SoftwareX 7 (2018) 245–254

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

A language and hardware independent approach to quantum–classical computing☆

A.J. McCaskey [a,b,*], E.F. Dumitrescu [a,c], D. Liakh [a,d], M. Chen [e,f], W. Feng [f], T.S. Humble [a,c,g]

[a] Quantum Computing Institute, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[b] Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[c] Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[d] Oak Ridge Leadership Computing Facility, Scientific Computing, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[e] Department of Physics, Virginia Tech, Blacksburg, VA 24060, USA
[f] Department of Computer Science, Virginia Tech, Blacksburg, VA 24060, USA
[g] Bredesen Center for Interdisciplinary Research, University of Tennessee, Knoxville, TN 37996, USA

ARTICLE INFO

Article history:
Received 30 April 2018
Received in revised form 24 July 2018
Accepted 25 July 2018

Keywords:
Quantum computing
Quantum software

ABSTRACT

Heterogeneous high-performance computing (HPC) systems offer novel architectures which accelerate specific workloads through judicious use of specialized coprocessors. A promising architectural approach for future scientific computations is provided by heterogeneous HPC systems integrating quantum processing units (QPUs). To this end, we present **XACC** (*e**X**treme-scale **ACC**elerator*) — a programming model and software framework that enables quantum acceleration within standard or HPC software workflows. XACC follows a coprocessor machine model that is independent of the underlying quantum computing hardware, thereby enabling quantum programs to be defined and executed on a variety of QPUs types through a unified application programming interface. Moreover, XACC defines a polymorphic low-level intermediate representation, and an extensible compiler frontend that enables language independent quantum programming, thus promoting integration and interoperability across the quantum programming landscape. In this work we define the software architecture enabling our hardware and language independent approach, and demonstrate its usefulness across a range of quantum computing models through illustrative examples involving the compilation and execution of gate and annealing-based quantum programs.

Code metadata

Current code version	v1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_48
Legal Code License	EPL and EDL
Code versioning system used	git
Software code languages, tools, and services used	C++, Python
Compilation requirements, operating environments & dependencies	C++11, Boost 1.59+, OpenSSL 1.0.2, CMake
Link to developer documentation/manual	https://xacc.readthedocs.io
Support email for questions	xacc-dev@eclipse.org, mccaskeyaj@ornl.gov

☆ This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. (http://energy.gov/downloads/doe-public-access-plan).

* Corresponding author at: Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA.
E-mail address: mccaskeyaj@ornl.gov (A.J. McCaskey).

https://doi.org/10.1016/j.softx.2018.07.007

1. Introduction

High-performance computing (HPC) architectures continue to make strides in the use of specialized computational accelerators, and future HPC designs are expected to increasingly take advantage of compute node heterogeneity [1]. Quantum processing units (QPUs) represent a unique coprocessor paradigm which leverages the information-theoretic principles of quantum physics for computational purposes. Several small-scale experimental QPUs, including the publicly available IBM quantum computer [2], already exist and their sophistication, capacity, and reliability continues to improve [3]. As a potential HPC accelerator, the emergence of mature QPU technologies requires careful consideration for how to best integrate these devices with conventional computing environments. While the hardware infrastructure for early QPUs is likely to limit their usage to remote access models and state-of-the-art HPC systems [4], there are clear use cases where hybrid algorithms may judiciously leverage both conventional and quantum computational resources for near-term scientific applications [5,6]. A hybrid computing paradigm is poised to broadly benefit scientific applications that are ubiquitous within research fields such as modeling and simulation of quantum many-body systems [7], applied numerical mathematics [8], and data analytics [9].

