Introduction to TGCC quantum stack

Harold Erbin

IPHT, CEA Saclay (France)

2024–2026







Outline: 1. Introduction

Introduction

Onboarding

Running quantum computations

TGCC QPUs and Bull's ecosystem

Real-world research setup

Conclusion

TGCC: Très Grand Centre de calcul

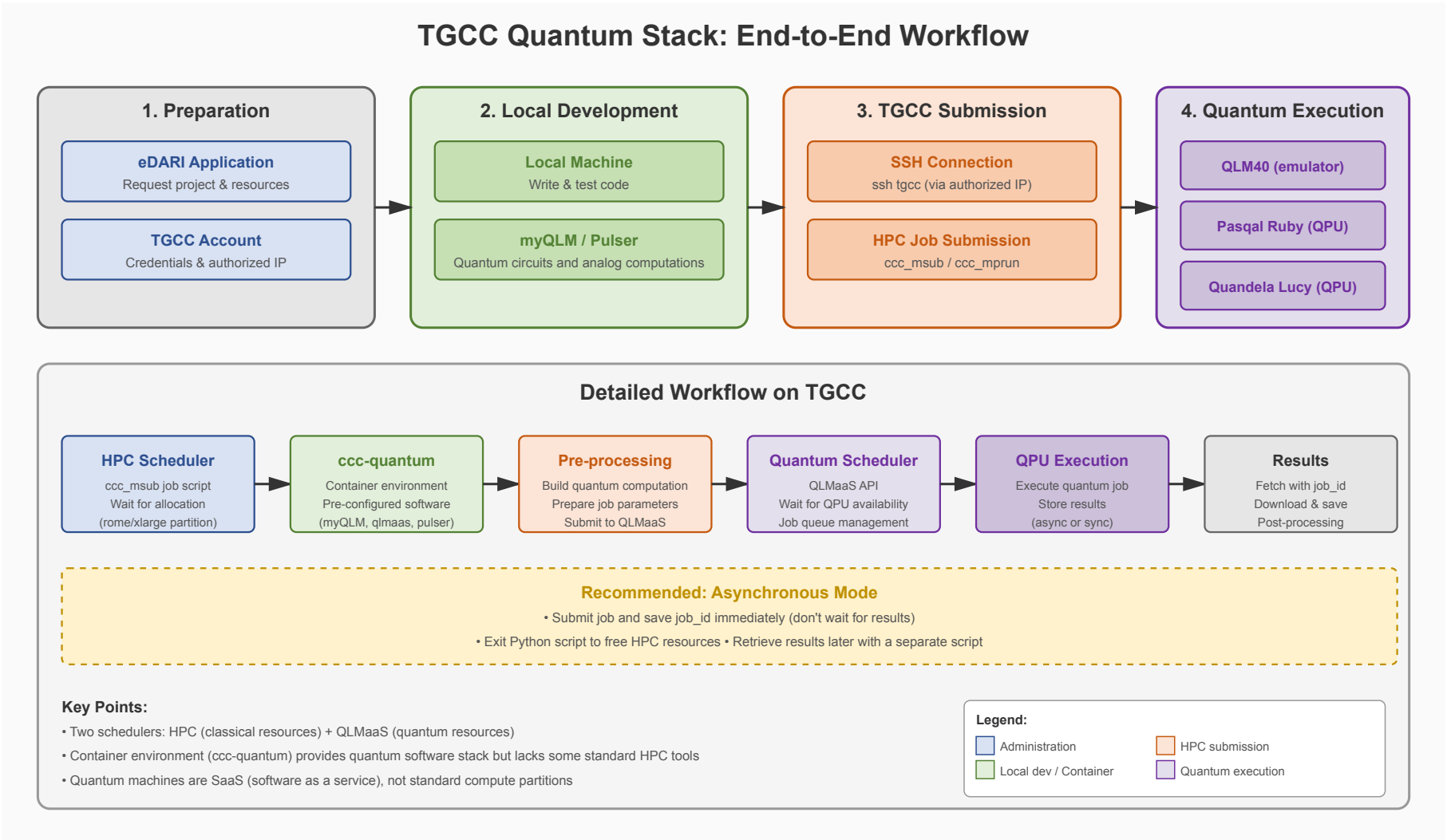
- National HPC center hosted by CEA (France)
- Provides computational resources for research projects (allocated via eDARI)
  - classical partitions (Rome, GPU clusters)
  - quantum resources (Bull emulator, Pasqal and Quandela QPUs)

