 block

$$
\begin{aligned}
|J=3/2, M=+3/2, P=000\rangle &= |000\rangle \\
|J=3/2, M=+1/2, P=000\rangle &= \sqrt{\frac{1}{3}}(|001\rangle + |010\rangle + |100\rangle) \\
|J=3/2, M=-1/2, P=000\rangle &= \sqrt{\frac{1}{3}}(|110\rangle + |011\rangle + |101\rangle) \\
|J=3/2, M=-3/2, P=000\rangle &= |111\rangle
\end{aligned}
\tag{5a}
$$

This is the J=1/2 block from J=1, multiplicity zero

$$
\begin{aligned}
|J=1/2, M=+1/2, P=001\rangle &= +\sqrt{\frac{2}{3}}|001\rangle - \sqrt{\frac{1}{6}}(|010\rangle + |100\rangle) \\
|J=1/2, M=-1/2, P=001\rangle &= -\sqrt{\frac{2}{3}}|110\rangle + \sqrt{\frac{1}{6}}(|011\rangle + |101\rangle)
\end{aligned}
\tag{5b}
$$

This is the J=1/2 block from J=0, multiplicity one

$$
\begin{aligned}
|J=1/2, M=+1/2, P=010\rangle &= \frac{1}{\sqrt{2}}(|010\rangle - |100\rangle) \\
|J=1/2, M=-1/2, P=010\rangle &= \frac{1}{\sqrt{2}}(|011\rangle - |101\rangle)
\end{aligned}
\tag{5c}
$$

this is in matrix form,

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 & \sqrt{\frac{1}{3}} & 0 & 0 & 0 \\
0 & 0 & 0 & \sqrt{\frac{1}{3}} & 0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & \sqrt{\frac{2}{3}} & -\sqrt{\frac{1}{6}} & 0 & -\sqrt{\frac{1}{6}} & 0 & 0 & 0 \\
0 & 0 & 0 & \sqrt{\frac{1}{6}} & 0 & \sqrt{\frac{1}{6}} & -\sqrt{\frac{2}{3}} & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0
\end{bmatrix}
\begin{bmatrix}
000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111
\end{bmatrix}
=
\left[\begin{array}{c}
|J=3/2, M=3/2\rangle \\
|J=3/2, M=1/2\rangle \\
|J=3/2, M=-1/2\rangle \\
|J=3/2, M=-3/2\rangle \\ \hline
|J=1/2, M=1/2, P=0\rangle \\
|J=1/2, M=-1/2, P=0\rangle \\ \hline
|J=1/2, M=1/2, P=1\rangle \\
|J=1/2, M=-1/2, P=1\rangle
\end{array}\right]
\tag{6}
$$

The CG transform for 3 qubits Eq. 6 can be rearranged to a block diagonal form which looks like it could be implemented in a circuit.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{2}{3}} & -\sqrt{\frac{1}{6}} & -\sqrt{\frac{1}{6}} & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 \\ 0 & 0 & 0 & 0 & \sqrt{\frac{1}{6}} & \sqrt{\frac{1}{6}} & -\sqrt{\frac{2}{3}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 000 \\ 001 \\ 010 \\ 100 \\ 011 \\ 101 \\ 110 \\ 111 \end{bmatrix} \tag{7}$$

A.2 3 Qubit phase encoding

This is the J=3/2 block

$$\begin{aligned} |J = 3/2, M = +3/2, P = 000\rangle &= |000\rangle \\ |J = 3/2, M = +1/2, P = 000\rangle &= \sqrt{\frac{1}{3}}(|001\rangle + |010\rangle + |100\rangle) \\ |J = 3/2, M = -1/2, P = 000\rangle &= \sqrt{\frac{1}{3}}(|110\rangle + |011\rangle + |101\rangle) \\ |J = 3/2, M = -3/2, P = 000\rangle &= |111\rangle \end{aligned} \tag{8a}$$

This is the J=1/2 block from J=1, multiplicity zero

$$\begin{aligned} |J = 1/2, M = +1/2, P = 001\rangle &= \frac{1}{\sqrt{3}}(|001\rangle + e^{2\pi i/3}|100\rangle + e^{4\pi i/3}|010\rangle) \\ |J = 1/2, M = -1/2, P = 001\rangle &= \frac{1}{\sqrt{3}}(|011\rangle + e^{2\pi i/3}|101\rangle + e^{4\pi i/3}|110\rangle) \end{aligned} \tag{8b}$$

This is the J=1/2 block from J=0, multiplicity one

$$\begin{aligned} |J = 1/2, M = +1/2, P = 010\rangle &= \frac{1}{\sqrt{3}}(|001\rangle + e^{4\pi i/3}|100\rangle + e^{2\pi i/3}|010\rangle) \\ |J = 1/2, M = -1/2, P = 010\rangle &= \frac{1}{\sqrt{3}}(|011\rangle + e^{4\pi i/3}|101\rangle + e^{2\pi i/3}|110\rangle) \end{aligned} \tag{8c}$$