The generalization of HPC programming paradigms to include new accelerators is not without precedent. Integrating graphical processing units (GPUs) into HPC systems was also a challenge for many large-scale scientific applications because of the fundamentally different way programmers interact with the hardware. Hardware-specific solutions provide language extensions [10] that enable programming natively in the local dialect. Hardware-independent solutions define a hybrid programming specification for offloading work to attached classical accelerators (GPUs, many-integrated core, field-programmable gate array, etc.) in a manner that masks or abstracts the underlying hardware type [11]. These hardware-agnostic approaches have proven useful because they retain a wide degree of flexibility for the programmer by automating those aspects of compilation that are overly complex. Programming models for QPUs will pose additional challenges because of the radically different logical features and physical behaviors of quantum information, such as the no cloning principle and reversible computation. The underlying technology (superconducting, trapped ion, etc.) and models (gate, adiabatic, topological, etc.) will further distinguish QPU accelerators from conventional computing devices. It is therefore necessary to provide flexible classical–quantum programming models and integrating software frameworks to handle the variability of quantum hardware to promote robust application benchmarking and program verification and validation.

Approaches for interfacing domain computational scientists with quantum computing have progressed over the last few years. A variety of quantum programming languages have been developed with a similar number of efforts under way to implement high-level mechanisms for writing, compiling, and executing quantum code. State-of-the-art approaches provide embedded domain-specific languages for quantum program expression. Examples include the languages and tools from vendors such as Rigetti [12], Microsoft [13], Google [14], and IBM [15], which each enable assembly-level quantum programming alongside existing Pythonic code. Individually, these implementations provide self-contained software stacks that optionally target the vendor's unique hardware implementation or simulator backend. The increasing variability in languages and platforms raises concerns for managing multiple programming environments and compilation tool-chains. The current lack of integration between software stacks increases application development time, decreases portability, and complicates benchmarking analysis. Methods that enable cross-compilation for QPUs will support the broad adoption of experimental quantum computing through faster development time and reusable code.

To address these unique challenges, we present a programming model and extensible compiler framework that integrates quantum computing devices into an accelerator-based execution model. The eXtreme-scale ACCelerator (XACC) framework is designed for robust and portable QPU-accelerated application programming by enabling quantum language and hardware interoperability. XACC defines interfaces and abstractions that enable compilation of hybrid programs composed of both conventional and quantum programming languages. The XACC design borrows concepts from existing heterogeneous programming models like OpenCL [11] by providing a hardware-independent interface for off-loading quantum subroutines to a quantum coprocessor. Moreover, XACC enables language interoperability through a low-level quantum intermediate representation.

The structure of this work is as follows: first, we present related work with regards to quantum programming and detail inherent unique challenges that XACC seeks to address; second, we define the XACC software architecture, including platform, programming, and memory models; finally, we detail unique demonstrations of the model's flexibility through demonstrations using both gate and annealing quantum computing models.

2. Related Work

Programming, compilation, and execution of quantum programs on physical hardware and simulators has progressed rapidly over the last few years. During this time, much research and development has gone into exploring high-level programming languages and compilers [16–19]. Moreover, there has been a recent surge in the development of embedded domain specific languages that enable high-level problem expression and automated reduction to low-level quantum assembly languages [12,14,15]. However, despite progress there are still numerous challenges that currently impede adoption of quantum computing within existing classical scientific workflows [20]. Most approaches that target hardware executions are implemented via Pythonic frameworks that provide data structures for the expression of one and two qubit quantum gates; essentially providing a means for the programming of low-level quantum assembly (QASM). Compiler tools provided as part of these frameworks enable the mapping of an assembly representation to a hardware-specific gate set as well as mapping logical to physical connectivity. The arduous task of complex compiler workflow steps, including efficient instruction scheduling, routing, and robust error mitigation are left as a manual task for the user. This hinders broad adoption of quantum computation by domain computational scientists whose expertise lies outside of quantum information.

Higher-level languages exist, but do not explicitly target any physical hardware. Therefore, users can compile these high-level languages to a representative quantum assembly language, but such instructions must be manually mapped to the set of instructions specified by a given hardware gate set. This translation process is often performed by re-writing the assembly code in terms of a Pythonic execution frameworks targeting a specific device. Moreover, high-level languages have in the past assumed a fault-tolerant perspective of quantum computation. However, this interpretation is at odds with practical near-term noisy computations, for which the user must provide robust compilation tools to enable a variety of error mitigation strategies. To this end, domain specific languages enabling problem expression at higher levels of abstraction [21–23] for non-fault-tolerant quantum computing have recently been developed. These represent promising pathways for enabling a broad community of computational scientists to benefit from quantum computation.