Survey of Imperative and Object Oriented Quantum Computer Programming Languages

Vivek Kumar, Anuranjan Misra

Abstract - In the academic world a variety of languages are studied and used. But with the exception of a few applications, most languages utilized for commercial applications are written in imperative and object oriented languages. A partial list of these languages includes many that would be familiar to any commercial developer: Visual Basic, C#, Java, Python, Fortran, Cobol, and so on. For the power of a quantum computer to be utilized economically in commercial applications, the programming must be easy for existing commercial developers to learn and utilize. This is best done by piggy backing off of the languages and techniques they are already familiar with- this means that successful quantum languages for existing commercial developers will likely be related to one of more of these languages, or quantum frameworks (libraries) for these languages. It should be pointed out that the popularity of languages changes with time, so as new languages come into popularity their potential for quantum computing also needs to be kept in mind. Many of today's popular languages were not designed to easily take advantages of multiple cores or processors. Consequently it is quite feasible that other languages that take advantage of these parallel processing capabilities will rise in popularity in the near future and be excellent candidates extending to carry out quantum computing.

Keywords:- C#, Java, Python, Fortran, Cobol, Visual Basic, libraries

I. INTRODUCTION

Quantum computers have the potential for solving certain types of problems much faster than classical computers. Speed and efficiency are gained because quantum bits can be placed in superposition's of one and zero, as opposed to classical bits, which are either one or zero. Moreover, the logic behind the coherent nature of quantum information processing often deviates from intuitive reasoning, leading to some surprising effects.

II. Different Approach (s)

The structure of quantum programming languages differs from existing classical languages in that the limitation must be enforced. Depending on the proposed approach, defiance of these limitations may be caught at compile time or at run time. The quantum languages typically include statements for initializing the quantum state of the system, manipulating it through (unary) operations, and finally measurement. When Knill's QRAM approach is utilized these are frequently additions to some existing classical programming techniques.

Revised Version Manuscript Received on August 20, 2015.

Mr. Vivek Kumar, M.Tech Student, Department of Electronics and Communication, Noida International University, Delhi National Capital Region Noida, India.

Dr. Anuranjan Misra, Professor & Head, Department of Computer Science and Engineering, Noida International University, Delhi National Capital Region Noida, India.

(a) Knill Approach

Knill has introduced pseudo code conventions. His pseudo code is based on imperative program techniques, as it utilizes variables and flow control statements based on that methodology. Within his paper he also provides several elementary examples of the use of his proposed pseudo code. As mentioned previously, the importance of Knill's paper lies not necessarily in the proposed pseudo code conventions, but in the use of his quantum random access machine model (QRAM). While Knill's work is an important step forward, pseudo code it has little use for writing actual applications. Even though, it is a step in the right direction.

$a \leftarrow$ MEASUREDFOURIER($\underline{a}, d$)

Input: A quantum register $\underline{a}$ with d qubits. The most significant qubit has index $d-1$.

Output: The amplitudes of $\underline{a}$ are Fourier transformed over $\mathbb{Z}_{2^d}$, and then measured. The most significant bit in the output has index 0, that is the ordering is reversed. The input quantum register is returned to a classical state in the process.

$\omega \leftarrow e^{i2\pi/2^d}$
$\phi \leftarrow 0$
for $i = d-1$ **to** $i = 0$
  $\mathcal{R}_\phi(\underline{a_i})$
  $\mathcal{H}(\underline{a_i})$
  $a_i \leftarrow \underline{a_i}$
  $\phi \leftarrow (\phi + a_i\pi)/2$
  C: *The expression on the right of this assignment statement requires a_i to be in a classical state as it involves operations not allowed for quantum registers.*

Figure1. Measured Fourier transform utilizing Knill's pseudo code

(b) Sanders and Zuliani Approach

Sanders and Zuliani developed the programming language qGCL as a means to express quantum algorithms. The primary purpose of the language is for program derivation, correctness of proof, and teaching. As the authors point out, qGCL does not aim to do numerical simulations of quantum algorithms like Omer's QCL, which will be covered later. Within the paper they first describe a probabilistic extension to Dijsktra's guarded command language (GCL), which they appropriately call pGCL. They then extend pGCL to invoke quantum procedures and call the resulting language qGCL. Thus qGCL is like many other proposed quantum programming techniques where the computation is controlled by a classical computer utilizing a quantum sub system. The three quantum procedures they outline and place emphasis on are fundamental to any system carrying

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

out quantum computation: initialization, evolution, and finalization (or observation). They also provide implementations of several quantum algorithms, including Shor's and Grover's. Since GCL was proposed in 1975, and qGCL is an augmentation to it, qGCL may be too limited and dated to construct commercial applications. Like Knill's pseudo code, qGCL also suffers from a very mathematical syntax- something that is harder for commercial programmers to understand and even type. As the authors point out though, this simplicity makes it an effective tool for teaching the basics of quantum programming.

