# Qlafoutea

## Baby steps towards compiling for analog quantum devices

David “Yoric” Teller (david.teller@pasqal.com)

Pasqal

2025-02-01

# about:

# about:Pasqal

- We build Quantum Computers (aka QPUs).
- Also Quantum Programming Tools/Languages.
- Also Quantum Application Libraries.
- Much of our work is FOSS[1].

[1] (or in the pipelines)

# about:Pasqal

- We build Quantum Computers (aka QPUs).
- Also Quantum Programming Tools/Languages.
- Also Quantum Application Libraries.
- Much of our work is FOSS[1].
- Remind me to tell you about our Community Portal!

[1](or in the pipelines)

# about:Yoric

- Open-source coordinator at Pasqal.
- Compiler background.
- Contributor to Firefox, JavaScript, Rust, Matrix, …

# about:Yoric

- Open-source coordinator at Pasqal.
- Compiler background.
- Contributor to Firefox, JavaScript, Rust, Matrix, …
- *Not* a Quantum Computing background! 🫣

# about:This talk

- Side project.
- Early-stage experiments.
- Hopefully a different way to look at Quantum Computing.
- Don’t expect brand new results.

# about:Quantum Computing

$$H(a,b,c) = \sum_{i=1}^{N}(a_i \cdot \sigma_i^x + b_i \cdot \sigma_i^z) + \sum_{1 \leq j \leq i \leq N} c_{i,j} \cdot \tau_i^z \cdot \tau_j^z$$

This is a $2^N \cdot 2^N$ matrix.

- On a classical computer, all interesting operations on $H(a,b,c)$ have a **time cost** that is at least **exponential** in $N$.
- On a quantum computer, many interesting operations $H(a,b,c)$ have a **probabilistic time cost** that is **polynomial** in $N$.

# about:Quantum Computing

$$H(a,b,c) = \sum_{i=1}^{N}(a_i \cdot \sigma_i^x + b_i \cdot \sigma_i^z) + \sum_{1 \leq j \leq i \leq N} c_{i,j} \cdot \tau_i^z \cdot \tau_j^z$$

This is a $2^N \cdot 2^N$ matrix.

- On a classical computer, all interesting operations on $H(a,b,c)$ have a **time cost** that is at least **exponential** in $N$.
- On a quantum computer, many interesting operations $H(a,b,c)$ have a **probabilistic time cost** that is **polynomial** in $N$.

Corollary: quantum computing *might* change the world 🐱 .

# about:Digital Quantum vs. Analog Quantum



- Digital Quantum Computing:
  - Try to tame $H$ by decomposing it into “simple” *gates*.
  - Error correction is… complicated.
  - Library: qiskit…

# about:Digital Quantum vs. Analog Quantum

- Digital Quantum Computing:
  - Try to tame $H$ by decomposing it into "simple" *gates*.
  - Error correction is… complicated.
  - Library: qiskit…
- Analog Quantum Computing:
  - Embrace the matrix.
  - One large complex system instead of many simple ones.
  - Library: pulser…

# about:Digital Quantum vs. Analog Quantum



- Digital Quantum Computing:
  - Try to tame $H$ by decomposing it into “simple” *gates*.
  - Error correction is… complicated.
  - Library: qiskit…
- Analog Quantum Computing:
  - Embrace the matrix.
  - One large complex system instead of many simple ones.
  - Library: pulser…
- Today, we’ll talk about Analog.

# about: Quantum Limitations



- Did I mention probabilistic?
- No `if`, `while`.
- No *memory*.

