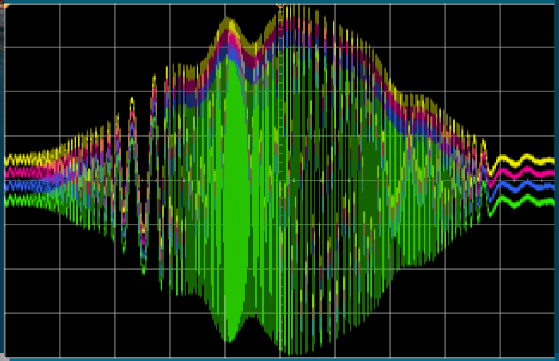
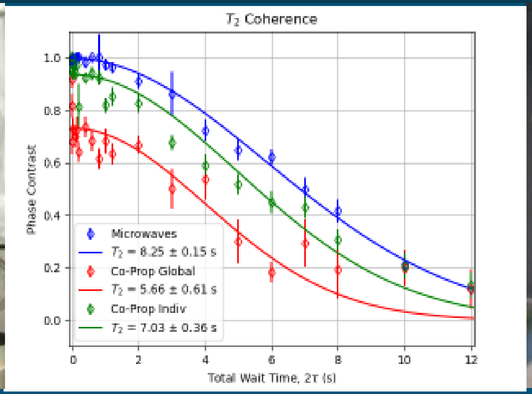
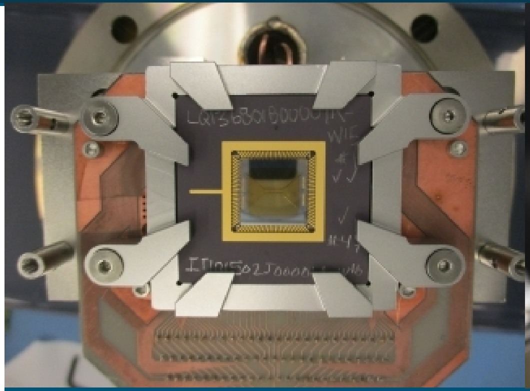
SAND2022-12317C

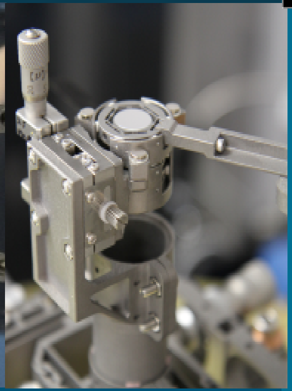
Batching Circuits to Reduce Compilation in Quantum Control Hardware



QSCOUT



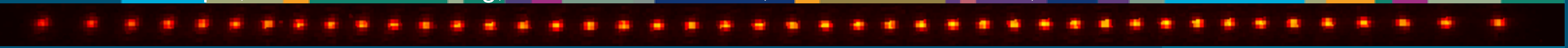




U.S. DEPARTMENT OF ENERGY
Office of Science

PRESENTED BY

Ashlyn D. Burch, Daniel S. Lobser, Christopher G. Yale, Jay W. Van Der Wall, Oliver G. Maupin, Joshua D. Goldberg, Matthew N. H. Chow, Melissa C. Revelle, Susan



Technical paper for 2022 IEEE Quantum Week

a problem shared amongst all quantum computers: parameter control

QSCOUT

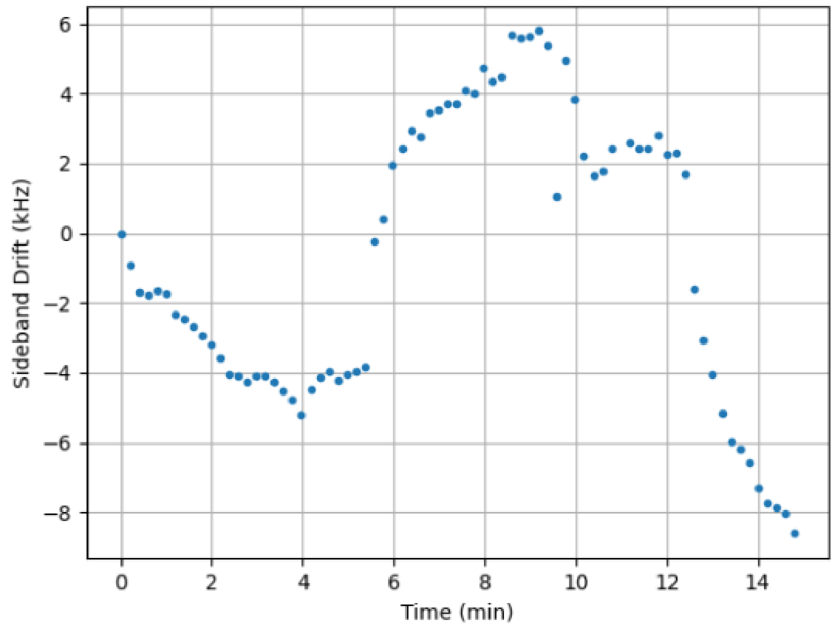
Aim: greater hardware control, reduced overhead

The whole purpose of having a smaller testbed platform is to explore the capabilities of user hardware and software control, making it easier to scale to larger systems.

Tunable parameter and potential sources of drift on trapped ion systems:

- Magnetic field fluctuations
- RF power instabilities
- Beam power or frequency fluctuations
- Spatial beam drifts
- Stray electric field effects
- Relative phases between gates

All of which may or may not be affecting the others! They all may also be affected on different timescales.



T. Proctor, M. Revelle, et al. Nature Communications **11**, 5396 (2020).
J. Kelly, R. Barends, et al. Phys. Rev. A **94**, 032321 (2016).

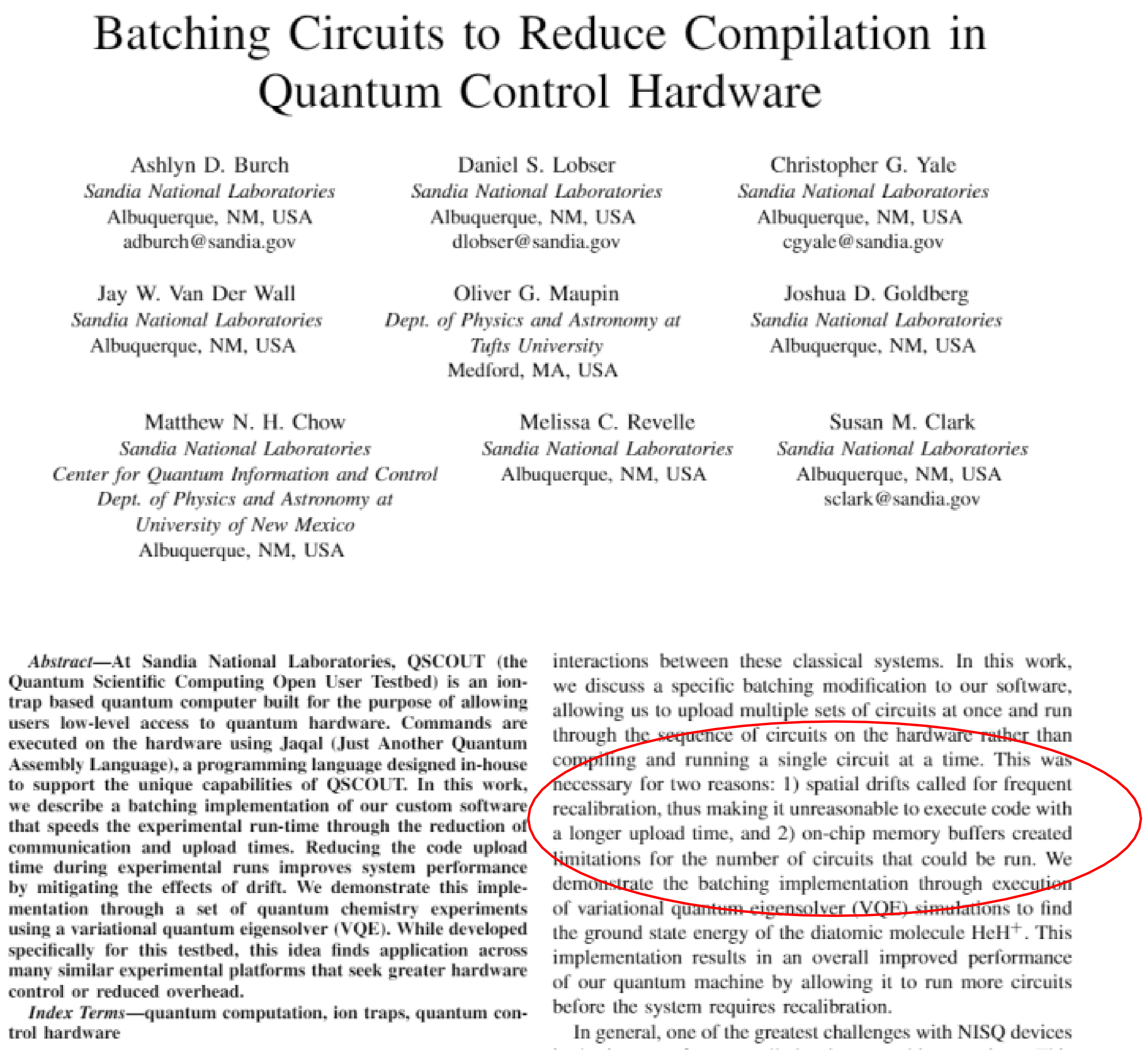
discovering our current limitations

Current main limitations:

