Quantum Task Offloading with the OpenMP API

Joseph K. L. Lee (EPCC), Martin Ruefenacht (LRZ), Johannes Doerfert (LLNL), Oliver Thomson Brown (EPCC),
Mark Bull (EPCC), Michael Klemm (AMD & OpenMP ARB), Martin Schulz (TUM & LRZ)

Abstract

Most of the widely used quantum programming languages and libraries are not designed for the tightly coupled nature of hybrid quantum-classical algorithms, which run on quantum resources that are integrated on-premise with classical HPC infrastructure. We propose a programming model using the API provided by OpenMP to target quantum devices, which provides an easy-to-use and efficient interface for HPC applications to utilize quantum compute resources. We have implemented a variational quantum eigensolver using the programming model, which has been tested using a classical simulator. We are in the process of testing on the quantum resources hosted at the Leibniz Supercomputing Centre (LRZ).

The (Sad) Current State

- Quantum programs consist of sequence of gates, applied on single or multiple Qubits
- Currently most popular way to compile and program quantum circuits is via Python libraries
  - ➢ e.g., Qiskit, Cirq, Pennylane…

Qiskit Cirq PENNYLANE

- Combined with pay-as-you-go cloud access to quantum resources
  - ➢ Low adoption barrier
- Disadvantages:
  - ➢ High latency
  - ➢ Security
  - ➢ Python not optimal for classical HPC performance

Intermediate Representations

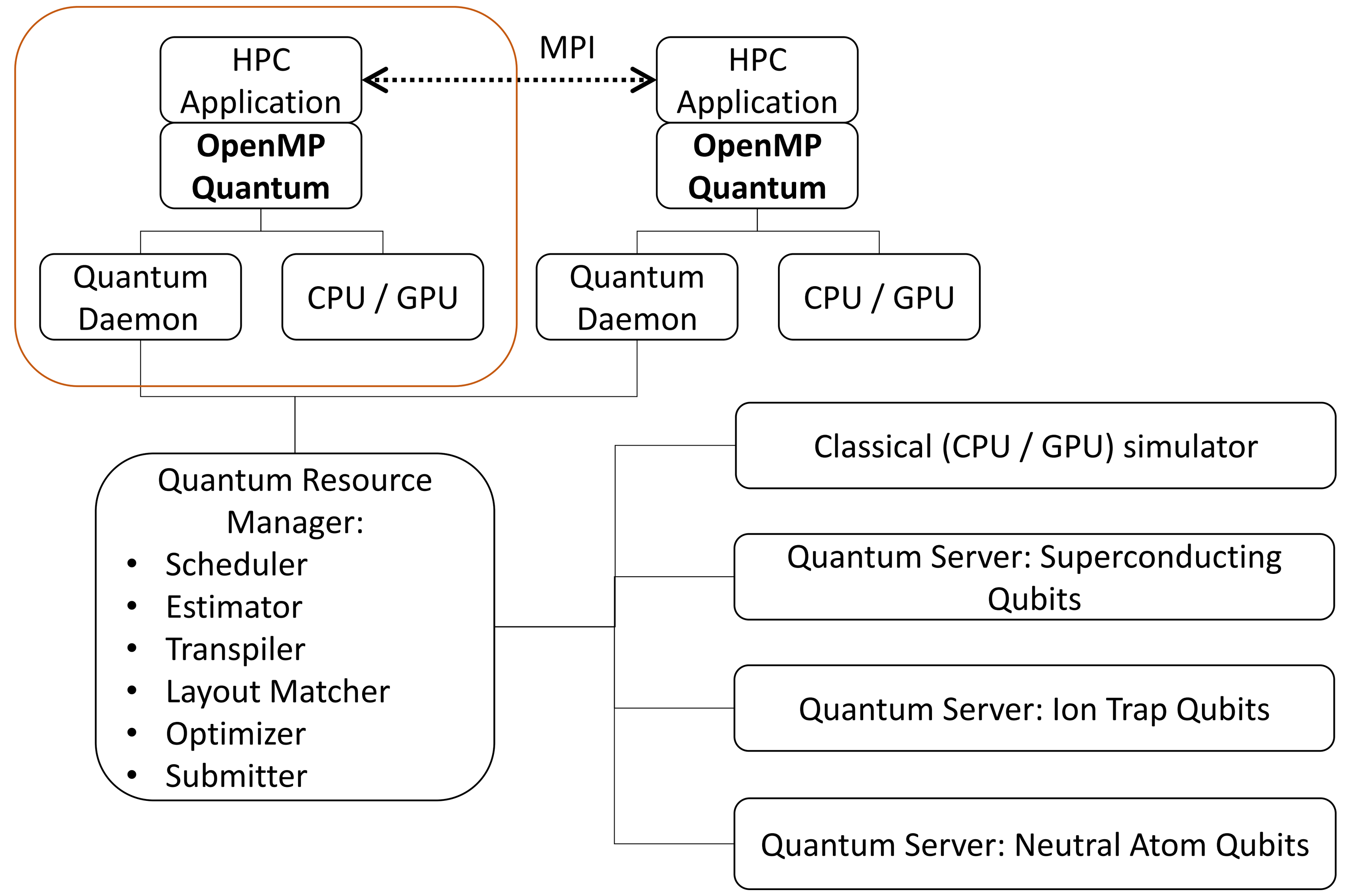
- Facilitate optimizations
- Used to target different hardware backends
  - ➢ OpenQASM: Assembly/C like language for describing quantum circuits
  - ➢ QIR: Based on LLVM IR

