Simulazione di Algoritmi Quantistici usando lo strumento Q#

Daniel Biasiotto

June 8, 2022

CONTENTS

1 INTRODUZIONE

Lo sviluppo di software quantistici si é confermato come un'area di grande interesse negli ultimi decenni, offrendo grandi possibilità di superare i limiti computazionali attualmente raggiunti in diverse aree di ricerca. É possibile che in futuro algoritmi quantistici possano sostituire controparti classiche in diverse applicazioni:

- crittografia
- problemi di ricerca, l'algoritmo di Grover in questo campo é quadraticamente più veloce del analogo classico
- utilizzo di computer quantistici per la simulazione di sistemi quantistici
- machine learning
- computazione biologica
- chimica generativa

Ovviamente il calcolo classico non verrà abbandonato, l'approccio classico e quello quantistico differiscono nelle loro forze e debolezze. Mentre gli attuali computer diventano sempre più veloci e le tecniche industriali permettono miniaturizzazioni sempre maggiori gli hardware quantistici rimangono estremamente complicati da costruire e distribuire. Il modo attualmente più congeniale di utilizzare questi hardware per effettuare testing di software rimane quello della condivisione di risorse attraverso le tecnologie di *cloud computing*.

Ma questo non significa che non sia possibile o non sia utile studiare i problemi e le soluzioni algoritmiche che il calcolo quantistico offre a livello teorico e di sviluppo software. Un importante punto di forza di questa tecnologia é che ci aspettiamo che i computer quantistici in generale siano molto più lenti di computer classici in termini di velocità di clock e architettura delle porte logiche ma che le risorse di cui dispongono nel contesto di alcune classi di problemi da risolvere scalino diversamente, in particolare permettendo per i problemi giusti, più adatti, di superare di gran lunga la *performance* dei migliori algoritmi classici. Questo perché l'approccio algoritmico permesso da questa tecnologia é profondamente diverso da quello classico e porta a miglioramenti importanti in termini di complessità computazionale. D'altra parte non ci si deve aspettare che questa sia una soluzione perfetta, ci sono problemi che rimarranno difficili da risolvere nonostante le nuove possibilità della computazione quantistica.

Per molto tempo l'approccio a questo campo é rimasto accessibile solo a matematici e fisici con le grandi conoscenze specifiche necessarie a comprendere appieno la meccanica quantistica e le sue sfumature tecniche. Questo sta cambiando velocemente negli ultimi

anni da quando l'industria ha cominciato a sviluppare strumenti e piattaforme che permettano a nuovi sviluppatori di interagire e imparare in quest'ambito astraendo dalla maggior parte della complessità della teoria matematica alla base di questo modello.

1.1 Definizione di un Computer Quantistico

Per prima cosa definiamo un *computer* come uno strumento che prendendo dei dati in input esegue delle operazioni su questi dati. I *computer* che conosciamo e utilizziamo attualmente possono e sono definiti nei confini teorici della fisica classica, ovvero attraverso le leggi di Newton e l'elettromagnetismo. Data questa definizione é semplice definire un *computer quantistico* come uno strumento che prendendo dei dati in input esegue delle operazioni si questi dati con processi descrivibili solo utilizzando i concetti della fisica quantistica. Detto questo la differenza tra un computer classico e un computer quantistico é la stessa che esiste tra la fisica classica e quella quantistica. La principale differenza é la scala su cui operano in termini di dimensioni e energia. I calcolatori quantistici sono in ogni caso controllati tramite interfacce e strumenti classici con cui é più semplice interagire e tramite cui i dati sono riconvertiti in dati di tipo classico utilizzabili dalle `CPU` che comunemente utilizziamo.

1.2 Limiti Hardware

Attualmente i software sviluppati per hardware quantistici possono essere eseguiti su simulatori, sempre software, oppure sfruttando reali macchine quantistiche in remoto. Questo é necessario in quanto i requisiti e limiti tecnici della costruzione di un calcolatore quantistico sono estremamente complessi e richiedono ambienti fisici attualmente incompatibili con un utilizzo desktop. Questi limiti sono in particolare l'estrema sensibilità dei registri all'interno di un computer simile ai segnali elettromagnetici esterni, per ovviare a problemi di interferenza é necessario mantenere l'hardware in temperature vicine allo zero termico (0K o $-273.15°C$). Queste condizioni non sono ovviamente replicabili in un ambiente non strettamente controllato e isolato.

