Kalpa Publications in Computing
Volume 20, 2024, Pages 25–34
Proceedings of 2024 Concurrent Processes Architectures and Embedded Systems Hybrid Virtual Conference

Could Communicating Sequential Processes be Used to Make Quantum Computing More Tractable?

Jeremy M. R. Martin
Lloyd's of London

Abstract

Quantum computers are notoriously difficult to program. This is largely because they operate predominantly on complex algebraic structures which are not usually part of the vocabulary of conventional programmers who are more used to working with integers, floating point numbers and Boolean values. Also, the atomic operations carried out by quantum gates require a level of understanding of counter-intuitive properties of quantum mechanics, such as superposition, measurement, and entanglement, which is a considerable step beyond the familiar Boolean logic which is implemented by gates in classical circuits. Therefore, there is a significant challenge for software developers who have been trained in how to create code for conventional computers in making the transition to quantum. Quantum programming is currently the preserve of mathematicians and theoretical physicists. In this paper I shall consider whether the process algebra of CSP could be useful tool in this context to make the field of Quantum Computing more accessible.

Keywords - Quantum Computing, CSP, Occam.

1 Introduction

Quantum Computing [1] is a field that leverages the principles of quantum mechanics to perform computations in a fundamentally different way from classical computers. It was originally proposed in the early 1980s by Feynman [2] and Manin [3], independently. At the time of writing, small-scale quantum computers have become a reality and are available for hire from cloud service providers. However, they are not yet sufficiently powerful to carry out useful computations, and therefore must be currently considered only to be a research tool.

The allure of Quantum Computing is that it may make certain significant computations feasible that are currently out of the reach of classical computers. Theoretical work has shown that certain calculations have a lower quantum complexity than computational complexity. For instance a quantum computer could potentially carry out integer factorisation exponentially faster than a traditional computer (using Shor's algorithm, developed by the American mathematician Peter Shor in 1994 [4]). This would make it feasible to break many of the cryptographic systems in use today, with potentially devastating consequences for global commerce and national security. And this realisation has been a key driver for a massive multi-billion-pound investment in this subject in recent years.

But there is also a school of thought that it might never be possible to create sufficiently powerful quantum computers to realise this potential because of problems with managing increasing error-rates as the number of components increases. This phenomenon is known as 'Quantum Noise' [5].

Quantum computers are analogous to classical computers in that they consist of circuits with gates that process bits of information. Whereas a classical bit only has two discrete states, 0 or 1, a qubit can be in infinitely-many different quantum states, represented by two complex numbers, α and β, subject to a normalisation constraint: |α2| + |β2| = 1. Technically a qubit state is a unit length complex vector in a Hilbert vector space, which can take an infinite number of possible values [1].

In a sense the qubit exists in a simultaneous combination of both the classical discrete states, with certain probabilities, this property is known as of superposition. However, when the qubit's state is measured, it becomes fixed at either zero or one.

Quantum states can be conveniently represented as a point on the surface of a unit sphere, known as the Bloch sphere.

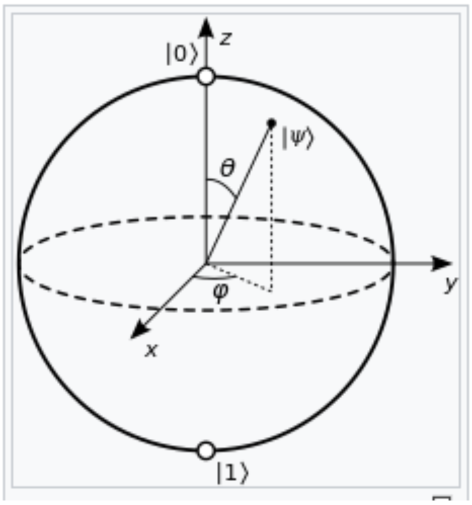


Figure 1. The Bloch Sphere

Quantum computers also exhibit the strange phenomenon of entanglement between qubits. If one qubit is entangled with another, then its state will also become fixed by the measurement of the other one.