# about: Quantum Limitations

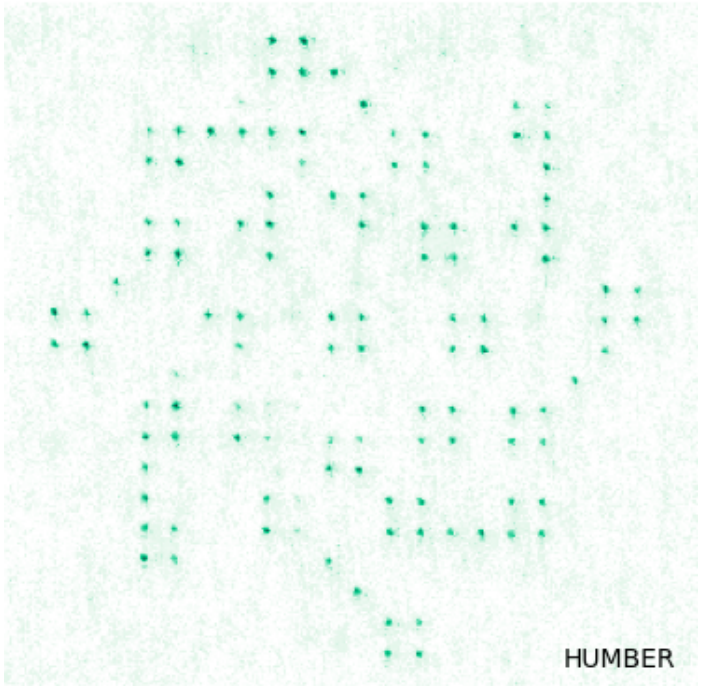


- Did I mention probabilistic?
- No `if`, `while`.
- No *memory*.
- ...unless you also have a classical computer!

# Qlafoutea

# Why a compiler?

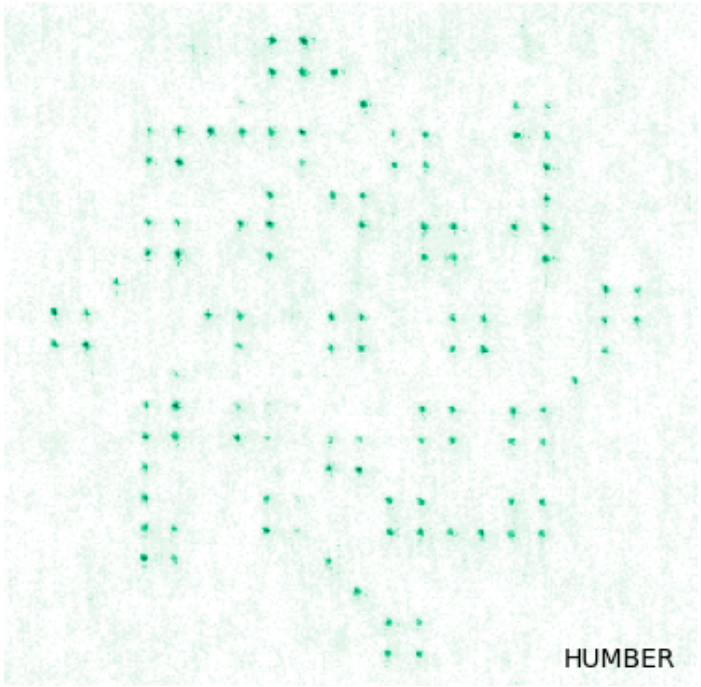
Beats the alternative:



# Why a compiler?



Beats the alternative:




aka Quantum machine code.

# What's in a compiler?



Source code $\mapsto$ Machine Code.

# What's in a compiler?



Source code $\mapsto$ Machine Code.

- Often accompanied by a Runtime.
- Ideally accompanied by a Machine.

# Compiling!

Source code
$\mapsto$ Intermediate Representation 1
$\mapsto$ ...
$\mapsto$ Intermediate Representation N
$\mapsto$ Machine Code
$\mapsto$ Machine or VM
$\mapsto$ Execution results

# Compiling!



Source code

$\mapsto$ Intermediate Representation 1

$\mapsto$ ...

$\mapsto$ Intermediate Representation N

$\mapsto$ Machine Code

$\mapsto$ Machine or VM

$\mapsto$ Execution results (*Bitstring counters*)

# Compiling!

Source code

$\mapsto$ Intermediate Representation 1

$\mapsto$ ...

$\mapsto$ Intermediate Representation N

$\mapsto$ Machine Code

$\mapsto$ Machine or VM (*QPU* or *Emulator*)

$\mapsto$ Execution results (*Bitstring counters*)

# Compiling!

Source code

$\mapsto$ Intermediate Representation 1

$\mapsto$ ...

$\mapsto$ Intermediate Representation N

$\mapsto$ Machine Code (*Pulse Sequences*)

$\mapsto$ Machine or VM (*QPU* or *Emulator*)

$\mapsto$ Execution results (*Bitstring counters*)

# Source code

- $A \land (A \Rightarrow B)$, is $B$ true?

# Source code

- $A \wedge (A \Rightarrow B)$, is $B$ true?
- Yes, not the most sophisticated of languages, but still the base of e.g. Prolog, type inference, etc.

