Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

BAB II

LANDASAN TEORI

2.1 Quantum Bit

Quantum Bit (*qubit*) pada komputer kuantum memiliki sebuah *state* yang sama seperti bit pada komputer klasik, yaitu 0 atau 1 (Nielsen dan Chuang, 2010). *State* yang mungkin pada *qubit* adalah *state* $|0\rangle$ atau $|1\rangle$. Notasi $|\ \rangle$ disebut sebagai notasi Dirac yang digunakan sebagai standar notasi pada mekanika kuantum (Nielsen dan Chuang, 2010). Perbedaan *qubit* dengan bit adalah *qubit* memiliki *state* selain $|0\rangle$ atau $|1\rangle$. *Qubit* dapat memiliki *state* kombinasi linear dari dua atau lebih *state* yang disebut dengan *superposition* (Nielsen dan Chuang, 2010). Kombinasi linear dapat dilihat pada Persamaan 2.1.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \ldots(2.1)$$

dengan $|\psi\rangle$ merupakan *wave function* sedangkan α dan β merupakan bilangan kompleks yang memenuhi Persamaan 2.2 .

$$|\alpha|^2 + |\beta|^2 = 1 \quad \ldots(2.2)$$

Ketika *qubit* berada pada *state* $\alpha|0\rangle + \beta|1\rangle$, *qubit* dapat direpresentasikan dalam bentuk vektor 2 dimensi (Nielsen dan Chuang, 2010) seperti

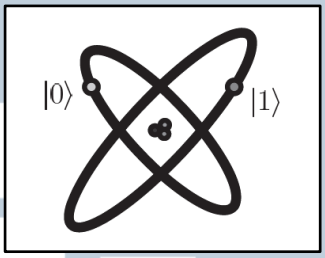
$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$$

dengan α melambangkan *amplitude* dari *state* $|0\rangle$ dan β melambangkan *amplitude* dari *state* $|1\rangle$.

Qubit juga dapat direpresentasikan dengan dua elektron pada sebuah atom. Pada Gambar 2.1, terdapat dua buah *state* yaitu $|0\rangle$ dan $|1\rangle$. Dengan memberikan cahaya yang cukup pada atom, *state* dapat berubah dari $|0\rangle$ ke $|1\rangle$ dan sebaliknya

Implementasi algoritma kuantum..., Anthony, FTI UMN, 2019

(Nielsen dan Chuang, 2010). Tetapi, dengan mengurangi cahaya yang diberikan, *state* dapat berada diantara $|0\rangle$ dan $|1\rangle$.



Gambar 2.1 Representasi *qubit* dengan dua elektron pada sebuah atom (Nielsen dan Chuang, 2010)

2.2 Quantum Gate

Quantum gate digunakan untuk membawa dan memanipulasi informasi kuantum (Nielsen dan Chuang, 2010). *Quantum gate* pada *single qubit* dapat direpresentasikan dengan matriks 2 x 2. Syarat agar matriks tersebut dapat digunakan sebagai *quantum gate* adalah matriks tersebut harus *unitary*, yaitu $U^{\dagger}U = I$. $U^{\dagger}$ adalah *adjoint* dari matriks U yang didapat dengan tranposisi dan konjugasi dari matriks U, sedangkan I adalah matriks identitas. Salah satu *quantum gate* yang penting yaitu Hadamard *gate* (Nielsen dan Chuang, 2010). Hadamard *gate* dapat dilihat pada Persamaan 2.3.

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad ...(2.3)$$

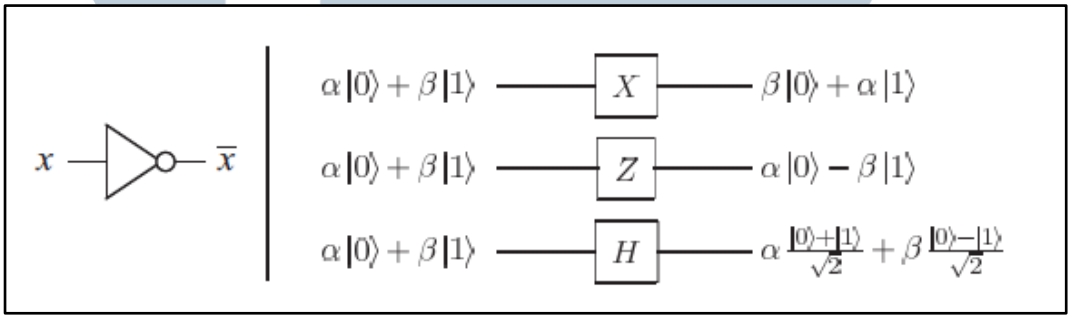
Sebagai contoh, jika Hadamard *gate* dioperasikan pada sebuah *qubit*, maka matriks yang dihasilkan sebagai berikut.