Quantum gates are used to process the qubits. Although these are analogous to classical logic gates they tend to perform far more complicated operations. Whereas the gates of a digital circuit perform simple Boolean operations, like AND, OR, and NOT, on classical bits, quantum circuits require a more advanced level of mathematics to describe their states and transformations.

Quantum gates perform operations on one or more qubits by applying a tensor product between the gate's matrix representation and the state vector(s) of the qubits.

The mathematical properties of superposition and entanglement can be used to design highly efficient quantum algorithms for solving important problems, potentially processing vast amounts of information in parallel.

However, Quantum Computing, like Quantum Mechanics, is hard to grasp intuitively because of the strangeness of the concepts of superposition and entanglement. The reason why quantum computers operate in this strange manner is straightforward: it is the way that nature works below the scale of atoms. This makes the entry bar seem rather high for becoming a quantum developer.

But there once was a time when programming a classical computer would have required a significant understanding of electronic circuits and components. And we have moved a very long way from that in the past eighty years with a stack of abstractions for digital circuits, microprocessors, distributed memory protocols, machine code, assembly language, and high-level programming languages.

Simulators exist for quantum circuits, using classical computing, with mathematical concepts, such as complex vectors, matrices and tensor products used to represent the entities and operations. One way for traditional programmers to understand Quantum Circuits is as a kind of restricted programming language, where the only data type is qubit, and the operators are defined as matrices. But, because of the nature of Quantum Physics, some of these operations can lead to more efficient execution than would be possible in a simulator.

Building abstraction models for quantum computing has become a rich area of research. Languages, such as Q#, Qiskit, or Cirq, provide abstractions for programming quantum algorithms. Programmers can express quantum operations and algorithms using familiar syntax. Quantum Libraries are available for these languages providing pre-built quantum algorithms and circuits that programmers can use as building blocks. This is essentially at the same level as assembly language for classical computing, and further layers of abstraction and simplicity are needed to drive progress.

So, the race is on to come up with an abstract model for quantum computing, that makes the field more accessible to conventional programmers without requiring them to have an advanced understanding of Quantum Physics.

It would be interesting to explore whether use of a process algebra, such as CSP [11, 12], could be instrumental in achieving this. After all, entanglement is a similar concept to synchronous communication, and circuits represent a process workflow.

2 Quantum Circuits

Quantum circuits as used today are essential acyclic. Information travels from a source along wires from left to right passing through gates. An input signal consisting of a sequence of bits, or qubits, is passed through a sequence of gates, where each gate carries out an operation on its inputs and outputs the results. There is no splitting of wires to create multiplexed signals due to the "no cloning theorem" [6] which states that quantum states cannot be duplicated into multiple independent copies.

To perform iterative algorithms or computations with quantum computers, the normal approach is to embed a quantum circuit within classical circuitry acting as a control system. This is akin to the concept of graphical or mathematical coprocessors that are used in modern processing chips.