  [genci.fr/tres-grand-centre-de-calcul-du-cea-tgcc]
- Quantum resources are accessed through a hybrid architecture
  - classical HPC scheduler for job management
  - SaaS (software as a service) layer for quantum machines
  - containerized environment (`ccc-quantum`) with pre-configured quantum software
- Access requires
  - a Genci account (attached to a project with allocated hours)
  - SSH connection through authorized IP (e.g., IPHT login node)
  - familiarity with both HPC job submission and quantum API workflows

Objectives

- Give a general introduction to the quantum stack at TGCC
- Explain the different methods for running code through the quantum stack
- Describe Bull's ecosystem (myQLM, Qaptiva, QLMaaS) and the available QPUs at TGCC
- Show a basic script to submit a job to Pasqal Ruby or the QLM40 emulator
- Present a real-world research pipeline from theory to hardware
- Share practical tips and common pitfalls

General overview



Outline: 2. Onboarding

Introduction

Onboarding

Running quantum computations

TGCC QPUs and Bull's ecosystem

Real-world research setup

Conclusion

Access problem: QLM40

- Even though someone is attached to a project with hours on the QLM40 **does not mean** that they are authorized to use it
- This is true even if the account creation email is saying that one can access the QLM40
- Diagnostic: see next slide
- Solution: write to `hotline.tgcc@cea.fr` and ask to check the access permissions

Access problem: QLM40 (diagnosis)

```
> from qlmaas.qpus import AnalogQPU
Traceback (most recent call last):
        File "connection.py", line 19, in qlmaas.connection
        File "connection.py", line 201, in
        ↪  qat.qlmaas.connection.QLMaaSConnection.__init__
        File "https.py", line 496, in
        ↪  qat.qlmaas.https.SecuredConnection.__init__
        File "https.py", line 343, in
        ↪  qat.qlmaas.https._check_response_code
RuntimeError: Code 401 - Unauthorized: Invalid certificate

The above exception was the direct cause of the following
↪  exception:
Traceback (most recent call last):
        File "<string>", line 1, in <module>
        File "__init__.py", line 34, in init qlmaas.__init__
        File "connection.py", line 23, in init qlmaas.connection
ImportError: Could not connect to the QLMaaS server. Please
↪  ensure a QLMaaS configuration file exists on your computer
```

HPC-QC – Hybrid computing

- Quantum machines $\neq$ standard compute partitions (like Rome or Skylake) but cloud SaaS (software as a service)
  - accessed through API
  - at TGCC: encapsulated in a container `ccc-quantum` with pre-configured softwares (myQLM/Qaptiva, etc.)
- `ccc-quantum` $\neq$ standard HPC environment:
  - no `module` command (no environment modules)
  - no `ccc_*` commands (`ccc_msub`, `ccc_myproject`, etc.)
  - limited Python libraries
  - GPU access is more complicated (requires additional flags)
  - fewer development tools (compilers, debuggers, version control)
- Monitoring of quantum resources usage is still limited
  - `ccc_myproject`: only for standard compute resources
  - no user-visible dashboard for quantum hours consumed
  - yearly email notification on quantum resource usage (?)
- Missing packages in container: write to TGCC hotline explaining why the package is needed

