## CONTENTS

1 Intel Quantum Simulator .......................... 3
   1.1 Build instructions .................................. 3
   1.2 Docker: build image and run/execute container .......... 5
   1.3 Getting started with Intel-QS ........................ 5
   1.4 How to contribute ................................... 6
   1.5 How to contact us ................................. 6
   1.6 How to cite ........................................ 6

2 Getting started with Intel Quantum Simulator: Examples .... 7
   2.1 Import Intel QS library ............................ 7
   2.2 Example 1 .......................................... 7
   2.3 Example 2 .......................................... 9

3 Example using the extra features for QAOA circuits ......... 13
   3.1 Import Intel QS library ............................ 13
   3.2 Initialize the Max-Cut problem instance via its adjacency matrix ........................ 13
   3.3 Collect the results and visualize them in a histogram .................. 15
   3.4 Simple test ........................................ 16

4 Contributing ............................................. 19

5 API References .......................................... 21
   5.1 Class Hierarchy ..................................... 21
   5.2 File Hierarchy ..................................... 21
   5.3 Full API ........................................... 21

6 Indices and tables ..................................... 61

Index ......................................................... 63
Intel Quantum Simulator (Intel-QS), also known as qHiPSTER (The Quantum High Performance Software Testing Environment), is a simulator of quantum circuits optimized to take maximum advantage of multi-core and multi-nodes architectures. It is based on a complete representation of the qubit state, but avoids the explicit representation of gates and other quantum operations in terms of matrices. Intel-QS uses MPI (message-passing-interface) protocols to handle the communication between distributed resources that are used to store and manipulate the quantum state.

1.1 Build instructions

Intel-QS builds as a shared library which, once linked to the application program, allows to take advantage of the high-performance implementation of circuit simulations. The library can be built on a variety of different systems, from laptop to HPC server systems.

The directory structure of the repository can be found in intel-qs/docs/directory_structure.md.

The library object is: /builb/lib/libiqs.so

1.1.1 Requirements

The following packages are required by the installation:

- CMake tools version 3.12+
- MPICH3 library for enabling the distributed communication
- optional: MKL for distributed random number generation
- optional: PyBind11 (installed via conda, not pip) required by the Python bunding of Intel-QS

The first step is cloning the repository:

```bash
git clone https://github.com/iqusoft/intel-qs.git
cd intel-qs
```
1.1.2 Use Intel Parallel Studio compilers to build Intel-QS

If you wish to build Intel-QS using the latest Intel compiler technologies, then you need to configure your environment properly according to that tool’s documentation. Assuming that you have installed Intel Parallel Studio in the standard location on your system, you should invoke the following scripts through the source command on Linux.

```
source /opt/intel/bin/compilervars.sh -arch intel64 -platform linux
source /opt/intel/compiler_and_libraries/linux/mpi/intel64/bin/mpivars.sh
```

Now, use CMake to generate the appropriate makefiles to use the Intel Parallel Studio compilers. The installation follows the out-of-source building and requires the creation of the directory build. This directory is used to collect all the files generated during the installation process.

```
mkdir build
cd build
CXX=mpiicpc cmake -DIqsMPI=ON -DIqsUtest=ON ..
make
```

By default, MKL is required when Intel compilers are used.

To re-build Intel-QS with different settings or options, we recommend to delete all content of the build directory and then restart from the CMake command.

1.1.3 Use standard GNU tools to build Intel-QS

If you wish to build Intel-QS using only standard GNU compilers type:

```
mkdir build
cd build
CXX=g++ cmake -DIqsMPI=OFF ..
make
```

By default, MKL is not required when GNU compilers are used. Optionally, MPI can be included by setting the option -DIqsMPI=ON instead. You must ensure that you have at least version 3.1 of MPICH installed for the build to succeed. https://www.mpich.org

1.1.4 Enable MPI protocol for distributed memory use

The above installation enables MPI functionalities to deploy Intel-QS on High Performance Computing and Cloud Computing infrastructures. There is the option of disabling MPI: simply set the CMake option selection to -DIqsMPI=OFF (or just omit the option selection since MPI is disabled by default in the CMake build).

1.1.5 Enable Latest Vector Capability

To compile with the latest instruction set supported by your architecture, there is the option -DIqsNative. Compiled with -DIqsNative=ON, the latest vector instructions available on your machine, e.g. AVX2, AVX512, are used. By default, -DIqsNative=OFF.

If the machine you compile and the machine you run have different vector capabilities, turning on IqsNative=ON might cause run-time problems.

Underneath, this option uses -xhost with Intel compilers and -march=native with GNU compilers.
1.1.6 Enable Python binding (only available without MPI)

By default, whenever MPI is disabled, the building process includes the Python binding for Intel-QS. The binding code uses the Pybind11 library which needs to be installed via ‘conda’ (and not simply with pip) to include the relevant information in CMake. See this page for more info on this issue.

To disable the Python wrap, even without MPI, set the CMake option selection to -DIqsPython=OFF.

1.1.7 Unit test

By default, with MPI either enabled or disabled, the building process includes a suite of unit tests written in the googletest framework. Following the recommended integration, the CMake building process automatically downloads the up-to-date repository of gtest and installs it in the build path.

To disable the unit tests, set the CMake option selection to -DIqsUtest=OFF.

To run the unit tests, from /build launch the executable ./bin/utest.

1.1.8 Recommended build for HPC.

The recommended building process requires Intel Math Kernel Library and the MPI-ICPC compiler.

When the program is run in hybrid configuration (OpenMP+MPI), we recommend to manage the OpenMP affinity directly. Affinity settings can be set using the syntax: KMP_AFFINITY=compact,1,0,granularity=fine. A quick look at the options can be found at this page.

1.2 Docker: build image and run/execute container

Dockerfile includes the instructions to build the docker image of an Ubuntu machine with Intel-QS already installed. The image can be ‘run’ to create a container. The container can be ‘executed’ to login into the machine.

```
docker build -t qhipster .
docker run -d -t qhipster
docker ps
docker exec -itd <container_id> /bin/bash
```

If Docker is used on a Windows host machine, the last line should be substituted by: winpty docker exec -itd <container_id> //bin/bash.

1.3 Getting started with Intel-QS

The simplest way of familiarize with the Intel Quantum Simulator is by exploring the tutorials provided in the directory tutorials/. In particular, the code tutorials/get_started_with_IQS.cpp provides step-by-step description of the main commands to: define a qubit register object, perform quantum gates, measure one or multiple qubits.

If the Python bindings were enabled, the same learning can be performed using the iPython notebook tutorials/get_started_with_IQS.ipynb.
1.4 How to contribute

Thanks for your interest in the project! We welcome pull requests from developers of all skill levels. If you would like to contribute to Intel-QS, please take a look to our contributing policy and also to the code of conduct. For any bug, we use GitHub issues. Please submit your request there.

1.5 How to contact us

If you have a question or want to discuss something, feel free to send an email to Justin Hogaboam, Gian Giacomo Guerreschi, or to Fabio Baruffa.

1.6 How to cite

When using Intel Quantum Simulator for research projects, please cite:


The original implementation is described here:

Tutorial on the basic use of Intel QS through its Python interface: Two examples are provided.

**NOTE**: Currently, the Python implementation only allows for single-core execution and does not take advantages of the MPI protocol. However the user can familiarize with the same functionalities available in the distributed implementation (only C++ at the moment) and the transition should be relatively straightforward since all methods maintain name and effect.

## 2.1 Import Intel QS library

Let’s start by importing the Python library with the class and methods defined in the C++ implementation.

```python
# Import the Python library with the C++ class and methods of Intel Quantum Simulator.
# If the library is not contained in the same folder of this notebook, its path has to be added.
import sys
sys.path.insert(0, '../lib_python')
import intelqs as simulator

# Import NumPy library with Intel specialization.
import numpy as np
from numpy import random_intel

# Import graphical library for plots.
import matplotlib.pyplot as plt
```

## 2.2 Example 1

Create the state of a quantum register, having \( N > 3 \) qubits.

The state is initialized as a computational basis state (using the keyword “base”) corresponding to the index 0.

The index corresponds to a \( N \)-bit integer in decimal representation. With \( N \) qubits there are \( 2^N \) indices, from 0 to \( 2^N-1 \).
Let us allocate memory for the quantum register's state and initialize it to $|000...0\rangle$.

```python
num_qubits = 4;
if num_qubits<3:
    num_qubits = 4;
psi1 = simulator.QubitRegister(num_qubits, "base", 0, 0);
```

Let us apply a X Pauli gate on qubit 0, effectively flipping it from $|0\rangle$ to $|1\rangle$, followed by the Hadamard gate on all other qubits.

```python
psi1.ApplyPauliX(0);
for q in range(1,num_qubits):
    psi1.ApplyHadamard(q);
```

In addition to one-qubit gates, universal quantum computation can be achieved via 2-qubit entangling gates. For example, we now apply a CNOT between qubit 2 (here the control qubit) and qubit 1 (target qubit).

```python
control = 2;
target = 1;
psi1.ApplyCPauliX(control, target);
```

To extract information from the quantum register, one can obtain the probability of measuring a certain qubit in the computational basis and obtaining the outcome “1” (meaning that the state is in $|1\rangle$). In this example we measure qubit 1. Once the probability is known, one can draw a random number to simulate the stochastic outcome of the measurement and collapse the wavefunction accordingly.

```python
# Compute the probability of qubit 1 being in state $|1\rangle$.
measured_qubit = 1;
prob = psi1.GetProbability(measured_qubit);

print("Probability that qubit {}, if measured, is in state $|1\rangle$ = {}\n".format(measured_qubit, prob));

# Draw random number in [0,1)
r = np.random.rand()
if r < prob:
    # Collapse the wavefunction according to qubit 1 being in $|1\rangle$.
    print("Simulated outcome is 1. Collapse the function accordingly."")
    psi1.CollapseQubit(measured_qubit,True);
else:
    # Collapse the wavefunction according to qubit 1 being in $|0\rangle$
    print("Simulated outcome is 0. Collapse the function accordingly."")
    psi1.CollapseQubit(measured_qubit,False);

# In both cases one needs to re-normalize the wavefunction:
psi1.Normalize();
```

Probability that qubit 1, if measured, is in state $|1\rangle = 0.0$

Simulated outcome is 0. Collapse the function accordingly.
2.3 Example 2

Create the state of a quantum register, having \( N > 3 \) qubits.