$$H \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} \alpha + \beta \\ \alpha - \beta \end{bmatrix} \quad ...(2.4)$$

UNIVERSITAS MULTIMEDIA NUSANTARA

state $|0\rangle$ memiliki *amplitude* $\frac{1}{\sqrt{2}}\alpha + \frac{1}{\sqrt{2}}\beta$ dan dengan Persamaan 2.2 maka *state* $|0\rangle$ memiliki probabilitas sebesar 0,5 untuk hasil 0 dan probabilitas sebesar 0,5 untuk hasil 1. *State* $|1\rangle$ memiliki *amplitude* $\frac{1}{\sqrt{2}}\alpha - \frac{1}{\sqrt{2}}\beta$ dan dengan Persamaan 2.2 maka *state* $|1\rangle$ memiliki probabilitas sebesar 0,5 untuk hasil 0 dan probabilitas sebesar 0,5 untuk hasil 1.

Beberapa *single qubit gate* yang penting dapat dilihat pada Gambar 2.2. X melambangkan NOT *gate* dan Z *gate* berfungsi untuk membalikkan tanda pada state (Nielsen dan Chuang, 2010). Namun pada penelitian ini hanya digunakan Hadamard *gate*.



Gambar 2.2 Beberapa *single qubit gate* (Nielsen dan Chuang, 2010)

2.3 Algoritma Kuantum Grover

Permasalahan yang diselesaikan dari algoritma kuantum Grover adalah terdapat data sebanyak N dimana hanya ada 1 data yang memenuhi persyaratan, dan data tersebut harus diambil (Grover, 1996). Algoritma kuantum Grover adalah algoritma untuk mencari sebuah data pada *database* tidak terstruktur dalam waktu $O(\sqrt{N})$. *Input* dari algoritma ini adalah *state* sebanyak $N = 2^n$ *state* dengan n adalah jumlah *qubit* (Grover, 1996). *State-state* tersebut direpresentasikan dalam bentuk *n bitstring* dan terdapat sebuah *state* yang memenuhi persyaratan (Grover,

Implementasi algoritma kuantum..., Anthony, FTI UMN, 2019

1996). *Output* yang dihasilkan yaitu sebuah *state* yang memenuhi persyaratan tersebut.

Dengan menggunakan *unitary matrix* yang disebut Oracle, data dapat direpresentasikan dalam *qubit* pada komputer kuantum (IBM Research and The IBM QX team, 2017). Fungsi Oracle mengembalikan nilai 1 untuk data yang sesuai dan mengembalikan nilai 0 untuk data yang tidak sesuai. Oracle *matrix* U_f dapat dilihat pada Persamaan 2.5

$$U_f|x\rangle = (-1)^{f(x)}|x\rangle \quad ...(2.5)$$

dengan $|x\rangle$ adalah *state quantum* dan $f(x)$ merupakan fungsi Oracle.

Algoritma ini dimulai dengan komputer berada pada *state* $|0\rangle^{\otimes n}$ (Nielsen dan Chuang, 2010). Hadamard *transform* dilakukan pada setiap *qubit* untuk membuat komputer berada pada *state superposition* seperti pada persamaan 2.6

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \quad ...(2.6)$$

Setelah dilakukan Hadamard *transform* pada setiap *qubit*, setiap *state* memiliki *amplitude* yang sama, yaitu $\frac{1}{\sqrt{N}}$. Langkah ini didapat dalam $O(\log N)$ langkah (Grover, 1996).

Langkah selanjutnya yaitu proses *amplitude amplification*, dimana *amplitude* pada *state* yang sesuai akan diperkuat, sedangkan *amplitude* pada *state* yang tidak sesuai akan diperkecil (Hui, 2018). Proses *amplification* ini diperoleh dari pengulangan sebanyak $\frac{\pi}{4}\sqrt{N}$ (Boyer, dkk., 1996). Pengulangan tersebut berisi langkah penerapan fungsi Oracle dan Diffusion *transform* (Grover, 1996). Fungsi Oracle akan membalikkan *amplitude state* yang sesuai, sedangkan *amplitude state* yang tidak sesuai tidak berubah. Diffusion *transform* akan membalikkan

Implementasi algoritma kuantum..., Anthony, FTI UMN, 2019

amplitude disekitar rata-rata *amplitude*. Matriks Diffusion D dapat dilihat pada Persamaan 2.7

$$D_{ij} = \frac{2}{N} \; if \; i \neq j \; \& \; D_{ii} = -1 + \frac{2}{N} \qquad ...(2.7)$$

dengan i adalah indeks baris dan j adalah indeks kolom.

Setelah dilakukan proses perulangan, akan terdapat sebuah *state* unik, dimana fungsi Oracle mengembalikan nilai 1 terhadap *state* tersebut, yang memiliki probabilitas lebih dari sama dengan 0,5 (Grover, 1996). Lakukan *measure* kepada hasil *state-state* tersebut.