- Drifts – RF power fluctuations, spatial beam drifts that call for frequent recalibration
- Limitations of on-chip memory buffers that make the experiment infeasible – unique for certain experiments

Collaboration on experiments with Tufts

University and Oak Ridge National Laboratories exposed a problem that we needed to fix!



Batching Circuits to Reduce Compilation in Quantum Control Hardware

Ashlyn D. Burch
Sandia National Laboratories
Albuquerque, NM, USA
adburch@sandia.gov

Daniel S. Lobser
Sandia National Laboratories
Albuquerque, NM, USA
dlobser@sandia.gov

Christopher G. Yale
Sandia National Laboratories
Albuquerque, NM, USA
cgyale@sandia.gov

Jay W. Van Der Wall
Sandia National Laboratories
Albuquerque, NM, USA

Oliver G. Maupin
Dept. of Physics and Astronomy at Tufts University
Medford, MA, USA

Joshua D. Goldberg
Sandia National Laboratories
Albuquerque, NM, USA

Matthew N. H. Chow
Sandia National Laboratories
Center for Quantum Information and Control
Dept. of Physics and Astronomy at University of New Mexico
Albuquerque, NM, USA

Melissa C. Revelle
Sandia National Laboratories
Albuquerque, NM, USA

Susan M. Clark
Sandia National Laboratories
Albuquerque, NM, USA
sclark@sandia.gov

***Abstract*—At Sandia National Laboratories, QSCOUT (the Quantum Scientific Computing Open User Testbed) is an ion-trap based quantum computer built for the purpose of allowing users low-level access to quantum hardware. Commands are executed on the hardware using Jaqal (Just Another Quantum Assembly Language), a programming language designed in-house to support the unique capabilities of QSCOUT. In this work, we describe a batching implementation of our custom software that speeds the experimental run-time through the reduction of communication and upload times. Reducing the code upload time during experimental runs improves system performance by mitigating the effects of drift. We demonstrate this implementation through a set of quantum chemistry experiments using a variational quantum eigensolver (VQE). While developed specifically for this testbed, this idea finds application across many similar experimental platforms that seek greater hardware control or reduced overhead.**

***Index Terms*—quantum computation, ion traps, quantum control hardware**

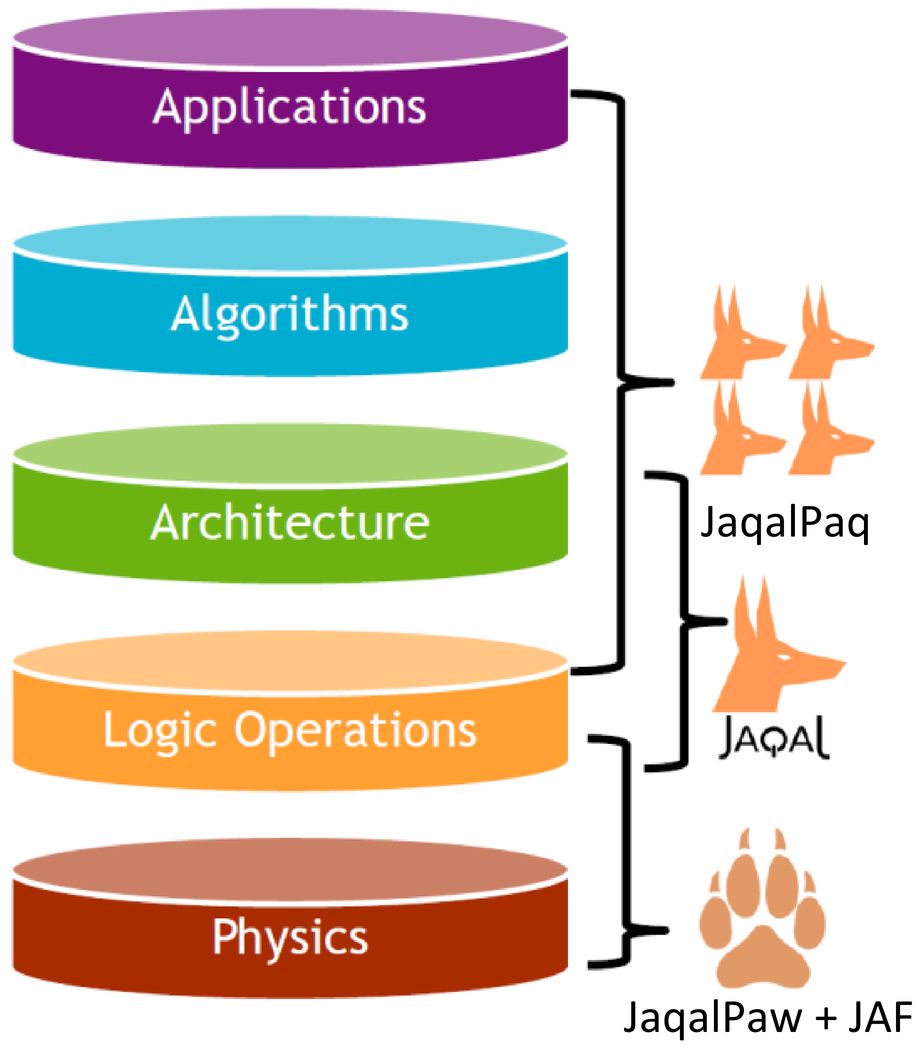
interactions between these classical systems. In this work, we discuss a specific batching modification to our software, allowing us to upload multiple sets of circuits at once and run through the sequence of circuits on the hardware rather than compiling and running a single circuit at a time. This was necessary for two reasons: 1) spatial drifts called for frequent recalibration, thus making it unreasonable to execute code with a longer upload time, and 2) on-chip memory buffers created limitations for the number of circuits that could be run. We demonstrate the batching implementation through execution of variational quantum eigensolver (VQE) simulations to find the ground state energy of the diatomic molecule HeH^+. This implementation results in an overall improved performance of our quantum machine by allowing it to run more circuits before the system requires recalibration.

In general, one of the greatest challenges with NISQ devices

outline

- **Exploring the Jaqal-verse**
  - Jaqal
  - JaqalPaq
  - JaqalPaw
  - Jaqal Application Framework
- **Implementing experiments on the hardware**
  - QSCOUT
  - Octet
- **Integration of batching in Jaqal**
  - Tufts University - Richardson extrapolation requiring scaling factor
    - ➢ Batching with let parameter dictionaries
  - Oak Ridge National Laboratories - randomized compiling requiring large number of circuits to be run
    - ➢ Batching through indexing

the Jaqal-verse



Jaqal (Just Another Quantum Assembly Language) – quantum assembly language created for explicit and transparent qubit control

- Preparation of all qubits in the z basis
- parameterized single qubit rotation gates that can be rotated about any axis on the equatorial plane of the Bloch sphere
- Parameterized single qubit virtual Z gates that act as a phase advance on subsequent waveforms
- Native two-qubit (Mølmer-Sørensen) gates between any two ions, with user-defined phase and rotation angle
- Measurement of all qubits in the z basis

JaqalPaq – python software package for easy programming with sub-packages that include compiler, emulator, code generator, etc.