The state is initialized as a random state (using the keyword “rand”):

This requires a random number generator (RNG), that we initialize just before the second register. Notice that ‘777’ plays the role of the seed to initialize the RNG.

```python
[8]: num_qubits = 4;
    if num_qubits<3:
        num_qubits = 4;
    psi2 = simulator.QubitRegister(num_qubits, "rand", 777, 0);
```

Let us apply one- and two-qubit gates as in the previous example.

```python
[9]: # Let us apply a X Pauli gate on qubit 0, effectively flipping it from |0> to |1>.
    psi2.ApplyPauliX(0);

    # Let us apply an Hadamard gate on all other qubits.
    for q in range(1,num_qubits):
        psi2.ApplyHadamard(q);
```

One can define an arbitrary single-qubit gate and apply it to the chosen qubit.

In addition one can apply a custom one-qubit gate conditionally on the state of a control qubit.

```python
[10]: # Define an arbitrary single qubit gate and apply it to the chosen qubit.
    # The quantum gate G is given by a 2x2 unitary matrix, here using a bi-dimensional NumPy array.
    G = np.zeros((2,2),dtype=np.complex_);
    G[0,0] = 0.592056606032915 + 0.459533060553574j;
    G[0,1] = -0.314948020757856 - 0.582328159830658j;
    G[1,0] = 0.658235557641767 + 0.070882241549507j;
    G[1,1] = 0.649564427121402 + 0.373855203932477j;

    qubit = 0;
    psi2.Apply1QubitGate(qubit,G);

    # It is also possible to apply the arbitrary gate specified by G controlled on the state of another qubit.
    # G is applied conditioned on the control qubit being in |1>.
    control = 1;
    target = 2;
    psi2.ApplyControlled1QubitGate( control, target, G);

    # Notice that this output is directed to the terminal and not re-directed to the iPython notebook.
    psi2.Print("After all gates.")
```

To extract information from the quantum register, one can obtain the expectation value of Pauli strings.

For example, consider the Pauli string given by:

\[
X_0 \otimes id_1 \otimes Z_2 \otimes Y_3
\]

Such observable is defined by: - the position of the non-trivial Pauli matrices, in this case \( \{0,2,3\} \) - the corresponding Pauli matrices (\(X=1, Y=2, Z=3\)).
To facilitate the verification of the expectation value, we reinitialize the quantum state to $|+01\rangle$.

We also consider the Pauli string

$$X_0 \otimes id_1 \otimes Z_2 \otimes Z_3$$

[12]: # Prepare the state $|+01\rangle$
index = 2+8;
psi2.Initialize("base",index);
# Notice that GetProbability() does not change the state.
for qubit in range(0,num_qubits):
    prob = psi2.GetProbability( qubit );
    print("Probability that qubit {}, if measured, is in state \(|1\rangle = \{|\n".format(qubit, prob));

psi2.ApplyHadamard(0);
psi2.ApplyHadamard(1);

# The Pauli string given by: $X_0 \cdot id_1 \cdot Z_2 \cdot Y_3$
# Such observable is defined by the position of the non-trivial Pauli matrices:
qubits_to_be_measured = [0,2,3]
# And by the corresponding Pauli matrices ($X=1, Y=2, Z=3$)
observables = [1,3,2]

# The expectation value $\langle psi2|X_0.id_1.Z_2.Y_3|psi2\rangle$ is obtained via:
average = psi2.ExpectationValue(qubits_to_be_measured, observables, 1.);print("Expectation value $\langle+01|X_0.id_1.Z_2.Y_3|+-01\rangle = \{|\n".format(average));

# The expectation value $\langle psi2|X_0.id_1.Z_2.Y_3|psi2\rangle$ is obtained via:
qubits_to_be_measured = [0,2,3]
observables = [1,3,3]
average = psi2.ExpectationValue(qubits_to_be_measured, observables, 1.);
print("Expectation value $\langle+01|X_0.id_1.Z_2.Z_3|+-01\rangle = \{|\n".format(average));

# Trivial expectation:
average = psi2.ExpectationValue([0],[1], 1.);
print("Expectation value $\langle+01|X_0|+-01\rangle = \{|\n".format(average));

# The expectation value $\langle psi2|X_0.id_1.id_2.Z_3|psi2\rangle$ is obtained via:
average = psi2.ExpectationValue([0,3],[1,3], 1.);
print("Expectation value $\langle+01|X_0.Z_3|+-01\rangle = \{|\n".format(average));

Probability that qubit 0, if measured, is in state $|1\rangle = 0.0$
Probability that qubit 1, if measured, is in state $|1\rangle = 1.0$
Probability that qubit 2, if measured, is in state $|1\rangle = 0.0$
Probability that qubit 3, if measured, is in state $|1\rangle = 1.0$
Expectation value $\langle+01|X_0.id_1.Z_2.Y_3|+-01\rangle = 0.0$
Expectation value $\langle+01|X_0.id_1.Z_2.Z_3|+-01\rangle = -0.9999999999999989$
Expectation value $\langle+01|X_0|+-01\rangle = 1.0$

(continues on next page)
Expectation value $\langle+01|X_0.Z_3|+01\rangle = -1.0000000000000004$

[13]:

# Prepare the state $|+01\rangle$
index = 2+8;
psi2.Initialize("base",index);
# Notice that GetProbability() does not change the state.
for qubit in range(0,num_qubits):
    prob = psi2.GetProbability(qubit);
    print("Probability that qubit {}, if measured, is in state $|1\rangle = {}$\".
          format(qubit, prob));

psi2.ApplyHadamard(0);
psi2.ApplyHadamard(1);
# The Pauli string given by: $X_0 . id_1 . Z_2 . Y_3$
# Such observable is defined by the position of the non-trivial Pauli matrices:
qubits_to_be_measured = [0,2,3]
# And by the corresponding Pauli matrices ($X=1$, $Y=2$, $Z=3$)
observables = [1,3,2]
# The expectation value $\langle psi2|X_0.id_1.Z_2.Y_3|psi2\rangle$ is obtained via:
average = psi2.ExpectationValue(qubits_to_be_measured, observables, 1.);
print("Expectation value $\langle+01|X_0.id_1.Z_2.Y_3|+01\rangle = {}$\".format(average));

# The expectation value $\langle psi2|X_0.id_1.Z_2.Y_3|psi2\rangle$ is obtained via:
qubits_to_be_measured = [0,2,3]
observables = [1,3,3]
average = psi2.ExpectationValue(qubits_to_be_measured, observables, 1.);
print("Expectation value $\langle+01|X_0.id_1.Z_2.Z_3|+01\rangle = {}$\".format(average));

Probability that qubit 0, if measured, is in state $|1\rangle = 0.0$
Probability that qubit 1, if measured, is in state $|1\rangle = 1.0$
Probability that qubit 2, if measured, is in state $|1\rangle = 0.0$
Probability that qubit 3, if measured, is in state $|1\rangle = 1.0$
Expectation value $\langle+01|X_0.id_1.Z_2.Y_3|+01\rangle = 0.0$
Expectation value $\langle+01|X_0.id_1.Z_2.Z_3|+01\rangle = -0.9999999999999989$

[14]:

# Extra expectation values.
# Prepare the state $|+01\rangle$
index = 2+8;
psi2.Initialize("base",index);
psi2.ApplyHadamard(0);
psi2.ApplyHadamard(1);
# The expectation value of $X_0$:
average = psi2.ExpectationValue([0],[1], 1.);
print("Expectation value $\langle+01|X_0|+01\rangle = {}$\".format(average));
The expectation value of $X_0.Z_3$:
\[
\text{average} = \text{psi2.ExpectationValue}([0,3],[1,3], 1.);
\]
\[
\text{print}("\text{Expectation value } <+01|X_0.Z_3|+-01> = \{\text{average}\}\)."
\]

The expectation value of $X_0.Z_2$:
\[
\text{average} = \text{psi2.ExpectationValue}([0,2],[1,3], 1.);
\]
\[
\text{print}("\text{Expectation value } <+01|X_0.Z_2|+-01> = \{\text{average}\}\)."
\]

The expectation value of $X_1.Z_2$:
\[
\text{average} = \text{psi2.ExpectationValue}([1,2],[1,3], 1.);
\]
\[
\text{print}("\text{Expectation value } <+01|X_1.Z_2|+-01> = \{\text{average}\}\)."
\]

Expectation value $<+01|X_0|+-01> = 1.0$

Expectation value $<+01|X_0.Z_3|+-01> = -1.0000000000000002$

Expectation value $<+01|X_0.Z_2|+-01> = 0.999999999999998$

Expectation value $<+01|X_1.Z_2|+-01> = -1.0000000000000002$
EXAMPLE USING THE EXTRA FEATURES FOR QAOA CIRCUITS

The Quantum Approximate Optimization Algorithm (QAOA) is a variational algorithm to solve combinatorial problems. Here we provide the syntax to quickly define and simulate QAOA circuits.

As a concrete example, we consider the MaxCut problem on a linear graph of 6 vertices. It is trivially solved analytically, but the numerical procedure extends to more complicated instances.

NOTE: Currently, the Python implementation only allows for single-core execution and does not take advantages of the MPI protocol.

3.1 Import Intel QS library

We start by importing the Python library with the class and methods defined in the C++ implementation.

```python
# Import the Python library with the C++ class and methods of Intel Quantum Simulator.
# If the library is not contained in the same folder of this notebook, its path has
# to be added.
import sys
sys.path.insert(0, '../build/lib')
import intelqs_py as simulator

# Import NumPy library with Intel specialization.
import numpy as np
from numpy import random_intel

# Import graphical library for plots.
import matplotlib.pyplot as plt
```

3.2 Initialize the Max-Cut problem instance via its adjacency matrix

Specific instance: 0 – 1 – 2 – 3 – 4 – 5

We describe the instance by its adjacency matrix $A$, represented as a bidimensional NumPy array.

