Kvantové sčítání za použití metody Ripple-Carry

Michal Forgó

February 12, 2025

Contents

1 Úvod

Kvantové počítání přináší nové způsoby provádění aritmetických operací, včetně binárního sčítání. Tento dokument vysvětluje implementaci kvantového sčítání pomocí metody extbfripple-carry.

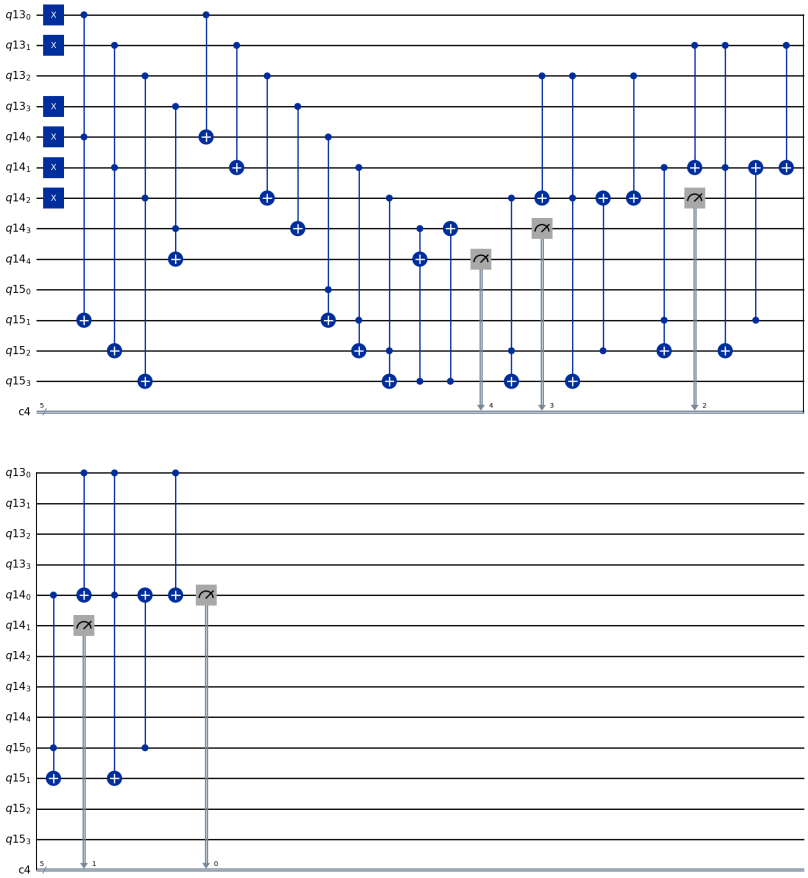
2 Sčítání s přenosem v kvantovém počítání

Metoda ripple-carry funguje tak, že postupně počítá součet dvou binárních čísel a propaguje přenosové bity. Proces zahrnuje:

1. Použití **CCNOT** hradla (CCX) k výpočtu přenosového bitu.
2. Použití **CNOT** hradla (CX) k výpočtu součtu na každé pozici.
3. Obrácení nepotřebných operací pro zajištění reverzibility kvantového výpočtu.

3 Reprezentace kvantového obvodu

Kvantové počítání umožňuje efektivní aritmetické operace pomocí reverzibilní logiky. Jednou ze základních operací v kvantové aritmetice je sčítání, které lze implementovat pomocí kvantových hradel. Diagram níže znázorňuje kvantový 4bitový ripple-carry sčítač, který provádí bitové sčítání pomocí řízených operací.



Obrázek 4: Reprezentace kvantového obvodu ripple-carry sčítání.

4 Použitá kvantová hradla

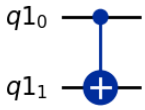
4.1 NOT

- **NOT (X) hradlo**: Používá se k nastavení vstupních hodnot v kvantových registrech. Překlápí qubit ze stavu $|0\rangle$ do $|1\rangle$ a naopak.

Obrázek 1: NOT hradlo.