The phase encoding matrix is given by,

$$\frac{1}{\sqrt{3}}\begin{bmatrix} \sqrt{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{3} \\ 0 & e^{2\pi i/3} & e^{4\pi i/3} & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{2\pi i/3} & 0 & e^{4\pi i/3} & 1 & 0 \\ 0 & e^{4\pi i/3} & e^{2\pi i/3} & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{4\pi i/3} & 0 & e^{2\pi i/3} & 1 & 0 \end{bmatrix}\begin{bmatrix} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{bmatrix} = \begin{bmatrix} |J=3/2, M=3/2\rangle \\ |J=3/2, M=1/2\rangle \\ |J=3/2, M=-1/2\rangle \\ |J=3/2, M=-3/2\rangle \\ \hline |J=1/2, M=1/2, P=0\rangle \\ |J=1/2, M=-1/2, P=0\rangle \\ \hline |J=1/2, M=1/2, P=1\rangle \\ |J=1/2, M=-1/2, P=1\rangle \end{bmatrix} \tag{9}$$

Where this is a different form to the other basis for 3 qubits Eq. 6.

A.3 4 Qubit CG coefficients

The J=2 block, P=0000, (J=1/2, J=1, J=3/2, J=2)

$$\begin{aligned}
|J=2, M=+2, P=0000\rangle &= |0000\rangle \\
|J=2, M=+1, P=0000\rangle &= \frac{1}{2}(|0001\rangle + |0010\rangle + |0100\rangle + |1000\rangle) \\
|J=2, M=\ \ 0, P=0000\rangle &= \sqrt{\frac{1}{6}}(|0011\rangle + |0101\rangle + |1001\rangle + |1100\rangle + |1010\rangle + |0110\rangle) \\
|J=2, M=-1, P=0000\rangle &= \frac{1}{2}(|1110\rangle + |1101\rangle + |1011\rangle + |0111\rangle) \\
|J=2, M=-2, P=0000\rangle &= |1111\rangle
\end{aligned} \tag{10a}$$

The J=1 (0) block, P=0001, (J=1/2, J=1, J=3/2, J=1)

$$\begin{aligned}
|J=1, M=+1, P=0001\rangle &= +\sqrt{\frac{3}{4}}|0001\rangle - \sqrt{\frac{1}{12}}(|0010\rangle + |0100\rangle + |1000\rangle) \\
|J=1, M=\ \ 0, P=0001\rangle &= \sqrt{\frac{1}{6}}(|0011\rangle + |0101\rangle + |1001\rangle - |1100\rangle - |1010\rangle - |0110\rangle) \\
|J=1, M=-1, P=0001\rangle &= -\sqrt{\frac{3}{4}}|1110\rangle + \sqrt{\frac{1}{12}}(|1101\rangle + |1011\rangle + |0111\rangle)
\end{aligned} \tag{10b}$$

The J=1 (1) block, P=0010, (J=1/2, J=1, J=1/2, J=1)

$$
\begin{aligned}
|J=1, M=+1, P=0010\rangle &= +\sqrt{\frac{2}{3}}\,|0010\rangle - \sqrt{\frac{1}{6}}(|0100\rangle + |1000\rangle) \\
|J=1, M=\quad 0, P=0010\rangle &= \sqrt{\frac{1}{3}}(|0011\rangle - |1100\rangle) + \sqrt{\frac{1}{12}}(|0110\rangle + |1010\rangle - |0101\rangle - |1001\rangle) \\
|J=1, M=-1, P=0010\rangle &= -\sqrt{\frac{2}{3}}\,|1101\rangle + \sqrt{\frac{1}{6}}(|1011\rangle + |0111\rangle)
\end{aligned}
\tag{10c}
$$

The J=1 (2) block, P=0100, (J=1/2, J=0, J=1/2, J=1)

$$
\begin{aligned}
|J=1, M=+1, P=0100\rangle &= +\sqrt{\frac{1}{2}}(|0100\rangle - |1000\rangle) \\
|J=1, M=\quad 0, P=0100\rangle &= \frac{1}{2}(|0101\rangle - |1001\rangle + |0110\rangle - |1010\rangle) \\
|J=1, M=-1, P=0100\rangle &= -\sqrt{\frac{1}{2}}(|0111\rangle - |1011\rangle)
\end{aligned}
\tag{10d}
$$

The J=0 block, P=0011, (J=1/2, J=1, J=1/2, J=0)

$$
|J=0, M=0, P=0011\rangle = \sqrt{\frac{1}{3}}(|0011\rangle + |1100\rangle) - \sqrt{\frac{1}{12}}(|0101\rangle + |1001\rangle + |0110\rangle + |1010\rangle) \tag{10e}
$$

The J=0 block, P=0101, (J=1/2, J=0, J=1/2, J=0)

$$
|J=0, M=0, P=0101\rangle = \frac{1}{2}(|0101\rangle - |1001\rangle - |0110\rangle + |1010\rangle) \tag{10f}
$$

A.4 Rotation matrix for J & M values

J=0 values

$$
\begin{aligned}
\left|J=0, M'=+1/2\right\rangle &= R = I \\
\left|J=0, M'=-1/2\right\rangle &= R = XZ
\end{aligned}
\tag{11a}
$$

J=1/2 values

$$
\begin{aligned}
\left|J=1/2, M'=+1\right\rangle &= R = I \\
\left|J=1/2, M'=\quad 0\right\rangle &= R = XH \\
\left|J=1/2, M'=-1\right\rangle &= R = XZ
\end{aligned}
\tag{11b}
$$