Each of the $2^6$ bipartitions of the 6 vertices is associated with a cut value (the number of edges connecting vertices of different color).

```python
# Number of vertices.
num_vertices = 6;
# Adjacency matrix.
A = np.zeros((num_vertices,num_vertices),dtype=np.int32);
```

(continues on next page)
Since A is sparse, fill it element by element.
A[0,1] = 1;
A[1,0] = 1;
A[1,2] = 1;
A[2,1] = 1;
A[2,3] = 1;
A[3,2] = 1;
A[3,4] = 1;
A[4,3] = 1;
A[4,5] = 1;
A[5,4] = 1;
print("The adjacency matrix of the graph is:
")
print(A)
#list(A.flatten())

# Allocate memory for the diagonal of the objective function.
diag_cuts = simulator.QubitRegister(num_vertices, "base", 0, 0);
max_cut = simulator.InitializeVectorAsMaxCutCostFunction( diag_cuts, list(A.flatten()) );
print("The max value of the cut is : {0:2d}".format(max_cut))

The adjacency matrix of the graph is:

```
[[0 1 0 0 0 0]
 [1 0 1 0 0 0]
 [0 1 0 1 0 0]
 [0 0 1 0 1 0]
 [0 0 1 0 1 0]
 [0 0 0 1 0 0]]
```

The max value of the cut is : 5

### Implement a p=2 QAOA circuit

- initialize the state in |000000⟩
- prepare the state in |+++++++⟩
- iterate through the QAOA steps (here p=2)
- each step is composed by the global operation defined by the cost function C and the transverse field mixing

```python
# Number of qubits.
num_qubits = num_vertices;
# Allocate memory for the quantum register's state and initialize it to |000000⟩.
psi = simulator.QubitRegister(num_qubits, "base", 0, 0);

# Prepare state |+++++++⟩
for qubit in range(num_qubits):
    psi.ApplyHadamard(qubit);

# QAOA circuit:
qaoa_depth = 2;
# Random choice of QAOA parameters.
np.random.seed(7777);
```

(continues on next page)
gamma = np.random.random_sample((qaoa_depth,))*3.14159;
beta = np.random.random_sample((qaoa_depth,))*3.14159;

for p in range(qaoa_depth):
    # exp(-i gamma C)
    simulator.ImplementQaoaLayerBasedOnCostFunction(psi, diag_cuts, gamma[p]);
    # exp(-i beta B)
    for qubit in range(num_qubits):
        psi.ApplyRotationX(qubit,beta[p]);

# At this point |psi> corresponds to the state at the end of the QAOA circuit.

3.3 Collect the results and visualize them in a histogram

histo = simulator.GetHistogramFromCostFunction(psi, diag_cuts, max_cut);
print("The probabilities of the cut values are:")
for c in range(max_cut+1):
    print("cut={0:2d} : {1:1.4f}".format(c,histo[c]))

# Plot histogram.
x = np.arange(max_cut+1)
fig = plt.bar(x, histo, align='center', alpha=0.5)
plt.xticks(x)
plt.xlabel('cut value')
plt.ylabel('probability of cut')
plt.title('Summary of results')
plt.show()

The probabilities of the cut values are:
cut= 0 : 0.0397
cut= 1 : 0.1778
cut= 2 : 0.3483
cut= 3 : 0.2986
cut= 4 : 0.1120
cut= 5 : 0.0236
3.4 Simple test

This instance represents a disconnected graph with only two edges, namely: 0 – 1 2 3 4 – 5

We describe the instance by its adjacency matrix \( A \), represented as a bidimensional NumPy array.

Each of the \( 2^6 \) bipartitions of the 6 vertices is associated with a cut value (the number of edges connecting vertices of different color). For Half of the bipartitions the 0–1 edge can be cut and for half of the bipartitions, independently of the previous consideration, the 5–6 edge can be cut. The histogram has three bins (cut values = \{0,1,2\}) and ratio 1:2:1.

```python
[5]: # Number of vertices.
num_vertices = 6;
# Adjacency matrix.
A = np.zeros((num_vertices,num_vertices),dtype=np.int32);
# Since A is sparse, fill it element by element.
A[0,1] = 1;
A[1,0] = 1;
A[num_vertices-2,num_vertices-1] = 1;
A[num_vertices-1,num_vertices-2] = 1;

# Allocate memory for the diagonal of the objective function.
diag_cuts = simulator.QubitRegister(num_vertices, "base", 0, 0);
max_cut = simulator.InitializeVectorAsMaxCutCostFunction( diag_cuts, list(A.ravel()) );

# Number of qubits.
num_qubits = num_vertices;
# Allocate memory for the quantum register's state and initialize it to |000000>.
psi = simulator.QubitRegister(num_qubits, "base", 0, 0);

# Prepare state |+++++++>
for qubit in range(num_qubits):
    psi.ApplyHadamard(qubit);

# The form of the histogram has been discussed privately.
histo = simulator.GetHistogramFromCostFunction(psi, diag_cuts, max_cut);
print(histo)

# Plot histogram.
x = np.arange(max_cut+1)
fig = plt.bar(x, histo, align='center', alpha=0.5)
#plt.xticks(x)
plt.xlabel('cut value')
plt.ylabel('probability of cut')
#plt.title('Summary of results')
plt.show()
[0.2499999999999999, 0.4999999999999999, 0.2499999999999999]
```
3.4. Simple test
Thanks for your interest in the project!

Any contribution, project participation and pull request from developers are welcome. Please follow this process:

- Clone this repository or fork it. Create a new branch and add your modifications.
- Remember to add unit tests to verify the desired behavior of the new features or methods.
- When the first version of the code is ready, create a pull-request to the development branch of iqusoft/intel-qs.
- Edits may be required to resolve conflicts since it is possible that the repository has changed while you worked on your new contribution. Please resolve the merge conflicts.
- Verify that all unit tests, and not only the one added in the contribution, run correctly.
- The IQS team will help with the revision of the pull-request.

To facilitate the review consider starting with small contributions (a couple files and ~100 lines). For contributions that require substantial changes or changes in many files, please contact the IQS team to discuss the most effective strategy.

Creating your pull request from a fork? We suggest allowing edits from maintainers. Then, anyone with Write access to the upstream repository will be able to add commits to your branch. This can make the review process easier for maintainers since they can make a small change themselves instead of asking you to make the change.

If you would like to contribute to IQS, please visit our wiki page https://github.com/iqusoft/intel-qs/wiki/Contribute.
CHAPTER
FIVE

API REFERENCES

5.1 Class Hierarchy

5.2 File Hierarchy

5.3 Full API

5.3.1 Namespaces

Namespace qaoa

Contents

• Functions

Functions

• Template Function qaoa::GetExpectationValueFromCostFunction
• Template Function qaoa::GetExpectationValueSquaredFromCostFunction
• Template Function qaoa::GetHistogramFromCostFunction
• Template Function qaoa::GetHistogramFromCostFunctionWithWeightsBinned
• Template Function qaoa::GetHistogramFromCostFunctionWithWeightsRounded
• Template Function qaoa::ImplementQaoaLayerBasedOnCostFunction
• Template Function qaoa::InitializeVectorAsMaxCutCostFunction
• Template Function qaoa::InitializeVectorAsWeightedMaxCutCostFunction
Namespace qhipster

Contents

• Namespaces
• Classes
• Functions

Namespaces

• Namespace qhipster::detail
• Namespace qhipster::mpi

Classes

• Template Struct AlignedAllocator::rebind
• Template Class AlignedAllocator
• Template Class RandomNumberGenerator
• Template Class TinyMatrix

Functions

• Template Function qhipster::floor_power_of_two
• Template Function qhipster::highestBit
• Template Function qhipster::ilog2
• Template Function qhipster::isPowerOf2
• Function qhipster::popcnt(uint64_t)
• Function qhipster::popcnt(uint32_t)
• Template Function qhipster::ShuffleFisherYates
• Template Function qhipster::toString
• Function qhipster::WhatCompileDefinitions
Namespace qhipster::detail

Contents

• Functions

Functions

• Function qhipster::detail::BX
• Template Function qhipster::detail::highestBitImpl

Namespace qhipster::mpi

Contents

• Classes
• Functions

Classes

• Class Environment
• Class Exception

Functions

• Function qhipster::mpi::Barrier
• Function qhipster::mpi::PoolBarrier
• Function qhipster::mpi::PoolPrint
• Function qhipster::mpi::Print
• Function qhipster::mpi::StateBarrier
• Function qhipster::mpi::StatePrint

Namespace std

5.3.2 Classes and Structs

Template Struct extract_value_type

• Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_qreg.hpp

5.3. Full API 23
Struct Documentation

template<typename T>
struct extract_value_type

Public Types

typedef T value_type

Template Struct extract_value_type<X T>

• Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_qureg.hpp

Struct Documentation

template<template<typename> class X, typename T>
struct extract_value_type<X<T>>

Public Types

typedef T value_type

Template Struct AlignedAllocator::rebind

• Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_alignedallocator.hpp

Nested Relationships

This struct is a nested type of Template Class AlignedAllocator.

Struct Documentation

template<typename U>
struct qhipster::AlignedAllocator::rebind

Public Types

typedef AlignedAllocator<U, Alignment> other
Class GateCounter

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_gate_counter.hpp

Class Documentation

class GateCounter

The GateCounter class serves two main purposes: 1) To count the number of gates applied, divided by kind. 2) To estimate the circuit depth if scheduled in a greedy way.

Public Functions

GateCounter (int new_num_qubits)
~GateCounter ()
void Reset ()
int GetTotalGateCount ()
int GetOneQubitGateCount ()
int GetTwoQubitGateCount ()
int GetParallelDepth ()

void OneQubitIncrement (int qubit)
    Update the counters and depth due to the action of a one-qubit gate.

void TwoQubitIncrement (int qubit_0, int qubit_1)
    Update the counters and depth due to the action of a two-qubit gate.

void Breakdown ()
    Print the values of counters and depth.