4.2 CNOT

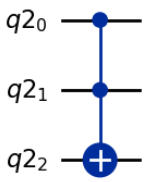
- **CNOT (CX) hradlo**: Řízené NOT hradlo, které překlápí cílový qubit, pokud je řídicí qubit ve stavu $|1\rangle$. Používá se pro výpočet součtu.



Obrázek 2: CNOT hradlo.

4.3 CCNOT

- **CCNOT (CCX) hradlo**: Řízené řízené NOT hradlo, které funguje jako generátor přenosu, překlápějící cílový qubit, pokud jsou oba řídicí qubity ve stavu $|1\rangle$.



Obrázek 3: CCNOT hradlo.

5 Postupná analýza kódu

5.1 Inicializace

Nejprve je nutné importovat potřebné knihovny pro správnou funkci programu.

```
from qiskit import QuantumCircuit, transpile
from qiskit import QuantumRegister, ClassicalRegister
from qiskit_aer import AerSimulator
```

Následně program potřebuje metodu pro zadávání vstupů. Můžeme hodnoty pevně nastavit takto:

```
firstBinaryNumber = "11001"
secondBinaryNumber = "0101"
```

Nebo můžeme umožnit uživateli jejich zadání:

```
while True:
    firstBinaryNumber = input("Zadejte binární číslo: ")
    secondBinaryNumber = input("Zadejte další binární číslo: ")

    # Ověření správnosti vstupu
    if len(firstBinaryNumber) > 8 or len(secondBinaryNumber) > 8 or \
    not set(firstBinaryNumber).issubset({'0','1'}) or \
    not set(secondBinaryNumber).issubset({'0','1'}):
        print("Zadejte platné binární číslo. Zkuste to znovu.")
    else:
        break  # Ukončí cyklus při správném vstupu
```

Nezáleží na použité metodě, důležité je zajistit, aby vstup obsahoval pouze jedničky a nuly a nebyl příliš dlouhý. My jsme zvolily 7 bitů.

Pro provedení kvantového sčítání musíme alokovat prostor pro uložení hodnot. Nejprve určíme největší vstup a zjistíme jeho velikost. Na základě toho definujeme potřebné kvantové a klasické registry:

```
maxInputLength = max(len(firstBinaryNumber), len(secondBinaryNumber))

regA = QuantumRegister(maxInputLength)
regB = QuantumRegister(maxInputLength + 1)
regC = QuantumRegister(maxInputLength)
regD = ClassicalRegister(maxInputLength + 1)
```

- *regA* (Kvantový registr, *maxInputLength* qubitů)
  - Tento registr uchovává první binární číslo.
  - Každý qubit reprezentuje bit vstupu a je inicializován do stavu $|0\rangle$ nebo $|1\rangle$ podle hodnoty binárního čísla.
- *regB* (Kvantový registr, $maxInputLength + 1$ qubitů)
  - Tento registr uchovává druhé binární číslo a zároveň slouží jako výstupní registr.
  - Přebytečný qubit (+1) je nezbytný pro zohlednění možného přenosu při součtu.
- *regC* (Kvantový registr, *maxInputLength* qubitů)
  - Tento registr se používá pro uchování přenosových bitů během sčítání.
  - Dočasně uchovává informace, které zajišťují správné provedení sčítání simulací klasického přenosu.
- *regD* (Klasický registr, $maxInputLength + 1$ bitů)
  - Tento klasický registr slouží k uložení konečného změřeného výsledku.
  - Po provedení kvantového výpočtu je výstup změřen a uložen do *regD*.

V tuto chvíli jsou registry prázdné. Proto do nich musíme vložit naše data. Protože jsou všechny jednotlivé qubity ve výchozím stavu $|0\rangle$, stačí použít NOT hradlo na qubity, které chceme přepnout na $|1\rangle$.

```
for idx, val in enumerate(firstBinaryNumber):
    if val == "1":
        qc.x(regA[len(firstBinaryNumber) - (idx+1)])

for idx, val in enumerate(secondBinaryNumber):
    if val == "1":
        qc.x(regB[len(secondBinaryNumber) - (idx+1)])
```

5.2 Logika přenosových hradel pro kvantové sčítání

V této sekci implementujeme logiku přenosových hradel potřebnou pro provedení binárního sčítání v kvantovém obvodu. Proces zahrnuje výpočet přenosových hodnot, jejich aktualizaci a zajištění správného resetu pro udržení korektního fungování.