J=1 values

$$\begin{aligned}
\left|J=1,M'=+3/2\right\rangle &= R = I \\
\left|J=1,M'=+1/2\right\rangle &= R = \frac{1}{\sqrt{3}}\begin{bmatrix}\sqrt{2} & -1 \\ 1 & \sqrt{2}\end{bmatrix} \\
\left|J=1,M'=-1/2\right\rangle &= R = \frac{1}{\sqrt{3}}\begin{bmatrix}1 & -\sqrt{2} \\ \sqrt{2} & 1\end{bmatrix} \\
\left|J=1,M'=-3/2\right\rangle &= R = XZ
\end{aligned} \tag{11c}$$

J=3/2 values

$$\begin{aligned}
\left|J=3/2,M'=+2\right\rangle &= R = I \\
\left|J=3/2,M'=+1\right\rangle &= R = \frac{1}{2}\begin{bmatrix}\sqrt{3} & -1 \\ 1 & \sqrt{3}\end{bmatrix} \\
\left|J=3/2,M'=\ \ 0\right\rangle &= R = XH \\
\left|J=3/2,M'=-1\right\rangle &= R = \frac{1}{2}\begin{bmatrix}1 & -\sqrt{3} \\ \sqrt{3} & 1\end{bmatrix} \\
\left|J=3/2,M'=-2\right\rangle &= R = XZ
\end{aligned} \tag{11d}$$

J=2 values

$$\begin{aligned}
\left|J=2,M'=+5/2\right\rangle &= R = I \\
\left|J=2,M'=+3/2\right\rangle &= R = \frac{1}{\sqrt{5}}\begin{bmatrix}2 & -1 \\ 1 & 2\end{bmatrix} \\
\left|J=2,M'=+1/2\right\rangle &= R = \frac{1}{\sqrt{5}}\begin{bmatrix}\sqrt{3} & -\sqrt{2} \\ \sqrt{2} & \sqrt{3}\end{bmatrix} \\
\left|J=2,M'=-1/2\right\rangle &= R = \frac{1}{\sqrt{5}}\begin{bmatrix}\sqrt{2} & -\sqrt{3} \\ \sqrt{3} & \sqrt{2}\end{bmatrix} \\
\left|J=2,M'=-3/2\right\rangle &= R = \frac{1}{\sqrt{5}}\begin{bmatrix}1 & -2 \\ 2 & 1\end{bmatrix} \\
\left|J=2,M'=-5/2\right\rangle &= R = XZ
\end{aligned} \tag{11e}$$

A Appendix- Circuits

For the general circuit structure, the M register is updated, Controlled rotation (R_θ) is applied to the qubit and then the J register is updated.

The case where $|S\rangle = |0\rangle$ means the spin is $+\frac{1}{2}$ so to add $\frac{1}{2}$ to M, 1 is added to the m_0 qubit. The first Quantum Adder (QAdd) uses Toffoli gates controlled on $|s\rangle = |0\rangle$ (denoted by the white control circle) with the current m_0 value and C_0 (an ancilla carry). This ensures that the case when $m_0 = 1$ and 1 is added to it, m_0 goes to 0 and m_1 is increased using the carry as $001 + 1 = 010$. The rest of the QAdd stages then just check the carry of the previous qubit to complete to M $+\frac{1}{2}$ addition as $|S\rangle = |0\rangle$ does not trigger any of the rest of the control gates.

The case where $|S\rangle = |1\rangle$ means the spin is $-\frac{1}{2}$ we do M $-\frac{1}{2}$ which is done by adding the binary string for $-\frac{1}{2}$ which is the all 1's string, 111. This time the very first QAdd does not trigger and $|s\rangle$ is then added to all of the bits of M using C-NOT gates with carries to check for overflow.