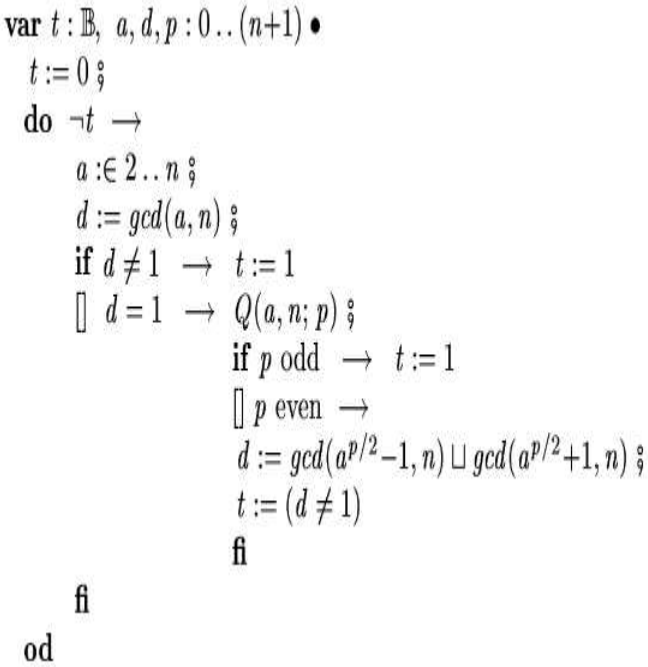

Figure 2. Shor's algorithm in Sanders and Zuliani's q GCL [51]

(c) Bettelli Approach

Bettelli has developed a preliminary extension for C++, in the form of a library, for quantum computer programming. This library exposes several classes that can be utilized for quantum computation. The use of classes provides the important benefit of encapsulating the workings of the library and hiding them from users. Furthermore, unlike some procedural implementations, rules can be better enforced and valid states maintained through the use of classes. Bettelli's implementation also generates quantum operations, and these byte codes could be piped to an actual quantum sub system or a simulator. While the library is in a preliminary form, Bettelli's paper also contains a list of features desirable for a scalable quantum programming language. One of the most important of these points is that a quantum programming language should be an extension of a classical language. Extensions can take a variety of forms: class libraries, dynamically linked libraries, and assemblies to name a few. Not only does extending a classical language make it easier for existing programmers to utilize quantum features, but it also helps to keep the library useful as the language surrounding it evolves to tackle classical problems. Thus the author of the quantum extention can focus on tackling only those issues that apply to quantum computing instead of all issues as must be done with a proprietary language. It is important to note that some languages, such as Python, are evolving iteratively through open source methods as opposed to large standards developed over a period of years as is the case with C and C++. C++ was developed in 1984, but the standard was not approved until 1998 enough time for processors to double in speed seven times in accordance with Moore's law. Additionally, there have been over 8,500 programming languages developed, yet only a select few of these are actually used in industry- further strengthening the argument for creating extensions of existing languages instead of new languages. Bettelli's work is the most useful to existing programmers because C++ is a widely used language and only the library needs to be learned, not an entire new language. As new languages are developed and speed and efficiency of a language are not as important due to increased computing power, C++ seems to be declining in popularity.

```
Qbitset run_Grover(bool(*f)(int), int n) {
   int repetitions = sqrt(pow(2.0,n));
   Qop phase_oracle(f,n);
   Qop invert_zero(f_0,n);
   Qop mixer = QHadamard(n);
   Qop invert_mean = mixer & invert_zero & mixer;
   Qop grover_step = phase_oracle & invert_mean;
   Qreg input(n);
   mixer(input);
   for (int i=0; i<repetitions; ++i) grover_step(input);
   return input.measure();
}
```

Figure 1. Grover's algorithm in Bettelli's C++ extension [32]

Over a period of six years, 1998 – 2004, Omer has developed what is arguably the most complete quantum programming language to date: Quantum Computation Language, or QCL. QCL is a language that has a structure similar to C, making it easy to learn for many programmers because C and its decedents such as C++, C#, and Java are popular languages. However this strength of basing QCL on C is also part of its downfall. C is still used for low level applications such as drivers, but not for cutting edge commercial software. As a result, QCL does not have many of the features available in modern languages. By being a proprietary language QCL would be difficult to adopt in the real world for many programmers writing applications since it does not have the power and libraries available to modern languages. Omer has also created a complete simulator for QCL programs, including an interpreter. Having an interpreter for QCL allows for students of the language to create and see how code behaves in real time. In a benefit to all studying quantum computing, Omer has also made the source code of the interpreter available. While the inclusion of the interpreter and source code makes QCL useful, the fact that it is a new language does present an obstacle to those wishing to learn quantum computer programming. As