# IR 1 (SAT)

...make it simpler

- $A \wedge (B \vee \neg A)$
- observe $B$

# IR 2 (3SAT)

...make it closer to the machine

- system
  - $A \vee X_0 \vee X_1 == \texttt{true}$
  - $\neg A \vee B \vee X_0 == \texttt{true}$
- observe $B, X_0, X_1$
- restrict to $X_0 == \texttt{false}$
- restrict to $X_1 == \texttt{false}$

...make it closer to the machine

- system
  - $A + X_0 + X_1 - A \cdot X_0 - X_0 \cdot X_1 - A \cdot X_1 + A \cdot X_0 \cdot X_1 == 1$
  - $\neg A + B + X_0 - (\neg A) \cdot B - B \cdot X_0 - (\neg A) \cdot X_0 + (\neg A) \cdot B \cdot X_0 == 1$
- observe $B, X_0, X_1$
- restrict to $X_0 == 1$
- restrict to $X_1 == 1$

...make it closer to the machine

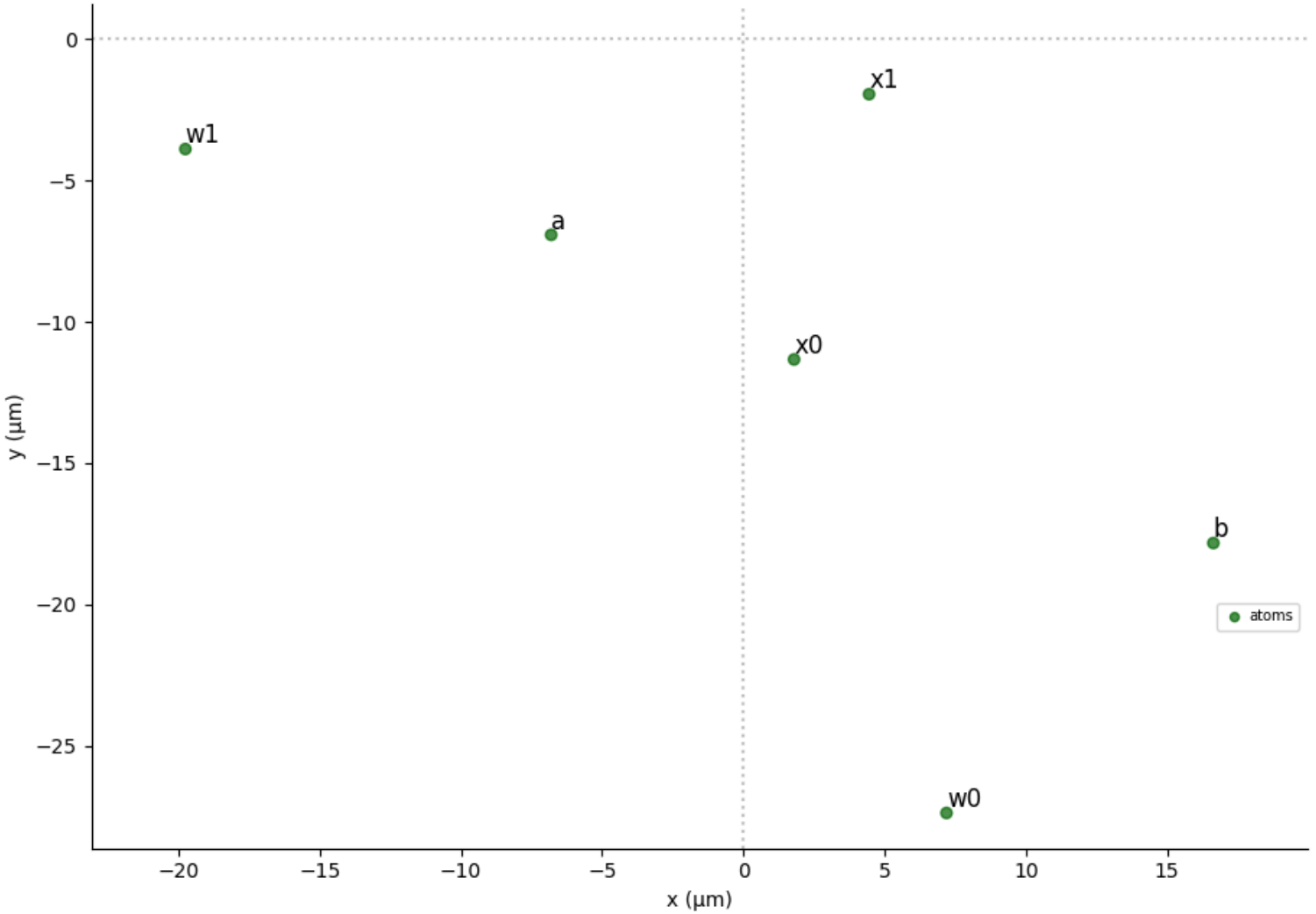
- maximize $\{(1 + W_0) \cdot (A + X_0 + X_1) - A \cdot X_0 - X_0 \cdot X_1 - A \cdot X_1 - 2 \cdot W_0 + (1 + W_1) \cdot (1 - A + B + X_0) - (1 - A) \cdot B - B \cdot X_0 - (1 - A) \cdot X_0 + W_1\}$
- observe $B, X_0, X_1$
- restrict to $X_0 == 1$
- restrict to $X_1 == 1$
- ignore $W_0, W_1$

...make it closer to the machine

$$\begin{pmatrix} 0 & -2 & 1 & 1 & 0 & 0 \\ -2 & 2 & -1 & -1 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

...