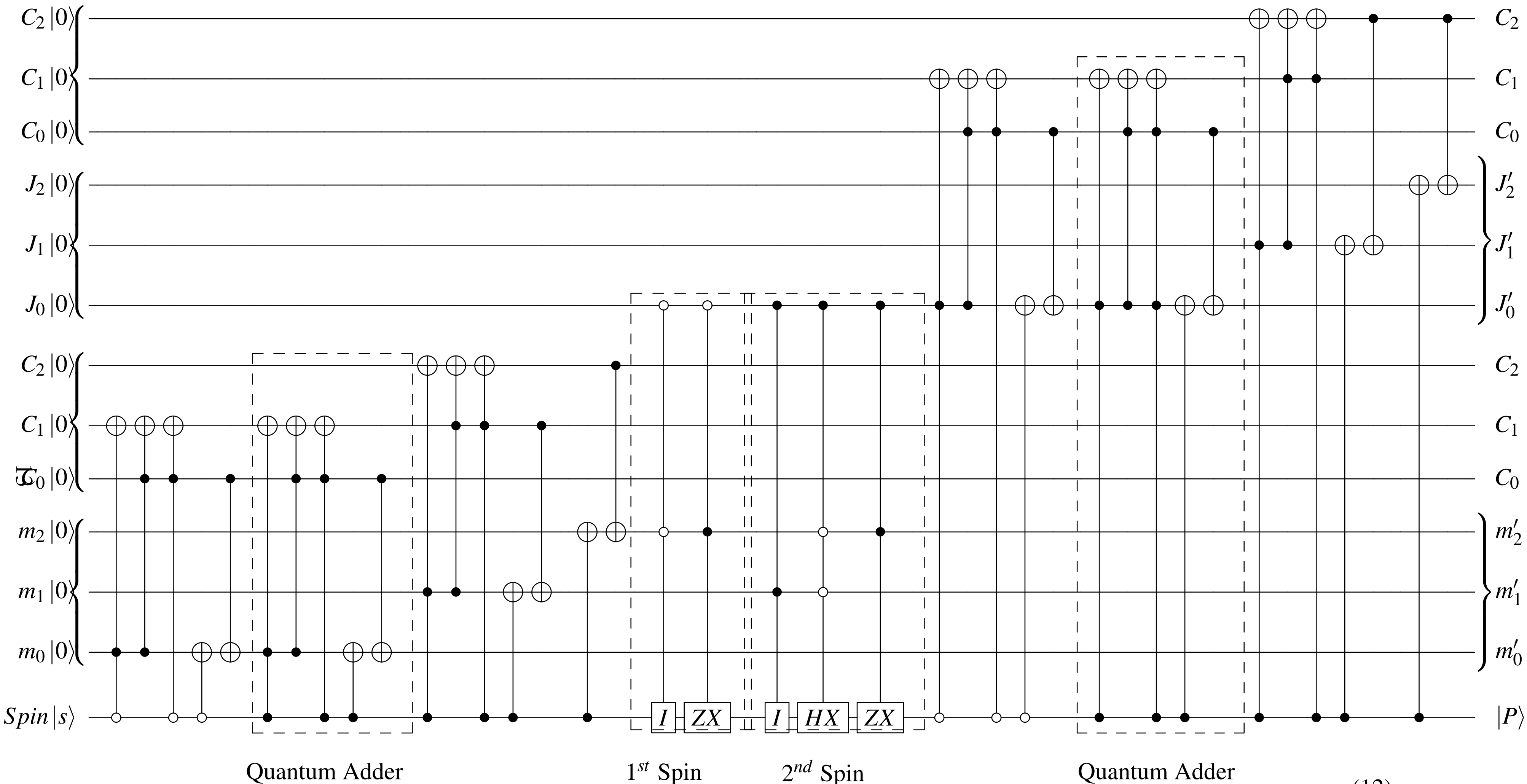
The Unitary is then performed on $|S\rangle$ depending on the values of the newly calculated M' and J registers using $R_y(\theta_{J,m'})$ Eq. 4. The J register is then updated to J' by adding the value of $|P\rangle$ to J using the QAdd sequence of gates.

To add the second qubit in the values of J' and M' are passed in as the initial register values. It is easy to extend this to many qubits being streamed in one at a time by carefully conditioning the controls on the unitaries, in the general case you need at most N controls for coupling up to N qubits in one at a time. The circuit written here has redundancy in the Identity and ZX gates appearing twice which is shown in the circuit for completenessFig. 4. registers.

J_2	J_1	J_0	J
0	0	0	0
0	0	1	$\frac{1}{2}$
0	1	0	1
0	1	1	$\frac{3}{2}$
1	0	0	-2
1	0	1	-$\frac{3}{2}$
1	1	0	-1
1	1	1	-$\frac{1}{2}$

m_2	m_1	m_0	M
0	0	0	0
0	0	1	$\frac{1}{2}$
0	1	0	1
0	1	1	$\frac{3}{2}$
1	0	0	-2
1	0	1	-$\frac{3}{2}$
1	1	0	-1
1	1	1	-$\frac{1}{2}$

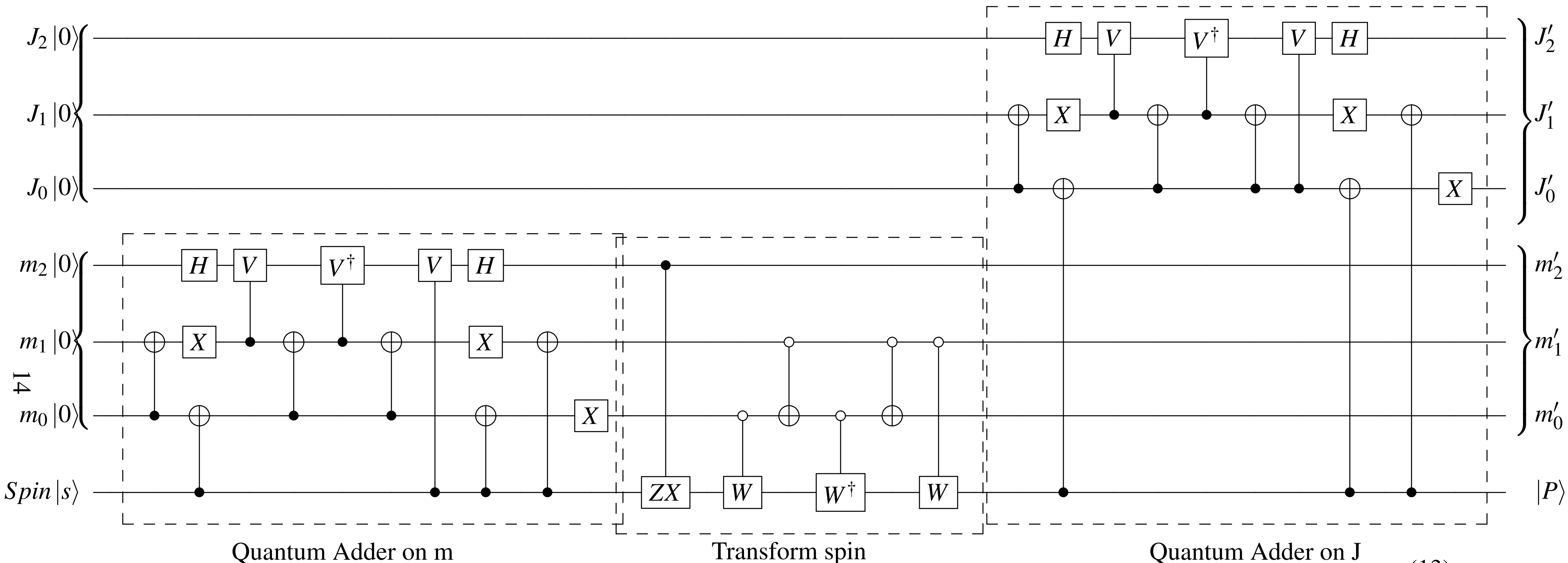
Table 1: Tables giving binary Two's complement encoding to spin values of the M and J registers