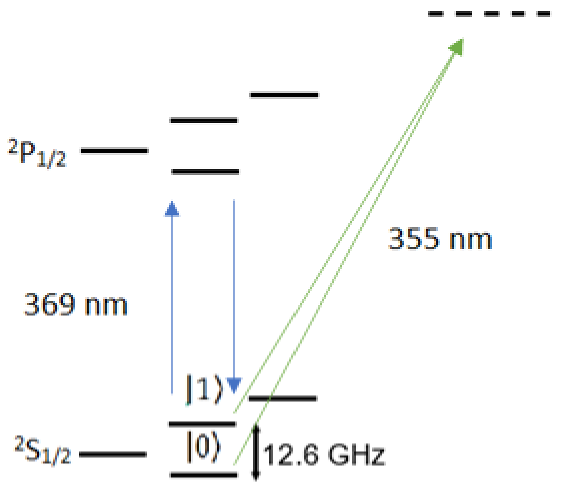
JaqalPaw (Jaqal Pulses and Waveforms) – a suite that is used to define pulses and waveforms for Jaqal to be run on the QSCOUT hardware

- The lower level counterpart to Jaqal that describes gates in terms of waveforms in a gate pulse file
- Users can construct macros for custom gates built off of native gates

JAF (Jaqal Application Framework) – application framework to execute Jaqal circuits on the hardware

A.J. Landahl, D.S. Lobser, et al (2020) "Jaqal, the Quantum Assembly Language for QSCOUT." arXiv: 2003.09382.

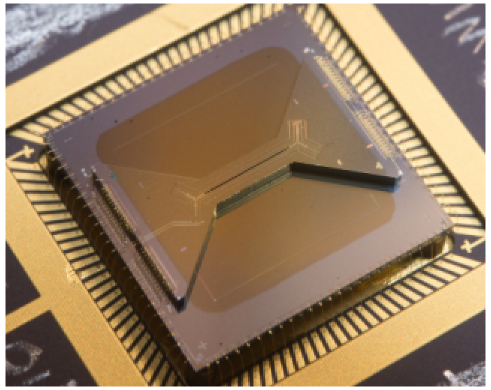
QSCOUT – testbed for open user access



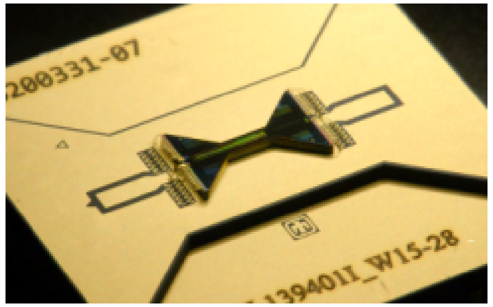
S. Olmschenk et al., PRA **76**, 052314 (2007)

- RT system based on $^{171}Yb^+$ trapped ions
- 1D linear chain of 2-11 qubits, with hardware support of up to 32
- Full connectivity between ions using radial vibrational modes

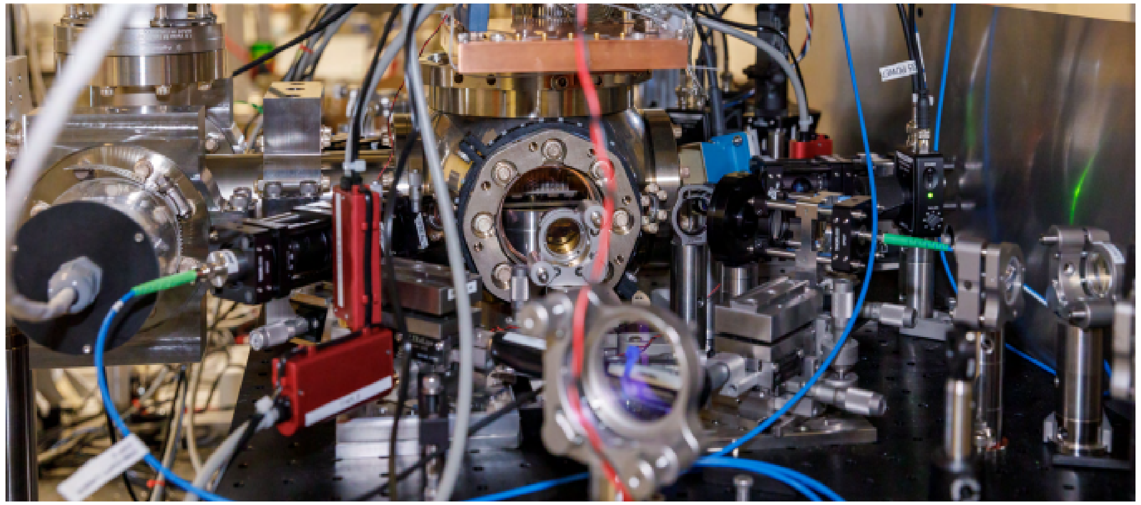
HOA 2.1 surface electrode trap



Peregrine surface trap



- Individual addressability with 355 nm Raman beams
- RF tones applied to AOMs for individual addressing and global beam are controlled by *Octet* (custom-designed hardware for advanced gate + pulse control)
- Provides unique parameterized single and two-qubit gate sets



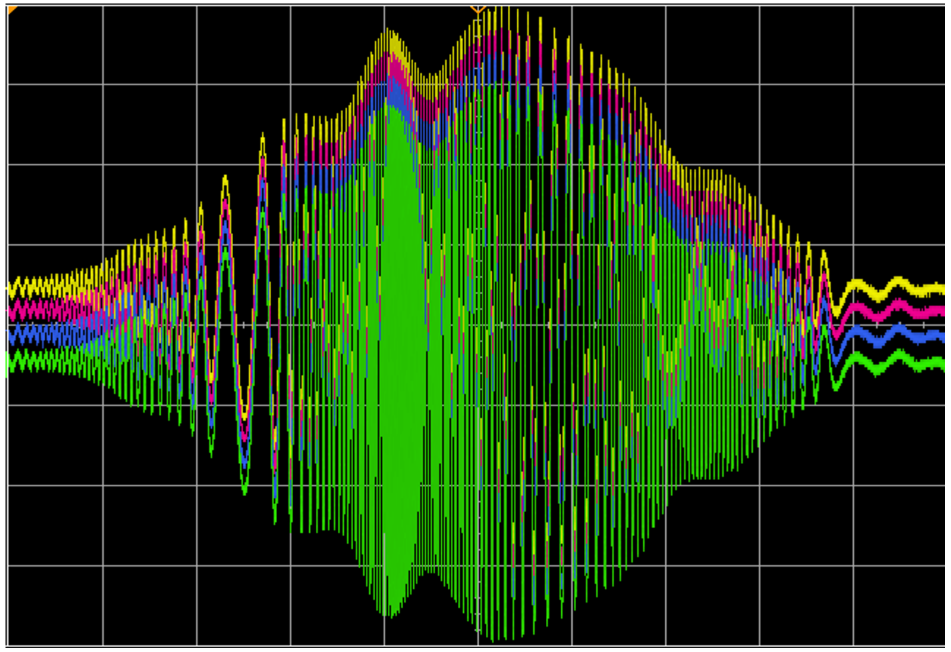
S.M. Clark, D. Lobser, et al (2021) "Engineering the Quantum Scientific Computing Open User Testbed." IEEE Transactions on Quantum Engineering vol. 2, pp. 1-32 Art no. 3102832, doi: 10.1109/TQE.2021.3096480.

