Capitolo 3
Ottimizzare l'utilizzo di un qubit

Una volta imparato a manipolare un qubit (e.g. saper tradurre un gate logico in un preciso impulso), si deve pensare a come usarlo per il suo fine ultimo: essere utilizzato per il quantum computing. Anche questo aspetto è di fondamentale importanza, in quanto, come dimostrato nel capitolo precedente, un qubit ha dei limiti intrinseci: errori di implementazione di singolo gate e tempi di decoerenza. L'utilizzo ideale di un qubit deve ridurre al minimo queste cause di errore.

Riduzione di errori dovuti ai gate

Per ridurre la probabilità che degli errori scaturiscano a causa di gate inseriti in un dato algoritmo, ci sono due strade percorribili: lavorare a livello di hardware o a livello di algoritmo stesso; in questo paragrafo si discute questa seconda.

Come prima soluzione di ottimizzazione si può sfruttare un po' di semplice algebra lineare: il prodotto tra matrici 2x2. Questo perché ogni gate è una trasformazione unitaria applicata ad un sistema a due livelli, dunque rappresentabile con una matrice che gode, per definizione, della proprietà:

$$U^{\dagger}U = \mathbb{I}$$

Ragionando in termini "*gate=matrice*" si capisce come applicare più gate ad un qubit (vettore-stato di dimensione 2) equivalga a moltiplicare un vettore per un certo numero di matrici 2x2. Il risultato sarà comunque una singola trasformazione unitaria; infatti il prodotto di due matrici unitarie è una matrice unitaria. E, siccome il risultato finale è ciò che conta davvero, è conveniente cercare di raggiungerlo nel minor tempo possibile e col minor numero di passaggi possibile. Come negli esercizi di matematica. A questo scopo, si compongono i gate applicati ad un qubit in modo da ottenerne uno solo: un gate U3 con opportuni parametri. Schematicamente:

$$G_i G_{i-1} \dots G_2 G_1 |\psi> \rightarrow\ U3(\theta, \varphi, \lambda)|\psi>$$

Dove G_i rappresenta il gate *i-esimo* applicato al qubit.

Per vedere come la riduzione di numero di gate comporti un vantaggio in termini di costo computazionale si riporta un esempio costruito utilizzando simulatori di computer quantistici.

Esempio:

Creo una generica sequenza (*seq1*) di gate i già noti X, Rz, P e Ry da applicare ad un qubit. Di seguito una rappresentazione (fig. 3.1).

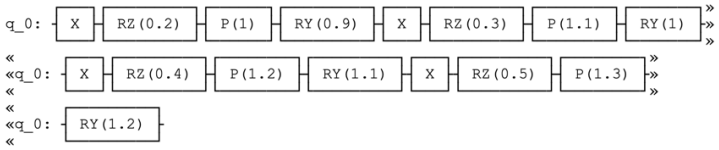
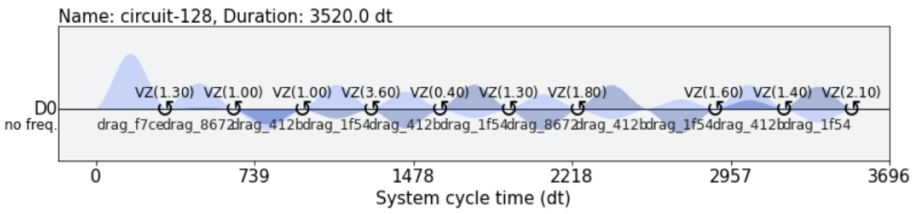


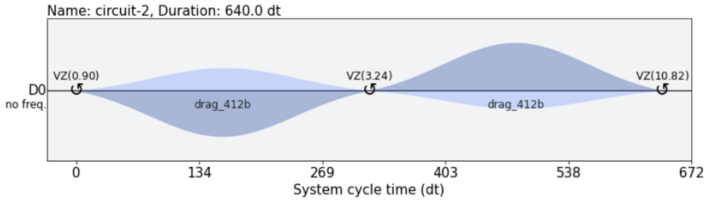
Fig. 3.1: Una sequenza arbitraria di gate applicati al qubit *q_0*

Utilizzando un simulatore (in questo caso *FakeArmonk*, in dotazione con le librerie Qiskit) mostro gli impulsi necessari per implementare questa sequenza:



Di questo grafico osservo un unico dato: la durata, in unità arbitrarie, pari a 3520dt.