(12)

Figure 4: general streaming circuit

This can be simplified, removing all of the carries as we take advantage of the fact that as only one qubit is coupled at a time the registers will either always be incremented by ± 1. This enables us to rewrite the circuit in this way and effectively use the m_1 qubit as a temporary carry. Here we have also expanded all of the multiple control gates into single control gates.



(13)

Figure 5: temporal multiplexed streaming

Where V is the phase gate, $V = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$, $V^{\dagger}V = I$ and $V^2 = Z$. V is used here to expand the double controlled Toffoli gate into single control gates in the quantum adder subroutine.

The W gate, $W^2 = HX$ with $W^{\dagger} = I$, is used to expand the HX gate into single control gates in the spin transform region.

The circuit checks that if (m_1 XNOR m_0) AND (m_0 XOR S_0) and will then change m_2. Then m_1 is updated using $m_1 = m_0$ XOR S_0. m_0 is always incremented by 1, if $|S\rangle = |0\rangle$ increment only m_0 by 1 corresponding to adding $\frac{1}{2}$ to the M register. $|S\rangle = |1\rangle$ corresponds to subtracting $\frac{1}{2}$ from the M register by adding the string 111 bitwise to M.

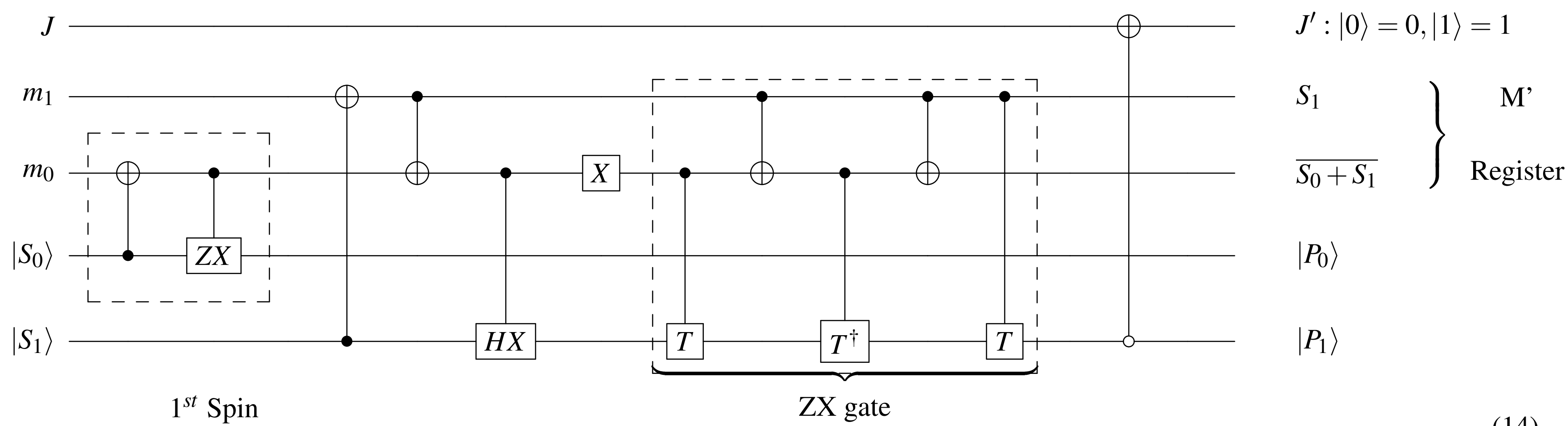
For the most positive values of M the Identity is performed on the spin corresponding to the strings $M = 001(J = \frac{1}{2}, M' = \frac{1}{2})$ for the first spin and $M = 010(J = 1, M' = 1)$ for the second coupled in spins.

The most negative values of M performs $XZ|S\rangle$ corresponding to the strings $M = 111(J = \frac{1}{2}, M' = -\frac{1}{2})$ for the first spin and $M = 110(J = 1, M' = -1)$ for the second spin.