Cost and resource awareness

- Quantum and classical resources allocated separately in eDARI
  - compute partition hours and quantum machine hours are independent budgets
- Classical compute partitions
  - check usage: `ccc_compuse` (standard) or `ccc_myproject`
  - over-consumption flagged if accessing partitions without allocated hours
- Quantum resources (QPU / emulator)
  - billing model not clear
  - no consumption dashboard yet
- Queue times vary significantly
  - QLM40: typically fast (emulator on dedicated HPC)
  - Ruby: no queue at TGCC (jobs execute immediately)
  - Lucy: ?
- Plan carefully: wasted hours on failed jobs or debugging cannot be recovered

Notes on compute partitions

- Access to several partitions (v100l, v100xl, hybrid, xlarge) is provided for visualization, but without hours
  $\rightarrow$ user is flagged as over-consuming
- Problematic to forget classical compute for different reasons:
  - a partition must be provided for submitting TGCC scripts
  - cannot run simulations (except for small systems)
  - cannot submit jobs which requires a lot of pre-processing
  - no access to GPU
- Diagnostic

```
$ ccc_compuse
genxxxx@hybrid      over-consumption
```

- Solution: request dynamical access to compute partitions when filling the eDARI form

Documentations

- Internal website: `www-eu.ccc.cea.fr` (`PDF version`)
  - more information
  - sometimes different
  - more complicated to access (need SSH dynamic port forwarding if IP address not registered at TGCC)
- Public website: `www-hpc.cea.fr` (`PDF version`)
  - `Virtualization (containers)` (but not useful for our purpose)
  - `Quantum stack`
- Documentation on the quantum stack is scarce and confusing (though it improved recently).
  - if you have a problem, don't stay stuck for a long time (it may be a system problem), but write to TGCC hotline for help
  - the hotline is responsive and very helpful (special thanks to Marc Joos)
  - example: the TGCC documentation points to Qaptiva documentation, which starts by explaining how to connect to a device; but this is not necessary (and impossible) at TGCC

Connecting to TGCC

- Security settings
  - Access only from authorized IP address
  - Cannot use SSH key, only password
    ($\rightarrow$ no automated login without `sshpass`)
- `~/.ssh/config`

```
# Example: lab or university server with authorized IP
Host gateway
    Hostname <gateway.fr>
    User <gateway_user>
    IdentityFile ~/.ssh/id_rsa

Match host tgcc !exec "ifconfig | grep <ip>"
    ProxyJump gateway
Host tgcc
    Hostname irene-eu.ccc.cea.fr
    User <tgcc_user>
```

  `ProxyJump` + `Match`: conditional routing through authorized IP
- Connect with: `ssh tgcc`

Outline: 3. Running quantum computations

Introduction

Onboarding

Running quantum computations
- Scripts
- Jobs

TGCC QPUs and Bull's ecosystem

Real-world research setup

Conclusion

General process

1. Run script or submit job to HPC job controller
2. Wait for resources allocations
3. Perform pre-processing
4. Send job to quantum job controller
5. Wait for QPU availability
6. QPU executes the job, stores and possibly returns the results

Outline: 3.1. Scripts

Introduction

Onboarding

Running quantum computations
- Scripts
- Jobs

TGCC QPUs and Bull's ecosystem

Real-world research setup

Conclusion

Interactive session (local launch)

```
$ pcocc-rs run ccc-quantum
$ ipython
```

- Pros
  - check quantum stack environnement
  - can run multiple actions within the stack
  - useful for testing on `ipython`
- Cons
  - run on the login node (cannot use too many resources)
  - inconvenient for managing jobs

Interactive session (remote launch)

```
$ ccc_mprun -C ccc-quantum -p hybrid -s
# can specify more options:
$ ccc_mprun -C ccc-quantum -p hybrid -c 2 -m work -s
# with GPU
$ ccc_mprun -C ccc-quantum -E '--ctr-module nvidia' -p hybrid -n
↪  1 -c 48 -s
```

- Pros
  - run a script with more resources (including GPU)
- Cons
  - requires compute partition
  - not really interesting for interactive session

Execute code with container (local launch)

```
$ pcocc-rs run ccc-quantum -- python3 sim.py
```