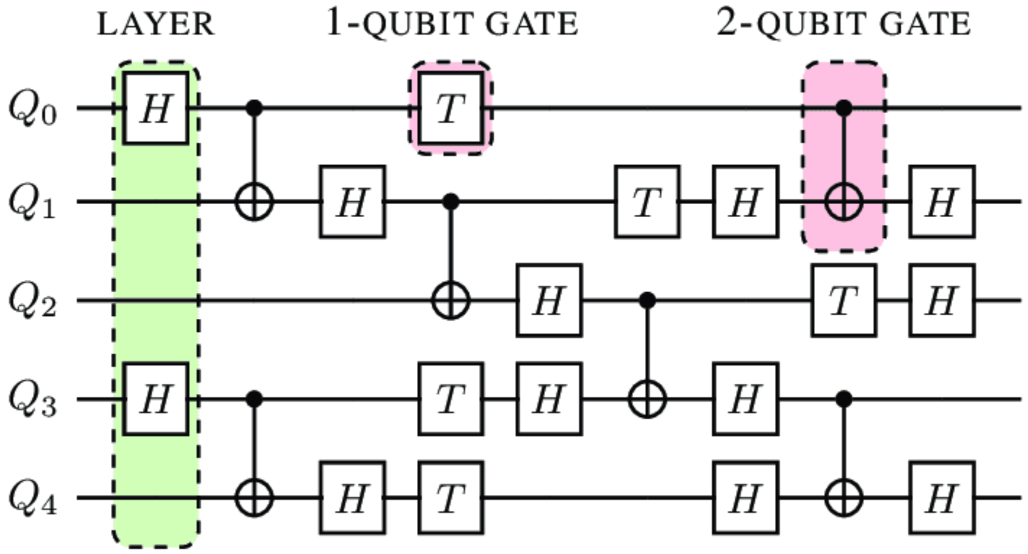
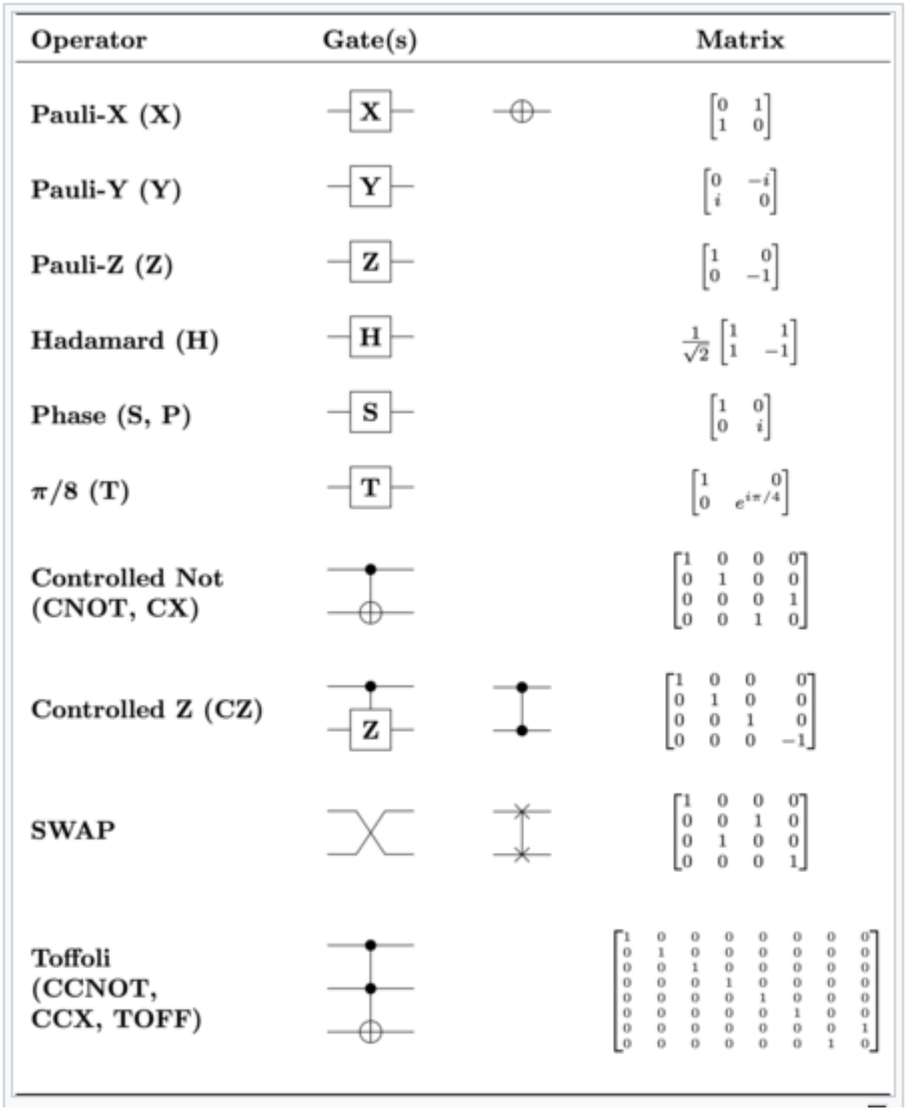


Figure 2. Example Quantum Circuit

A qubit (short for quantum bit) is the basic unit of information in quantum computing and counterpart to the bit (binary digit) in classical computing. A qubit plays a similar role to a bit, in terms of storing information, but it behaves much differently because of the quantum properties on which it's based.