Dati questi limiti é facile dire che per molto tempo ancora questo tipo di hardware sarà limitato a un utilizzo da remoto, servizi come `Amazon Quantum` renderanno sempre più facilmente disponibile l'accesso a questi strumenti.

I simulatori e le macchine reali condividono interfacce condivise che permettono lo sviluppo di software che abbia la possibilità di essere testato in maniera indiscriminata su un qualsiasi di questi.

1.3 Utilizzi della tecnologia

I computer quantistici offrono nuove possibilità nella risoluzione di diverse classi di problemi, la ricerca in questo campo ha trovato alcuni esempi di algoritmi che utilizzando le proprietà particolari di questo approccio per sviluppare un vantaggio nei confronti dell'approccio classico.

Ad esempio:

- L'**algoritmo di Grover** effettua una ricerca in una lista di N elementi in tempo $O(\sqrt{N})$
- L'**algoritmo di Shor** fattorizza velocemente grandi numeri, in particolare permette di fattorizzare con un grado di errore arbitrariamente piccolo e un numero polinomiale di passi rispetto alla lunghezza in bit dell'input
- L'**algoritmo di Deutsch-Jozsa** verifica se una funzione é costante o bilanciata in tempo costante $O(1)$
- L'**algoritmo di Simon**, ispirazione per il sopraccitato algoritmo di Shor, risolve in tempo esponenzialmente più veloce rispetto all'approccio classico il problema di determinare se una data funzione f *blackbox* sia **uno-a-uno** o **due-a-uno**

In aggiunta ai problemi di cui sopra, la cui soluzione é legata a un algoritmo in particolare, sono state trovate applicazioni per i computer quantistici in diversi altri ambiti:

- La stima di una somma di Gauss, un tipo di somma esponenziale, con precisione polinomiale e in tempo polinomiale contro il tempo esponenziale degli algoritmi classici
- La valutazioni di formule booleane complesse può essere velocizzata tramite un approccio quantistico
- Questi computer permettono di simulare sistemi quantistici permettendone uno studio più approfondito
- La generazioni di numeri casuali é un'importante componente della crittologia e utilizzando tecniche quantistiche é possibile generarne che siano davvero casuali e non più pseudo-casuali come necessario in computer classici

Questi sono risultati importanti e in particolare l'algoritmo di Shor pone dei dubbi sulla sicurezza degli attuali protocolli crittografici che si basano sulla difficoltà computazionale della fattorizzazione di grandi numeri interi. Se tale algoritmo fosse facilmente eseguibile significherebbe che un attaccante potrebbe facilmente violare questi

protocolli di sicurezza correntemente alla base della comunicazione via Internet.

Rimane difficile trovare possibili algoritmi quantistici che diano un vantaggio computazionale nel campo dell'apprendimento automatico dove é fondamentale l'accesso casuale a una grande quantità di dati.

In generale é più probabile che un problema che abbia una piccola mole di dati in entrata e in uscita ma una grande quantità di manipolazioni per arrivare all'output sia un buon candidato per l'utilizzo di computer quantistici.

2 AMBIENTE

Per lo sviluppo di software quantistici sono disponibili diversi ambienti e framework, tra i più conosciuti troviamo **Microsoft Azure** con il proprio Quantum Development Kit (QDK) o l'ambiente di sviluppo di IBM **Qiskit**. Altri *Software Development Kit* che possono essere utilizzati per eseguire circuiti quantistici su prototipi di device quantistici o simulatori sono:

- Ocean
- ProjectQ
- Forest
- t|ket>
- Strawberry Fields
- PennyLane

Molti di questi progetti sono open-source e sviluppati sulla base di Python.

Per questo lavoro abbiamo utilizzato gli strumenti offerti da Microsoft per l'ottima documentazione consultabile sulle loro pagine web e in quanto questo strumento era utilizzato dalla nostra fonte principale *Learn Quantum Computing with Python and Q#*.