- Pros
  - run a single script from command line
  - run several jobs from a bash script
- Cons
  - run on the login node (cannot use too many resources)

Execute code with container (remote launch)

```
$ ccc_mprun -C ccc-quantum -p hybrid -- python3 sim.py
```

- Pros
  - run a script with more resources (including GPU)
  - run several jobs with `ccc_msub`
  - necessary for long pre-processing
- Cons
  - requires compute partition

Visual session (1)

1. Book graphical session session:

```
# TGCC terminal
$ ccc_visu virtual -p hybrid
```

2. Wait for resources to be allocated
3. Open the virtual desktop NiceDV (inside a browser):
   3.1 If not on authorized computer (like CEA intranet), create dynamic port forwarding

   ```
   # local computer terminal
   $ ssh -v -N -D 1338 authorized_gateway
   ```

   and configure your browser to use this SOCKS proxy
   3.2 Visit the URL indicated in the terminal

   ```
   https://visu-eu.ccc.cea.fr/...
   ```

4. Run what you want in NiceDV (jupyter notebook. . .):

```
# graphical session terminal
$ pcocc-rs run ccc-quantum
$ jupyter lab
$ firefox http://127.0.0.1:8888/lab?token=...
```

5. When done, close the browser and the initial session

Visual session (2)

- Pros
  - use Jupyter notebooks on quantum stack
  - display plots in live
- Cons
  - may need SSH dynamic port forwarding
  - wait time to initialize (must request node)
  - requires compute partition

Outline: 3.2. Jobs

Introduction

Onboarding

Running quantum computations
- Scripts
- Jobs

TGCC QPUs and Bull's ecosystem

Real-world research setup

Conclusion

Submitting jobs: bash

```
#!/bin/bash

for L in $(seq 3 6); do
   echo "L = $L"
   pcocc-rs run ccc-quantum -- python3 sim.py -L $L
done
```

- Pros
  - simple, not need to use ccc
  - perfect for running many quantum jobs with little pre-processing
- Cons
  - run on the login node (cannot use too many resources)

Submitting jobs: ccc_msub (1)

```
#!/bin/bash

#MSUB -r qlm_test              # job name
#MSUB -A genxxxx               # project id
#MSUB -q hybrid                # partition
#MSUB -n 2                     # number of tasks
#MSUB -T 300                   # time limit (s)
#MSUB -o msg_test_%I.out       # standard output
#MSUB -e msg_test_%I.err       # error output
#MSUB -F                       # Flux plugin

# Send email when job begin and end
#MSUB -@ harold.erbin@cea.fr:begin,end

set -x
# directory where job has been submitted
cd ${BRIDGE_MSUB_PWD}

export SCRIPT_DIR=${BRIDGE_MSUB_PWD}/scripts

for L in $(seq 3 6); do
   pcocc-rs run ccc-quantum -- python3 $SCRIPT_DIR/sim.py $L
done
```

Submitting jobs: ccc_msub (2)

- Pros
  - run a script with more resources (including GPU)
  - can perform long pre-processing
- Cons
  - requires compute partition
  - script goes through two job schedulers (HPC and quantum)

Notes:

- submit with: `ccc_msub job.sh`
- could also replace last line with (but less recommended):
  `ccc_mprun -C ccc-quantum python3 sim.py`
- Flux: run multiple jobs in parallel (background execution within a single allocation)
- cannot use option: `#MSUB -C ccc-quantum`
  $\rightarrow$ container must be used with `pcocc-rs` or `ccc_mprun -C`

Outline: 4. TGCC QPUs and Bull's ecosystem

Introduction

Onboarding

Running quantum computations

TGCC QPUs and Bull's ecosystem

Real-world research setup

Conclusion

myQLM, Qaptiva, QLMaaS

- Packages (PyPI)
  - `myqlm`: main package to build a quantum program and run jobs
  - `qlmaas`: server interaction (QLM Access, QLM as a Service)
- Free and paid tiers
  - free: accessible by everyone (myQLM: general software, dummy emulators)
  - paid: QPU server, improved emulators (Qaptiva: available at TGCC through `ccc-quantum`)
- TGCC configuration file (inside `ccc-quantum`)

```
$ echo $QLMAAS_DIR
/.qlmaas/config.ini
```

