CS 5341

Oscar Galindo

**Extending C++ for Heterogenous Quantum-Classical Computing**

This paper begins by noting that “The recent availability of programmable quantum computers over the cloud has enabled a number of small-scale experimental demonstrations” hence, the authors say, “These demonstrations point toward a future computing landscape whereby classical and quantum computing resources maybe used in a hybrid, heterogenous manner…”. Furthermore, the authors mention “… the novelty and utility of heterogenous quantum-classical compute models will only be effective if there is an enabling software infrastructure that promotes efficiency, programmability, and extensibility.” Lastly, the authors mention that there is a “void” in the quantum computing software stack where they hope that their contribution, qcor could help.

At a high level, qcor is a compiler that enables language extensions to C++ through Clang plugin implementations. Such extension of the C++ language permits a dichotomy where code targeted for the quantum “accelerators” will be compiled following a specific procedure required to integrate well with the hardware of the accelerator, and “regular”/C++ code will be compiled the usual way. As described in the research document, program executions meant for the quantum accelerator are noted by the “\_\_qpu\_\_” tag at the beginning of the function declaration. It is important to note that every function for the accelerator must return “void” and can take up to any number of arguments, but it must at least have a quantum register “qreg” as part of the arguments to secure that the execution results are saved. By including “\_\_qpu\_\_” in the signature of the method the compiler performs a process of tokenization from which the resulting tokens are converted into an implementation of a class called QuantumKernel. Then, the code is translated into a LLVM IR (Intermediate Representation) by Clang, and finally qcor utilizes the implementations offered by XACC to optimize the resulting circuit in terms of depth, as well as exploiting the versatility of XACC to make the resulting compilation be runnable by different implementations of the accelerator, like D-Wave and IBM’s implementations. Another nice feature of qcore is that quantum code can be written in any quantum computing programming language. All of this is done through a series of compilation steps that translate code into a format, or that create implementations of compiler-defined classes which are then used as means to conform with the needed structures of code of Clang and XACC.

In the words of the authors, the fact that the qcor “is built upon the XACC framework, [means] it is well positioned to serve as an integration framework for state-of-the-art quantum compilation strategies coming from experts in the field.” Furthermore, qcor offers compilation flags that permit to define the accelerator that will be used, and it also permits to define different error/noise correction methodologies for the results after an execution. qcor also offers the nice feature that execution of binaries is similar to common C++ (i.e., ./program.out). The paper finalized by offering different examples of what type of works can be done with the qcor compiler.

I think this paper overall does a poor job at explaining so many dependencies that exist as the compilation process occurs. I felt most of the time lost understanding what was going on and why the translation had to be done, although, granted there were reasons stated along the reading. On the bright side, I think the tool, the compiler is of great value because it permits to do similar work to OpenMPI and OpenACC for parallelization. But in this case it allows to use the usual flags of optimization, target machine, etc. and the unique compiler like “acc” to produce runnable code. Overall, great in terms of results and contribution to the community.