Class Header

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_timer.hpp

Class Documentation

class Header

It was a struct, but this is slightly nicer (due to having a constructor).
Public Functions

Header()
Header(int num_qubits_, int num_procs_, int num_records_)
std::string sprint()

Public Members

std::size_t num_qubits
std::size_t num_procs
std::size_t num_records

Template Class NoisyQureg

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_NoisyQureg.hpp

Inheritance Relationships

Base Type

- public QubitRegister< Type > (Template Class QubitRegister)

Class Documentation

template<class Type = ComplexDP>
class NoisyQureg : public QubitRegister<Type>
Class that expand QubitRegister states by adding noise between “logical” gates.

Parameters

- num_qubit: is the number of qubits When we refer to “experimental” gates, it means that noise gates are excluded. For the simulation to be faithful (i.e. with one-to-one correspondence with the experimental gates), we include among the experimental gates both the gates for the algorithm and those to schedule it according to the connectivity of the specific hardware.

Public Functions

NoisyQureg(unsigned num_qubits, unsigned RNG_seed = 12345, BaseType T1 = 2000, BaseType T2 = 1000)
Constructor.

~NoisyQureg()
Default destructor.

void Initialize(std::string style, std::size_t base_index)

void ResetTimeForAllQubits()
Reset to zero the time elapsed for each and every qubit in the register.
void ApplyNoiseGatesOnAllQubits()
    Apply the noise gates on each and every qubit. Then reset to time counter.

    This is useful, for example, at the end of a circuit before measuring the quantities of interest: One has to
    apply the noise corresponding to the idle evolution between the last logical gate and the final time. The
    time from last logical gate is then resetted to zero for every qubit.

void SetDecoherenceTime (BaseType, BaseType)
    Set the decoherence time in terms of T_1 and T_2 values (in accordance to the new noise model).

void SetGateDurations (BaseType, BaseType)
    Update the duration of single- and two- qubit gates.

unsigned GetTotalExperimentalGateCount()
    Return the current number of (experimental) 1- and 2-qubit gates.

unsigned GetOneQubitExperimentalGateCount()
    Return the current number of (experimental) single-qubit gates.

unsigned GetTwoQubitExperimentalGateCount()
    Return the current number of (experimental) two-qubit gates.

std::vector<unsigned> GetExperimentalGateCount(unsigned q1)
    Return the number of (experimental) gates involving qubit q.

unsigned GetExperimentalGateCount(unsigned q1, unsigned q2)
    Return the number of (experimental) gates involving qubits q1,q2.

void AddNoiseOneQubitGate(unsigned const)
    Include and execute the noise gate corresponding to the idle time of a single qubit.

void AddNoiseTwoQubitGate(unsigned const, unsigned const)
    Include and execute the noise gate corresponding to the idle time of two qubits.

void NoiseGate(unsigned const)
    Noise gate corresponding to single-qubit rotation with appropriate (stochastic) angle.

    Each noise gate is the product of three rotations around X,Y,Z axis (by a small angle each). We compute
    their product before applying it to the quantum register.

void NoiseGate_OLD(unsigned const)
    Noise gate corresponding to single-qubit rotation with appropriate (stochastic) angle.

    ** OLD OLD OLD OLD OLD OLD **

    Kept for historical reasons, it should be deletead.

    To obtain a single rotation around an arbitrary axis we use the relations: \[ a b c \mid h-f \mid R = l d e f \mid > u = l \]
    \[ c-g \mid > \text{abs}(u) = 2 \sin(\ 'angle'\ ) \mid g h i l d-b \mid > u/\text{abs}(u) = \text{rotation axis} \]

void Apply1QubitGate(unsigned const, qhipster::TinyMatrix<Type, 2, 2, 32>)
void ApplyHadamard(unsigned const)
void ApplyRotationX(unsigned const, BaseType)
void ApplyRotationY(unsigned const, BaseType)
void ApplyRotationZ(unsigned const, BaseType)
void ApplyCPauliX(unsigned const, unsigned const, qhipster::TinyMatrix<Type, 2, 2, 32>)
void ApplyControlled1QubitGate(unsigned const, unsigned const, qhipster::TinyMatrix<Type, 2, 2, 32>)
Class Permutation

• Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_permute.hpp

Class Documentation

class Permutation

Public Functions

unsigned operator[](std::size_t i)
unsigned operator[](unsigned i)
int operator[](int i)
std::size_t size()
std::string GetMapStr()
std::string GetImapStr()

Permutation (std::size_t num_qubits)
Permutation (std::vector<std::size_t> m)
std::size_t Find (std::size_t position)
void SetNewPermutation (std::vector<std::size_t> m)
std::string dec2bin (std::size_t in, std::size_t num_bits)
std::size_t bin2dec (std::string in)
std::size_t lin2perm_ (std::size_t v)
std::size_t perm2lin_ (std::size_t v)
std::string lin2perm (std::size_t v)
std::string perm2lin (std::string s)
std::string perm2lin (std::string s)
void prange ()

Public Members

std::vector<std::size_t> map
std::vector<std::size_t> imap
std::size_t num_qubits
Template Class AlignedAllocator

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_alignedallocator.hpp

Nested Relationships

Nested Types

- Template Struct AlignedAllocator::rebind

Class Documentation

template<typename T, unsigned int Alignment>
class qhipster::AlignedAllocator
An allocator returning aligned memory.
This class provides an aligned C++98 and C++11 conforming allocator.
Pre The alignment must be a power of 2.

Public Types

typedef T *pointer
typedef T const *const_pointer
typedef T &reference
typedef T const &const_reference
typedef T value_type
typedef std::size_t size_type
typedef std::ptrdiff_t difference_type

Public Functions

AlignedAllocator()
AlignedAllocator(AlignedAllocator const&)
template<typename U>
AlignedAllocator(AlignedAllocator<U, Alignment> const&)

pointer allocate(size_type n)
void deallocate(pointer p, size_type)
size_type max_size() const
void construct(pointer p, const_reference t)
template<typename C>
void destroy(C *c)
bool operator==(AlignedAllocator const&) const
bool operator!= (AlignedAllocator const&) const

template<typename U, unsigned int UAlignment>
bool operator== (AlignedAllocator<U, UAlignment> const&) const

bool operator!=(AlignedAllocator<U, UAlignment> const&) const

template<typename U>
struct rebind

Public Types

typedef AlignedAllocator<U, Alignment> other

Class Environment

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_mpi_env.hpp

Class Documentation

class qhipster::mpi::Environment

A trimmed down version of the BOOST::MPI environment. Its purpose is to initialize the MPI library and
partition the cluster or single threaded environment for parallel operations. In preparation of the Monte-Carlo
simulations required for the noisy implementation, we provide a communicator over the ranks involved in a
single MC simulation.

Specifically, we store two communicators:

  a. pool_communicator: spanning all the useful ranks
  b. state_communicator: spanning those ranks used in a single MC simulation (i.e. state)

Public Functions

Environment (int &argc, char **&argv)

Initialize the MPI Environment.

It receives the same argc and argv arguments passed to the main function. If MPI is present, but has not
been initialized, then MPI_Init will be called.

~Environment ()

Finalize the MPI Environment

If MPI is present and has been initialized in the constructor then MPI_Finalize will be called here.

Environment (Environment const&) = delete

Environment &operator= (Environment const&) = delete
Public Static Functions

void UpdateStateComm (int num_states, bool do_print_info = true)
    Update the state and pool communicators.
    
    Pre This can only be called when all ranks are still active.

bool IsUsefulRank ()
    Check whether the rank is useful or not.

int GetPoolRank ()
    The rank of the current MPI process: pool or state.
    
    The PoolRank may not correspond to that from MPI_COMM_WORLD due to dummy ranks. The rank
    is 0 if MPI is not present.
    
    Pre If MPI is present, this can only be called after initialising MPI.

int GetStateRank ()

int GetRank ()

int GetPoolSize ()
    Number of MPI processes.
    
    The PoolSize may not correspond to that from MPI_COMM_WORLD due to dummy ranks. The number
    of processes is 1 if MPI is not present.
    
    Pre If MPI is present, this can only be called after initialising MPI.

int GetStateSize ()

int GetSize ()

template<class Type>
Type IncoherentSumOverAllStatesOfPool (Type local_value)
    Get incoherent average over all states of the pool.

Parameters

• local_value: the address of the value stored in the local rank.

int GetNumRanksPerNode ()

int GetNumNodes ()

int GetNodeId ()

int GetStateId ()

int GetNumStates ()

void RemapStateRank (int newme)
Class Exception

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_mpi_exception.hpp`

Inheritance Relationships

Base Type

- public exception

Class Documentation

class qhipster::mpi::Exception: public exception
Catch-all exception class for MPI errors.

Similar to the MPI exception class in the Boost libraries. Instances of this class will be thrown when an MPI error occurs.

Public Functions

`Exception(const char *routine, int error_code)`
Build a new `Exception` exception.

Parameters

- `routine`: The MPI routine in which the error occurred. This should be a pointer to a string constant: it will not be copied.
- `error_code`: The result code returned from the MPI routine that aborted with an error.

`~Exception()`

`const char *what () const`
A description of the error that occurred.

`const char *routine () const`
Retrieve the name of the MPI routine that reported the error.

`int error_code () const`
Obtain the result code returned from the MPI routine that caused an error.

Template Class RandomNumberGenerator

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_rng_utils.hpp`
Class Documentation

template<typename Type>
class qhipster::RandomNumberGenerator

Used to generate random numbers that are local to each rank, or common to the state or the complete pool.

The generation of numbers and the method to skip ahead are more efficient when MKL (and in particular VSL) is used.