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

with all new languages, it also makes it harder to integrate quantum algorithms into existing code bases.

```
/* Define Oracle */

const coin1=(random()>=0.5);        // Define two random boolean
const coin2=(random()>=0.5);        //   constants

boolean g(boolean x) {              // Oracle function g
  if coin1 {                        // coin1=true  -> g is constant
    return coin2;
  } else {                          // coin1=false -> g is balanced
    return x xor coin2;
  }
}
qufunct G(quconst x,quvoid y) {     // Construct oracle op. G from g
  if g(false) xor g(true) { CNot(y,x); }
  if g(false) { Not(y); }
}

/* Deutsch's Algorithm */

operator U(qureg x,qureg y) {       // Bundle all unitary operations
  H(x);                             //   of the algorithm into one
  G(x,y);                           //   operator U
  H(x & y);
}

procedure deutsch() {               // Classical control structure
  qureg x[1];                       // allocate 2 qubits
  qureg y[1];
  int m;
  {                                 // evaluation loop
    reset;                          //   initialize machine state
    U(x,y);                         //   do unitary computation
    measure y,m;                    //   measure 2nd register
  } until m==1;                     // value in 1st register valid?
  measure x,m;                      // measure 1st register which
  print "g(0) xor g(1) =",m;        //   contains g(0) xor g(1)
  reset;                            // clean up
}
```

Figure4. Deutsch's algorithm expressed in Omer's QCL [58]

(d) Blaha Approach

Blaha has introduced a quantum assembly language and quantum C language. In his two language proposals the languages themselves are algebraic in nature, which he argues allows for better understanding of the language and proof of correctness if necessary. Within Blaha's work however, less than one page is dedicated to his quantum C language, and most of that involves an explanation of pointers in C. So while he proposes a quantum C language, there isn't much of an explanation of how it works other than defining the algebraic representation of the pointer operations. It is also interesting to note that Blaha was able to obtain trademarks for what would seem to be generic terms in the field of quantum computing, including "Probabilistic Grammar", "Quantum Grammar", and "Quantum Assembly Language". Like Bettelli's work, Blaha's use of C makes the approach very viable. However, without further details it is hard to gauge how easy it is to actually use.

(e) Markus Approach

Markus has devised a method to simulate quantum computing using Fortran. While not a true language or framework in itself, it is worth noting because it is an example of how such a library would work. Currently any quantum computing language or library must simulate the quantum system since quantum computers are currently unavailable for use in programming. Many languages are derived from Fortran, so Markus's paper gives a good insight on how to actually accomplish that for a variety of languages. Included in the paper is the full source code listing for the simulation, along with debugging statements. It is also notable that Fortran has been used as a parallel programming language in the Fortran-K implementation, which is a subset of Fortran-90. Nonetheless, more modern languages such as Fotress could also be used to simulate quantum computing and be more accessible. Providing the source code is invaluable for others developing quantum libraries as it provides a source of solutions for problems that may arise during implementation, and this is a benefit of the work Markus has done.

(f) Carini Approach

Carini has developed a method to simulate qubits using the programming language Ruby. Like Markus's Fortran simulation, even though it is not a language or framework it is noteworthy due to the implementation techniques. Carini's implementation involves simulating the states of a qubit on separate threads, although she admittedly ran into some scheduling issues. This is another important insight for the simulator of any proposed language or framework- the simulation should take advantage of today's multiprocessor systems. Doing so increases efficiency of the simulation, but presents challenges of its own through the need to implement parallel processing techniques. In particular this presents a problem for any framework or language built upon the Python programming language due to the global interpreter lock. While Python is a concise and easy to program in language, only one thread within a process can access Python objects at a time . This means that even with a multiprocessor system, multithreaded Python programs cannot take full advantage of it as they effectively use one processsor. The work around for this is to implement multiple processes within Python instead of multiple threads. Even with this difficulty Python is still a good candidate for building a quantum computing framework on. Python is platform independent, like Java, so it eliminates the need to port to different systems. Unlike Java though, it is an interpreted language, which allows for one to dynamically interact with the system like Omer's QCL.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

(g) Svore Approach

Svore and colleagues have developed a suite of tools for use in quantum computation . These tools include a language, compiler, optimizer, simulator, and layout tools. A key feature to the language, as others have pointed out as necessary, is that it is machine independent. For practical purposes quantum computers are not yet a reality, so any proposal for programming them must be independent of whatever solution is used to realize them. Within their paper they also propose translating their high level language into a quantum intermediate language (QIR) which then gets translated into a quantum assembly language (QASM), and finally a physical language (QCPOL). This is approach is the similar to many modern day classical languages. As with many other quantum programming proposals, this one also makes use of Knill's QRAM model. Another key to the proposal is that quantum error correction be implemented on a lower level and not within the higher level language itself. This higher level abstraction is akin to how modern day programmers are not concerned with error correction within RAM or through a network connection. While the purpose of the various languages and transitions between them are described, the work does not actually include specifications for the languages themselves. As such, the languages themselves remain an open problem as is pointed out at the end of the paper as an important challenge.

