High level programming language for quantum computing

Davide Camino

UNIVERSITÀ
DI TORINO

davide.camino@edu.unito.it

Obbiettivo del lavoro

Reasoning su quantum computer

1. Base di conoscenza classica + query
2. Embedding su quantum computer
3. Esecuzione e recupero risultati

Perché il quantum computing

Grazie a

- Entanglement
- Superposition

immagine

Violiamo

Strong Church-Turing Thesis

immagine

Due paradigmi di programmazione

Quantum Gate

immagine

Quantum Annealing

immagine

Quantum Gate

- Formalismo molto studiato
- Gate Reversibili
- Set di porte universali
- Algoritmi di Shore, ecc.

immagine

Quantum Annealer

immagine

- Ispirato a Simulated Annealling
- Effetto tunneling
- Adiabatic Quantum Computing (AQC)
- $\mathcal{H}(t) = s(t)\mathcal{H}_i + (i - s(t))\mathcal{H}_f$

QA-Prolog

Il progetto

- Sviluppato da Sott Pakin
- Proof of concept
- Trasformazioni successive
- Prolog + Query $\rightarrow \mathcal{H}_f$
- Interagisce direttamente con il solver
- Raccoglie e organizza i risultati

immagine

Pipeline

immagine

Trasformazioni

1. Prolog $\rightarrow$ Verilog (HDL)
2. Verilog $\rightarrow$ Circuito digitale
3. Circuito digitale $\rightarrow \mathcal{H}_f$ simbolica
4. $\mathcal{H}_f$ simbolica $\rightarrow \mathcal{H}_f$ fisica

QMASM

$\mathcal{H}_f$ simbolica $\rightarrow$ $\mathcal{H}_f$ fisica

Cos'è

- Quantum macro assembler
- Sviluppato in Python
- Basso livello di astrazione
- Si interfaccia con Ocean

Cosa permette di fare

- Riferimento simbolico a *qubit*
- *Qubit* "pinnati" a `TRUE` o `FALSE`
- Incapsulare pattern in macro
- Creazione di librerie di macro
- Pulizia dell'output:
  - solo *qubit* "interessanti"
  - no slack variables

QMASM

Esempio: Macro

```
# 2-input OR gate (Y = A or B)
!begin_macro or2
$A  0.3333
$B  0.3333
$Y -0.6667

$A $B  0.3333
$A $Y -0.6667
$B $Y -0.6667
!end_macro or2
```

Figure: **or** gate

```
# 1-input NOT gate (Y = not A)
!begin_macro not1
$A -0.5
$Y -0.5

$A $Y 1.0
!end_macro not1
```

Figure: **not** gate

```
# 2-input AND gate (Y = A or B)
!begin_macro or2
$A  0
$B  0
$Y  1

$A $B  0
$A $Y -0.7
$B $Y -0.7
!end_macro or2
```

Figure: **and** gate

possiamo racchiudere queste macro nel file `gates.qmasm`

QMASM

Esempio: $y = x_1 \wedge \neg(x_2 \vee x_3)$

```
!include <gates>

!use_macro or2 x2_or_x3
x2_or_x3.$A = x2
x2_or_x3.$B = x3
x2_or_x3.$Y = $x4

!use_macro not1 not_x4
not_x1.$A = $x4
not_x1.$Y = $x5

!use_macro and2 x1_and_x5
x4_and_x5.$A = x1
x4_and_x5.$B = $x5
x4_and_x5.$Y = y
```

Figure: CircSat problem

"Pinniamo" il valore di y per ottenere l'assegnamento delle x_i che verificano la formula logica:

```
qmasm --run --pin="x10 := true" circsat.qmasm
```

```
Solution #1 (energy = -80.00):

    Name(s)  Spin  Boolean
    -------  ----  -------
    x1         +1  True
    x2         -1  False
    x3         -1  False
    y          -1  True
```

Figure: CircSat solution

Yosys - edif2qmasm

Yosys - edif2qmasm

Esempio

QA-Prolog

QA-Prolog

Esempio

QAOA

Da $\mathcal{H}$ a operatori di Paoli

Esempio

Esempio

Risultato 1

Esempio

Risultato 2

Ontologie

OWL

Inferenze in OWL

Complessità dell'iInferenza

Da OWL-rdf a Prolog