If $M = 000(J = 0, M' = 0)$ do $XH|S\rangle$ Fig. 5.

B Spatial multiplexing

The registered circuit, which takes in two qubits can be constructed from 11 gates if the m register is not compressed. The spatially multiplexed minimal gate explicit J & M 2 qubit transform is shown in Fig. 6 which contains 11 two-qubit gates. Another CNOT can be added to compress the M' register, which means using the same values for M'=0, see Fig. 7 for the 12 two-qubit gate circuit.



(14)

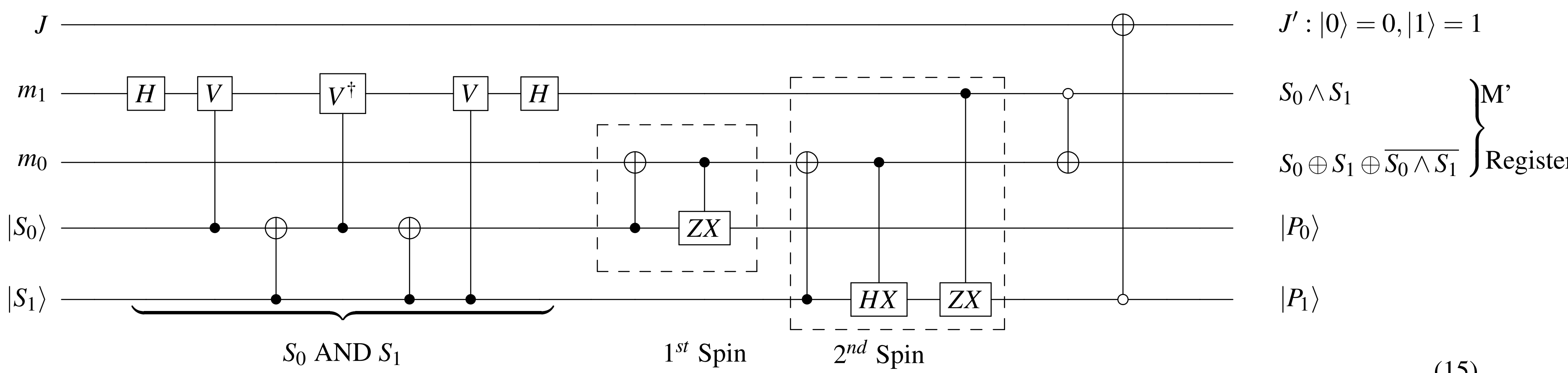
Figure 6: minimal gate spatial multiplexing

HX gate triggers if $M = 0$ meaning $S_0 \neq S_1$ which is implemented using an XOR between S_0 & S_1. The other gate (T) is triggered when $M = -1$ meaning $S_0 = S_1 = 1$ which is done using an AND (Toffoli) gate between, $S_0 = S_1$ AND $S_1 = 1$ which is decomposed into 5 two-qubit gates. T^2 is the ZX gate, meaning $T^2\,|S\rangle = XZ\,|S\rangle$.

The registered circuit, which takes in two qubits can be constructed from 11 gates if the m register is not compressed. The spatially multiplexed minimal gate explicit J & M 2 qubit transform is shown in Fig. 6 which contains 11 two-qubit gates.

Spin values		Circuit output		M value
S_1	S_0	S_1	$\overline{S_0 + S_1}$	M
0	0	0	1	M=+1
0	1	0	0	M=0
1	0	1	0	M=0
1	1	1	1	M=-1

Table 2: Table giving M register decoding for minimal gate number



(15)

Figure 7: Spatial multiplexed 2 qubit

Another CNOT can be added to compress the M' register, which means using the same values for M'=0, see Fig. 7 for the 12 two-qubit gate circuit.

Spin values		Circuit output		M value
S_1	S_0	m_1	m_0	M
0	0	0	1	M=+1
0	1	0	0	M=0
1	0	0	0	M=0
1	1	1	0	M=-1