(let's skip a few steps)


0
−5
−10
−15
−20
−25
y (µm)
−20
−15
−10
−5
0
5
10
15
x (µm)
x1
w1
a
x0
b
atoms
w0

# Machine Code

```
{
  'a': array([-6.80383692, -6.91849058]),
  'b': array([ 16.63635959, -17.82765376]),
  'x0': array([  1.7942145 , -11.32810108]),
  'x1': array([ 4.4480917 , -1.94023321]),
  'w0': array([  7.18946654, -27.38150432]),
  'w1': array([-19.7528544 ,  -3.88177107])
}
```

# Machine Code

```
{
  'a': array([-6.80383692, -6.91849058]),
  'b': array([ 16.63635959, -17.82765376]),
  'x0': array([  1.7942145 , -11.32810108]),
  'x1': array([ 4.4480917 , -1.94023321]),
  'w0': array([  7.18946654, -27.38150432]),
  'w1': array([-19.7528544 ,  -3.88177107])
}
```

Also, lasers (QAA).

# Does it work?



Let’s check it out!

# Conclusions

# What we have

Source code

$\mapsto$ Intermediate Representation 1 (SAT)

$\mapsto$ ...

$\mapsto$ Intermediate Representation N (Geometry)

$\mapsto$ Machine Code (Pulse Sequences)

$\mapsto$ Machine or VM (QPU or Emulator)

$\mapsto$ Machine or VM (Bitstring extraction)

$\mapsto$ Execution results (Variables)

# What we have

Source code

$\mapsto$ Intermediate Representation 1 (SAT)

$\mapsto$ ...

$\mapsto$ Intermediate Representation N (Geometry)

$\mapsto$ Machine Code (Pulse Sequences)

$\mapsto$ Machine or VM (QPU or Emulator)

$\mapsto$ Machine or VM (Bitstring extraction)

$\mapsto$ Execution results (Variables)

Demonstrated with logical formula, works for *optimization problems*.

# What we learnt

- We *can* write compilers for hybrid classical/quantum analog.
- Result reliability is something of an unusual problem (solutions have been demonstrated).

# What we learnt



- We *can* write compilers for hybrid classical/quantum analog.
- Result reliability is something of an unusual problem (solutions have been demonstrated).
- Not *terribly* different from compiling to *digital* quantum (same reliability difficulties, too).

# What we learnt



- We *can* write compilers for hybrid classical/quantum analog.
- Result reliability is something of an unusual problem (solutions have been demonstrated).
- Not *terribly* different from compiling to *digital* quantum (same reliability difficulties, too).
- So far, nothing like a speedup.

# What we learnt



- We *can* write compilers for hybrid classical/quantum analog.
- Result reliability is something of an unusual problem (solutions have been demonstrated).
- Not *terribly* different from compiling to *digital* quantum (same reliability difficulties, too).
- So far, nothing like a speedup.
- Rust works great for experiments!

# Todo

1. Improve reliability
    1. Better laser usage.
    2. Better QUBO (WIP).
2. Could we use fewer qubits?

# Let's not forget!

Pasqal Community Portal https://community.pasqal.com

- **Tutorials** and **advanced courses** on Quantum Computing.
- **Chat** & **collaborate** on Quantum Computing.
- **FOSS** software for low-level & high-level Quantum Development.
- Access **physical QPUs** and **high-performance emulators**.

# Let's not forget!

Pasqal Community Portal https://community.pasqal.com

- **Tutorials** and **advanced courses** on Quantum Computing.
- **Chat** & **collaborate** on Quantum Computing.
- **FOSS** software for low-level & high-level Quantum Development.
- Access **physical QPUs** and **high-performance emulators**.

Launch & webinar February 25th 10am, CET.

https://app.livestorm.co/pasqal/introducing-pasqal-community

# Thank you!


Any questions?

# Thank you!


Any questions?

Fork me on Github: https://github.com/pasqal-io/qlafoutea.