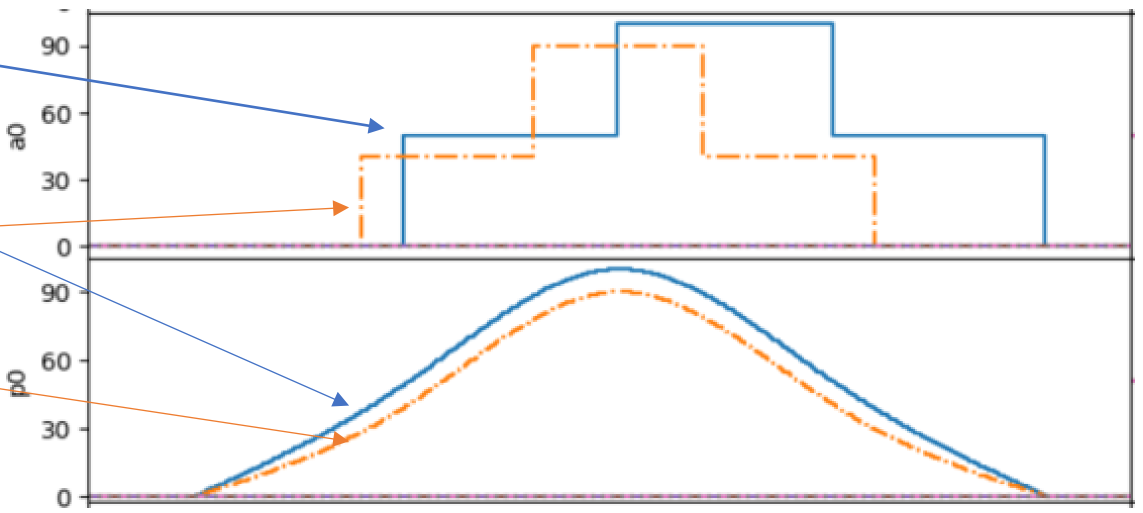
hardware for coherent pulse generation

QSCOUT

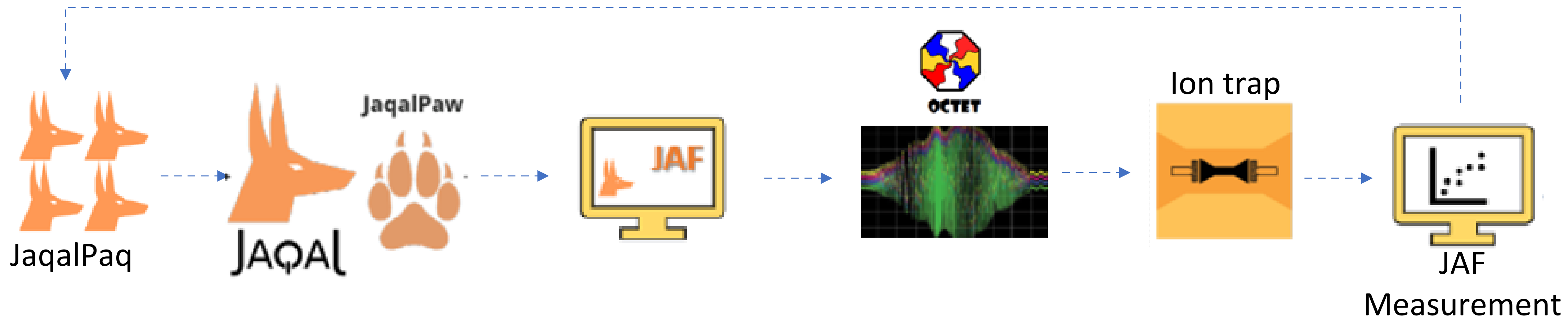
- RFSoC to control single and multi-channel AOMs
- Coherent output synchronized between all channels
- Generation of up to two tones per channel within AOMs
- Compact representation of gates for efficient circuit streaming
- Z gates are performed virtually in the software

```
def gate_Rcounter(self, channel, angle=np.pi, phase=0):
    duration = (angle/np.pi)*self.counter_resonant_pi_time
    return [PulseData(0, duration,
                freq0=self.freq0,
                amp0= [0, 50, 100, 50]
                phase0=(0, 50, 100, 50, 0),
                sync_mask=0b01, enable_mask=1),
            PulseData(channel, duration,
                freq0=self.freq0,
                freq1=self.ia_center_frequency,
                amp0=[0,40,90,40,0],
                phase0 = (0,40,90,40,0),
                amp1=self.amp1_counterprop_list[int(channel)],
                phase1=phase,
                sync_mask=0b11, enable_mask=2)]
```





running QSCOUT with the software



1) User builds their Jaqal circuits constructing JaqalPaq
2) These Jaqal circuits can construct new gates and pulses using JaqalPaw
3) The user code is uploaded to the JAF network service running through a Docker container
4) These circuits are sent to the hardware Octet, interpreted and then translated into laser pulses
5) Laser pulses are streamed out and quantum circuits are performed on the ions in the trap
6) Measurements are recorded in the Jaqal Application Framework and returned to the user

outline

- **Exploring the Jaqal-verse**
  - Jaqal
  - JaqalPaq
  - JaqalPaw
  - Jaqal Application Framework
- **Implementing experiments on the hardware**
  - QSCOUT
  - Octet
- **Integration of batching in Jaqal**
  - Tufts University - Richardson extrapolation requiring scaling factor
    - ➢ Batching with let parameter dictionaries
  - Oak Ridge National Laboratories - randomized compiling requiring large number of circuits to be run
    - ➢ Batching through indexing

2 variational quantum eigensolver experiments, modeling dynamics of molecules and extracts upper bound ground state energy of Hamiltonian

Quantum and classical hybrid – quantum processor with classical optimizer

outline

- **Exploring the Jaqal-verse**
  - Jaqal
  - JaqalPaq
  - JaqalPaw
  - Jaqal Application Framework
- **Implementing experiments on the hardware**
  - QSCOUT
  - Octet
- **Integration of batching in Jaqal**
  - Tufts University - Richardson extrapolation requiring scaling factor
    - ➢ Batching with let parameter dictionaries
  - Oak Ridge National Laboratories - randomized compiling requiring large number of circuits to be run
    - ➢ Batching through indexing

error mitigation experiments on QSCOUT

Tufts University goal: Use a short-depth VQE (ground state energy of HeH$^+$) circuits error mitigation technique that can be demonstrated using QSCOUT → Richardson Extrapolation for determining zero-noise extrapolation of the circuit which required a controlled scaling factor

1. Circuit is being driven by a time-dependent Hamiltonian

$$K(t) = \sum_\alpha J_\alpha(t) P_\alpha$$

2. Expectation value of the observable is expanded as a Taylor series

$$E_K(\lambda) = E^* + \sum_{k=0}^{n} a_k \lambda^k + \mathcal{O}(\lambda^{n+1})$$

E^* : ideal zero-noise expectation value

$\lambda \ll 1$: small noise parameter

3. Additional noisy estimates are added:

$$\hat{E}_K(c_i\lambda)$$

c_i : controlled scaling factor

Our biggest challenge – scaling by a time factor or gate depth factor!

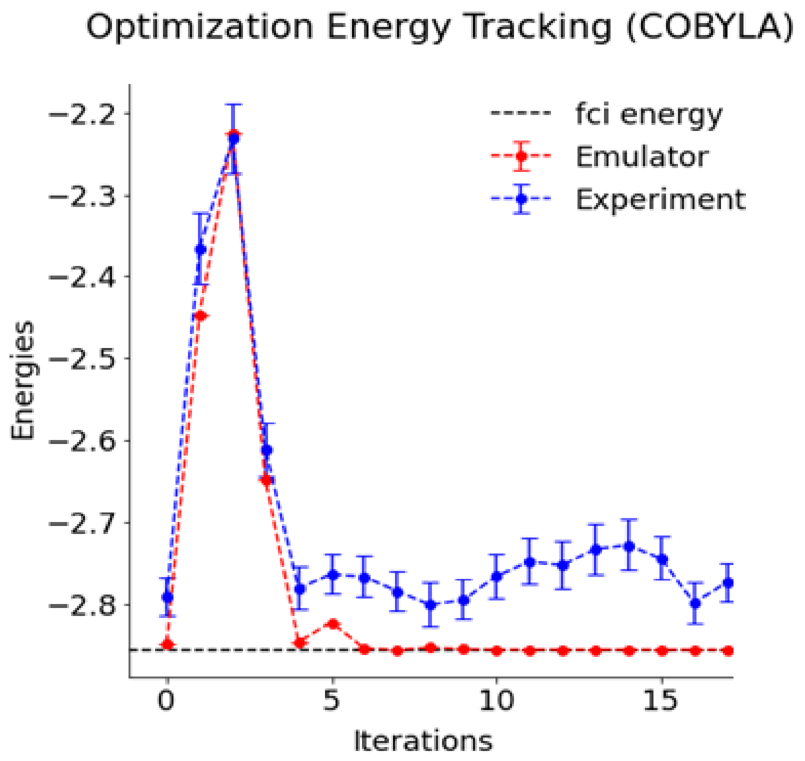
4. Measurements are extrapolated to single noiseless estimate:

$$\hat{E}_K^n(\lambda) = \sum_{i=0}^{n} \gamma_i \hat{E}_K(c_i\lambda)$$

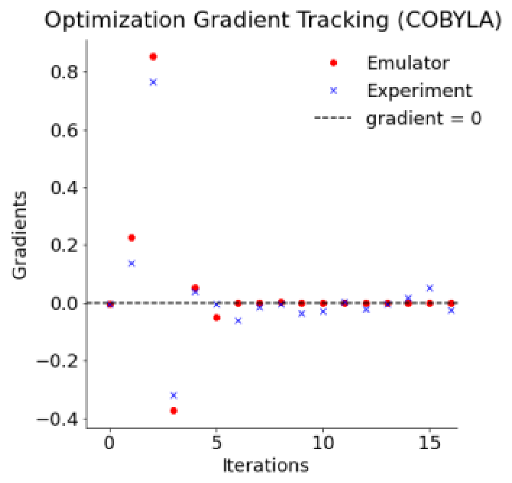
$$\sum_{i=0}^{n} \gamma_i = 1, \qquad \sum_{i=0}^{n} \gamma_i c_i^k = 1, \qquad k = 1 \ldots n$$

