Devoir d'information quantique
[IF217]

Compte-Rendu 03/10/2007

Évolution d'un registre de n qubits dans un circuit quantique à deux portes

Encadrant :

Yves LEROYER

Binôme { Marouane BEN LAKHAL
Mohamed Amine EL AFRIT

Table des matières

Chapitre 1

Introduction

On souhaite simuler l'évolution d'un registre de n qubits dans un circuit quantique ne comportant que des portes *Hadamard* et *C_NOT*.
On a choisi de réaliser cette simulation en language *C++*.
Dans ce compte rendu le lecteur trouvera les explications nécessaires pour l'utilisation du programme d'une part et quelques points techniques sur lesquels on s'est basé afin de réaliser le projet d'autre part.

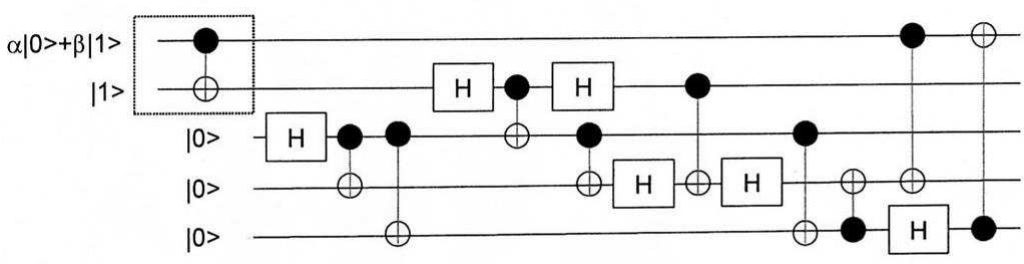

FIG. 1.1 – schéma du circuit

Chapitre 2

Conception

Dans toute cette patie n désigne la tailledu registre quantique.

2.1 Quelques éléments théoriques

Chaque porte quantique peut être representée par une matrice de taille $2^n \times 2^n$. Sachant qu'une porte agit sur un qubit ou un ensemble de qubits, il peut y exister plusieurs matrices dépendant des positions des qubits sur lesquels elles agissent.

2.2 Porte de Hadamard

Soit H une porte de *Hadamard* et $|q>=|q_1q_2...q_n>$ un système de n qubits. On peut affirmer qu'il existe n matrices de *Hadamard* H_i où i est la position de qubit sur lequel elle agit. Les autres qubits ne sont pas modifiés.

2.3 Porte *C-NOT*

Une porte *C-NOT* dépend de la position du bit de contrôle et de celle du bit cible. Ainsi on peut construire 2^{n-1} matrices *C-NOT* pour un système de n qubits.

Chapitre 3

Réalisation

Dans le rÃ©pertoire *Quantique* l'utilisateur peut trouver un makefile pour compiler le projet ainsi que les fichiers sources associés.
Le fichier *circuit.cpp* permet à l'utilisateur de définir sont circuit quantique et de simuler son fonctionnement.

3.1 Diagramme de classes

La classe *Matrice* définit un type abstrait de données réalisant plusieurs opérations, l'une de ces opérations est celle de la multiplication qui a été surchargée afin d'améliorer la lisibilité du code.

3.2 Implémentation de la porte de *Hadamard*

Cette implémentation est réalisée dans le fichier *Hadamard.cpp*. Le prototype du constructeur de cette classe est le suivant :

$$Hadamard \ \ (\ int\ Nqbit, int\ pemeBit\ )$$

Telle que :

Nqubit : est le nombre des qubits du registre d'entree
pemeBit : est la position du bit d'action de la porte.

Remarque

Hadamard est une matrice de la forme $\frac{1}{\sqrt{2}} \times H'$ où H' est une matrice dont les éléments sont des entiers. Pour des raisons de simplifications on a omis le $\frac{1}{\sqrt{2}}$. En revanche, pour ne pas faire d'erreurs on compte le nombre p de portes de *Hadamard* dans le circuit et on multiplie le résultat final par $\frac{1}{2^{\frac{p}{2}}}$

3.3 Implémentation de la porte *C-NOT*

La classe *C-NOT* est implémentée dans le fichier *C-Not.cpp*. Le prototype du constructeur de cette classe est le suivant :

$$C-Not \ \ (\ int\ Nqbit, int\ bControle, int\ bCible\ )$$

Telle que :

Nqubit : est le nombre des qubits du registre d'entree
bontrole : la position du bit de controle
bCible : La position du bit cible.

Chapitre 4

Résultat

Le circuit donné en exemple dans l'énoncé a été implémenté dans le fichier *circuit.cpp*.
Les valeurs de α et β sont données en macros dans le même fichier.