L'ambiente di esecuzione Q# può essere configurato sul editor Visual Studio Code tramite l'add-on proprietario di Microsoft. Quest'ultimo é disponibile solo sulla versione non FOSS del software, che é possibile installare tramite le repository opensource linux.

In alternativa o anche parallelamente é possibile sviluppare codice Q# ed eseguirlo tramite Jupyter Notebook tramite Python. Questo con i kernel necessari installati, avendo quindi l'ultima versione di dotnet disponibile.

Tramite anaconda si crea un ambiente con il necessario:

```
$ conda create -n qsharp-env -c microsoft qsharp
↪  notebook
$ conda activate qsharp-env
```

L'esecuzione del software Q# può essere testato localmente predisponendo un ambiente di simulazione tramite il pacchetto Python chiamato qsharp.

Ad esempio come nel listato qui sopra utilizziamo uno script host.py per creare un ambiente di simulazione per poter eseguire le operazioni Q# definite in Operation_One e Operation_Two. Il pacchetto automaticamente va a cercare nella directory locale le definizioni.

```
import qsharp
from QsharpNamespace import Operation_One, Operation_Two
var1 = 10
print("Simulation started...")
Operation_One.simulate(par1=var1)
Operation_Two.simulate(par2=var1,par3=5)
```

Listing 1: host.py

```
import qsharp

prepare_qubit = qsharp.compile("""
    open Microsoft.Quantum.Diagnostics;

    operation PrepareQubit(): Unit {
        using (qubit = Qubit()) {
            DumpMachine();
        }
    }
""")

if __name__ == "__main__":
    prepare_qubit.simulate()
```

Listing 2: qsharp-interop.py

Un esempio più complesso può essere quello definito in `qsharp-inteop.py` dove definiamo direttamente *inline* il contenuto del codice `Q#` che il `package qsharp` compila e simula.

3 ORACOLI

Gli oracoli che utilizziamo per testare gli algoritmi definiti in seguito sono: Qui sopra sono definite le versioni a singolo qbit e a n-qbit degli oracoli quantistici di alcune funzioni booleane costanti e bilanciate. In particolare abbiamo definito oracoli per le seguenti funzioni:

- $f(x) = 0$
- $f(x) = 1$
- $f(x) = x$
- $f(x) = \neg x$ ovvero $f(x) = 1 - x$
- $f(x, y) = x \oplus y$
  - dove x é l'input lungo n qbit e y é l'output

In questi casi le prime due funzioni sono costanti e le restanti sono bilanciate.

In `figura 1` vediamo un'altro esempio di oracolo bilanciato che applica 3 porte `CNOT` all'ultimo qubit:

- $q_3 = q_3 \oplus q_0 \oplus q_1 \oplus q_2$

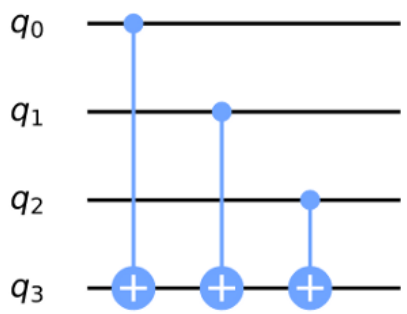

Figure 1: esempio di oracolo bilanciato utilizzando porte CNOT

4 ALGORITMO DI DEUTSCH-JOZSA

L'algoritmo di Deutsch-Jozsa ha interesse storico in quanto primo algoritmo quantistico in grado di superare in performance il miglior algoritmo classico corrispondente, mostrando che possono esistere vantaggi nel calcolo quantistico. Questo algoritmo ha spinto la ricerca in questa direzione per determinati problemi.

```
operation ApplyZeroOracle(control : Qubit, target :
↪  Qubit) : Unit {
}

operation ApplyOneOracle(control : Qubit, target :
↪  Qubit) : Unit {
  X(target);
}

operation ApplyZeroOracleN(control : Qubit[], target
↪  : Qubit) : Unit {
}

operation ApplyOneOracleN(control : Qubit[], target :
↪  Qubit) : Unit {
  X(target);
}

operation ApplyIdOracle(control : Qubit, target :
↪  Qubit) : Unit {
  CNOT(control,target);
}

operation ApplyXOROracleN(control : Qubit[], target :
↪  Qubit) : Unit {
  for qubit in control {
      CNOT(qubit,target);
  }
}

operation ApplyNotOracle(control : Qubit, target :
↪  Qubit) : Unit {
  X(control);
  CNOT(control,target);
  X(control);
}
```

