Quplexity User Manual

Jacob Liam Gill

Jan 2025

1. Abstract & Overview

Quantum Computer Simulators (QCS's) are often complex pieces of technology where performace is essential. Most modern QCS's like Qrack and Qiskit are written in either Python or C/C++. C++ being the more popular and suitable option for performant simulators. Despite C++ being performant and fast in nature, x86 and ARM/AMR64 Assembly, when written and utilised correctly proves to be significantly faster than C++ whilst also being extremely lightweight in nature. This paper looks at how I have successfully utilised the Assembly language to provide performance and "weight" benefits to QCS's and the like. All the code for this project can be viewed on the Quplexity github.

2. ARM and ARM64 Functions

This section will detail how to use and work with x86 and Intel based Quplexity functions in your C/C++ project. Before trying to use Quplexity functions ensure that you have downloaded the Quplexity repository and its located in the root directory of your project. If you have any issues, feel free to email me at: jacobygill@outlook.com **If you are using an M1/M2 Mac please remove the underscore (_) in your C/C++ code**.

2.1 Pauli-X Gate

The Pauli-X gate, also known as the quantum NOT gate, is a single-qubit quantum gate that flips the state of a qubit. If the input is $|0\rangle$, it outputs $|1\rangle$, and if the input is $|1\rangle$, it outputs $|0\rangle$. Mathematically, it is represented by the matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

This gate is analogous to the classical NOT gate but operates on quantum states. It can also act on superposition states, flipping the amplitudes between $|0\rangle$ and $|1\rangle$. Example use of the Quplexity Pauli-X gate:

```c
#include <stdio.h>

extern double _PX(double *qubit);

int main() {
    double qubit[2] = {0.0, 1.0};
    _PX(qubit);
    printf("%f, %f\n", qubit[0], qubit[1]);
    return 0;
}
```

2.2 Pauli-Z Gate

The Pauli-Z gate is a single-qubit quantum gate that applies a phase flip to the $|1\rangle$ state while leaving the $|0\rangle$ state unchanged. If the input is $|0\rangle$, the output remains $|0\rangle$, but if the input is $|1\rangle$, the output becomes $-|1\rangle$. Mathematically, it is represented by the matrix:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

This gate is a fundamental operation in quantum computing and is often used to manipulate the relative phase of quantum states. Example use of the Quplexity Pauli-Z gate:

```
#include <stdio.h>

extern double _PZ(double *qubit);

int main(){
    double qubit[2] = {0.0, 1.0};
    _PZ(qubit);
    printf("%f, %f\n", qubit[0], qubit[1]);
    return 0;
}
```

2.3 Hadamard Gate

The Hadamard gate is a single-qubit quantum gate that creates superposition states. It transforms the basis states $|0\rangle$ and $|1\rangle$ into equal superpositions:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Mathematically, it is represented by the matrix:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

The Hadamard gate is essential in quantum computing as it enables the creation of superpositions, a key feature of quantum algorithms.

Example use of the Quplexity Hadamard gate:

```
#include <stdio.h>

extern double _H(double *qubit);

int main(){
    double qubit[2] = {0.0, 1.0};
    _H(qubit);
    printf("%f, %f\n", qubit[0], qubit[1]);
    return 0;
}
```

2.4 Controlled Not Gate

The Controlled NOT (CNOT) gate is a two-qubit quantum gate that flips the state of the target qubit if the control qubit is in the $|1\rangle$ state. It acts as follows:

$$\text{CNOT}|00\rangle = |00\rangle, \quad \text{CNOT}|01\rangle = |01\rangle,$$

$$\text{CNOT}|10\rangle = |11\rangle, \quad \text{CNOT}|11\rangle = |10\rangle.$$

Mathematically, it is represented by the matrix:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The CNOT gate is a fundamental two-qubit operation in quantum computing and is crucial for creating entanglement between qubits. Example use of the Quplexity CNOT gate:

```
#include <stdio.h>

extern double _CNOT(double *qubit, double *qubit2);

int main(){
    double qubit[2] = {0.0, 1.0};
    double qubit2[2] = {0.0, 1.0};
    _CNOT(qubit, qubit2);
    printf("%f, %f\n", qubit2[0], qubit2[1]);
    return 0;
}
```

2.5 Controlled Controlled Not Gate

The Controlled-Controlled NOT (CCNOT) gate, or Toffoli gate, is a three-qubit quantum gate. It flips the state of the target qubit if and only if both control qubits are in the $|1\rangle$ state. The operation is defined as follows:

$$\text{CCNOT}|abc\rangle = |ab(c \oplus (a \wedge b))\rangle,$$

where $\oplus$ represents addition modulo 2 (XOR), and $\wedge$ represents the logical AND operation.