Public Functions

RandomNumberGenerator()
~RandomNumberGenerator()

RandomNumberGenerator(RandomNumberGenerator *source_rng)

Initialize RNG by copying the streams of the source RNG.

std::size_t GetSeed()

Get basic quantities.

std::size_t GetNumGeneratedOrSkippedLocalNumbers()

std::size_t GetNumGeneratedOrSkippedStateNumbers()

std::size_t GetNumGeneratedOrSkippedPoolNumbers()

void SetSeedStreamPtrs(std::size_t RNG_seed)

Set one different seed for each MPI rank (no MKL) or assign different streams (VSL).

void SkipAhead(std::size_t num_skip, std::string shared = "local")

Skip ahead.

void UniformRandomNumbers(Type *value, std::size_t size = 1UL, Type a = 0., Type b = 1., std::string shared = "local")

Generate random numbers in \([a,b)\):

- size indicates how many numbers
- shared can be: local, state, pool

void GaussianRandomNumbers(Type *value, std::size_t size = 1UL, std::string shared = "local")

Generate random gaussian numbers (mean value = 0, std.dev = 1):

- size indicates how many numbers
- shared can be: local, state, pool

void RandomIntegersInRange(int *value, std::size_t size = 1UL, int a = 0, int b = 2, std::string shared = "local")

Generate random integers in \([a,b)\), default being \([0,1)\).

- size indicates how many numbers
- shared can be: local, state, pool
Template Class TinyMatrix

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_tinymatrix.hpp

Class Documentation

template<class ValueType, unsigned M, unsigned N = M, unsigned align = alignof(ValueType)>
class qhipster::TinyMatrix

A small matrix with dimensions fixed at compile time.

The matrix is stored internally as a two-dimensional C array, and thus in row-major ordering.

Public Types

using value_type = ValueType
    the type of elements stored in the matrix

using pointer = ValueType*
    a pointer to elements of the matrix

using const_pointer = ValueType const*
    a pointer to elements of a const matrix

using reference = ValueType&
    a reference to elements of the matrix

using size_type = unsigned
    an integral type large enough to store the size of the matrix

using RowType = ValueType[N]
    the type for a row of the matrix

Public Functions

TinyMatrix()
    default-initialize all matrix elements

template<class U>
TinyMatrix(U init[M][N])
    initialize from a C-style array of the same dimensions

template<class U>
TinyMatrix(std::initializer_list<std::initializer_list<U>> const &init)
    initialize from an initializer list, i.e. a compile time given matrix

template<class U, unsigned alignrhs>
TinyMatrix(TinyMatrix<U, M, N, alignrhs> const &rhs)
    copy from a matrix with a potentially different type and alignment

TinyMatrix(TinyMatrix const&) = default
    the default copy constructor

TinyMatrix &operator=(TinyMatrix const&) = default
    the default assignment

template<class U, unsigned alignrhs>
TinyMatrix &operator=(TinyMatrix<U, M, N, alignrhs> const &rhs)
    assign from a matrix with a potentially different type and alignment

template<class U>
TinyMatrix &operator=(U const (&rhs)[M][N])
    assign from a C-style array

constexpr size_type numRows() const
    the number of matrix rows

constexpr size_type numCols() const
    the number of matrix columns

constexpr size_type size() const
    the size of the matrix, i.e. the number of matrix elements. This is the same as number of rows times number of columns

value_type operator()(unsigned i, unsigned j) const
    access a matrix element of a const matrix

    Pre  i<numRows() & j<numCols()!

Parameters
    • i: the row index
    • j: the column index

reference operator()(unsigned i, unsigned j)
    access a matrix element

    Pre  i<numRows() & j<numCols()!

Parameters
    • i: the row index
    • j: the column index

template<class U, unsigned alignrhs>
bool operator==(TinyMatrix<U, M, N, alignrhs> const &rhs) const
    compare two matrices element-wise for equality

template<class U, unsigned alignrhs>
bool operator!=(TinyMatrix<U, M, N, alignrhs> const &rhs) const
    compare two matrices element-wise for inequality

template<class U>
bool operator==(U const (&rhs)[M][N])
    compare two matrices element-wise for equality

template<class U>
bool operator!=(U const (&rhs)[M][N])
    compare two matrices element-wise for inequality

const_pointer getPtr() const
    obtain a pointer to the first element of the matrix

RowType &operator[](unsigned i)
    C-style array subscript

the TinyMatrix can be indexed both using the mat(i,j) syntax or the C-style mat[i][j] syntax
RowType const &operator[] (unsigned i) const
C-style array subscript for a const matrix

the TinyMatrix can be indexed both using the mat(i,j) syntax or the C-style mat[i][j] syntax

template<unsigned MSub, unsigned NSub = MSub>
TinyMatrix<ValueType, MSub, NSub, align> getSubMatrix (unsigned i_start = 0, unsigned j_start = 0,
unsigned i_stride = 1, unsigned j_stride = 1) const

Get submatrices

Returns the submatrix starting at i_start, j_start of size MSub, NSub using stride i_stride, j_stride

Pre Strides are strictly positive

Pre Parameters actually represent a submatrix (no index out of bounds)

Parameters

• i_start: The starting row index
• j_start: The starting column index
• i_stride: The row stride to use for accessing elements
• j_stride: The column stride to use for accessing elements

Template Parameters

• MSub: The number of rows of the submatrix
• NSub: The number of columns of the submatrix

void print (std::string name)
std::string tostr () const

Public Members

std::string name

Template Class QubitRegister

• Defined in file ~/home/docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts/docs/include/qureg.hpp

Inheritance Relationships

Derived Types

• public NoisyQureg< Type > (Template Class NoisyQureg)
• public QubitRegisterMetric< Type > (Template Class QubitRegisterMetric)
Class Documentation

template<class Type = ComplexDP>
class QubitRegister
    Subclassed by NoisyQureg<Type>, QubitRegisterMetric<Type>

Public Types

using value_type = Type
typedef extract_value_type<Type>::value_type BaseType

Public Functions

QubitRegister()
QubitRegister(std::size_t num_qubits, std::string style = "", std::size_t base_index = 0, std::size_t tmp_spacesize_ = 0)
QubitRegister(const QubitRegister &in)
QubitRegister(std::size_t num_qubits, Type *state, std::size_t tmp_spacesize_ = 0)
~QubitRegister()
void AllocateAdditionalQubit()
void Allocate(std::size_t new_num_qubits, std::size_t tmp_spacesize_)
void Initialize(std::size_t new_num_qubits, std::size_t tmp_spacesize_)
void Initialize(std::string style, std::size_t base_index)
Type &operator[](std::size_t index)
Type &operator[](std::size_t index) const
Type GetGlobalAmplitude(std::size_t index) const
std::size_t LocalSize() const
std::size_t GlobalSize() const
void Resize(std::size_t new_num_amplitudes)
std::size_t size() const
std::size_t NumQubits() const
Type *TmpSpace() const
size_t TmpSize() const
bool check_bit(std::size_t variable, std::size_t position) const
std::size_t set_bit(std::size_t variable, std::size_t position) const
std::size_t clear_bit(std::size_t variable, std::size_t position) const
void EnableStatistics()
void GetStatistics()
void DisableStatistics()
void \texttt{ResetStatistics} ()

void \texttt{Permute} (std::vector<std::size_t> \texttt{permutation\_new\_vec})

bool \texttt{Apply1QubitGate\_helper} (unsigned \texttt{qubit}, TM2x2<Type> \texttt{const \&m}, std::size_t \texttt{sstate\_ind},
\texttt{std::size\_t estate\_ind})

void \texttt{Apply1QubitGate} (unsigned \texttt{qubit}, TM2x2<Type> \texttt{const \&m})

bool \texttt{ApplyControlled1QubitGate\_helper} (unsigned \texttt{control\_qubit}, unsigned \texttt{target\_qubit},
TM2x2<Type> \texttt{const \&m}, std::size_t \texttt{sind},
\texttt{std::size\_t eind})

void \texttt{ApplyControlled1QubitGate} (unsigned \texttt{control\_qubit}, unsigned \texttt{target\_qubit},
TM2x2<Type> \texttt{const \&m})

bool \texttt{ApplySwap\_helper} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2}, TM2x2<Type> \texttt{const \&m})

void \texttt{ApplySwap} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2})

void \texttt{ApplyISwap} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2})

void \texttt{Apply4thRootISwap} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2})

void \texttt{ApplySqrtISwap} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2})

void \texttt{ApplyISwapRotation} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2}, TM2x2<Type> \texttt{const \&m})

void \texttt{Swap} (unsigned \texttt{b1}, unsigned \texttt{b2})

void \texttt{ApplyDiagSimp} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2}, TM4x4<Type> \texttt{const \&m})

void \texttt{ApplyDiag} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2}, TM4x4<Type> \texttt{const \&m})

void \texttt{ApplyDiagControl} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2}, TM4x4<Type> \texttt{const \&m})

void \texttt{ApplyDiagGeneral} (unsigned \texttt{qubit1}, unsigned \texttt{qubit2}, TM4x4<Type> \texttt{const \&m})

void \texttt{Apply2QubitGate} (unsigned \texttt{const qubit\_high}, unsigned \texttt{const qubit\_low}, TM4x4<Type> \texttt{const \&m})

void \texttt{ApplyRotationX} (unsigned \texttt{const qubit}, BaseType \texttt{theta})

void \texttt{ApplyRotationY} (unsigned \texttt{const qubit}, BaseType \texttt{theta})

void \texttt{ApplyRotationZ} (unsigned \texttt{const qubit}, BaseType \texttt{theta})

void \texttt{ApplyPauliX} (unsigned \texttt{const qubit})

void \texttt{ApplyPauliY} (unsigned \texttt{const qubit})

void \texttt{ApplyPauliZ} (unsigned \texttt{const qubit})

void \texttt{ApplyPauliSqrtX} (unsigned \texttt{const qubit})

void \texttt{ApplyPauliSqrtY} (unsigned \texttt{const qubit})

void \texttt{ApplyPauliSqrtZ} (unsigned \texttt{const qubit})

void \texttt{ApplyT} (unsigned \texttt{const qubit})

void \texttt{ApplyToffoli} (unsigned \texttt{const qubit1}, unsigned \texttt{const qubit2}, unsigned \texttt{const qubit3})