Listing 3: oracles.qs

L'algoritmo risponde a una domanda su una funzione f booleana con n bit in input

$$f(\{x_0, x_1, \cdots, x_n\}) \rightarrow 0 \text{ o } 1$$

Questa funzione su cui agisce l'algoritmo ha la proprietà di essere una di due forme:

- costante
- bilanciata

Definite come:

- Una funzione é costante se restituisce per tutti gli input $\{x_0, x_1, \cdots, x_n\}$ lo stesso risultato
- Una funzione é bilanciata se restituisce 0 esattamente per metà degli input, e 1 esattamente per metà degli input

4.1 La Soluzione Classica

Nella soluzione classica nel **caso migliore** due *query* all'oracolo sono sufficienti per riconoscere la funzione f come bilanciata. Per esempio supponiamo di avere due chiamate con i seguenti risultati:

$$f(0, 0, \cdots) \rightarrow 0$$

$$f(1, 0, \cdots) \rightarrow 1$$

Dato che é assunto che f sia *garantita* essere costante oppure bilanciata questi risultati ci dimostrano f come bilanciata.

Per quanto riguarda il caso peggiore tutte le nostre interrogazioni daranno lo stesso output, decidere in modo certo che f sia costante necessita di metà più uno interrogazioni. Dato che il numero di input possibili é 2^n questo significa che, nel caso peggiore, saranno necessarie $2^{n-1} + 1$ interrogazioni per essere certi che $f(x)$ sia costante.

4.2 La Soluzione Quantistica

Tramite la computazione quantistica é possibile risolvere questo problema con un'unica chiamata della funzione $f(x)$. Questo a patto che la funzione f sia implementata come un oracolo quantistico U_f, che mappi: $|x\rangle|y\rangle$ a $|x\rangle|y \oplus f(x)\rangle$ [1]

I passi dell'algoritmo in particolare sono:

1. prepara 2 registri di `qubit`, il primo di n `qubit` inizializzato a $|0\rangle$ e il secondo di un singolo `qubit` inizializzato a $|1\rangle$

1 dove $\oplus$ é l'addizione modulo 2 o `XOR`

2. applica `Hadamard` a entrambi i registri
3. applica l'oracolo quantistico U_f definito per f
4. a questo punto il secondo registro può essere ignorato, riapplica `Hadamard` al primo registro
5. misura il primo registro, questa risulta 1 per $f(x)$ costante e 0 altrimenti nel caso bilanciato

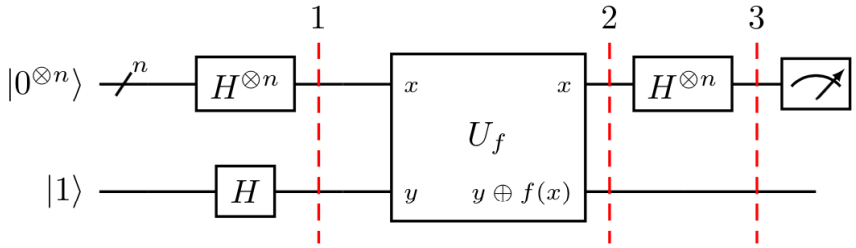

Figure 2: i passi dell'algoritmo in forma di circuito

Un punto fondamentale dell'algoritmo é l'utilizzo della porta `Hadamard`, chiamata anche trasformata di `Hadamard`. Questa é una generalizzazione delle trasformate di Fourier definita dalla matrice $H_m = 2^m \times 2^m$. Questa é definibile ricorsivamente a partire dall'identità $H_0 = 1$ e, per $m > 0$:

$$H_m = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix}$$

e quindi alcuni esempi di porte di `Hadamard` sono:

$$H_0 = +(1)$$

$$H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H_2 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Il trasformato di `Hadamard` H_1 é la porta logica quantistica conosciuta come porta `Hadamard`, l'applicazione di questa porta a ciascun qubit di un registro a n-qubit parallelamente é equivalente alla trasformata H_n.