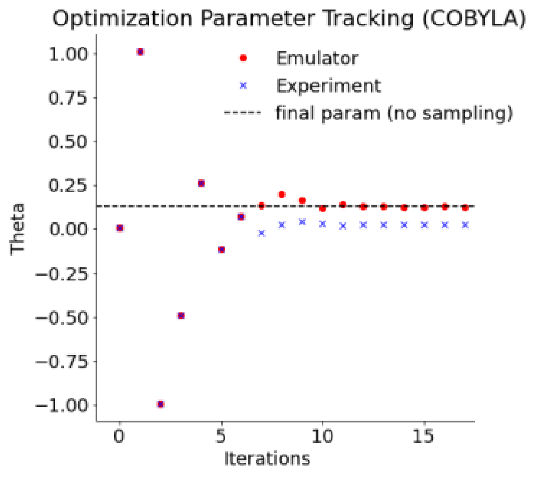
K. Temme, S. Bravyi, J.M. Gambetta, Error mitigation for short-depth quantum circuits. Physical Review Letters, **119** (18), (2017)

Richardson extrapolation



```
Px q[target1]
Px q[target2]
MS q[target2] q[target1] 0 pi2
Rz q[target2] theta
MS q[target2] q[target1] 0 npi2
```





*theta is tuned to construct the formation of the Hamiltonian

COBYLA method: iteratively approximates next candidate solution based on linear programming and continues to evaluate using original objective and constraint functions to find the minimum of the Hamiltonian

Adding an example stretch factor:

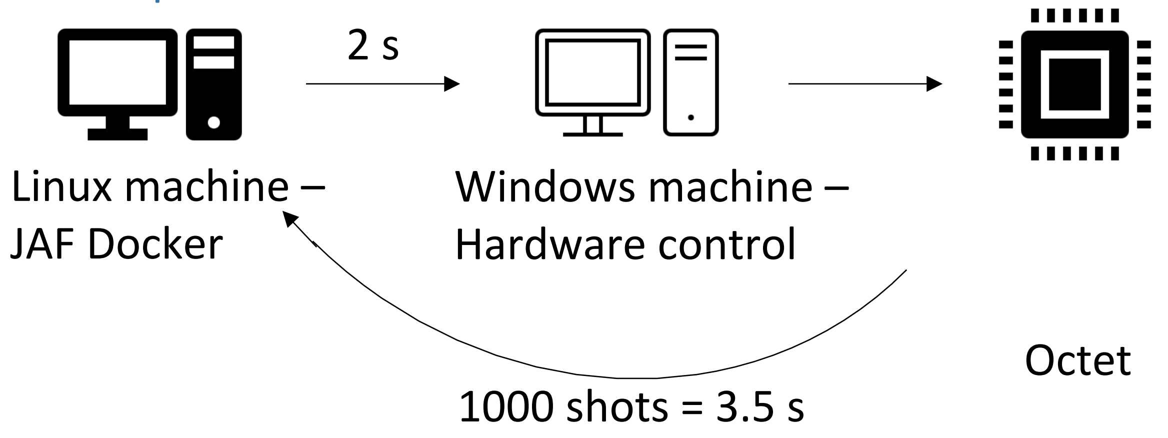
```
Px q[target1]
Px q[target2]
loop MS_loop{MS q[target2]
q[target1] 0 pi2
MS q[target2] q[target1] 0 npi2}
MS q[target2] q[target1] 0 pi2
Rz q[target2] theta
MS q[target2] q[target1] 0 npi2
loop MS_loop{MS q[target2]
q[target1] 0 pi2
MS q[target2] q[target1] 0 npi2}
```

running with JAF

How does (did) JAF work with the hardware?

- Fetches data from notebook or other application and stores as binary data in the docker container (QSCOUT unaware of application)
- When the compiler stores the data, it prohibits recalculation
- Those subcircuits are then executed on Octet

- In the older regime, JAF created a brand new Jaqal file for each new step of a user's code


2 s
Linux machine – JAF Docker
Windows machine – Hardware control
Octet
1000 shots = 3.5 s

Two main inconveniences:

- let parameters could only correspond to one value at a time
- A new Jaqal file was created for each new step of the code

```
let MS_loop 2
let ZZ 1
let XZ 0
let ZX 0
let XX 0

prepare_all
Px q[target1]
Px q[target2]
loop MS_loop{MS q[target2] q[target1] 0 pi2
MS q[target2] q[target1] 0 npi2}
MS q[target2] q[target1] 0 pi2
Rz q[target2] theta
MS q[target2] q[target1] 0 npi2
loop MS_loop{MS q[target2] q[target1] 0 pi2
MS q[target2] q[target1] 0 npi2}
loop XZ { Sy q[target1]
Px q[target1] }
loop ZX {Sy q[target2]
Px q[target2]}
loop XX{ Sy q[target1]
Px q[target1]
Sy q[target2]
Px q[target2]}
measure_all
```

batching with overrides for VQE

Batching Implementation:

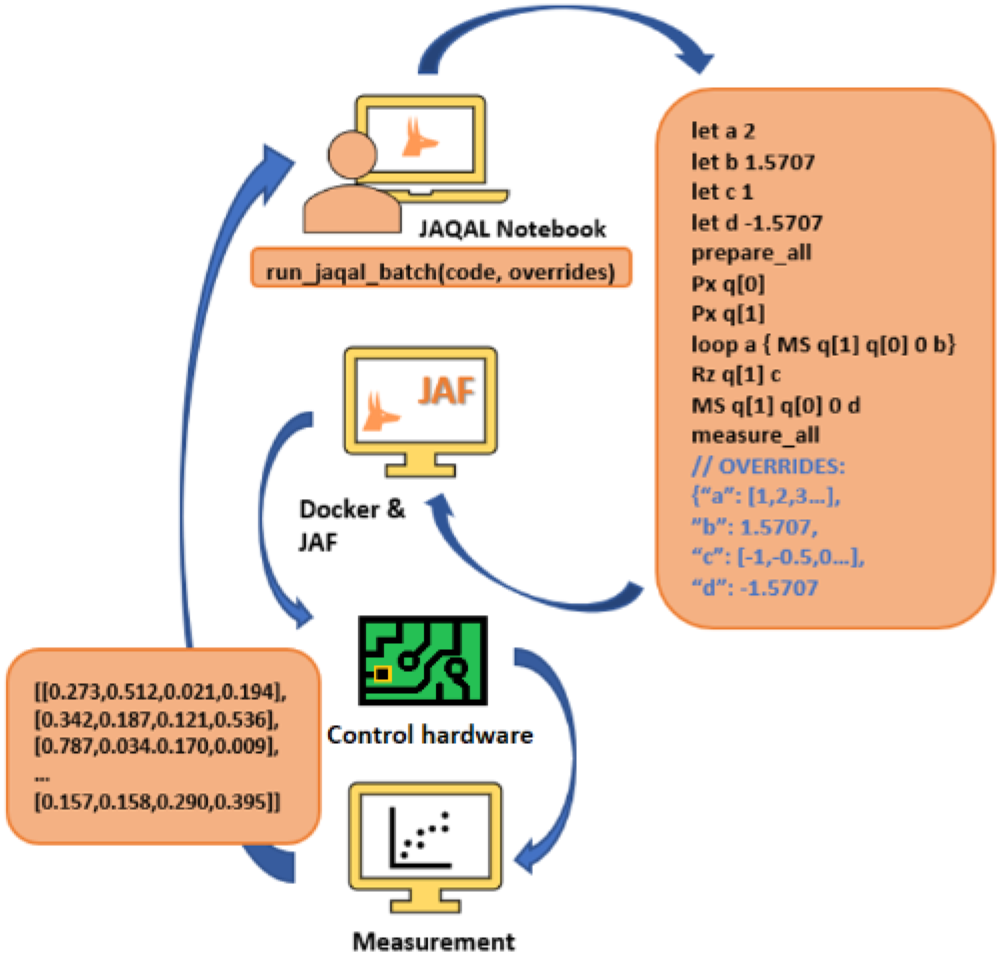
- Batching with overrides

●	●	●	●
ZZ x 4	XZ x 2	ZX x 2	XX

→ 9 Pauli terms in the Hamiltonian

- Identical circuits with varying parameters
- Θ = 0.01 (as an example)
- # of shots =1000 (as an example), although we also have the support for changing this amongst projections
- 1 batch = 1 optimizer step