Quantum gates have been discovered which act on one, two or three qubits and transform their states. These are analogous to the Boolean gates that are found in digital circuits and may be represented as unitary square matrices with dimension 2n where n is the number of qubits it works on.

A quantum state of multiple qubits is constructed as the tensor product of those qubits and is a vector of length 2n [1]. A quantum gate therefore acts on a quantum state by multiplication. It may take the state to an entangled one. This means that the resultant state cannot be decomposed into a tensor product of the states of separate qubits – they are no longer independent from each other.



Operator	Gate(s)		Matrix
Pauli-X (X)	$\boxed{X}$	$\oplus$	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$
Pauli-Y (Y)	$\boxed{Y}$		$\begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$
Pauli-Z (Z)	$\boxed{Z}$		$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$
Hadamard (H)	$\boxed{H}$		$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$
Phase (S, P)	$\boxed{S}$		$\begin{bmatrix}1 & 0\\0 & i\end{bmatrix}$
$\pi/8$ (T)	$\boxed{T}$		$\begin{bmatrix}1 & 0\\0 & e^{i\pi/4}\end{bmatrix}$
Controlled Not (CNOT, CX)			$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$
Controlled Z (CZ)	$\boxed{Z}$		$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&-1\end{bmatrix}$
SWAP			$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)			$\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{bmatrix}$

Figure 3. Matrix Representation of Quantum Gates

These matrix operations can be padded out so that they allow other qubits to pass through them unaltered, which means that any acyclic quantum circuit with n inputs can be represented as product of unitary matrices with dimension 2n applied to an initial state vector to produce a final state vector. This means that any quantum circuit can be represented as a single unitary matrix [13].

However, it is more intuitive for quantum programmers to work at the level of quantum gates, and it is possible to fabricate composite gates from collections of the standard building blocks, shown in figure 3, to build more sophisticated functions as patterns and provide a higher level abstraction of quantum algorithms. A special measurement gate can be applied within quantum circuits to observe a possible classical state of one or more qubits. Once observed, the quantum state collapses to the observed classical state.

The programming languages that are currently used for quantum computing are assembly languages rather than high level programming languages. There is a rich seam of research of quantum circuit optimisation, which seeks to optimise quantum circuit complexity by performing semantics-preserving transformations, e.g. [9].

3 Why is it Hard to Program Quantum Computers?

It is extremely tricky to fit together the somewhat arbitrary operations for which quantum gates have been discovered (defined in Hilbert Space) to build meaningful programs that solve familiar problems. This essentially relies on intuition and a degree of trial and error.

Also, the property of entanglement, which is useful for creating efficient quantum algorithms, leads to side effects, i.e. if two qubits are entangled, and one of them passes through a quantum gate but not the other, the states of both qubits will be altered nonetheless, because they cannot be separated. This is a common problem to deal with in object-oriented programming and design patterns exist to manage it, but in quantum computing it only adds to the difficulty of writing structured programs that are fit for purpose.

The “no cloning theorem” also makes Quantum Computing less tractable, because of the impossibility of taking a separate copy of a qubit.

4 The CSP Language and the FDR Proof Tool

The CSP language of CAR Hoare (Communicating Sequential Processes) [11, 12] is a notation for describing patterns of communication by algebraic expressions. It is widely used for design of parallel and distributed hardware and software and for the formal proof of vital properties of such systems. (However, without computer assistance it is often impractical to prove such properties, algebraically, other than for very simple systems.) The grammar of CSP is shown in Figure 4.