Si crede che applicando un circuito di `Hadamard` a un qubit nello stato $|0\rangle$ si crei uno stato sovrapposto tra gli stati $|0\rangle$ e $|1\rangle$ denominato $|+\rangle$. A livello matematico sono definite:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

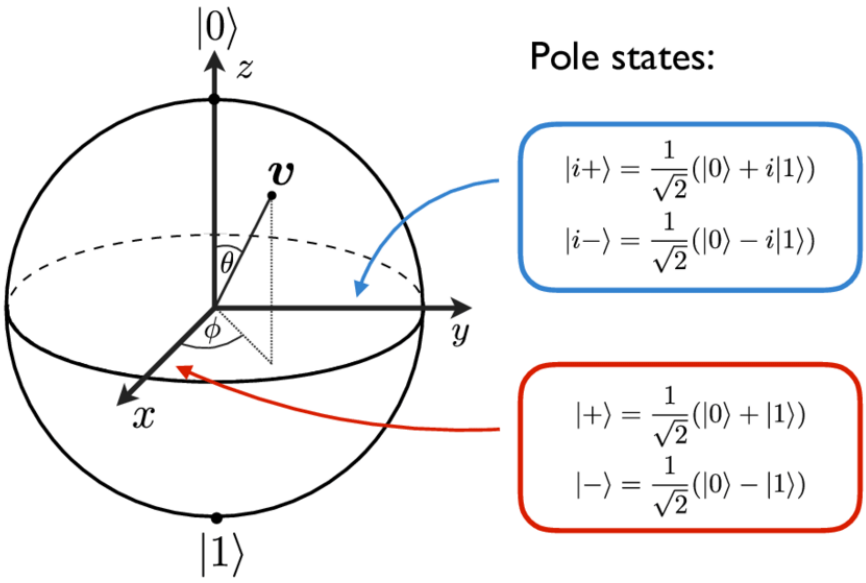

Figure 3: Rappresentazione geometrica di un qubit con la sfera di Bloch. Sono rappresentati come poli sull'asse z gli stati equivalenti allo 0 e 1 di un bit classico, sull'asse x invece i poli sono gli stati sopraccitati $|+\rangle$ e $|-\rangle$. Con questa rappresentazione é possibile notare come H non sia altro che una rotazione in questo spazio tridimensionale.

Inoltre una funzione f applicata a questa sovrapposizione si ottiene, nel caso $n = 1$, uno stato sovrappostro tra $f(0)$ e $f(1)$. Questo effetto é utilizzato dall'algoritmo in quanto riapplicando `Hadamard` si controlla in un solo passo se si ottiene la sovrapposizione di due stati uguali o di due stati diversi, o meglio se $f(0) = f(1)$ o meno. La riapplicazione di H restituirà 1 nel primo caso, 0 nel secondo.

4.2.1 single-bit Deutsch-Jozsa

```
operation DeutschJozsaSingleBit(oracle : (( Qubit,
 ↪  Qubit ) => Unit)) : Bool {
    use control = Qubit();
    use target = Qubit();

    H(control);
    X(target);
    H(target);

    oracle(control, target);

    H(target);
    X(target);

    return MResetX(control) == One;
}
```

4.2.2 n-bit Deutsch-Jozsa

```
operation DeutschJozsa(size : Int, oracle : ((Qubit[],
 ↪  Qubit ) => Unit) ) : Bool {
    use control = Qubit[size];
    use target = Qubit();

    ApplyToEachA(H, control);
    X(target);
    H(target);

    oracle(control, target);

    H(target);
    X(target);

    let result = MResetX(control[0]) == One;
    ResetAll(control);
    return result;
}
```

5 ENTANGLEMENT E TELETRASPORTO QUANTISTICO

6 CONCLUSIONI