(h) Tucci Approach

Tucci has developed quantum compiler that compiles steps of an algorithm into a sequence of elementary operations . The implementation of his compiler proposal is called "Qubiter", for which he has made the source code in C++ freely available. While still in a basic state as he admits and lacking a GUI it is still a valuable learning tool because the source code is available. Notable about his compiler is that it will also perform optimizations. These insights he provides on optimization would be useful for any other quantum programming system in order to increase efficiency. Tucci also received a patent for the ideas that Qubiter represent in 2002 .

```
ROTY 3 45.0000000
ROTY 2 45.0000000
ROTY 1 45.0000000
CPHA 1 T 180.000000
CPHA 2 T 180.000000
CPHA 3 T 180.000000
ROTY 0 45.0000000
CPHA 0 T 180.000000
```

Figure 2. Output of Tucci's Qubitter for the input 4 bit Hadamard matrix, which is also known as Hardamard-Walsh transform.

III. Conclusion

While there has been a small variety of quantum computing programming proposals utilizing the imperative or object oriented approach, none of them is equivalent to or utilizes the more wide spread modern programming languages such as C#, Visual Basic, Java, or Python. The lack of a quantum computing framework for any of these languages makes quantum computer programming less accessible to the average commercial developer. Just as important, usability has also been neglected. So while the languages and libraries presented could be used, the fact that they are not similar to or use modern languages represents a significant hurdle to their use by practicing commercial developers. The fact that modern languages are not utilized for quantum computer programming and usability has been largely ignored represents an excellent candidate for work in the field of quantum computer programming.

REFERENCES

1. T. J. Bergin, "A History of the History of Programming Languages," Communications. ACM, vol. 50, p. 5, May 2007 2007.
2. E. Knill, "Conventions for Quantum Pseudocode," Los Alamos National Laboratory LAUR-96-2724, 1996.
3. D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," Proceedings of the Royal Society of London, vol. A, pp. 97-117, 1985.
4. G. Fairbanks, D. Garlan, and W. Scherlis, "Design fragments make using frameworks easier," in Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications Portland, Oregon, USA: ACM, 2006.
5. W. E. Halal, "Technology's Promise: Expert Knowledge on the Transformation of Business and Society," 2007.
6. P. Strathern, The Big Idea: Turing and the Computer, 1 ed. New York, NY: Doubleday, 1997.
7. Turing, "On Computable Numbers, with an Application to Entscheid-ungsproblem," Proc. London Math Society, vol. 42, pp. 230-265, 1936.
8. Burda, Introduction to Quantum Computation, 1 ed. Boca Raton, FL: Universal Publishers, 2005.
9. M. Hivensalo, Quantum Computing, 2 ed. Berlin: Springer, 2004.
10. E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," Commun. ACM, vol. 18, pp. 453-457, 1975.
11. B. Cannon, "Guido, Some Guys, and a Mailing List: How Python is Developed." vol. 2007: Python.org, 2007.
12. R. W. Sebesta, Concepts of Programming Languages, 5 ed. Boston, MA: Addison-Wesley, 2002.
13. NCITS, "International Standard 14882 - Programming Language C++." vol. 2007: International Committe for Information Technology Standards, 1998.
14. B. Omer, "A Procedural Formalism for Quantum Computing," in Theoretical Physics. vol. Masters Vienna: Technical University of Viena, 1998, p. 93.
15. B. Omer, "Procedural Quantum Programming," AIP Conference Proceedings, vol. 627, pp. 276-285, 2001.
16. B. Omer, "Structured Quantum Programming," in Information Systems. vol. Ph.D. Vienna: Technical University of Vienna, 2003, p. 130.
17. B. Omer, "Classical Concepts in Quantum Programming," Internation Journal of Theoretical Physics, vol. 44, pp. 943-955, July 2005 2004.
18. B. Omer, "QCL - A Programming Language for Quantum Computers: Source and Binaries," 0.6.3 ed. vol. 2007: Omer, Bernhard, 2006, p. Source and binary downloads of the QCL interpreter.
19. S. Blaha, "Quantum Computers and Quantum Computer Languages: Quantum Assembly Language and Quantum C Language," in Cosmos and Consciousness, 1 ed: Janus Associates Inc., 2002, p. 292.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.