Ora mostro gli impulsi necessari nel caso in cui la sequenza sia ridotta ad un unico U3 gate:



Ora la durata è di 640dt! La cosa interessante da osservare è che questa durata non dipende in alcun modo dalla lunghezza della sequenza originale e questo perché il gate U3 viene realizzato con al più tre rotazioni; che sono tre "operazioni" elementari che richiedono un unico impulso ciascuna.

Con l'esempio si è mostrato un caso in cui la riduzione di tempo di esecuzione è di circa un fattore 1/6; questo risultato, però, merita una precisazione. È stata ottenuta una riduzione notevole in virtù del fatto che la sequenza trasformata in singolo gate fosse particolarmente lunga. Di fatto, nella quasi totalità degli algoritmi vengono utilizzati più di un qubit e, in genere, non capita che si debbano applicare molti gate su un singolo qubit senza che questo venga fatto interagire con un altro; dunque, ci si può aspettare dei risultati più modesti da questa riduzione. Per chiarezza, si osservi la fig 3.2. rappresentante un circuito più "plausibile" di quello dell'esempio precedente.

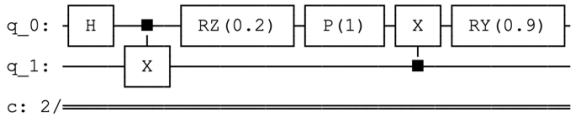


Fig 3.2: Nel circuito mostrato gli unici gate che possono essere composti sono Rz e P dato che i gate CNOT (indicati con una X e un segmento tra i due canali) implicano l'interazione tra qubit diversi.

Nelle librerie di Qiskit le composizioni tra gate su singolo qubit sono calcolate in automatico con il metodo *transpile()*.

```
compiled_circuits = transpile(qc, backend=backend)
```

Questa semplice linea di codice restituisce un circuito compilato in cui ogni sequenza d gate su singolo qubit è ridotta ad un unico U3 prendendo per argomento il circuito quantistico corrispondente ad un certo algoritmo e il backend sul quale si intende eseguire il circuito. Per onestà intellettuale, si intende specificare che il metodo *transpile()* fa in realtà molto di più, in quanto adegua un algoritmo ad un determinato backend a seconda delle caratteristiche di ogni suo singolo qubit e della sua architettura interna; ma questo è un argomento che esula da quanto trattato nel capitolo.

Utilizzare gate appena calibrati

Una volta minimizzato il numero di gate necessari per svolgere un determinato algoritmo quantistico, è bene massimizzare l'affidabilità di ogni singolo gate.

Per quest'ulteriore tecnica di ottimizzazione, non è necessario introdurre nulla di veramente nuovo: bisogna solo rifarsi al lavoro di calibrazione del capitolo 2. Si tratta, infatti, di sfruttare quanto fatto a livello hardware per avere vantaggi a livello di quantum computing. L'idea di fondo di questo capitolo è che i computer quantistici sono strumenti estremamente delicati e il loro utilizzo è fruttuoso solo nelle ore immediatamente successive ad una rigorosa calibrazione. Se non ci si volesse basare sulle calibrazioni giornaliere fatte da IBM (che, di conseguenza, hanno al più qualche ora di "età"), si può fare affidamento ad altre ancora più recenti: le nostre. Sì, perché quanto fatto su Qiskit Pulse può essere facilmente utilizzato in Qiskit. Ad esempio, la linea di codice necessaria per aggiungere un X gate al primo qubit ad un *QuantumCircuit* chiamato qc è:

```
qc.x(0)
```

In questo modo non si hanno informazioni sull'impulso di microonde inviato al qubit, si utilizza quello implementato con l'ultima calibrazione fatta al computer. Per usare, invece, l'impulso coi parametri appena ottimizzati si devono usare i seguenti metodi:

```
with pulse.build(backend) as pi_pulse:
    drive_duration = get_closest_multiple_of_16(pulse.seconds_to_samples(drive_duration_sec))
    drive_sigma = pulse.seconds_to_samples(drive_sigma_sec)
    drive_chan = pulse.drive_channel(qubit)
    pulse.play(pulse.Gaussian(duration=drive_duration,
                              amp=pi_amp,
                              sigma=drive_sigma,
                              name='pi_pulse'), drive_chan)
qc.add_calibration('x', [0], pi_pulse)
```

In cui durata, sigma e ampiezza dell'impulso gaussiano sono i parametri ottenuti con le procedure spiegate nel capitolo 2. Così facendo, si controlla pienamente la corrispondenza gate←→impulso. Dalla chiamata del metodo *qc.add_calibration()* il programma smetterà di usare il gate pre-impostato ed inizierà a usare l'impulso creato dall'utente.