(but not really useful)

Available quantum resources – Overview

	QLM40	**Ruby**	**Lucy**
Type	Emulator	QPU	QPU
Technology	Classical HPC	Neutral atoms	Photonic
Capacity	40 qubits (exact) $\sim$ 100 (MPS)	100 qubits	12 qubits 24 modes
Connectivity	N/A	Geometry-dependent	All-to-all
Best for	circuit simulations analog simulations debugging	many-body physics optimization quantum ML	gate-based circuits boson sampling quantum ML

All accessible through `qlmaas.qpus` inside `ccc-quantum` container

Available quantum resources – `qlmaas.qpus`

- Local computer (dummy emulator development)
  - `PyLinAlg` (gate-based)
  - `QutipQPU` (analog)
- QLM40 (Bull): classical computer dedicated to quantum emulation
  - `LinAlg` (gate-based)
  - `AnalogQPU` (analog, exact)
  - `MPSTraj` (analog, tensor networks)
  - etc.
- Ruby (Pasqal): neutral-atom QPU
  - `PasqalQPU`
  - packages: `pulser`, `qoolqit` (binding: `pulser_myqlm`)
- Lucy (Quandela): photonic QPU
  - `QuandelaQPU`
  - packages: `perceval`

Async mode

- To avoid wasting compute resources, run in async mode:
  - **do not** wait for QPU to return results (blocking call)
  - store job id (and other useful information) and exit Python script immediately after submission
  - run another script to retrieve the results when job is finished
- Why async? Three practical reasons:
  - SSH connection timeout
  - HPC compute hours are billed – keeping a node idle while waiting for QPU results wastes allocation
  - QPU queues can be long (minutes to hours) $\rightarrow$ better to submit and move on
- Remember that `ccc-quantum` is not a standard HPC environment (less features are available, etc.)

Submit quantum jobs: Pulser (1)

```
from pulser_myqlm.ising_aqpu import IsingAQPU
try:
    from qlmaas.qpus import AnalogQPU, PasqalQPU
except ImportError:
    QLMAAS_AVAILABLE = False
        # could use QutipQPU locally
else:
    QLMAAS_AVAILABLE = True
```

Explanations

- `qlmaas` can be imported only inside `ccc-quantum` at TGCC
- need to make sure that the code works on all platforms, and that we can diagnose if we are at TGCC or not

Submit quantum jobs: Pulser (2)

```python
def submit_job(seq, job_id, backend=None, shots=2000):
    if backend in ("pasqal", "qlm40") and QLMAAS_AVAILABLE is False:
        raise RuntimeError("QLMaaS is not available.")

    if backend == "pasqal":
        qpu = PasqalQPU()
    elif backend == "qlm40":
        qpu = AnalogQPU()
        nbshots = 0  # AnalogQPU returns probabilities
    else:
        qpu = None

    qpu = IsingAQPU.from_sequence(seq, qpu=qpu)
    job = IsingAQPU.convert_sequence_to_job(seq, nbshots=nbshots)
    results = qpu.submit(job)

    if backend in ("pasqal", "qlm40"):
        # wait for results; DON'T do this
        # results = results.join()
        print(f"Job info: {results.get_info()}")
        job_id = results.job_id
        # TODO: save for reusing later (json, etc.)

    return results
```

Submit quantum jobs: Pulser (3)

```
import qat

def fetch_job(job_id, path=None):
    connection = qat.qlmaas.connection.QLMaaSConnection()
    # 1: pending, 2: running, 3: done
    status = connection.get_status(job_id)

    results = None
    try:
        if status == "done":
            results = connection.get_result(job_id)
    except qat.comm.qlmaas.ttypes.QLMServiceException:
        pass

    if results is not None and path is not None:
        # save results
        results.dump(path)

    return results

# load results from file (from dump)
results = qat.core.Result.load(path)
```

