A quantum circuit modeling toolkit for high performance computing

Makhamisa Senekane[1], Bheki Zulu[1] and Francesco Petruccione[1,2]

[1] Centre for Quantum Technology, School of Chemistry and Physics, University of KwaZulu-Natal, P/Bag X54001 Durban, South Africa

[2] National Institute for Theoretical Physics (NITheP_KZN), P/Bag X54001 Durban, South Africa

E-mail: 211560527@stu.ukzn.ac.za, 210556489@stu.ukzn.ac.za, petruccione@ukzn.ac.za

Abstract. Theoretically, quantum computers are known to solve a certain class of problems more efficiently than their classical counterparts. This is due to parallelism which is inherent in quantum algorithms. However, a full-scale quantum computer has not been realised as yet. Therefore, in order to validate and debug quantum circuits, a classical computer is used. Since most of these circuits are simulated using personal computers (PCs), quantum circuits with a limited number of quantum bits (qubits) can only be simulated, due to computational limitations of PCs. In this work, we report the simulation of quantum circuits for a high performance platform using message passing interface for the Python (mpi4py) package.

1. Introduction

Quantum computation uses quantum mechanical principles such as entanglement, superposition and tunneling to process information. It dates back to the ideas of Manin [1], Feynman [2] and Bernioff [3] in the 1980s. It was mainly motivated by the limitations imposed on the conventional classical computation by the fundamental laws of physics. Additionally, the first universal quantum machine was proposed by Deutsch [4] in 1985. However, a major breakthrough came in 1994, with the work of Shor [5], who showed that a quantum computer can be used to break some of the conventional cryptographic algorithms. Another contribution was by Grover [6] two years later, who invented a quantum search algorithm which offers a quadratic speed-up over the best known classical search algorithm. Since then, there has been a concerted effort to develop more quantum algorithms for different applications [7, 8, 9].

In a classical computer paradigm, computer scalability can be governed by Moore's law [10]. However, as the transistors get smaller, the quantum effects would be too pronounced, and Moore's law would cease to apply. Therefore, since Moore's law does not apply anymore in the quantum regime, scalability of quantum computers cannot be determined by Moore's law. On the contrary, scalability in the quantum regime is dependent on the ability to isolate the quantum system (hence mitigate decoherence) and the advancement of fault-tolerant quantum computation mechanisms.

Although quantum computation offers many interesting computational capabilities, the major drawback is that a scalable quantum computer is yet to be realized. So, in order to debug and validate quantum algorithms, a classical computer is used. Furthermore, the Gottesman-Knill theorem [11], makes it possible for the classical simulation of some quantum circuits, since it

states that a certain class of quantum circuits, known as stabilizer circuits, can be simulated efficiently on a classical computer. Many classical simulators of quantum circuits have been simulated [12]. A comprehensive list of these simulators can be found in Ref. [13]. It can be observed that most of the simulators developed thus far are intended for a stand-alone personal computer (PC). Some attempts have been made to develop simulators that make use of high performance computing (HPC) [14, 15, 16]. We report in this work a quantum circuit modeling toolkit which is intended for use on a high-performance computing platform. This simulator is developed using Python programming language, which ease of use and rapid development. Additionally, Python is freely available, with an open-source licensing. Finally, since Python is a vectorized language, it is suitable for high performance scientific computing. A python module that is used to develop this toolkit is message passing interface for Python (mpi4py).

The remainder of this manuscript is arranged as follows. The next section provides a background information on quantum computation and high performance computing. Section 3 discusses the implementation details of the simulator. It is followed by Section 4, which discusses the simulation results obtained using this simulator. The last section concludes this manuscript.