Contact

Joseph Lee, EPCC
j.lee@epcc.ed.ac.uk

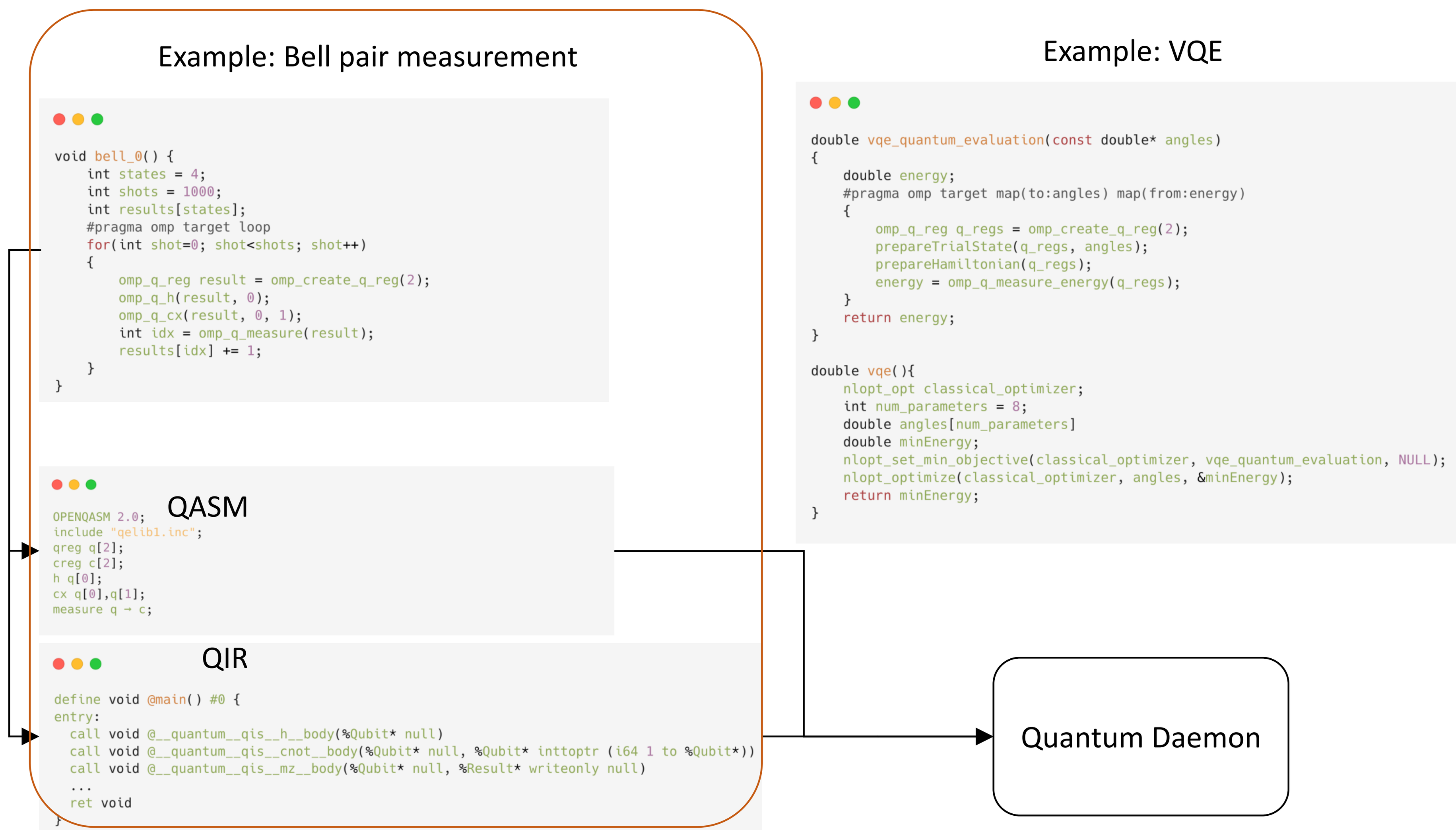
Hybrid Quantum-Classical Computing

- Hybrid algorithms are well suited to Noisy Intermediate-Scale Quantum (NISQ) computers
- Example: Variational Quantum Algorithms (VQAs) combine short-depth parametrized quantum circuits with classical variational loop
- Tightly coupled hybrid algorithms require low latency
- Classical components are computationally intensive
  - ➢ Requires classical HPC system
  - ➢ Benefit from optimizations from traditional compiled programs (usually in C/C++ or Fortran, and parallelized with MPI/OpenMP)
- Future systems (e.g. Euro-Q-Exa hosted by Leibniz Supercomputing Centre) provide on-premise integration
  - ➢ Quantum resources in close physical proximity to classical HPC infrastructure



OpenMP Quantum

- OpenMP provides flexible target interface
  - ➢ Send/receive data to target device
  - ➢ Execute control flow of target region
- Widely adopted for other accelerators, e.g., GPUs
- Implement quantum target offloading using the OpenMP API
- Transpile to OpenQASM or QIR
- Synchronising execution (default)
  - ➢ Suited to tightly coupled hybrid algorithm
- Asynchronous execution (via `nowait`)
  - ➢ Allow quantum resource manager allocation
- Implemented Variational Quantum Eigensolver (VQE) algorithm
  - ➢ Tested on classical simulator
  - ➢ Test on physical superconducting qubits at LRZ in progress



Example: Bell pair measurement

```
void bell_0() {
    int states = 4;
    int shots = 1000;
    int results[states];
    #pragma omp target loop