void \texttt{ApplyHadamard} (unsigned \texttt{const qubit})

void \texttt{ApplyCRotationX} (unsigned \texttt{const control\_qubit}, unsigned \texttt{const target\_qubit}, BaseType \texttt{theta})

void \texttt{ApplyCRotationY} (unsigned \texttt{const control\_qubit}, unsigned \texttt{const target\_qubit}, BaseType \texttt{theta})
void ApplyCRotationZ (unsigned const control_qubit, unsigned const target_qubit, BaseType theta)
void ApplyCPauliX (unsigned const control_qubit, unsigned const target_qubit)
void ApplyCPauliY (unsigned const control_qubit, unsigned const target_qubit)
void ApplyCPauliZ (unsigned const control_qubit, unsigned const target_qubit)
void ApplyCPauliSqrtZ (unsigned const control_qubit, unsigned const target_qubit)
void ApplyCHadamard (unsigned const control_qubit, unsigned const target_qubit)
void ApplyCPhaseRotation (unsigned const qubit, unsigned const qubit2, BaseType theta)
void TurnOnFusion (unsigned log2llc = 20)
void TurnOffFusion ()
bool IsFusionEnabled ()
void ApplyFusedGates ()
void TurnOnSpecialize ()
void TurnOffSpecialize ()
bool GetClassicalValue (unsigned qubit, BaseType tolerance = 1.e-13) const
bool IsClassicalBit (unsigned qubit, BaseType tolerance = 1.e-13) const
void CollapseQubit (unsigned qubit, bool value)
BaseType GetProbability (unsigned qubit)
BaseType ExpectationValueX (unsigned const qubit, BaseType coeff = 1.)
BaseType ExpectationValueY (unsigned const qubit, BaseType coeff = 1.)
BaseType ExpectationValueZ (unsigned const qubit, BaseType coeff = 1.)
BaseType ExpectationValueXX (unsigned const qubit, unsigned const qubit2, BaseType coeff = 1.)
BaseType ExpectationValueXY (unsigned const qubit, unsigned const qubit2, BaseType coeff = 1.)
BaseType ExpectationValueXZ (unsigned const qubit, unsigned const qubit2, BaseType coeff = 1.)
BaseType ExpectationValueYX (unsigned const qubit, unsigned const qubit2, BaseType coeff = 1.)
BaseType ExpectationValueYY (unsigned const qubit, unsigned const qubit2, BaseType coeff = 1.)
BaseType ExpectationValueYZ (unsigned const qubit, unsigned const qubit2, BaseType coeff = 1.)
BaseType ExpectationValueZZ (unsigned const qubit, unsigned const qubit2, BaseType coeff = 1.)
BaseType ExpectationValue (std::vector<unsigned> &qubits, std::vector<unsigned> &observables, BaseType coeff = 1.)
BaseType GetT1 ()

5.3. Full API
**Public Members**

```cpp
std::size_t num_qubits
std::vector<Type, qhipster::AlignedAllocator<Type, 256>> state_storage
Type *state
Permutation *permutation
Timer *timer
GateCounter *gate_counter
std::size_t llc_watermarkbit
bool imported_state
bool specialize
bool fusion
unsigned log2llc
std::vector<std::tuple<std::string, TM2x2<Type>, unsigned, unsigned>> fwindow
```
Public Static Functions

void SetDoPrintExtraInfo (bool value)

Template Class QubitRegisterMetric

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_QubitRegisterMetric.hpp

Inheritance Relationships

Base Type

- public QubitRegister< Type > (Template Class QubitRegister)

Class Documentation

template<class Type = ComplexDP>
class QubitRegisterMetric: public QubitRegister<Type>

Public Functions

QubitRegisterMetric (int iNQubits)
int GetTotalQubitGateCount ()
int GetOneQubitGateCount ()
int GetTwoQubitGateCount ()
int GetParallelDepth ()
void ApplyHadamard (int)
void ApplyRotationX (int, double)
void ApplyRotationY (int, double)
void ApplyRotationZ (int, double)
void ApplyCPauliX (int, int)
void ApplyControlled1QubitGate (int, int, qhipster:: TinyMatrix<Type, 2, 2, 32>)

Class Time

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_timer.hpp

5.3. Full API
Class Documentation

class Time
Stores the time spent in the various part of the computation/communication.

Public Functions

Time()
bool timed()
std::string sprint (bool combinedstats)

Public Members

double start
bool exists
std::size_t cpos
std::size_t tpos
std::size_t ncalls
double total
double sn_time
double sn_bw
double dn_time
double dn_bw
double tn_time
double tn_bw
double cm_time
double cm_bw
double flops
double gflops

Class Timer

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
qs_checkouts_docs_include_timer.hpp

42 Chapter 5. API References
Class Documentation

class Timer

The Timer class serves two purposes: 1) To provide a reliable and static call to Wtime() (since MPI_Wtime may not be available). 2) To provide a tidy way of profiling the code.

Public Functions

Timer (bool combinedstats = false)
Timer (int num_qubits_, int my_rank_, int num_procs_)
~Timer ()
void Reset ()
double Wtime ()
void Start (std::string s, std::size_t cpos, std::size_t tpos = 999999)
Start the timer.
void record_sn (double time, double bw)
void record_dn (double time, double bw)
void record_tn (double time, double bw)
void record_cm (double time, double bw)
void Stop ()
Stop the timer.
void Breakdown ()
Print the statistics to screen.

Public Members

std::map<std::string, Time>::iterator curiter

5.3.3 Functions

Template Function __attribute__

• Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_highperfkernels.hpp

Function Documentation

template<typename Type> __attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std __attribute__ ((noinline)) void Loop_TN(Type *state

5.3. Full API
Function GetQhipsterVersion

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_qreg_version.hpp

Function Documentation

std::string GetQhipsterVersion (void)

Function perm

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_permute.hpp

Function Documentation

std::size_t perm (std::size_t v, std::size_t *map, std::size_t num_qubits)

Template Function qaoa::GetExpectationValueFromCostFunction

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_qaoa_features.hpp

Function Documentation

template<typename Type>
QubitRegister<Type>::BaseType qaoa::GetExpectationValueFromCostFunction (const QubitRegister<Type> &psi, const QubitRegister<Type> &diag)

Template Function qaoa::GetExpectationValueSquaredFromCostFunction

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_qaoa_features.hpp
Function Documentation

template<typename Type> 
QubitRegister<Type>::BaseType qaoa::GetExpectationValueSquaredFromCostFunction(const 
QubitRegister<Type>& psi, 
const QubitRegister<Type>& diag)

Template Function qaoa::GetHistogramFromCostFunction

• Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
qs_checkouts_docs_include_qaoa_features.hpp

Function Documentation

template<typename Type> 
std::vector<typename QubitRegister<Type>::BaseType> qaoa::GetHistogramFromCostFunction(const 
QubitRegister<Type>& psi, 
const QubitRegister<Type>& diag, 
int max_value)

Template Function qaoa::GetHistogramFromCostFunctionWithWeightsBinned

• Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
qs_checkouts_docs_include_qaoa_features.hpp
Template Function qaoa::GetHistogramFromCostFunctionWithWeightsRounded

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qc_checkouts_docs_include_qaoa_features.hpp

Function Documentation

template<typename Type>
std::vector<typename QubitRegister<Type>::BaseType> qaoa::GetHistogramFromCostFunctionWithWeightsRounded

Template Function `qaoa::ImplementQaoaLayerBasedOnCostFunction`

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel qs_checkouts_docs_include_qaoa_features.hpp`

Function Documentation

```
template<typename Type>
void qaoa::ImplementQaoaLayerBasedOnCostFunction(QubitRegister<Type> &psi, QubitRegister<Type> &diag, typename QubitRegister<Type>::BaseType gamma)
```

Template Function `qaoa::InitializeVectorAsMaxCutCostFunction`

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel qs_checkouts_docs_include_qaoa_features.hpp`

Function Documentation

```
template<typename Type>
int qaoa::InitializeVectorAsMaxCutCostFunction(QubitRegister<Type> &diag, std::vector<int> &adjacency)
```

Template Function `qaoa::InitializeVectorAsWeightedMaxCutCostFunction`

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel qs_checkouts_docs_include_qaoa_features.hpp`

Function Documentation

```
template<typename Type>
QubitRegister<Type>::BaseType qaoa::InitializeVectorAsWeightedMaxCutCostFunction(QubitRegister<Type> &diag, std::vector<typename QubitRegister<Type>::BaseType> &adjacency)
```
Function qhipster::detail::BX

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_bitops.hpp

Function Documentation

```cpp
int qhipster::detail::BX(long x)
```

Template Function qhipster::detail::highestBitImpl

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_bitops.hpp

Function Documentation

```cpp
template<class Integral>
constexpr unsigned qhipster::detail::highestBitImpl(Integral i, unsigned pos)
```

Template Function qhipster::floor_power_of_two

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_bitops.hpp

Function Documentation

```cpp
template<class Integral>
unsigned qhipster::floor_power_of_two(Integral x)
```

Template Function qhipster::highestBit

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_bitops.hpp

Function Documentation

```cpp
template<class Integral>
constexpr unsigned qhipster::highestBit(Integral i)
```

returns the highest bit set in a non-zero integer

This function returns the highest bit set in a non-zero integer.

**Pre** The integer \( i \) is non-zero

**Parameters**

- \([\text{in}]\ i: \) the non-zero integer of which the highest bit is returned
Template Function qhipster::ilog2

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel
  - qs_checkouts_docs_include_bitops.hpp

**Function Documentation**

template<class Integral>
unsigned int qhipster::ilog2(Integral n)

Returns the logarithm base 2 of a non-zero integer.