Per avere un confronto tra i diversi approcci è stato eseguito un esperimento. Sempre utilizzando il backend ibmq-armonk, si sono eseguiti dei circuiti elementari (applicazione di un X gate e misura del qubit) in tre modi differenti:

1. Utilizzando le librerie Qiskit. Si tratta semplicemente di dichiarare un oggetto della classe QuantumCircuit, applicare un X gate e una misura. Il risultato atteso di questo circuito è, ovviamente, lo stato |1>, per stimare l'affidabilità di questo gate preimpostato si è commissionato al servizio cloud di IBM l'esecuzione di questo algoritmo 30 volte con 1000 tentativi ciascuno per ottenere della statistica.
2. Utilizzando le librerie QiskitPulse. In questo secondo caso si è ripercorso il programma di caratterizzazione fino alla dichiarazione del π-pulse; giunti a questo punto non si è proceduto con la stima di T1 e T2 ma si è costruito il medesimo circuito descritto nel punto 1; in questo, però, l'X gate è stato implementato con l'impulso appena calibrato. Il numero di esecuzioni richieste è pari a quelle precedenti.
3. Utilizzando le librerie QiskitPulse ma con un lavoro di calibrazione più completo. Come anticipato nel Capitolo 2, è possibile affinare la calibrazione del π-pulse iterando il processo. Di fatti, si è mostrato come costruire questo impulso utilizzando solo una stima approssimativa della frequenza di risonanza; questo perché si è immaginato di non esser ancora in grado di passare da |0> a |1>, il che sarebbe poi stato necessario per i vari esperimenti come il Rabi e il Ramsey. Ma, seguendo l'esempio del paper [REF_GatesCalib], ai fini di una migliore ottimizzazione dei parametri di questo impulso è consigliato seguire i seguenti step:
   -Trovare una prima stima di frequenza di risonanza utilizzando tecniche di spettroscopia all'output del qubit; come fatto nel paragrafo "Frequenza di risonanza" del Capitolo 2.
   -Impulsare il qubit a questa frequenza in modo da osservare le oscillazioni Rabi e ottenere da un fit con una funzione sinusoidale l'ampiezza necessaria per implementare un π-pulse.
   -Eseguire un esperimento di Ramsey in modo da ottenere una stima più precisa della frequenza di risonanza.
   -Eseguire un ulteriore esperimento per osservare le oscillazioni Rabi ed estrarre da un nuovo fit l'ampiezza definitiva dell'impulso necessario per passare dallo stato |0> allo stato |1>.
   Anche per questo terzo metodo sono stati eseguiti 30 volte set di 1000 circuiti ripetuti.

In tabella di riportano i risultati dei tre esperimenti; in particolare si mostrano i valori medi del numero di volte in cui si è ottenuto |1> su un set di 1000 esecuzioni.

Metodo 1	Metodo 2	Metodo 3
904 ± 2	903 ± 2	904 ± 2

Si reputano questi risultati estremamente indicativi delle prestazioni del qubit e della bontà delle calibrazioni giornaliere di IBM. La prima cosa che si vuole osservare è che si ottiene |1> come risultato della misura solo