Table 3: Table giving M register decoding for 2 qubit spatial multiplexing

A Appendix- Fortran code

```
!Oliver Thomas 2018 Bristol
program matrixmul
implicit none

integer, parameter :: dp=selected_real_kind(15,300)
integer :: n, qubits, numofdecomp, i, j, counter
integer, allocatable, dimension(:) :: p, gatenum

real(kind=dp), parameter :: invr2=1/sqrt(real(2,kind=dp)), invr3=1/sqrt(real(3,kind=dp)),
real(kind=dp), parameter :: r2=sqrt(real(2,kind=dp))

real(kind=dp), allocatable, dimension(:,:) :: unitary, ident, uprod
real(kind=dp), allocatable, dimension(:,:,:)  :: u, gateseq

counter=1
print*, 'Enter number of qubits, 2, 3 or 4'
read*, qubits
n= 2**qubits
numofdecomp=int(n*(n-1)/2.0_dp)

allocate(ident(n,n))
allocate(unitary(n,n))
allocate(u(n,n,n*n))
allocate(uprod(n,n))
allocate(p(n))
allocate(gateseq(n,n,n*n))
allocate(gatenum(numofdecomp))


ident=0.0_dp
unitary=0.0_dp
u=0.0_dp
gateseq=0.0_dp
gatenum=1

!#make identity
ident=identity(n)

!#make u's ident
do i=1, size(u,3)
  u(:,:,i)=identity(n)
end do
gateseq=u
```

```
!#init uprod as ident

!#make unitary
if (qubits==2) then
  p(1:n)=(/1,2,4,3/)

  unitary(1:n,1)=(/1.0_dp, 0.0_dp, 0.0_dp, 0.0_dp/)
  unitary(1:n,2)=(/0.0_dp, invr2, invr2, 0.0_dp/)
  unitary(1:n,3)=(/0.0_dp, 0.0_dp, 0.0_dp, 1.0_dp/)
  unitary(1:n,4)=(/0.0_dp, invr2, -invr2,0.0_dp/)

else if (qubits==3) then

  p(1:n)=(/1,2,4,3,7,8,6,5/)
!# col,row
  unitary(1,1)=1.0_dp

  unitary(2,2)=invr3
  unitary(2,5)=r2*invr3

  unitary(3,2)=invr3
  unitary(3,5)=-invr6
  unitary(3,7)=invr2

  unitary(4,3)=invr3
  unitary(4,6)=invr6
  unitary(4,8)=invr2

  unitary(5,2)=invr3
  unitary(5,5)=-invr6
  unitary(5,7)=-invr2

  unitary(6,3)=invr3
  unitary(6,6)=invr6
  unitary(6,8)=-invr2

  unitary(7,3)=invr3
  unitary(7,6)=-r2*invr3

  unitary(8,4)=1.0_dp

else if (qubits==4) then

  p(1:n)=(/1,2,4,3,7,8,6,5,13,9,10,12,11,15,16,14/)
  !# col,row
```

```
!0000
unitary(1,1)=1.0_dp
!0001
unitary(2,2)=0.5_dp
unitary(2,6)=sqrt(0.75_dp)
!0010
unitary(3,2)=0.5_dp
unitary(3,6)=-sqrt(1.0_dp/12.0_dp)
unitary(3,9)=sqrt(2.0_dp/3.0_dp)
!0011
unitary(4,3)=sqrt(1.0_dp/6.0_dp)
unitary(4,7)=sqrt(1.0_dp/6.0_dp)
unitary(4,10)=sqrt(1.0_dp/3.0_dp)
unitary(4,15)=sqrt(1.0_dp/3.0_dp)
!0100
unitary(5,2)=0.5_dp
unitary(5,6)=-sqrt(1.0_dp/12.0_dp)
unitary(5,9)=-sqrt(1.0_dp/6.0_dp)
unitary(5,12)=sqrt(1.0_dp/2.0_dp)
!0101
unitary(6,3)=sqrt(1.0_dp/6.0_dp)
unitary(6,7)=sqrt(1.0_dp/6.0_dp)
unitary(6,10)=-sqrt(1.0_dp/12.0_dp)
unitary(6,13)=0.5_dp
unitary(6,15)=-sqrt(1.0_dp/12.0_dp)
unitary(6,16)=0.5_dp
!0110
unitary(7,3)=sqrt(1.0_dp/6.0_dp)
unitary(7,7)=-sqrt(1.0_dp/6.0_dp)
unitary(7,10)=sqrt(1.0_dp/12.0_dp)
unitary(7,13)=0.5_dp
unitary(7,15)=-sqrt(1.0_dp/12.0_dp)
unitary(7,16)=-0.5_dp
!0111
unitary(8,4)=0.5_dp
unitary(8,8)=sqrt(1.0_dp/12.0_dp)
unitary(8,11)=sqrt(1.0_dp/6.0_dp)
unitary(8,14)=-sqrt(0.5_dp)
!1000
unitary(9,2)=0.5_dp
unitary(9,6)=-sqrt(1.0_dp/12.0_dp)
unitary(9,9)=-sqrt(1.0_dp/6.0_dp)
unitary(9,12)=-sqrt(0.5_dp)
!1001
unitary(10,3)=sqrt(1.0_dp/6.0_dp)
```

```
  unitary(10,7)=sqrt(1.0_dp/6.0_dp)
  unitary(10,10)=-sqrt(1.0_dp/12.0_dp)
  unitary(10,13)=-0.5_dp
  unitary(10,15)=-sqrt(1.0_dp/12.0_dp)
  unitary(10,16)=-0.5_dp
  !1010
  unitary(11,3)=sqrt(1.0_dp/6.0_dp)
  unitary(11,7)=-sqrt(1.0_dp/6.0_dp)
  unitary(11,10)=sqrt(1.0_dp/12.0_dp)
  unitary(11,13)=-0.5_dp
  unitary(11,15)=-sqrt(1.0_dp/12.0_dp)
  unitary(11,16)=0.5_dp
  !1011
  unitary(12,4)=0.5_dp
  unitary(12,8)=sqrt(1.0_dp/12.0_dp)
  unitary(12,11)=sqrt(1.0_dp/6.0_dp)
  unitary(12,14)=sqrt(0.5)
  !1100
  unitary(13,3)=sqrt(1.0_dp/6.0_dp)
  unitary(13,7)=-sqrt(1.0_dp/6.0_dp)
  unitary(13,10)=-sqrt(1.0_dp/3.0_dp)
  unitary(13,15)=sqrt(1.0_dp/3.0_dp)
  !1101
  unitary(14,4)=0.5_dp
  unitary(14,8)=sqrt(1.0_dp/12.0_dp)
  unitary(14,11)=-sqrt(2.0_dp/3.0_dp)
  !1110
  unitary(15,4)=0.5_dp