  18 steps vs. 162 steps (9 proj x 18 opt. steps) where each JAF call was 2 s
- Reduction from 29 minutes → 19.5 minutes



```
let a 2
let b 1.5707
let c 1
let d -1.5707
prepare_all
Px q[0]
Px q[1]
loop a { MS q[1] q[0] 0 b}
Rz q[1] c
MS q[1] q[0] 0 d
measure_all
// OVERRIDES:
{"a": [1,2,3…],
"b": 1.5707,
"c": [-1,-0.5,0…],
"d": -1.5707
```

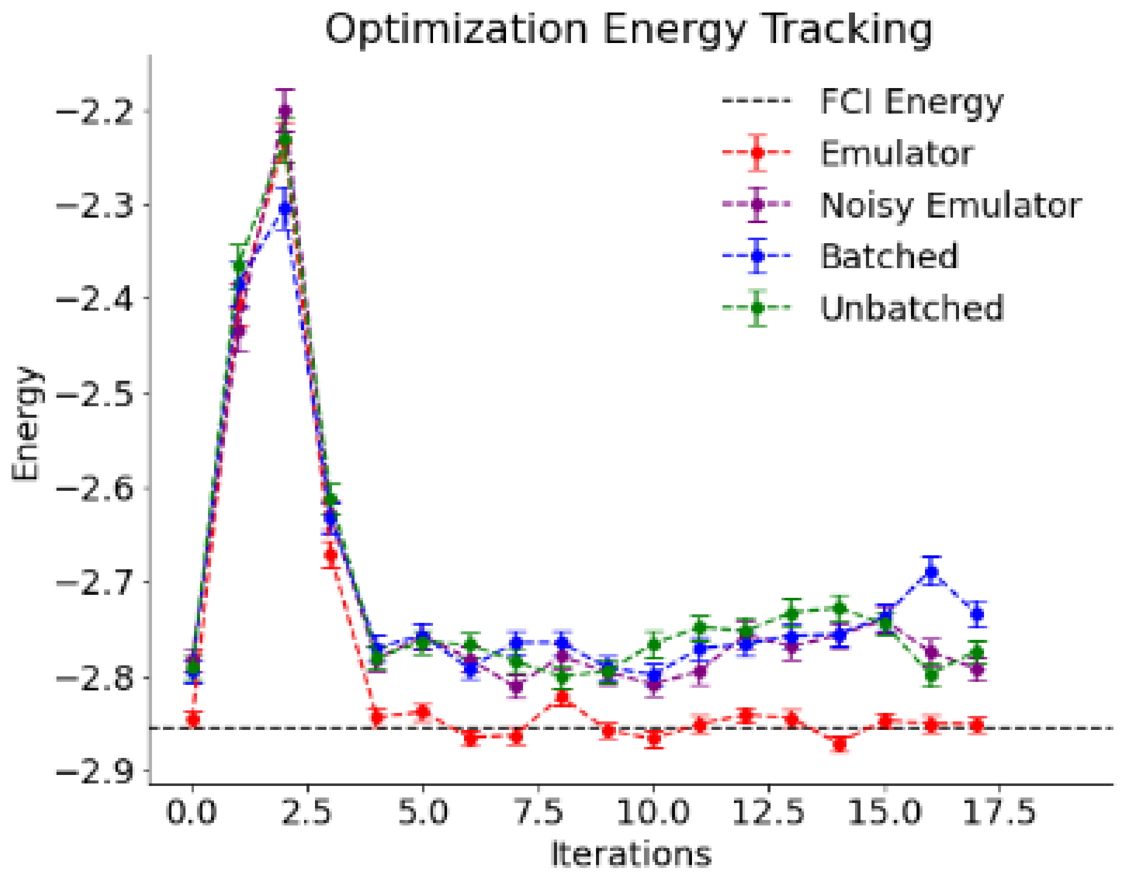
```
[[0.273,0.512,0.021,0.194],
[0.342,0.187,0.121,0.536],
[0.787,0.034.0.170,0.009],
…
[0.157,0.158,0.290,0.395]]
```

*When calibration parameters do not change and gate data is similar, the circuit is compressed by compiling everything upfront

results – batching Tufts VQE code

- num_MS = 1
- One theta value per optimizer step, although we have the ability to incorporate multiple thetas
- Number of experiments set to 1000
- Batched results are performing at the same accuracy in a shorter amount of time

Next and final step: vary across num_MS to incorporate the scaling factor to perform Richardson extrapolation



Noisy emulator COBYLA: simulated error model using common experimental control errors (power, frequency, phase, etc.)

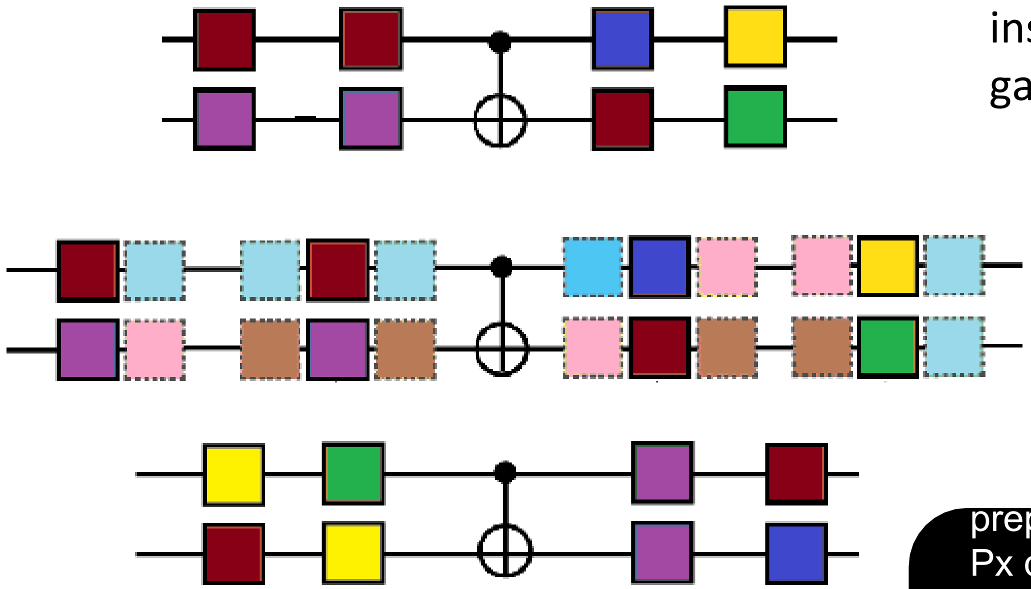
outline

- **Exploring the Jaqal-verse**
  - Jaqal
  - JaqalPaq
  - JaqalPaw
  - Jaqal Application Framework
- **Implementing experiments on the hardware**
  - QSCOUT
  - Octet
- **Integration of batching in Jaqal**
  - Tufts University - Richardson extrapolation requiring scaling factor
    - ➢ Batching with let parameter dictionaries
  - Oak Ridge National Laboratories - randomized compiling requiring large number of circuits to be run
    - ➢ Batching through indexing

randomized compiling comparison

ORNL goal: To characterize and mitigate coherent errors in a trapped ion quantum processor using hidden inverses → *How does randomized compiling compare to the use of hidden inverses?*

- For randomized compiling, twirling gates are inserted before and after each single qubit gate and are then compiled together



- Each set of circuits starts as a combination of single qubit gates and CNOT gates

```
prepare_all
Px q[1]
Sy q[1]
Sxd q[0]
CNOT q[1] q[0]
Rz q[0] alpha
CNOT† q[1]
q[0]
Syd q[1]
Sx q[0]
measure_all
```

```
prepare_all
Px q[1]
Sy q[1]
Sxd q[0]
Twirl q[0]/q[1]
CNOT q[1] q[0]
Twirl q[0]/q[1]
Rz q[0] alpha
Twirl q[0]/q[1]
CNOT q[1] q[0]
Twirl q[0]/q[1]
Syd q[1]
Sx q[0]
measure_all
```

```
Px
q[1]
Py
q[0]
or
Pz
q[1]
or
Py
q[0]
Py
q[1]
or…
```

```
Py
q[1]
Pz
q[0]
or
Pz
q[1]
or
Px
q[1]
Pz
q[0]
or…
```

```
Py
q[1]
Py
q[0]
or
Pz
q[0]
or
Pz
q[1]
or…
```

```
Px
q[1]
Pz
q[0]
or
Pz
q[1]
Pz
q[0]
or
```