5.2.1 Výpočet a propagace přenosu

Přenosové bity jsou vypočítány a propagovány skrze obvod pomocí CCNOT (CCX) a CNOT (CX) hradel.

```
# Implementace logiky přenosových hradel
for i in range(maxInputLength - 1):
    qc.ccx(regA[i], regB[i], regC[i + 1])  # Výpočet přenosu
    qc.cx(regA[i], regB[i])  # Částečný součet
    qc.ccx(regC[i], regB[i], regC[i + 1])  # Aktualizace přenosu
```

Každá iterace smyčky zpracovává dvojici bitů z `regA` a `regB`, vypočítává přenos pomocí CCNOT hradla a aktualizuje částečný součet pomocí CNOT hradla.

5.2.2 Finální výpočet přenosu

Pro zpracování nejvýznamnějšího bitu (MSB) je proveden závěrečný výpočet přenosu, který zajišťuje správné šíření přenosu.

```
qc.ccx(regA[maxInputLength - 1], regB[maxInputLength - 1],
        regB[maxInputLength])
qc.cx(regA[maxInputLength - 1], regB[maxInputLength - 1])
qc.ccx(regC[maxInputLength - 1], regB[maxInputLength - 1],
        regB[maxInputLength])
```

5.3 Resetování přenosových bitů

Aby byla zajištěna správná funkce při následných výpočtech, přenosové operace jsou provedeny zpětně, čímž se všechny přenosové bity resetují do stavu $|0\rangle$.

```
# Vrácení poslední přenosové operace pro resetování stavu
qc.cx(regC[maxInputLength - 1], regB[maxInputLength - 1])

# Obrácení přenosových operací k resetování všech přenosových bitů do
for i in range(maxInputLength - 1):
    qc.ccx(regC[(maxInputLength - 2) - i],
        regB[(maxInputLength - 2) - i],
        regC[(maxInputLength - 1) - i])
    qc.cx(regA[(maxInputLength - 2) - i],
        regB[(maxInputLength - 2) - i])
    qc.ccx(regA[(maxInputLength - 2) - i],
        regB[(maxInputLength - 2) - i],
        regC[(maxInputLength - 1) - i])

    # Tyto operace překlápějí regB, pokud je řídicí bit |1>
    qc.cx(regC[(maxInputLength - 2) - i],
    regB[(maxInputLength - 2) - i])
    qc.cx(regA[(maxInputLength - 2) - i],
    regB[(maxInputLength - 2) - i])
```

Aplikací zpětných přenosových operací v opačném pořadí obvod zajistí, že všechny mezilehlé přenosové hodnoty se vrátí do původního stavu.

5.4 Měření konečného výsledku

Po dokončení výpočtu je konečný součet získán měřením qubitů v `regB` a uložením klasického výsledku do `regD`.

```
# Měření qubitů a uložení výsledků do klasického registru
for i in range(maxInputLength + 1):
    qc.measure(regB[i], regD[i])
```

Tento krok zhroucením kvantového stavu převede výstup na klasickou binární hodnotu, která reprezentuje součet dvou vstupních čísel.

5.5 Spuštění a vizualizace výsledků

Nyní, když je obvod kompletně sestaven, je potřeba jej simulovat, abychom získali výsledky. Inicializujeme kvantový simulátor, přeložíme obvod pro spuštění a necháme jej vypočítat výsledek.

```
simulator = Aer.get_backend('aer_simulator')
circ = transpile(qc, simulator)
result = simulator.run(circ).result()
counts = result.get_counts(circ)
print(*counts)
```

6 Závěr

Kvantové sčítání pomocí metody ripple-carry je klíčovým krokem k provádění složitějších aritmetických operací v kvantovém počítání. Použití bran CCNOT a CNOT zajišťuje logickou správnost výpočtu a zároveň zachovává kvantovou reverzibilitu.