  unitary(15,8)=-sqrt(3.0_dp/4.0_dp)
  !1111
  unitary(16,5)=1.0_dp

end if

21 format ( 16F7.3)
write(*,21) unitary
!#!!! make unitary gates

print*,
write(*,21) matmul(unitary,transpose(unitary))

uprod=unitary
do i=1,n !#col
  do j=1,n-1 !#row
    if(p(n-j+1).ne.p(i)) then
```

```
      call makeunitary(p(n-j),p(n-j+1),p(i), uprod, u(:,:,(i-1)*n+j))
    end if
  end do
end do

print*,
print*, 'unitaries'

uprod=unitary
do i=1, n*n
  if (icheck(u(:,:,i))==0) then
    uprod=matmul(uprod(:,:),u(:,:,i))
  end if
end do

  print*, 'THIS IS UNITARY'
  call invert(u,gateseq,counter)
  call gateset(gateseq)

do i=1, counter
  uprod=matmul(uprod,gateseq(:,:,i))
end do

print*, '-------------------------------------------------------------------'
print*, 'Unitary matrix from', counter-1, 'gates'
print*, '-------------------------------------------------------------------'

print*,

deallocate(ident)
deallocate(unitary)
deallocate(u)
deallocate(uprod)
deallocate(p)
deallocate(gateseq)
deallocate(gatenum)

!#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
contains

!# print non identity elements
subroutine gateset(matrix)
  real(kind=dp), dimension(:,:,:), intent(in) :: matrix
  integer :: n, i
  n=size(matrix,3)
```

```
  do i=1, n
    if (icheck(matrix(:,:,i))==0) then
    end if
  end do
end subroutine gateset

!# transpose and invert array
subroutine invert(matrix,inverted,count)
  real(kind=dp), dimension(:,:,:), intent(inout) :: inverted
  real(kind=dp), dimension(:,:,:), intent(in) :: matrix
  integer :: i, n, count
  n=size(matrix,3)
  count=1
  do i=1,n
    if (icheck(matrix(:,:,n-i+1))==0) then
      inverted(:,:,count) = transpose(matrix(:,:,n-i+1))
      count=count+1
    end if
  end do

end subroutine invert

!#!!!! Check product gives identity
function icheck(uni)
  real(kind=dp) :: icheck
  real(kind=dp), dimension(:,:), intent(in) :: uni
  integer :: i, j

icheck=1
!#check ident
itest:do i=1, size(uni,1)
  do j=1, size(uni,1)
    if (i.ne.j) then
      if (abs(uni(i,j))>=1e-10) then
        icheck=0
        exit itest
      end if
    else if (i.eq.j) then
      if (abs(abs(uni(i,j))-1)>=1e-10) then
        icheck=0
        exit itest
      end if
    end if
  end do
end do itest
```

```
end function icheck

!#!!!! Check product gives identity
function unitarycheck(umatrices, uni)
  real(kind=dp) :: unitarycheck
  real(kind=dp), dimension(:,:,:), intent(in) :: umatrices
  real(kind=dp), dimension(:,:), intent(in) :: uni
  real(kind=dp), dimension(:,:), allocatable :: uprod
  integer :: i, j

unitarycheck=1
uprod=uni
!#do u_n*u_n-1*...*u1*Unitary=Ident
do i=1, size(umatrices,3)-1
  if (icheck(umatrices(:,:,n*n-i))==0) then
    uprod=matmul(uprod(:,:), umatrices(:,:,i))
  end if
end do

unitarycheck=icheck(uprod)

end function unitarycheck

!#!!! find type of u
subroutine makeunitary(row1,row2,col,ucurrent, ugate)
  real(kind=dp) :: c,s,r
  real(kind=dp), dimension(:,:), intent(inout) :: ucurrent, ugate
  integer, intent(in) :: row1, row2, col

  ugate=identity(size(ucurrent,1))
  c=0.0
  s=0.0
  r=0.0

call givensrot(ucurrent(col,row1), ucurrent(col,row2), c,s,r)
  ugate(row1,row1)=c
  ugate(row1,row2)=-s
  ugate(row2,row1)=s
  ugate(row2,row2)=c
ucurrent=matmul(ucurrent,ugate)
end subroutine makeunitary

!#!!! Calc givens rotation
subroutine givensrot(a, b, c, s, r)
  real(kind=dp) :: a, b, c, s, r, h, d
```

```
h=0.0
d=0.0
 !#write(*,*) 'a',a,'b',b,'c',c,'s',s
if (abs(b)>=1e-1) then
  h=hypot(a,b)
  d=1.0_dp/h
  c=abs(a)*d
  s=sign(d,a)*b
  r=sign(1.0_dp,a)*h
else
  c=1.0_dp
  s=0.0_dp
  r=a

end if
end subroutine givensrot


!#!!!!! make identiy matrix dim n
function identity(n)
  real(kind=dp), dimension(n,n) :: identity
  integer :: n, i

identity=0.0_dp
do i=1, n
  identity(i,i) =1.0_dp
end do
end function identity

end program matrixmul
```