nel 90% dei casi (circa). Questo è piuttosto sorprendente. Di fatti si è considerato il circuito più semplice da eseguire, in cui compare un unico gate che non ha nemmeno bisogno di essere scomposto in operazioni più elementari; ci si poteva aspettare un valore decisamente più prossimo a 100.
In seconda battuta, si vuole commentare la perfetta compatibilità dei tre metodi utilizzati. L'unico a sembrare un po' meno performante rispetto agli altri sembra essere quello che si basa sulla stima "grezza" della frequenza di risonanza; ce lo si poteva aspettare. C'è però un dettaglio che si vorrebbe rimarcare: per svolgere questa tesi non si ha avuto a completa disposizione un qubit, tutti i circuiti inviati al servizio cloud sono stati messi "in coda" con altri di altri utenti e questo va a discapito delle calibrazioni fatte con QiskitPulse e poi utilizzate nei circuiti. Il caso ideale, che si è descritto a inizio paragrafo, prevede che le esecuzioni dei circuiti siano immediatamente successive alle calibrazioni, il che potrebbe portare una maggiore affidabilità del metodo 3.
Altro aspetto non trascurabile per il commento dei risultati è l'errore nella misura dello stato di un qubit. Nella scheda tecnica del computer ibmq-armonk si legge *"Avg. Readout Error: 3.86e-2"*. Il che significa che l'errore di lettura dello stato di un qubit è affetto da un'incertezza relativa nell'ordine del 4%! Il che è ben al disopra delle deviazioni standard dalla media riportate in tabella. Per poter dire in che misura i risultati ottenuti siano dovuti all'errata implementazione del gate o all'errore di lettura dello stato, bisognerebbe condurre ulteriori approfondimenti. Questi, però, esulano dallo scopo di questo lavoro interessato al confronto tra gate preimpostati e gate creati dall'utente; i quali sono stati qui confrontati a parità di hardware, dunque a parità di apparato condizioni.

As Late As Possible (ALAP)
"*As Late As Possible"* è l'approccio alla schedulazione utilizzato di default da Qiskit. Sebbene si possa utilizzare il suo opposto "*As Soon As Possible"*, c'è un ottimo motivo per scegliere l'approccio ALAP: la decoerenza. L'idea di base è quella che un qubit lasciato allo stato fondamentale ha poche probabilità di cambiare di stato, mentre un qubit in una sovrapposizione di stati o esattamente nello stato |1> è soggetto sia a rilassamento trasverso che longitudinale.
Chiaramente, questo discorso è valido esclusivamente per circuiti in cui è presente un numero plurale di qubit, in cui è possibile che solo un certo sottogruppo venga utilizzato nella prima parte di algoritmo e solo più tardi vengano "chiamati in causa" i restanti.

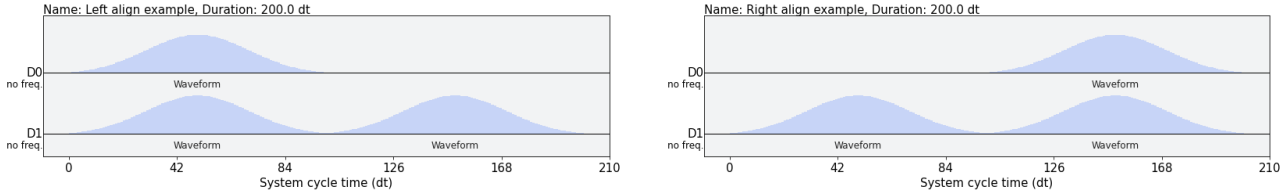


Fig. 3.2: In figura sono rappresentati i due diversi approcci di schedulazione ALAP e ASAP.
Importante notare come nel caso destro (ALAP) nessun qubit venga lasciato interagire liberamente con l'ambiente dopo il primo impulso.

Durata di singolo gate
Si riprenda il terzo criterio di DeVincenzo: *"tempi di decoerenza sufficientemente lunghi"*. Come già commentato, non c'è alcuna indicazione di tempi tipici i quali possono essere considerati "sufficientemente lunghi" in virtù del fatto che questi vanno messi in relazione con i quantum gate; è necessario, in genere, aver dei tempi di decoerenza T1 e T2 di quattro o cinque ordini di grandezza maggiori della durata di una singola operazione. Dato che T1 e T2, in fin dei conti, sono intrinseci al qubit fisico, viene spontaneo pensare che sia conveniente ridurre la durata di un impulso che implementa un gate. Ebbene, questo è quello che si tenta di fare; ma va fatto considerando un altro aspetto: un impulso troppo breve può non portare a risultati performanti.[REF_GatesCalib e un'altra]
Per mostrare ciò, è stato eseguito un esperimento in cui si ripercorre la parametrizzazione dell'impulso π per diverse ampiezze temporali della gaussiana che lo rappresenta.