Intel® Quantum SDK Getting Started

February 15, 2024

Release Version 1.1

Contents:

1.0 Getting Started

The Intel® Quantum SDK is a complete quantum computing stack using simulation. It gives developers the programming tools for developing applications that incorporate results from quantum algorithms.

Included are a quantum compiler, quantum runtime, language extensions, and a suite of qubit simulator backends (ranging from abstract to physics-based). The language extensions provide the data types to represent quantum resources and build powerful quantum expressions. The quantum compiler comprehends that logic, and optimizes the quantum instructions to reduce the number and complexity. The quantum runtime enables the dynamic flow of data between the qubit simulator backend and the C++ data structures. And the qubit simulators apply various simulation techniques to return a result from a given quantum system.

Users can find information here to: * Get familiar with new tools and quantum computing concepts. * Check on hardware requirements. * Learn about how to use the SDK.

1.1 System Requirements

The Intel® Quantum SDK has the following memory requirements:

1.1.1 Memory Requirements

The memory requirement of applications compiled using the Intel® Quantum SDK will be the sum of the traditional footprint created by the data structure in the C++ code and the memory required by the qubit simulator.

The backends utilizing state-vector simulations require an amount of memory which increases with the number of qubits. Representing n qubits requires 2^n complex numbers. Representing a complex `double` value requires 16 bytes, 2^4. So the memory required to simulate n qubits is

$$\text{memory} = 2^{(4+n)} \text{ bytes}$$

As an example, simulating different-sized qubit systems will require at least:

Table 1: Memory Requirements

qubits	memory
10	17 KB
20	17 MB
30	17.2 GB

Note that adding 1 additional qubit doubles the required memory.

1.2 How to Use?

In the simplest terms, using the Intel® Quantum SDK follows the pattern:

1. Write the quantum algorithm with a `.cpp` extension as a separate file. This file will include the `quantum_kernel` definitions, initialization for quantum hardware, and the logic for the quantum/classical algorithms (see **Writing New Algorithms** section for details). Example: `my_new_algo.cpp`
2. Build and Run. Invoke the `intel-quantum-compiler` to use the Intel® Quantum SDK:

   ```
   $ ./intel-quantum-compiler [compiler flags] new_algo_start_here.cpp
   ```

   To see a list of the flag options, run

   ```
   $ ./intel-quantum-compiler -h
   ```

   Run the executable produced by the compiler:

   ```
   $ ./new_algo_start_here
   ```

See Developers Guide and Reference (Known Limitations and Issues) for the limitations.

2.0 Writing New Algorithms

Writing a new algorithm is as easy as writing C++ code. A basic template and example file can be found in `new_algo_start_here.cpp`. For a new `.cpp` file, the only additions needed to access quantum-specific functionality are:

1. Add the `#include <clang/Quantum/quintrinsics.h>` header which defines the quantum gates listed in Developers Guide and Reference (Supported Quantum Logic Gates), and add the `#include <quantum_full_state_simulator_backend.h>` header which defines the `FullStateSimulator` class.
2. Declare the variable(s) that represent your quantum algorithm's qubits as globally defined `qbit` data types.
3. Before invoking any function containing quantum instructions, prepare the backend that will serve as the quantum hardware by instantiating an object of the desired class. The `FullStateSimulator` class provides access to both the Intel® Quantum Simulator and a Quantum Dot Simulator as backends. For more details on using the class methods, see Developers Guide and Reference (Configuring the FullStateSimulator), or see API Reference for a complete description of the class' methods. For details on using the simulators, see Developers Guide and Reference (Configuring the FullStateSimulator) or Developers Guide and Reference (Quantum Dot (QD) Simulator).
4. Place all calls to quantum gates inside a function or method, not directly in `main()`. Add the function specifier `quantum_kernel` to the definition of functions and methods including quantum gates or other `quantum_kernel` functions.

   These functions can declare and operate on classical (non-quantum) data types using typical scope rules. The compiler will move the classical calculation to have it available to a `quantum_kernel` while also preventing the classical instruction from being sent to quantum hardware. A `quantum_kernel` function can have parameters of `qbit` type as long as the logic ultimately resolves unambiguously to a globally defined `qbit` variable. See Developers Guide and Reference (In-lining & quantum_kernel functions) for a more detailed description.
5. Just like any C++ program, use the `int main()` function to run your primary computation. Conceptually, your program is a hybrid of traditional or "classical" components and quantum components. The quantum components are sent to the quantum backend when called in `main()`.

3.0 Next Steps

To read about individual components of the Intel® Quantum SDK, see the accompanying documents Developers Guide and Reference, Tutorials, FLEQ Guide and Reference and API Reference.

To explore examples included with the Intel® Quantum SDK, look in the `/<path to IQSDK>/quantum_examples/` and `/<path to IQSDK>/python-quantum-examples/` directories.

3.1 Fundamentals

To find out more about the core concepts of quantum computing, such as qubit mathematics or quantum circuit representations, we recommend using a textbook such as

1. "Quantum Computation and Quantum Information" by Nielsen and Chuang [NICH2010].
2. "Quantum Computing for Computer Scientists" by Yanofsky and Mannucci [YAMA2008].
3. "Quantum Computing: An Applied Approach" by Hidary [HIDA2019].

To brush up on C++, we recommend the latest edition of "Tour of C++" by Bjarne Stroustrup [STRO2022]. In our code, we strive to adhere to the LLVM style standard to simplify working with our APIs.

Bibliography

[NICH2010] M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition (Cambridge University Press, 2010). https://doi.org/10.1017/CBO9780511976667

[YAMA2008] Yanofsky, N., & Mannucci, M. (2008). Quantum Computing for Computer Scientists. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511813887

[HIDA2019] Hidary, J.D. (2019). Quantum Computing: An Applied Approach. Springer, Cham. https://doi.org/10.1007/978-3-030-23922-0

[STRO2022] Stroustrup, B. (2022). A Tour of C++. Addison-Wesley. https://www.stroustrup.com/tour3.html