2. Background Information

2.1. Quantum Computation

As opposed to a classical computer, which has a binary digit (bit) as its basic unit of information, a quantum computer has a quantum bit (qubit) as its unit of information. Additionally, a bit, which can only exist in state 0 or 1, a qubit can exist in the superposition of states $|0\rangle$ and $|1\rangle$, where states $|0\rangle$ and $|1\rangle$ are known as computational basis states. These computational basis state can be represented as column vectors, where

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{1}$$

and

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2}$$

Mathematically, a qubit can be represented as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{3}$$

where α and β, which are known as probability amplitudes, satisfy the condition

$$|\alpha|^2 + |\beta|^2 = 1. \tag{4}$$

Analogous to their classical counterparts, the building blocks of quantum circuits are quantum gates. However, unlike some of the classical gates, which are not reversible, all quantum gates are reversible. This owes to the fact that basically, a quantum gate is any unitary operation, and unitaries are invertible, hence reversible. Some of the common quantum gates are

a) Hadamard gate, H

$$\mathrm{H} = 1/\sqrt{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

b) Phase shift gate, R

$$\mathrm{R} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

2.2. High Performance Computing

Although it is generally believed that classical computers cannot efficiently simulate universal quantum computers, it is nevertheless possible to classically simulate some of the quantum circuits, especially if such circuits are small (using few qubits). Because a stand-alone PC has limited memory and computing speed, and computational resources for simulating quantum circuits on a classical computer grow exponentially with an increase in input qubits, a PC puts huge limitations on the quantum circuits that can be efficiently simulated. These limitations can be overcome by classical simulation of quantum circuits on a high performance computing platform. HPC deploys the aggregation of computing power among different nodes so as to yield high overall computational power. It makes use of supercomputers and parallel processing schemes to solve compute-intensive problems [17].

High performance computing can be implemented using either of the two computing architectures, namely shared memory or distributed memory architectures. In the former architecture, all the processors have access to the entire memory of the machine. In contrast, in the latter architecture, each machine has a processing element (PE), which contains its processor and its memory, where each processor could only access the memory in its PE.

2.3. Message Passing Interface

MPI is an industry-wide standard protocol for passing messages between parallel processors [18] though generally, it can also run on shared memory machines. It is a *de facto* standard for parallel programming on distributed memory architecture systems [19]. It's goals are high performance, scalability and portability. Furthermore, message interface passing supports both point-to-point and collective communication.

It is worth noting that MPI is in itself neither an implementation nor a language, but rather a specification or an application programmer interface (API). MPI has a number of components such as user-defined datatypes, communication ports, communication operations and communication contexts [20, 21]. Typical communication operations include broadcast, reduce, scatter and gather [21].

2.4. Message Passing Interface for Python

One of the implementations of MPI standard for Python programming language is mpi4py [22]. It provides an object oriented approach to message passing which provides MPI bindings for Python. In the work presented in this manuscript, mpi4py package was used because it offers a number of benefits. These benefits include: clean and efficient MPI interface for Python, extensible and compatible implementation and easy of use.

3. Implementation

The toolkit was implemented using a single physical host, with a dual-processor PC at the clock frequency of 2.7 Gigahertz per processor, with a random access memory (RAM) of 4 Gigabytes. However, three virtual machines (VMs) were created and clustered together on this physical host machine, with each VM having a RAM of 1 Gigabytes. Different quantum circuits were then simulated, using various virtual nodes. Some of the circuits simulated include Einstein-Podolsky-Rosen (EPR) pair generation circuit, Greenberger-Horne-Zeilinger (GHZ) state preparation circuit and a three-qubit quantum Fourier Transform circuit [7, 12]. The QFT circuit was further expanded to simulate up to nine qubits, so as to make a more realistic comparison with the PC-based results obtained in [12]. The simulations were implemented using Python programming language, using mpi4py for MPI implementation.

Quantum gate operations can include either multiplication of state vectors with unitary matrices or tensor product for composite systems. As an example, consider the EPR pair generation circuit in figure 1. Also, consider that the two inputs are $|a\rangle = 0$ and $|b\rangle = 0$ respectively. Then

this can be mathematically represented as

$$\text{epr_generation}(|00\rangle) \rightarrow 1/\sqrt{2}* \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \left[\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right]$$
$$= 1/\sqrt{2}* \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

This result can be naturally decomposed into smaller tensor matrices, which can then be distributed over different prosessors for HPC [14, 23, 24]. A similar approach was taken for other quantum circuits. The simulation results obtained are provided in the next section.