    for(int shot=0; shot<shots; shot++)
    {
        omp_q_reg result = omp_create_q_reg(2);
        omp_q_h(result, 0);
        omp_q_cx(result, 0, 1);
        int idx = omp_q_measure(result);
        results[idx] += 1;
    }
}
```

QASM

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
h q[0];
cx q[0],q[1];
measure q → c;
```

QIR

```
define void @main() #0 {
entry:
  call void @__quantum__qis__h__body(%Qubit* null)
  call void @__quantum__qis__cnot__body(%Qubit* null, %Qubit* inttoptr (i64 1 to %Qubit*))
  call void @__quantum__qis__mz__body(%Qubit* null, %Result* writeonly null)
  ...
  ret void
}
```

Example: VQE

```
double vqe_quantum_evaluation(const double* angles)
{
    double energy;
    #pragma omp target map(to:angles) map(from:energy)
    {
        omp_q_reg q_regs = omp_create_q_reg(2);
        prepareTrialState(q_regs, angles);
        prepareHamiltonian(q_regs);
        energy = omp_q_measure_energy(q_regs);
    }
    return energy;
}

double vqe(){
    nlopt_opt classical_optimizer;
    int num_parameters = 8;
    double angles[num_parameters]
    double minEnergy;
    nlopt_set_min_objective(classical_optimizer, vqe_quantum_evaluation, NULL);
    nlopt_optimize(classical_optimizer, angles, &minEnergy);
    return minEnergy;
}
```

Part of this research has been conducted as part of the Munich Quantum Valley (MQV), which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. Further funding comes from the German Federal Ministry of Education and Research (BMBF) through the project MuniQC-SC/ATOMS as well as Q-Exa.