This function returns the the logarithm base 2 of a non-zero integer

**Pre** The integer i is a power of 2

**Parameters**

- [in] `i`: the non-zero integer of which the logarithm base 2 is returned

Template Function qhipster::isPowerOf2

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel
  - qs_checkouts_docs_include_bitops.hpp

**Function Documentation**

template<class Integral>
constexpr bool qhipster::isPowerOf2(Integral i)

checks whether an integer is a power of 2

**Parameters**

- [in] `i`: an integer to be checked

Function qhipster::mpi::Barrier

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel
  - qs_checkouts_docs_include_mpi_env.hpp

**Function Documentation**

void qhipster::mpi::Barrier()
Function `qhipster::mpi::PoolBarrier`

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_mpi_env.hpp`

**Function Documentation**

```cpp
void qhipster::mpi::PoolBarrier()

An MPI barrier.

It waits until all MPI processes have reached this call. This function does nothing if MPI is not present.
```

Function `qhipster::mpi::PoolPrint`

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_mpi_env.hpp`

**Function Documentation**

```cpp
void qhipster::mpi::PoolPrint (std::string s, bool all = false)

Print from all MPI processes.

It prints a string from all processes if `all` is true or just from the master process with rank 0 if `all` is false. If MPI is not present it prints the string.

If `all` is set, the string is prefixed by the number of the MPI process.

**Parameters**

- `s`: the string to be printed
- `all`: a flag to specify if all processes should print or just the master process
```

Function `qhipster::mpi::Print`

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_mpi_env.hpp`

**Function Documentation**

```cpp
void qhipster::mpi::Print (std::string s, bool all = false)
```
Function qhipster::mpi::StateBarrier

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_mpi_env.hpp

Function Documentation

void qhipster::mpi::StateBarrier()

Function qhipster::mpi::StatePrint

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_mpi_env.hpp

Function Documentation

void qhipster::mpi::StatePrint (std::string s, bool all = false)

Function qhipster::popcnt(uint32_t)

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_bitops.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “qhipster::popcnt” with arguments (uint32_t) in doxygen xml output for project “My Project” from directory: ./doxyoutput/xml. Potential matches:

- long popcnt (uint32_t x)
- long popcnt (uint64_t x)

Function qhipster::popcnt(uint64_t)

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_bitops.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve multiple matches for function “qhipster::popcnt” with arguments (uint64_t) in doxygen xml output for project “My Project” from directory: ./doxyoutput/xml. Potential matches:

- long popcnt (uint32_t x)
- long popcnt (uint64_t x)
Template Function qhipster::ShuffleFisherYates

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_rng_utils.hpp

Function Documentation

template<typename Type, typename TypeFloat>
void qhipster::ShuffleFisherYates (std::vector<Type> &array, RandomNumberGenerator<TypeFloat> *rnd_generator_ptr, std::string shared = "local")

Template Function qhipster::toString

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_conversion.hpp

Function Documentation

template<class T>
std::string qhipster::toString (T const &val)
convert to a string

This function converts any value to a string, by writing it into a string stream.

Pre Writing into a std::istream using operator<< needs to be implemented for the type

Parameters

- [in] val: the value to be converted to a string

Function qhipster::WhatCompileDefinitions

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_utils.hpp

Function Documentation

void qhipster::WhatCompileDefinitions ()
Utility method to inform on the currently set compiler flags.

Template Function ScaleState

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_highperfkernels.hpp
template<typename Type>
void ScaleState(std::size_t start, std::size_t end, Type *state, const Type &s, Timer *timer)

Function Documentation

double time_in_seconds(void)

5.3.4 Variables

Variable c11

Variable Documentation
__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t c11

Variable c12

Variable Documentation
__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t c12

Variable c13

Variable Documentation
__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t c13

5.3. Full API
Variable Documentation

__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t c13

Variable c21

• Defined in file_home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_highperfkernels.hpp

Variable Documentation

__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t std::size_t c22

Variable c22

• Defined in file_home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_highperfkernels.hpp

Variable Documentation

__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t std::size_t std::size_t c23

Variable c23

• Defined in file_home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_highperfkernels.hpp

Variable Documentation

__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t c31

Variable c31

• Defined in file_home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_highperfkernels.hpp

Variable Documentation

__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t c31
Variable documentation:

**Variable c32**
- Defined in file `/home/docs/checkouts/readthedocs.org/user_builds/intel-qs/checkouts/docs/include/highperfkernels.hpp`

**Variable Documentation**

`__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t c32`  

**Variable ind_shift**
- Defined in file `/home/docs/checkouts/readthedocs.org/user_builds/intel-qs/checkouts/docs/include/highperfkernels.hpp`

**Variable Documentation**

`__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t ind_shift`  

**Variable m**
- Defined in file `/home/docs/checkouts/readthedocs.org/user_builds/intel-qs/checkouts/docs/include/highperfkernels.hpp`

**Variable Documentation**

`__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t TM2x2< Type > const & m`  

**Variable specialize**
- Defined in file `/home/docs/checkouts/readthedocs.org/user_builds/intel-qs/checkouts/docs/include/highperfkernels.hpp`

**Variable Documentation**

`__attribute__((noinline)) void Loop_SN(std __attribute__((noinline)) void Loop_DN(std std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t std::size_t specialize`  

**Variable timer**
- Defined in file `/home/docs/checkouts/readthedocs.org/user_builds/intel-qs/checkouts/docs/include/highperfkernels.hpp`

**Variable Documentation**

`bool QubitRegister::specialize`
Variable Documentation

*Timer* *QubitRegister::*timer

5.3.5 Defines

**Define __str__**

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_utils.hpp`

**Define Documentation**

__str__(s)

**Define D**

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_utils.hpp`

**Define Documentation**

D(x)

**Define DO_PRAGMA**

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_utils.hpp`

**Define Documentation**

DO_PRAGMA(x)

**Define INFO**

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_intel-qs_checkouts_docs_include_utils.hpp`
Define Documentation

INFO (x)

Define noexcept

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qqs_checkouts_docs_include_alignedallocator.hpp

Define Documentation

noexcept

Define QHIPSTER_MPI_CHECK_RESULT

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qqs_checkouts_docs_include_mpi_exception.hpp

Define Documentation

QHIPSTER_MPI_CHECK_RESULT (MPIFunc, Args)

Call the MPI routine MPIFunc with arguments Args (surrounded by parentheses). Checks the return value of MPIFunc call and throws a qhipster::mpi::exception if the result is not MPI_SUCCESS.

Define QHIPSTER_VERSION_STRING

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qqs_checkouts_docs_include_qureg_version.hpp

Define Documentation

QHIPSTER_VERSION_STRING

Define sec

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-qqs_checkouts_docs_include_utils.hpp

5.3. Full API
5.3.6 Typedefs

**Typedef BaseType**

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_intel-
  qs_checkouts_docs_include_qureg.hpp
typedef QubitRegister::BaseType NoisyQureg::BaseType

typedef ComplexDP

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-
qs_checkouts_docs_include_utils.hpp

using ComplexDP = std::complex<double>

typedef ComplexSP

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-
qs_checkouts_docs_include_utils.hpp

using ComplexSP = std::complex<float>

typedef TM2x2

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-
qs_checkouts_docs_include_qureg.hpp

using TM2x2 = qhipster::TinyMatrix<Type, 2, 2, 32>

typedef TM4x4

- Defined in file __home_docs_checkouts_readthedocs.org_user_builds_intel-
qs_checkouts_docs_include_qureg.hpp

using TM4x4 = qhipster::TinyMatrix<Type, 4, 4, 32>
CHAPTER
SIX

INDICES AND TABLES

• genindex
• modindex
• search
## Symbols

__str__ (C macro), 56

C

ComplexDP (C++ type), 59
ComplexSP (C++ type), 59

D

D (C macro), 56
DO_PRAGMA (C macro), 56

E

extract_value_type (C++ struct), 24
extract_value_type::value_type (C++ type), 24
extract_value_type<X<T>> (C++ struct), 24
extract_value_type<X<T>>::value_type (C++ type), 24

G

GateCounter (C++ class), 25
GateCounter::~GateCounter (C++ function), 25
GateCounter::Breakdown (C++ function), 25
GateCounter::GateCounter (C++ function), 25
GateCounter::GetOneQubitGateCount (C++ function), 25
GateCounter::GetParallelDepth (C++ function), 25
GateCounter::GetTotalGateCount (C++ function), 25
GateCounter::GetTwoQubitGateCount (C++ function), 25
GateCounter::OneQubitIncrement (C++ function), 25
GateCounter::Reset (C++ function), 25
GateCounter::TwoQubitIncrement (C++ function), 25
GetHipsterVersion (C++ function), 44

H

Header (C++ class), 25

Header::Header (C++ function), 26
Header::num_procs (C++ member), 26
Header::num_qubits (C++ member), 26
Header::num_records (C++ member), 26
Header::sprint (C++ function), 26