Process	:==	$STOP$	Deadlock
	|	$SKIP$	Termination
	|	$CHAOS$	Might do anything
	|	event $\rightarrow$ Process	Event prefix
	|	channel?x $\rightarrow$ Process	Input
	|	channel!x $\rightarrow$ Process	Output
	|	Process_1; Process_2	Sequential composition
	|	$\text{Process}_1 \mid[\text{alph}_1 \mid \text{alph}_2]\mid \text{Process}_2$	Parallel Composition
	|	$\text{Process}_1 \sqcap \text{Process}_2$	Non-deterministic choice
	|	$\text{Process}_1 \square \text{Process}_2$	Deterministic choice
	|	**if** B **then** Process_1 **else** Process_2	Conditional
	|	Process $\backslash$ event	Event hiding
	|	f(Process)	Event relabelling
	|	name	

Figure 4. Grammar of the core CSP Language

The FDR tool (Failures Divergences Refinement) provides an automated proof system for the CSP language when it is expressed in a machine-readable format.

5 How Might CSP Help with Quantum Programming?

So why might CSP [11, 12] be a good language for creating quantum programs? Well we already have a very good textbook for writing simulated quantum computer programs in Python[1]. But Python is not a concurrent programming language - we have already seen there is plenty of concurrency in quantum computing. CSP is an algebraic language for defining and reasoning about concurrent systems. But it also has a representation as a fully working programming language that has been widely used for building distributed systems. That language is called Occam [10].

Let us now consider how the circuit from figure 4 might be represented as an Occam program.

```
SEQ
    PAR
            H(Q0)
            H(Q3)
    PAR
            CNOT(Q0, Q1)
            CNOT(Q3, Q4)
    PAR
            H(Q1)
            H(Q4)
    PAR
            T(Q0)
            CNOT(Q1, Q2)
            T(Q3)
            T(Q4)
    PAR
            H(Q2)
            H(Q3)
    PAR
            T(Q1)
            CNOT(Q2, Q3)
    PAR
            H(Q1)
            H(Q3)
            H(Q4)
    PAR
            CNOT(Q0, Q1)
            T(Q2)
            CNOT(Q3, Q4)
    PAR
            H(Q1)
            H(Q2)
            H(Q4)
```

We have represented the various gates as a collection of processes composed in a hierarchy of sequential and parallel constructs. Each of these processes takes combinations of qubits as inputs and modifies their state. This sounds simple however the phenomenon of entanglement might get in the way. Because as already described above some of the qubits that are not inputs to a gate may still have their values changed due to this phenomenon. We would need to have a modified version of Occam that allows for entangled state of inputs. Potentially this would be more intuitive and easier to use than Python. And we could create a compiler for translating from this quantum version of Occam down to machine readable CSP which would be suitable for analysis by the FDR proof tool to establish formal properties of the underlying system.

Unfortunately, state explosion could cause issues here - we will need some way of representing in an abstract way the essential characteristics of data passing through the system without having to get down into the messy numerical details.

5.1 Modelling a standard acyclic quantum 'circuit'.

So, would it make sense to model a standard acyclic quantum circuit in "Quantum Occam"? The main benefit would seem to be in the intuitive representation of a distributed system as computer programming code. But a secondary benefit would be the possibility of carrying out algebraic proofs and automated final proofs using the FDR tool by translating from Quantum Occam down to machine readable CSP with a suitable level of data abstraction to contain the state explosion problem.

5.2 Designing and constructing composite quantum gates out of fundamental components.

It might also be possible to reason about the correctness of novel quantum algorithms using CSP, but it would very much depend on suitable abstractions away from the numerical complexity to make this tractable. And as already mentioned we need to find an elegant treatment for the phenomenon of entanglement to support this work.

5.3 Modelling hybrid classical/quantum systems.

Earlier in this paper we discussed the principle of embedding quantum computers within classical circuitry purely as deterministic computational devices – essentially as "maths coprocessors". Now in this scenario there would be an obvious use for the CSP language for modelling the hybrid circuits - essentially concentrating on the classical aspects of these rather than the quantum functions, which would be treated as 'black boxes'.