The truth table for the CCNOT gate is:

$$\begin{aligned} |000\rangle &\to |000\rangle, & |100\rangle &\to |100\rangle, \\ |001\rangle &\to |001\rangle, & |101\rangle &\to |101\rangle, \\ |010\rangle &\to |010\rangle, & |110\rangle &\to |111\rangle, \\ |011\rangle &\to |011\rangle, & |111\rangle &\to |110\rangle. \end{aligned}$$

Mathematically, it is represented by the 8x8 matrix:

$$\text{CCNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

The CCNOT gate is significant in quantum computing for its ability to perform reversible universal classical computation and is commonly used in quantum algorithms. Example use of the Quplexity CCNOT gate:

```c
#include <stdio.h>

extern double _CCNOT(double *qubit, double *qubit2, double *qubit3);

int main(){
    double qubit[2] = {0.0, 1.0};
    double qubit2[2] = {0.0, 1.0};
    double qubit3[2] = {0.0, 1.0};
    _CCNOT(qubit, qubit2, qubit3);
    printf("%f, %f\n", qubit3[0], qubit3[1]);
    return 0;
}
```

2.6 Controlled-Z Gate

The Controlled-Z (CZ) gate is a two-qubit quantum gate that applies a phase flip (Z gate) to the target qubit if the control qubit is in the $|1\rangle$ state. It acts as follows:

$$\mathrm{CZ}|00\rangle = |00\rangle, \quad \mathrm{CZ}|01\rangle = |01\rangle,$$
$$\mathrm{CZ}|10\rangle = |10\rangle, \quad \mathrm{CZ}|11\rangle = -|11\rangle.$$

Mathematically, it is represented by the matrix:

$$\mathrm{CZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

The CZ gate is essential in quantum computing for creating entangled states and is often used in conjunction with other gates to build quantum circuits. Example use of the Quplexity CZ gate:

```c
#include <stdio.h>

extern double _CZ(double *qubit, double *qubit2);

int main(){
    double qubit[2] = {0.0, 1.0};
    double qubit2[2] = {0.0, 1.0};
    _CZ(qubit, qubit2);
    printf("%f, %f\n", qubit2[0], qubit2[1]);
    return 0;
}
```

2.7 SWAP Gate

The SWAP gate is a two-qubit quantum gate that exchanges the states of two qubits. It acts as follows:

$$\mathrm{SWAP}|a, b\rangle = |b, a\rangle,$$

where $|a\rangle$ and $|b\rangle$ are the states of the two qubits.

Mathematically, the SWAP gate is represented by the matrix:

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The SWAP gate is useful in quantum computing for rearranging qubits in a circuit. Example use of the Quplexity SWAP gate:

```
#include <stdio.h>

extern double _SWAP(double *qubit, double *qubit2);

int main(){
    double qubit[2] = {1.0, 0.0};
    double qubit2[2] = {0.0, 1.0};
    _SWAP(qubit, qubit2);
    printf("%f, %f\n", qubit[0], qubit[1]);
    printf("%f, %f\n", qubit2[0], qubit2[1]);
    return 0;
}
```

2.8 FREDKIN (Controlled-SWAP) Gate

The Fredkin gate, also known as the Controlled-SWAP gate, is a three-qubit quantum gate that swaps the states of two target qubits if and only if the control qubit is in the $|1\rangle$ state. It acts as follows:

$$\text{Fredkin}|c, a, b\rangle = \begin{cases} |c, b, a\rangle, & \text{if } c = 1, \\ |c, a, b\rangle, & \text{if } c = 0. \end{cases}$$

Mathematically, the Fredkin gate is represented by the 8x8 matrix:

$$\text{Fredkin} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The Fredkin gate is significant in reversible computing and is used in quantum circuits to conditionally exchange qubits. Example use of the Quplexity SWAP gate: **(Not Currently Working)**

```
#include <stdio.h>

extern double _FREDKIN(double *qubit, double *qubit2);

int main(){
    double qubit[2] = {1.0, 0.0};
    double qubit2[2] = {0.0, 1.0};
    _FREDKIN(qubit, qubit2);
    printf("%f, %f\n", qubit[0], qubit[1]);
    printf("%f, %f\n", qubit2[0], qubit2[1]);
    return 0;
}
```

2.9 Controlled Phase Shift (CP) Gate

The Controlled Phase Shift (CP) gate is a two-qubit quantum gate that applies a phase shift to the target qubit if the control qubit is in the $|1\rangle$ state. The operation is defined as follows:

$$\mathrm{CP}|ab\rangle = \begin{cases} |ab\rangle, & \text{if } a = 0, \\ e^{i\phi}|ab\rangle, & \text{if } a = 1, \end{cases}$$

where a represents the state of the control qubit, b represents the state of the target qubit, and ϕ is the phase angle.

The truth table for the CP gate is:

$$\begin{aligned} |00\rangle &\to |00\rangle, \qquad & |10\rangle &\to |10\rangle, \\ |01\rangle &\to |01\rangle, \qquad & |11\rangle &\to e^{i\phi}|11\rangle. \end{aligned}$$

Mathematically, it is represented by the 4×4 matrix:

$$\mathrm{CP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix}.$$

The CP gate is significant in quantum computing for implementing controlled phase shifts, which are essential for many quantum algorithms, including those that leverage entanglement and interference.

Example use of the Quplexity CP gate:

```
#include <stdio.h>

extern void _CP(double* control_qubit, double* target_qubit,
                double* phase_angle);

int main(){
    double control_qubit[2] = {1.0, 0.0};
    double target_qubit[2] = {0.0, 1.0};
    double phase_angle[2] = {3.14159 / 2.0, 0.0};

    _CP(control_qubit, target_qubit, phase_angle);

    printf("%f, %f)\n", target_qubit[0], target_qubit[1]);
    return 0;
}
```

3. Intel and x86 Functions

CURRENTLY BEING WRITTEN...

4. Using Quplexity in a Python Project