QPack: An application-oriented benchmark for NISQ computers

TQCI seminar, 11 May 2023 Thales TRT – Palaiseau

Matthias Möller, Zaid Al-Ars, Koen Mesman, Huub Donkers
About

Matthias Möller
Associate Professor of Numerical Analysis
Department of Applied Mathematics

Zaid Al-Ars
Associate Professor at the Computer Engineering Lab
Department of Quantum & Computer Engineering

Koen Mesmann
PhD candidate – QAIMS lab

Huub Donkers
former MSc student – QCE
About

Vision
QPUs as special-purpose hardware accelerators in future high-performance computing systems

Research
NISQ & FTQ algorithms, high-level quantum programming SDKs, quantum benchmarks

TU Delft
QPack benchmark

Motivation

Variety of quantum benchmarks but lack of an accepted standard, e.g., "QLINPACK"

Goal

Unbiased, vendor- and qubit-technology-neutral benchmark for NISQ computers

Design criteria

- Hardware agnostic implementation (write-once-run-anywhere)
- Variety of application-oriented scalable test cases
- Multiple metrics – single score
LibKet

application developer

one-API high-level quantum programming SDKs for C, C++, and Python

algorithm developer

building block layer: quantum primitives, NISQ algorithms

"we"

abstraction layer: filters, gates, and device-specific features

embedded Python engine

C++ engine

Atos QLM  Cirq  IBM-Q  IonQ  Rigetti  YOURS?  OpenGL  QuEST  QX  YOURS?
LibKet – by example

#include <LibKet.hpp>

using namespace LibKet;
using namespace LibKet::circuits;

auto expr = measure(qft(init()));

try {
    QDevice<QDeviceType::qi_26_simulator, 6> qpu; qpu(expr);
    auto result = qpu.eval(1024);

    QInfo << result << std::endl;
    QInfo << "job ID : " << qpu.get<QResultType::id>(result) << std::endl;
    QInfo << "best : " << qpu.get<QResultType::best>(result) << std::endl;
    QInfo << "histogram : " << qpu.get<QResultType::histogram>(result) << std::endl;
} catch(const std::exception &e) {
    QWarn << e.what() << std::endl;
}
LibKet – by example

#include <LibKet.hpp>

using namespace LibKet;
using namespace LibKet::circuits;

auto expr = measure(qft(init()));

try {
    QDevice<QDeviceType::qi_26_simulator, 6> qpu; qpu(expr);
    auto result = qpu.eval(1024);

    QInfo << result << std::endl;
    QInfo << "job ID : " << qpu.get<QResultType::id>(result) << std::endl;
    QInfo << "best : " << qpu.get<QResultType::best>(result) << std::endl;
    QInfo << "histogram : " << qpu.get<QResultType::histogram>(result) << std::endl;
} catch(const std::exception &e) {
    QWarn << e.what() << std::endl;
}
LibKet – by example

```cpp
#include <LibKet.hpp>

using namespace LibKet;
using namespace LibKet::circuits;
auto expr = measure(qft(init()));

try {
    QDevice<QDeviceType::qi_26_simulator, 6> qpu; qpu(expr);
    auto result = qpu.eval(1024);
    QInfo << result << std::endl;
    QInfo << "job ID : " << qpu.get<QResultType::id>(result) << std::endl;
    QInfo << "best : " << qpu.get<QResultType::best>(result) << std::endl;
    QInfo << "histogram : " << qpu.get<QResultType::histogram>(result) << std::endl;
} catch(const std::exception &e) {
    QWarn << e.what() << std::endl;
}
```

Create 6-qubit device on the 26-qubit QI simulator and upload the expression
LibKet – by example

#include <LibKet.hpp>

using namespace LibKet;
using namespace LibKet::circuits;

auto expr = measure(qft(init()));

try {
    QDevice<QDeviceType::qi_26_simulator, 6> qpu; qpu(expr);
    auto result = qpu.eval(1024);

    QInfo << result << std::endl;
    QInfo << "job ID : " << qpu.get<QResultType::id>(result) << std::endl;
    QInfo << "best : " << qpu.get<QResultType::best>(result) << std::endl;
    QInfo << "histogram : " << qpu.get<QResultType::histogram>(result) << std::endl;
} catch(const std::exception &e) {
    QWarn << e.what() << std::endl;
}
LibKet – by example

Views

auto expr = all(qft(sel<0,3,4,6>(...)));

Switch to another device

QDevice<QDeviceType::ibmq_seattle, 433> qpu;

Non-blocking execution

auto job = qpu.execute_async(1024);
while (!job->query()) {
    // do something else
}
auto result = job->get();

Manual coding

QProgram prog;

prog.rx  ( 3.141, {0,1,2} );
prog.h   ( {0,1,2} );
prog.h   ( 3 );
prog.rx  ( 3.141, {3,4,5} );
prog.cnot ( {3,4,5}, {6,7,8} );
prog.measure ( {0,1,2,3,4,5,6,7,8} );
qpu(prog.to_string());

Advanced features

- CUDA-like streams
- JIT-compilation of quantum expressions
- Rule-based optimization
QPack: application-oriented scalable test cases

Quantum Approximate Optimization Algorithm (QAOA)
- Max-cut problem (MCP)
- Dominating set problem (DSP)
- Travelling salesperson problem (TSP)
- Maximum independent set problem (MIS)

Variational Quantum Eigensolver (VQE)
- Random diagonal Hamiltonian (RH)
- Transverse Ising chain (IC)

QPack: application-oriented scalable test cases

Quantum Approximate Optimization Algorithm (QAOA)
- Max-cut problem (MCP)
- Dominating set problem (DSP)
- Travelling salesperson problem (TSP)
- Maximum independent set problem (MIS)

Variational Quantum Eigensolver (VQE)
- Random diagonal Hamiltonian (RH)
- Transverse Ising chain (IC)
QPack: multiple metrics

QPack: multiple metrics

Capacity
- maximal number of qubits for which QPU achieves prescribed relative error relative to QuEST simulator

Scalability
- power law fitting $T_q = (\text{problem size})^\alpha$

Accuracy
- average relative error between expectation value of ideal simulator (QuEST) and QPU under testing

Runtime
- average #gates per second over all problem sizes

---

QPack: single score

QPack: preliminary result on remote simulators

QPack: preliminary result on hardware QPU

Summary and outlook

▪ QPack is an application-oriented scalable benchmark for NISQ computers
▪ Extension of test suite and benchmarking of other QPUs is ongoing (support & access is welcome!)

References


Thank you!