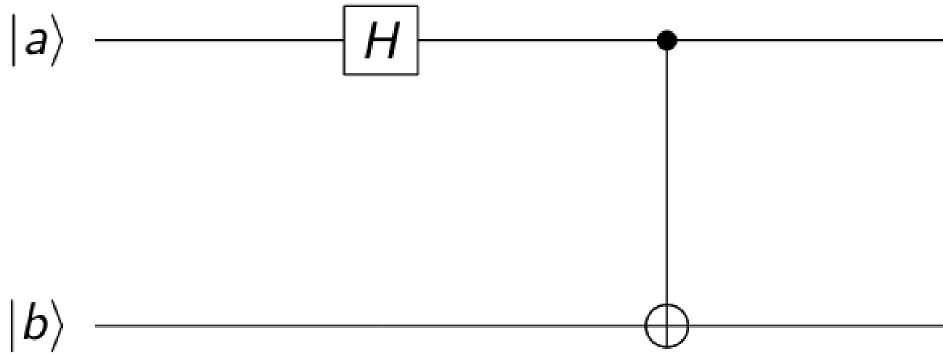

Figure 1. A circuit diagram of EPR pair generation circuit. It consists of two inputs $|a\rangle$ and $|b\rangle$, together with two quantum gates (Hadamard and control NOT (CNOT) gates.

4. Evaluation

Efficiency tests were run for this high performance computing quantum circuit simulation toolkit. Three quantum circuits were used, and the execution times of each circuit with respect to the number of processes were recorded. The timer functionality *Wtime()* was used to record the execution times. Figures 2, 3 and 4 show the results obtained for EPR pair generation, GHZ state generation and three-qubit QFT circuits respectively. The results obtained underline the utility of the simulator for quantum circuit simulation, since an increase in the number of processes used results in a significant decrease in execution time. Furthermore, the results demonstrate the feasibility of a portable, parallel simulation toolkit for quantum computing, written in Python. A measure that is employed for the utility of the simulator reported in this manuscript is the speed-up as compared to the stand-alone PC. This is given as:

$$speedup = \frac{T_{serial}}{T_{parallel}}, \tag{5}$$

where T_{serial} is the execution time for a stand-alone PC given in [12], while $T_{parallel}$ is the execution time obtained in figures 2, 3 and 4. From the results, the speed-up of the EPR pair generation circuit can be up to 3.4, that of GHZ circuit can be up to 3.2 while that of three-qubit QFT can be up to 2.5. These speed-ups provide a good support for the utility of this quantum circuit modeling toolkit.

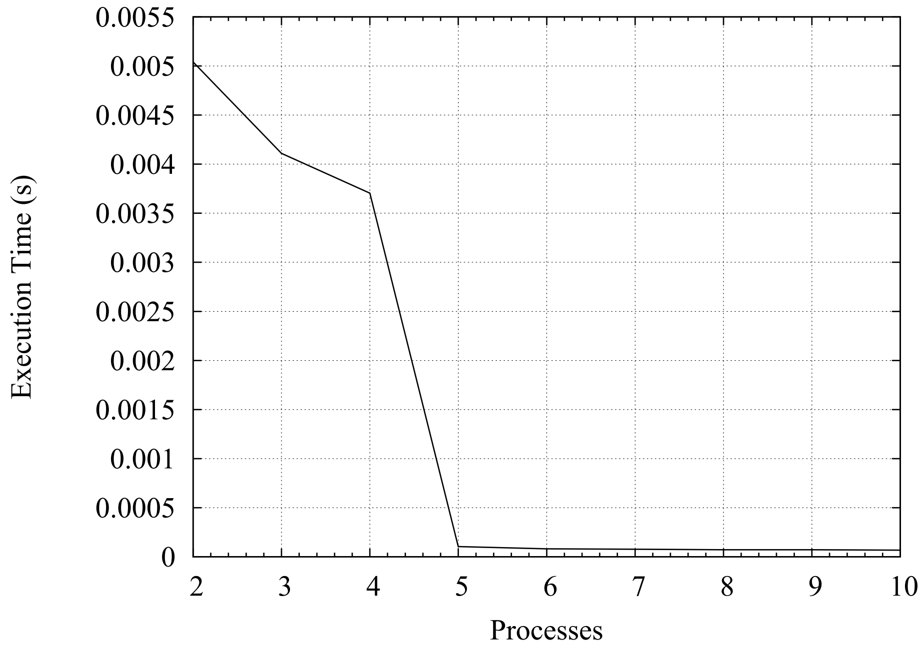

Figure 2. Execution time for EPR pair generation circuit as a function of number of processes.