2.4 Quantum Computing Playground

Quantum Computing Playground merupakan WebGL Chrome Experiment berbasis web yang dibuat oleh sekelompok *engineer* Google pada tahun 2014 (Quantum Computing Playground, 2014). Jumlah *qubit* yang disediakan memiliki ukuran antara 6 sampai 22 *qubit.* Simulasi komputer kuantum dapat dilakukan dengan menggunakan *script language* QScript (Quantum Computing Playground, 2014. Terdapat beberapa simulasi yang dapat dilakukan, salah satunya adalah simulasi algoritma Grover. Langkah-langkah dan visualisasi dari algoritma Grover dapat dilihat pada Quantum Computing Playground ini. Selain itu, informasi seperti *amplitude* juga bisa didapat.

Waktu eksekusi berisi informasi *simulated time*, *user time, system time*, dan *elapsed time* (Wicaksana dan Tang, 2017). *Simulated time* adalah waktu simulasi dalam *loosely timed* (Wicaksana dan Tang, 2017). *User time* adalah *wall-clock time* yang ditetapkan pada simulator (Wicaksana dan Tang, 2017). *System time* adalah waktu yang digunakan untuk melakukan proses sistem pada simulator (Wicaksana dan Tang, 2017). *Elapsed time* sesuai dengan *wall-clock time* dari

Implementasi algoritma kuantum..., Anthony, FTI UMN, 2019

mulai eksekusi sampai selesai (Wicaksana dan Tang, 2017). *Wall-clock time* adalah waktu dari mulai eksekusi sampai selesai, termasuk waktu ketika menunggu aktivitas sistem atau aplikasi lain jika ada (Black, dkk., 2009). Performa Quantum Computing Playground diukur dari sisi waktu eksekusi *user time*.

2.5 Rigetti Forest SDK

Rigetti Forest SDK dibuat oleh Rigetti Computing pada tahun 2019. Rigetti Forest SDK digunakan untuk melakukan pemrograman kuantum dan simulasi komputasi (Rigetti Computing, 2019). Untuk membuat program kuantum di Forest, diperlukan objek Program (Rigetti Computing, 2019). Untuk membuat objek Program, dapat dilihat potongan *code* pada Gambar 2.3.

```
from pyquil import Program
from pyquil.gates import *

p = Program()
```

Gambar 2.3 Pembuatan objek Program dengan pyQuil (Rigetti Computing, 2019)

Untuk mengimplementasikan *standard gate*, seperti Hadamard dan CNOT *gate*, dapat dilihat pada Gambar 2.4.

```
from pyquil import Program
from pyquil.gates import *

p = Program()
ro = p.declare('ro', 'BIT', 2)
p += H(0)
p += CNOT(0, 1)
```

Gambar 2.4 Implementasi Hadamard dan CNOT *gate* dengan pyQuil (Rigetti Computing, 2019).

Implementasi algoritma kuantum..., Anthony, FTI UMN, 2019

Selain *standard gate*, *gate* baru dapat ditambahkan ke dalam program. *Gate* baru direpresentasikan dalam bentuk *array* 2 dimensi. Gambar 2.5 merupakan potongan kode untuk membuat *gate* baru.

```
import numpy as np

from pyquil import Program
from pyquil.quil import DefGate

# First we define the new gate from a matrix
sqrt_x = np.array([[ 0.5+0.5j,  0.5-0.5j],
                   [ 0.5-0.5j,  0.5+0.5j]])

# Get the Quil definition for the new gate
sqrt_x_definition = DefGate("SQRT-X", sqrt_x)
# Get the gate constructor
SQRT_X = sqrt_x_definition.get_constructor()

# Then we can use the new gate
p = Program()
p += sqrt_x_definition
p += SQRT_X(0)
```

Gambar 2.5 Implementasi kode pembuatan *gate* baru
(Rigetti Computing, 2019).

Wavefunction simulator dapat digunakan untuk memeriksa *wavefunction* dari *state quantum* pada program. *Wavefunction* berfungsi untuk *debugging* pada sistem kuantum kecil dan tidak mungkin diperoleh pada prosesor kuantum (Rigetti Computing, 2019). Gambar 2.6 merupakan potongan kode untuk *wavefunction simulator*.

```
from pyquil import Program
from pyquil.gates import *
from pyquil.api import WavefunctionSimulator
wf_sim = WavefunctionSimulator()
coin_flip = Program(H(0))
wf_sim.wavefunction(coin_flip)
```

Gambar 2.6 Implementasi kode *wavefunction simulator*
(Rigetti Computing, 2019)

UNIVERSITAS MULTIMEDIA NUSANTARA