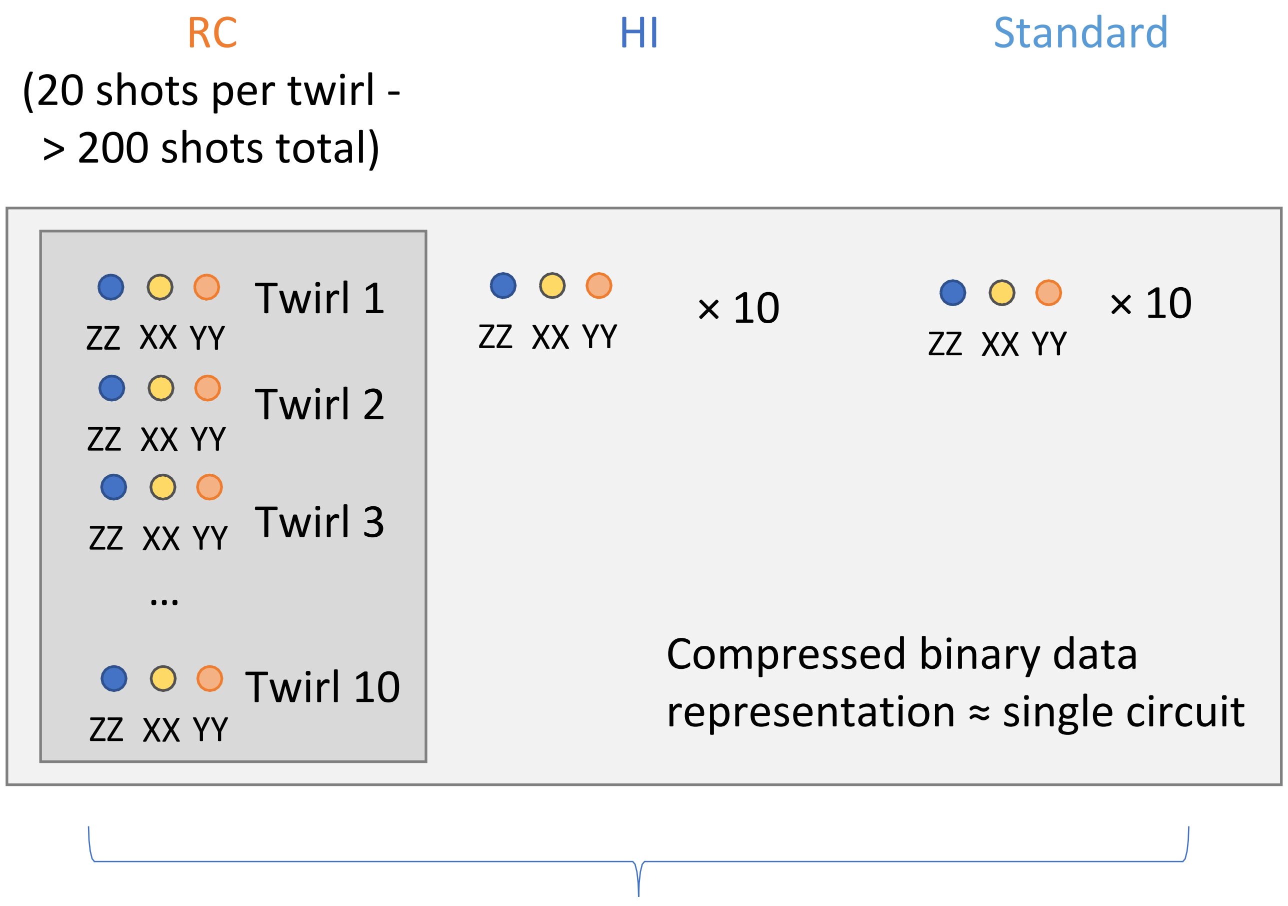
Our biggest challenge was performing all these randomizations – over 2 hours, creating new Jaqal file at each step

S. Majumder, C.G. Yale. (2022) "Characterizing and mitigating coherent errors in a trapped ion quantum processor using hidden inverses." arXiv:2205.14225.

batching VQE through indexing

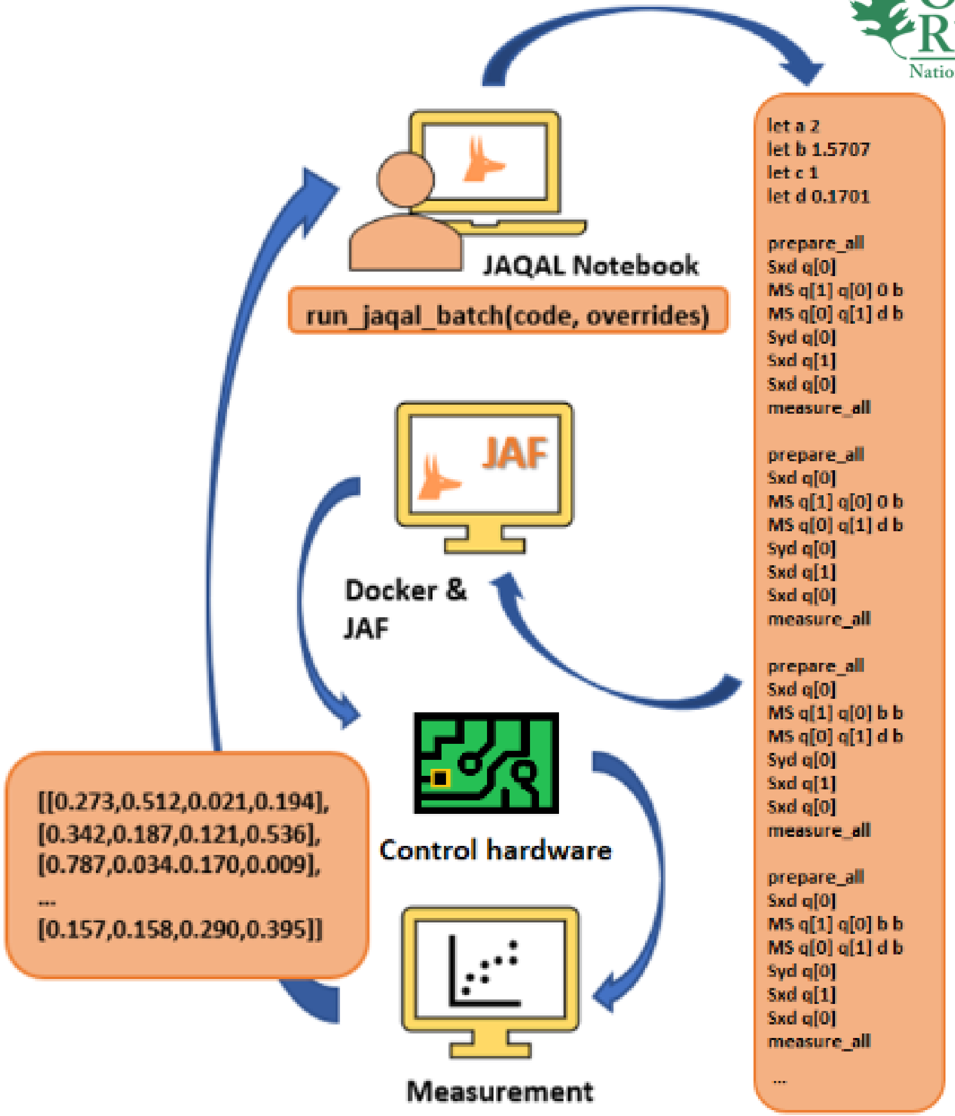
Batching implementation:

- Batching with indexing
- Cannot have different values for # shots within indexing batch



Compressed binary data representation ≈ single circuit

1 batch = 90 points ((10 RC + 10 HI + 10 standard) × 3 proj.)
Total 41 batches (41 α) = 3690 points