I

INFO (C macro), 57

N

noexcept (C macro), 57
NoisyQureg (C++ class), 26
NoisyQureg::~NoisyQureg (C++ function), 26
NoisyQureg::AddNoiseOneQubitGate (C++ function), 27
NoisyQureg::AddNoiseTwoQubitGate (C++ function), 27
NoisyQureg::Apply1QubitGate (C++ function), 27
NoisyQureg::ApplyControlled1QubitGate (C++ function), 27
NoisyQureg::ApplyCPauliX (C++ function), 27
NoisyQureg::ApplyHadamard (C++ function), 27
NoisyQureg::ApplyNoiseGatesOnAllQubits (C++ function), 26
NoisyQureg::ApplyRotationX (C++ function), 27
NoisyQureg::ApplyRotationY (C++ function), 27
NoisyQureg::ApplyRotationZ (C++ function), 27
NoisyQureg::BaseType (C++ type), 59
NoisyQureg::GetExperimentalGateCount (C++ function), 27
NoisyQureg::GetOneQubitExperimentalGateCount (C++ function), 27
NoisyQureg::GetTwoQubitExperimentalGateCount (C++ function), 27
NoisyQureg::Initialize (C++ function), 26
NoisyQureg::NoiseGate (C++ function), 27
<table>
<thead>
<tr>
<th>Class/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoisyQureg::NoiseGate_OLD (C++ function)</td>
<td>27</td>
</tr>
<tr>
<td>NoisyQureg::NoisyQureg (C++ function)</td>
<td>26</td>
</tr>
<tr>
<td>NoisyQureg::ResetTimeForAllQubits (C++ function)</td>
<td>26</td>
</tr>
<tr>
<td>NoisyQureg::SetDecoherenceTime (C++ function)</td>
<td>27</td>
</tr>
<tr>
<td>NoisyQureg::SetGateDurations (C++ function)</td>
<td>27</td>
</tr>
<tr>
<td>Perm (C++ function)</td>
<td>44</td>
</tr>
<tr>
<td>Permuation (C++ class)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::bin2dec (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::dec2bin (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::Find (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::GetMapStr (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::imap (C++ member)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::lin2perm (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::lin2perm_ (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::map (C++ member)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::num_qubits (C++ member)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::operator[] (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::perm2lin (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::perm2lin_ (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::prange (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::SetNewPermutation (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Permuation::size (C++ function)</td>
<td>28</td>
</tr>
<tr>
<td>Qaoa::GetExpectationValueFromCostFunction (C++ function)</td>
<td>44</td>
</tr>
<tr>
<td>Qaoa::GetExpectationValueSquaredFromCostFunction (C++ function)</td>
<td>45</td>
</tr>
<tr>
<td>Qaoa::GetHistogramFromCostFunction (C++ function)</td>
<td>45</td>
</tr>
<tr>
<td>Qaoa::GetHistogramFromCostFunctionWithWeightsBinned (C++ function)</td>
<td>46</td>
</tr>
<tr>
<td>Qaoa::GetHistogramFromCostFunctionWithWeightsBinned (C++ function)</td>
<td>46</td>
</tr>
<tr>
<td>Qaoa::ImplementQaoaLayerBasedOnCostFunction (C++ function)</td>
<td>47</td>
</tr>
<tr>
<td>Qaoa::InitializeVectorAsMaxCutCostFunction (C++ function)</td>
<td>47</td>
</tr>
<tr>
<td>Qaoa::InitializeVectorAsWeightedMaxCutCostFunction (C++ function)</td>
<td>47</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator (C++ class)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::allocate (C++ function)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::const_pointer (C++ type)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::const_reference (C++ type)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::construct (C++ function)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::deallocate (C++ function)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::destroy (C++ function)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::difference_type (C++ type)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::max_size (C++ function)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::operator!= (C++ function)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::operator== (C++ function)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::pointer (C++ type)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::rebind (C++ struct)</td>
<td>24, 30</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::rebind::other (C++ type)</td>
<td>24, 30</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::reference (C++ type)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::size_type (C++ type)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::AlignedAllocator::value_type (C++ type)</td>
<td>29</td>
</tr>
<tr>
<td>Qhipster::detail::BX (C++ function)</td>
<td>48</td>
</tr>
<tr>
<td>Qhipster::detail::highestBitImpl (C++ function)</td>
<td>48</td>
</tr>
<tr>
<td>Qhipster::floor_power_of_two (C++ function)</td>
<td>48</td>
</tr>
<tr>
<td>Qhipster::highestBit (C++ function)</td>
<td>48</td>
</tr>
<tr>
<td>Qhipster::ilog2 (C++ function)</td>
<td>49</td>
</tr>
<tr>
<td>Qhipster::isPowerOf2 (C++ function)</td>
<td>49</td>
</tr>
<tr>
<td>Qhipster::mpi::Barrier (C++ function)</td>
<td>49</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment (C++ class)</td>
<td>30</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment::~Environment (C++ function)</td>
<td>30</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment::Environment (C++ function)</td>
<td>30</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment::GetPoolRank (C++ function)</td>
<td>31</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment::GetNodeIds (C++ function)</td>
<td>31</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment::GetNumNodes (C++ function)</td>
<td>31</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment::GetNumRanksPerNode (C++ function)</td>
<td>31</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment::GetNumStates (C++ function)</td>
<td>31</td>
</tr>
<tr>
<td>Qhipster::mpi::Environment::GetPoolRank (C++ function)</td>
<td>31</td>
</tr>
</tbody>
</table>
qhipster::WhatCompileDefinitions (C++ function), 52
QHIPSTER_MPI_CHECK_RESULT (C macro), 57
QHIPSTER_VERSION_STRING (C macro), 57
QubitRegister (C++ class), 37
QubitRegister::~QubitRegister (C++ function), 37
QubitRegister::Allocate (C++ function), 37
QubitRegister::AllocateAdditionalQubit (C++ function), 37
QubitRegister::Apply1QubitGate (C++ function), 38
QubitRegister::Apply1QubitGate_helper (C++ function), 38
QubitRegister::Apply2QubitGate (C++ function), 38
QubitRegister::Apply4thRootISwap (C++ function), 38
QubitRegister::ApplyCHadamard (C++ function), 39
QubitRegister::ApplyControlled1QubitGate (C++ function), 38
QubitRegister::ApplyControlled1QubitGate_helper (C++ function), 38
QubitRegister::ApplyCPauliSqrtZ (C++ function), 39
QubitRegister::ApplyCPauliX (C++ function), 39
QubitRegister::ApplyCPauliY (C++ function), 39
QubitRegister::ApplyCPauliZ (C++ function), 39
QubitRegister::ApplyCPhaseRotation (C++ function), 39
QubitRegister::ApplyCRotationX (C++ function), 38
QubitRegister::ApplyCRotationY (C++ function), 38
QubitRegister::ApplyCRotationZ (C++ function), 38
QubitRegister::ApplyDiag (C++ function), 38
QubitRegister::ApplyDiagControl (C++ function), 38
QubitRegister::ApplyDiagGeneral (C++ function), 38
QubitRegister::ApplyDiagSimp (C++ function), 38
QubitRegister::ApplyFusedGates (C++ function), 38
QubitRegister::ApplyHadamard (C++ function), 38
QubitRegister::ApplyISwap (C++ function), 38
QubitRegister::ApplyISwapRotation (C++ function), 38
QubitRegister::ApplyNoiseGate (C++ function), 40
QubitRegister::ApplyPauliSqrtX (C++ function), 38
QubitRegister::ApplyPauliSqrtY (C++ function), 38
QubitRegister::ApplyPauliSqrtZ (C++ function), 38
QubitRegister::ApplyPauliX (C++ function), 38
QubitRegister::ApplyPauliY (C++ function), 38
QubitRegister::ApplyPauliZ (C++ function), 38
QubitRegister::ApplyRotationX (C++ function), 38
QubitRegister::ApplyRotationY (C++ function), 38
QubitRegister::ApplyRotationZ (C++ function), 38
QubitRegister::ApplySqrtISwap (C++ function), 38
QubitRegister::ApplySwap (C++ function), 38
QubitRegister::ApplySwap_helper (C++ function), 38
QubitRegister::ApplyT (C++ function), 38
QubitRegister::ApplyToffoli (C++ function), 38
QubitRegister:: BaseType (C++ type), 37
QubitRegister::check_bit (C++ function), 37
QubitRegister::clear_bit (C++ function), 37
QubitRegister::CollapseQubit (C++ function), 39
QubitRegister::ComputeNorm (C++ function), 40
QubitRegister::ComputeOverlap (C++ function), 40
QubitRegister::DisableStatistics (C++ function), 37
QubitRegister::dumpbin (C++ function), 40
QubitRegister::EnableStatistics (C++ function), 37
QubitRegister::Entropy (C++ function), 40
QubitRegister::ExpectationValue (C++ function), 39
QubitRegister::ExpectationValueX (C++ function), 39
QubitRegister::ExpectationValueXX (C++ function), 39
QubitRegister::ExpectationValueXY (C++ function), 39
QubitRegister::ExpectationValueXZ (C++ function), 39
QubitRegister::ExpectationValueY (C++ function), 39
QubitRegister::ExpectationValueXX (C++ function), 39
QubitRegister::ExpectationValueXY (C++ function), 39
QubitRegister::ExpectationValueXZ (C++ function), 39
QubitRegister::ExpectationValueY (C++ function), 39
QubitRegister::ExpectationValueXX (C++ function), 39
QubitRegister::ExpectationValueXY (C++ function), 39
QubitRegister::ExpectationValueXZ (C++ function), 39
QubitRegister::ExpectationValueY (C++ function), 39
QubitRegisterMetric::GetParallelDepth (C++ function), 41
QubitRegisterMetric::GetTotalQubitGateCount (C++ function), 41
QubitRegisterMetric::GetTwoQubitGateCount (C++ function), 41
QubitRegisterMetric::QubitRegisterMetric (C++ function), 41

S
ScaleState (C++ function), 53
sec (C macro), 58

T
Time (C++ class), 42
Time::cm_bw (C++ member), 42
Time::cm_time (C++ member), 42
Time::cpos (C++ member), 42
Time::dn_bw (C++ member), 42
Time::dn_time (C++ member), 42
Time::exists (C++ member), 42
Time::flops (C++ member), 42
Time::gflops (C++ member), 42
Time::ncalls (C++ member), 42
Time::sn_bw (C++ member), 42
Time::sn_time (C++ member), 42
Time::sprint (C++ function), 42
Time::start (C++ member), 42
Time::Time (C++ function), 42
Time::timed (C++ function), 42
Time::tn_bw (C++ member), 42
Time::tn_time (C++ member), 42
Time::total (C++ member), 42
Time::tpos (C++ member), 42
time_in_seconds (C++ function), 53
Timer (C++ class), 43
Timer::~Timer (C++ function), 43
Timer::Breakdown (C++ function), 43
Timer::curiter (C++ member), 43
Timer::record_cm (C++ function), 43
Timer::record_dn (C++ function), 43
Timer::record_sn (C++ function), 43
Timer::record_tn (C++ function), 43
Timer::Reset (C++ function), 43
Timer::Start (C++ function), 43
Timer::Stop (C++ function), 43
Timer::Timer (C++ function), 43
Wtime (C++ function), 43
TM2x2 (C++ type), 59
TM4x4 (C++ type), 59
TODO (C macro), 58

U
UL (C macro), 58