5.4 Modelling entanglement between qubits, how it propagates through the circuit, to avoid side effects of measurement.

One of the difficult aspects of quantum computing, as already described above, is managing the side effects of entanglement. Once quantum states have become entangled then the state of one can be altered due to the other one passing through a quantum gate. This is similar to the issue of side effects that are sometimes found in object-oriented programming when changing an object's value might indirectly change the state of another one because of the use of reference datatypes. Xia and Zhao [14] have created a pre-processing quantum compiler which analyses the entanglement relationships by static code analysis in quantum programmes. This is potentially an area where a tool like FDR could gain a foothold - especially by using the principle of data independence to conceal mathematical complexity.

5.5 Modelling error correction with redundancy

One of the unknown problems about quantum computing is to what extent will issues of random errors and faulty circuitry limit the expansion of quantum computers to much larger numbers of qubits so that they may carry out meaningful calculations. There is a school of thought that this problem will proliferate to such an extent that there will never be useful quantum computers. Current efforts to remediate this problem are founded upon error correction using redundancy. However, this is a non-trivial problem within the quantum domain due to the no cloning theorem.

Essentially splitting one logical qubit across multiple physical qubits can reduce the error rate. A common belief is that around 1000 physical qubits will be needed to model each logical qubit, but this will surely depend on the physical implementation of the qubits.

CSP/FDR might be useful in establishing the equivalence of virtual qubits and their physical counterparts if certain constraints are assumed to be true.

6 Conclusions

I have explained some of what makes quantum computing so difficult for programmers with conventional training and have considered ways that CSP might be used to help make pure quantum computing more tractable. However, perhaps the most promising use of CSP will be to model hybrid systems consisting of classical circuitry with multiple embedded quantum coprocessors working concurrently. This approach might emerge as a highly productive programming model for exploiting quantum computers. Discovery of novel efficient quantum algorithms will remain something of a black art performed exclusively by mathematicians and theoretical physicists, potentially exploiting AI assistants, engaged in trial and error to piece together circuitry and quantum gates to create efficient solutions to specific deterministic computational problems.

Please note that the opinions expressed here are those of the author and do not necessarily represent those of Lloyds of London.

References

1. Robert Hundt, “Quantum Computing for Programmers”, Cambridge University Press 2022.

2. Feynman, Richard. "Simulating Physics with Computers", International Journal of Theoretical Physics. 21 (6/7): 467–488 1982

3. Yuri I. Manin, “Mathematics as Metaphor. Selected Essays of Yuri I. Manin” 77–78, American Mathematical Society, 2007.

4. Shor, P.W., “Algorithms for quantum computation: discrete logarithms and factoring”. In: Proceedings 35th annual symposium on foundations of computer science, pp. 124–134 1994.

5. A. A. Clerk, M. H. Devoret, S. M. Girvin, F. Marquardt, and R. J. Schoelkopf, “Introduction to quantum noise, measurement, and amplification,” Rev. Mod. Phys. 82, 1155–1208 (2010)

6. Wootters, W., Zurek, W. “A single quantum cannot be cloned.” Nature 299, 802–803 1982

7. Park, James. "The concept of transition in quantum mechanics". Foundations of Physics. 1 (1): 23–33. 1970.

8. Jeremy M. R. Martin, “Concurrency and Models of Abstraction: Past, Present and Future”, Proceedings of 2023 Concurrent Processes Architectures and Embedded Systems Hybrid Virtual Conference. Kalpa Publications in Computing 2023

9. Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. “Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus”. Quantum 4, 279 (2020)

10. D. May, R. Taylor, “Occam-an overview”. Microprocessors and microsystems, 1984 – Elsevier

11. C. A. R. Hoare. “Communicating sequential processes”, Prentice-Hall, 1985.

12. A. W. Roscoe. “The theory and practice of concurrency”, Prentice Hall, 1998.

13. “How to interpret a quantum circuit as a matrix?” https://quantumcomputing.stackexchange.com/questions/2299/how-to-interpret-a-quantum-circuit-as-a-matrix Access date 30th June 2024.

14. S. Xia and J. Zhao, "Static Entanglement Analysis of Quantum Programs," in 2023 IEEE/ACM 4th International Workshop on Quantum Software Engineering (Q-SE), Melbourne, Australia, 2023 pp. 42-49.