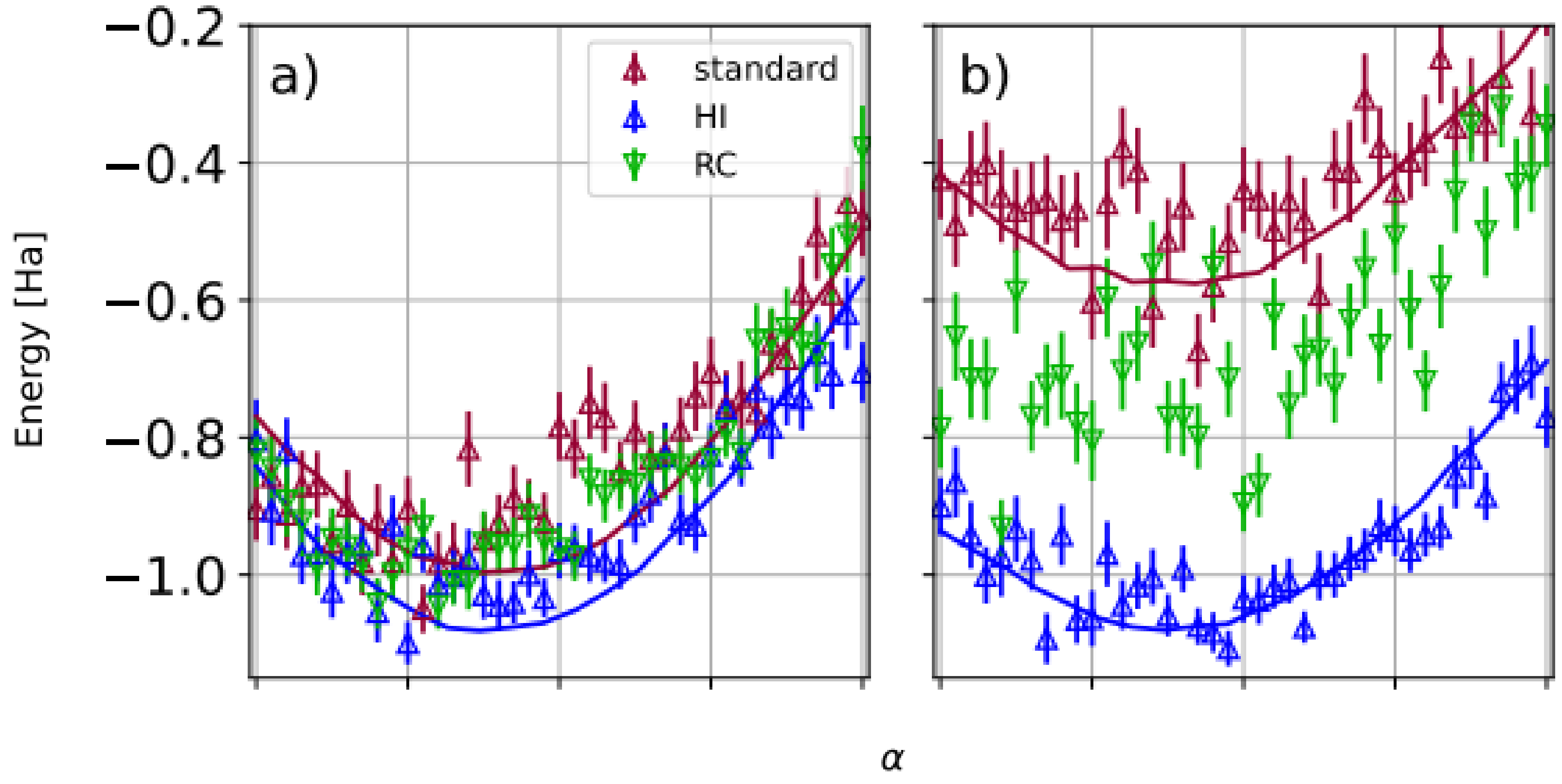


```
run_jaqal_batch(code, overrides)
```

```
[[0.273,0.512,0.021,0.194],
[0.342,0.187,0.121,0.536],
[0.787,0.034.0.170,0.009],
...
[0.157,0.158,0.290,0.395]]
```

```
let a 2
let b 1.5707
let c 1
let d 0.1701

prepare_all
Sxd q[0]
MS q[1] q[0] 0 b
MS q[0] q[1] d b
Syd q[0]
Sxd q[1]
Sxd q[0]
measure_all

prepare_all
Sxd q[0]
MS q[1] q[0] 0 b
MS q[0] q[1] d b
Syd q[0]
Sxd q[1]
Sxd q[0]
measure_all

prepare_all
Sxd q[0]
MS q[1] q[0] b b
MS q[0] q[1] d b
Syd q[0]
Sxd q[1]
Sxd q[0]
measure_all

prepare_all
Sxd q[0]
MS q[1] q[0] b b
MS q[0] q[1] d b
Syd q[0]
Sxd q[1]
Sxd q[0]
measure_all

...
```

results – batching ORNL VQE code



- Intentionally introduced coherent errors to compare RC to HI (refer to arXiv paper for details)
- Significant reduction in total JAF calls and dead time (41 vs. 3690 JAF calls)

S. Majumder, C.G. Yale. (2022) “Characterizing and mitigating coherent errors in a trapped ion quantum processor using hidden inverses.” arXiv:2205.14225.

summary

- Dominant sources of error in many ion trap testbed systems is drift due to variation in often controllable parameters. Speeding up experiment run-time helps to mitigate the effects of drift.

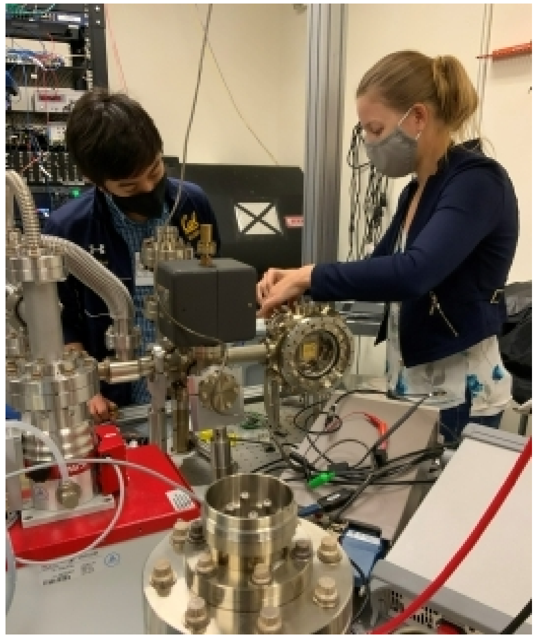
- Batching in Jaqal speeds experimental run-time through the reduction of communication and upload times.
  - ➢ **Batching with let parameters** creates a let parameter override dictionary that is used when circuits are identical but need to be run using different values for calibration parameters.
  - ➢ **Batching through indexing** compresses the representation of all circuits into subcircuits so that multiple circuits can be calculated once and executed by uploading minimal sets.

- Both techniques were useful in running completely different sets of code in Jaqal in collaboration with Tufts University and Oak Ridge National Laboratories, making running their code more feasible and/or convenient.

QSCOUT Team

Email: qscout@sandia.gov (mailing list) **Web:** https://qscout.sandia.gov **Jaqal:** https://gitlab.com/jaqal/jaqalpaq

QSCOUT Experimental Team



<u>Experimental</u>
Susan Clark, PI
Christopher Yale
Dan Lobser
Melissa Revelle
Matt Chow
Ashlyn Burch
Megan Ivory
Theala Redhouse
Craig Hogle
Dan Stick

Mechanical & Optical <u>Engineering</u>
Brad Salzbrenner
Madelyn Kosednar
Ted Winrow
Bill Sweatt
Dave Bossert
Josh Lane

Theory & <u>Software</u>
Andrew Landahl
Ben Morrison
Kenny Rudinger
Antonio Russo
Brandon Ruzic
Jay Van Der Wall
Josh Goldberg
Tim Proctor
Kevin Young

Trap Fabrication <u>and Packaging</u>
Becky Loviza
Ed Heller
Chris Nordquist
Ray Haltli
Tipp Jennings
Ben Thurston
John Rembetski
Eric Ou
Matt Delaney
Zach Meinelt
Nick Jimenez

<u>Collaborators/Users</u>
Ken Brown, Duke
Peter Love, Tufts
Oliver Maupin, Tufts
William Simon, Tufts
Titus Morris, ORNL
Swarnadeep Majumder, ORNL
Jacek Jakowski, ORNL
Raphael Pooser, ORNL

2021 R&D 100 WINNER