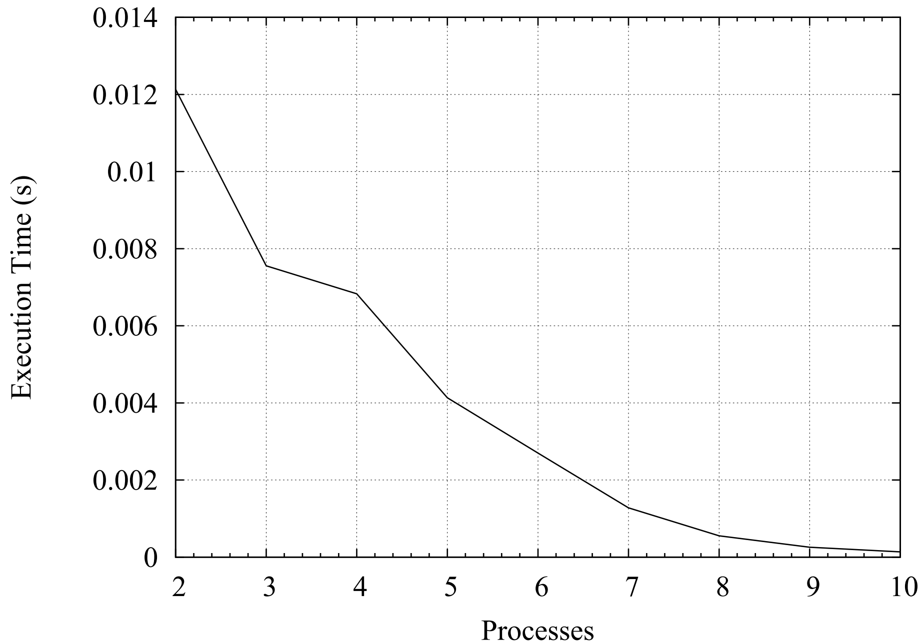

Figure 3. Execution time for GHZ state generation circuit as a function of number of processes.

Finally, coupled with the speed-up mentioned above, since this simulator uses Python programming language, it can be easily extended to simulate other quantum circuits, with a very rapid development time. Additionally, since Python is one of the three leading progamming languages in high performance scientific computing (the other two being C/C++ and Fortran), coupled with the fact that it is the only one of the three that is vectorized, it makes it a good candidate for quantum computing simulation toolkit. This circuit can then be used to simulate different type of quantum circuits mentioned in this manuscript.

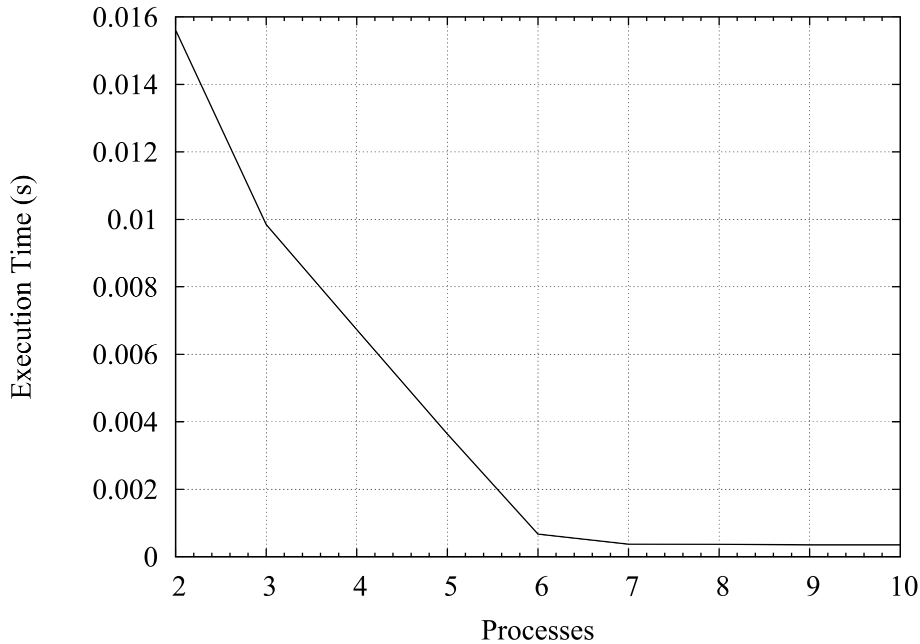

Figure 4. Execution time for three-qubit QFT circuit as a function of number of processes.