Exemples d'exécutions

α	β	$\|\psi>$
1	0	$\|01>$
0	1	$\|10>$
$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}} \times \|01> + \frac{1}{\sqrt{2}} \times \|10>$
$\frac{1}{\sqrt{3}}$	$\frac{\sqrt{2}}{\sqrt{3}}$	$\frac{1}{\sqrt{3}} \times \|01> + \frac{\sqrt{2}}{\sqrt{3}} \times \|10>$

Chapitre 5

Conclusion

Selon les résultats précédents on peut postuler que

$$|\psi>=\alpha|01>+\beta|10>$$

On peut alors déduire que si l'on effectue une mesure sur le 1^{er} qubit son état sera projeté sur l'état $|0>$ avec une probabilié de $|\alpha|^2$. Et si l'on effectue une mesure sur le 2^{eme} qubit son état sera projeté sur l'état $|1>$ avec la même probabilié $|\alpha|^2$

Cela nous ramène à déduire que la mesure du 1^{er} qubit projete systématiquement l'état du 2^{eme} qubit dans l'état opposé du 1^{er}.

```
#include <cstdio>
#include <math.h>
#include "Matrice.hpp"
#include "Hadamard.hpp"
#include "C_not.hpp"

#define ALPHA 1/sqrt(3)
#define BETA sqrt(2)/sqrt(3)

int main()
{
  Hadamard h2(5,2);
  Hadamard h3(5,3);
  Hadamard h4(5,4);
  Hadamard h5(5,5);

  C_not c12(5,1,2);
  C_not c34(5,3,4);
  C_not c35(5,3,5);
  C_not c23(5,2,3);
  C_not c24(5,2,4);
  C_not c54(5,5,4);
  C_not c14(5,1,4);
  C_not c51(5,5,1);

  float* vect = new float[32];
  float* res;
  int i;

  vect[8]=ALPHA;
  vect[24]=BETA;

  res =
    c51*
    (h5*
     (c14*
      (c54*
       (c35*
        (h4*
         (c24*
          (h4*
           (c34*
            (h2*
             (c23*
              (c35*
               (c34*
                (h3*
                 (h2*
                  (c12*
                   vect)))))))))))))));

  for (i=0;i<32;i++)
    if (res[i]!=0)
      printf(" %fl%d> +",res[i],i);

  printf("\b ");
  printf("\n");

  delete[] res;
  return 0;
}
```

```
#ifndef MATRICE_HPP
#define MATRICE_HPP
class Matrice
{
protected :
  int _taille;
  int** _tab;
public:
  Matrice();
  Matrice(int);
  ~Matrice();
  int * produit(int *) ;
  int get_element(int,int);
  void set_element(int,int,int);
  void afficher();
  int pemeBit(int,int);
  float* operator*(float*) const;
};
#endif
```

```cpp
#include <cstdio>
#include <math.h>
#include "Matrice.hpp"

Matrice::Matrice():_taille(32)
{
  int i;
  _tab = new int*[32];
  for (i=0;i<32;i++)
    _tab[i] = new int[32];
}

Matrice::Matrice(int Nqbit)
{
  int i;
  _taille = (int)pow(2,Nqbit);
  _tab = new int*[_taille];
  for (i=0;i<_taille;i++)
    _tab[i] = new int[_taille];
}

int Matrice::get_element(int i,int j)
{
  return _tab[i][j];
}

void Matrice::set_element(int elt,int i,int j)
{
  _tab[i][j]=elt;
}

void Matrice::afficher()
{
  int i,j;
  for (i=0;i<_taille;i++)
    for (j=0;j<_taille;j++)
      if(j==0)
        printf("|%d ",_tab[i][j]);
      else if (j==_taille-1)
        printf("%d|\n",_tab[i][j]);
      else
        printf("%d ",_tab[i][j]);
}

int Matrice::pemeBit(int elt,int pemePos)
{
  int x = (int)pow(2,pemePos);
  return (elt&x)!=0;
}

float* Matrice::operator*(float* vecteur) const
{
  float* resultat = new float[_taille];
  int i,j;
  for (i=0;i<_taille;i++)
    resultat[i]=0;
  for (i=0;i<_taille;i++)
    for(j=0;j<_taille;j++)
      resultat[i]=resultat[i]+_tab[i][j]*vecteur[j];

  for(i=0;i<_taille;i++)
    {
      vecteur[i]=resultat[i];
    }
  delete[] resultat;
  return vecteur;
}
```



```cpp
}

Matrice::~Matrice()
{
  int i;
  for (i=0;i<_taille;i++)
    delete[] _tab[i];
  delete[] _tab;
}
```

```
#ifndef HADAMARD_HPP
#define HADAMARD_HPP
#include "Matrice.hpp"

class Hadamard : public Matrice
{
private:
  int _pemeBit;
public :
  Hadamard(int,int);
};
#endif
```

```
#include <cstdio>
#include <math.h>
#include "Hadamard.hpp"

Hadamard::Hadamard(int Nqbit,int pemeBit):Matrice(Nqbit),_pemeBit(pemeBit)
{
  int j=0;
  int d = (int)pow(2,Nqbit-_pemeBit);
  for (j=0;j<_taille;j++)
    if (this->pemeBit(j,Nqbit-_pemeBit)==0)
      {
        _tab[j][j]=1;
        _tab[j+d][j]=1;
      }
    else
      {
        _tab[j-d][j]=1;
        _tab[j][j]=-1;
      }
}
```

```
#ifndef C_NOT_HPP
#define C_NOT_HPP
#include "Matrice.hpp"

class C_not:public Matrice
{
public:
  C_not(int,int,int);
};
#endif
```

```
#include <cstdio>
#include "C_not.hpp"
#include "math.h"

C_not::C_not(int Nqbit, int bControle, int bCible):Matrice(Nqbit)
{
  int j;
  for(j=0;j<_taille;j++)
    if(this->pemeBit(j,Nqbit-bControle)==0)
      _tab[j][j]=1;
    else{
      int j1=j^(int)pow(2,Nqbit-bCible);
      _tab[j1][j]=1;
    }
}
```