It is worth noting that this current simulator in its current form is limited in terms of functionality, since very few quantum circuits can be efficiently simulated. Additionally, the focus is only on the increase in the number of processes in order to effect a speed-up. The work is still on-going to increase the number of quantum circuits that can be simulated using this toolkit, and the running of tests on a physical (as opposed to virtual) HPC platform with multiple number of nodes and communication delays between the nodes.

5. Conclusion

In this manuscript, we have reported a quantum circuit modeling toolkit for HPC. This toolkit was evaluated using standard stabilizer circuits. The simulation results confirm the utility of the toolkit. However, this simulator still has some limitations. The quantum circuit that can be simulated by this toolkit are very few. This is because the work presented in this manuscript is still in progress. The next focus will be on the optimization of the simulator code, in order to make it extensible and scalable. Additionally, it (future work) will focus on extending the simulated circuits to any generalized quantum digit circuit, and implementing such circuits on a physical HPC platform.

Acknowledgments

This work is based on research supported by the South African Research Chair Initiative of the Department of Science and Technology and National Research Foundation.

References

[1] Knill E, Laflamme R, Barnum H, Dalvit D, Dziarmaga J, Gubernatis J, Gurvits L, Ortiz G, Viola L and Zurek W 2002 *Los Alamos Science* **27**
[2] Feynman R P 1982 *International journal of theoretical physics* **21** 467–488
[3] Benioff P 1980 *Journal of Statistical Physics* **22** 563–591
[4] Deutsch D 1985 *Proceedings of the Royal Society of London A. Mathematical and Physical Sciences* **400** 97–117

[5] Shor P W 1994 *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on* (IEEE) pp 124–134
[6] Grover L K 1996 *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (ACM) pp 212–219
[7] Nielsen M A and Chuang I L 2010 *Quantum computation and quantum information* (Cambridge University Press)
[8] Childs A M and Van Dam W 2010 *Reviews of Modern Physics* **82** 1
[9] Bacon D and VAn DAm W 2010 *Communications of the ACM* **53** 84–93
[10] Moore G E *et al.* 1998 *Proceedings of the IEEE* **86** 82–85
[11] Gottesman D 1998 *arXiv preprint quant-ph/9807006*
[12] Senekane M, Mohapi L and Petruccione F 2013 *AFRICON, 2013* (IEEE) pp 1–4
[13] "Quantiki" "2010 (Accessed: March 16, 2014)" *"List of QC simulators"* URL ‘‘http://www.quantiki.org/wiki/List_of_QC_simulators"
[14] Obenland K M and Despain A M 1998 *arXiv preprint quantph9804039*
[15] Niwa J, Matsumoto K and Imai H 2002 *Unconventional Models of Computation* (Springer) pp 230–251
[16] De Raedt K, Michielsen K, De Raedt H, Trieu B, Arnold G, Richter M, Lippert T, Watanabe H and Ito N 2007 *Computer Physics Communications* **176** 121–136
[17] Hager G and Wellein G 2010 *Introduction to high performance computing for scientists and engineers* (CRC Press)
[18] Gropp W, Lusk E and Thakur R 1999 *Using MPI-2: Advanced features of the message-passing interface* (MIT press)
[19] Jin H, Jespersen D, Mehrotra P, Biswas R, Huang L and Chapman B 2011 *Parallel Computing* **37** 562–575
[20] Pacheco P S 1997 *Parallel programming with MPI* (Morgan Kaufmann)
[21] Pacheco P 2011 *An introduction to parallel programming* (Elsevier)
[22] Dalcín L, Paz R, Storti M and D‘Elía J 2008 *Journal of Parallel and Distributed Computing* **68** 655–662
[23] Patz G 2003 *A parallel environment for simulating quantum computation* Ph.D. thesis Massachusetts Institute of Technology
[24] Steeb W H 1997 *Matrix calculus and Kronecker product with applications and C++ programs* (World Scientific)