Post-processing and analysis

- Results structure (`qat.core.Result`)
- Typical post-processing pipeline
  - load results from file: `qat.core.Result.load(path)`
  - aggregate shots, compute expectation values
  - noise mitigation (readout correction, etc.)
  - compare with emulator / theoretical predictions
- Important: run analysis outside the `ccc-quantum` container
  - download results to your local machine or a standard partition

Outline: 5. Real-world research setup

Introduction

Onboarding

Running quantum computations

TGCC QPUs and Bull's ecosystem

Real-world research setup

Conclusion

Job handling

- Script to manage jobs:
  - save job information after submission
  - fetch all jobs and check status
  - gather information on failed jobs and delete them
  - download results for finished jobs and perform post-processing
- Script to submit multiple jobs (most efficient: CLI)
  - determine if Ruby is available
  - no queue system: jobs are executed immediately when submitted (no waiting list)
    $\rightarrow$ good for fast iteration, but means one must manage concurrency and failed jobs oneself
  - issues with timeout (trick to bypass for now if needed)
- Other scripts
  - automate exchange between TGCC / laptop / institute cluster (makefile)
  - notify on phone on QPU status changes
  - retrieve QPU specs
    - needed for prototyping
    - not available in `pulser`

From theory to hardware

- Map theoretical problem to Rydberg Hamiltonians
  - encoding of degrees of freedom
  - level of approximation (truncate interactions...)
  - pulse sequence design
  - initial state preparation
- Adapt to physical devices
  - hardware constraints (number of qubits, interatomic distances, pulse magnitude...)
  - measurement constraints
  - timescales
  - sequence modulation
  - noise characterization / mitigation

First end-user application: Many-body quantum Score

Quantum many-body Hamiltonian

$$H_{\text{Ising}} = J \sum_i \sigma_i^z \sigma_{i+1}^z + h \sum_i \sigma_i^x$$

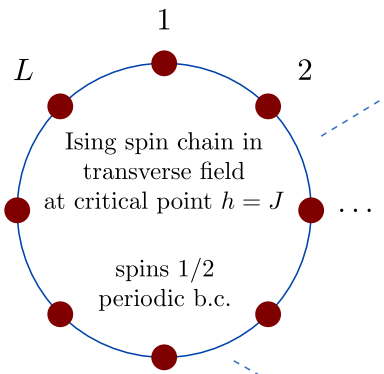
Hamiltonian dynamics (quench)

$$|\psi(t)\rangle = \exp\left(-\mathrm{i}Ht\right)|\psi(0)\rangle$$

Final state correlations at time $t_* \sim L$

$$\langle \sigma_i^z \sigma_j^z \rangle - \langle \sigma_i^z \rangle \langle \sigma_j^z \rangle$$

Benchmark score



Initial state:
$|\psi(0)\rangle = |\downarrow \cdots \downarrow\rangle$

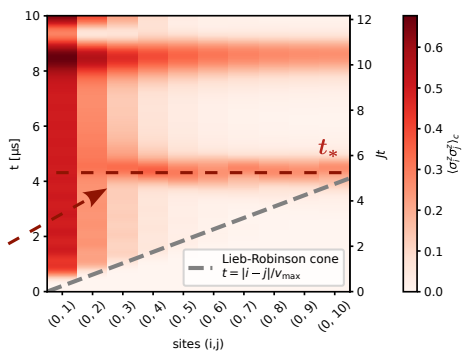
global correlation surge

QPU

- **analog**: possible when $H_{\text{hardware}} \approx H_{\text{Ising}}$
- **gate-based**: discretized time-evolution (Trotter) → circuit

CPU

theoretical solution: free fermions
→ $\boldsymbol{O(L^3)}$ complexity



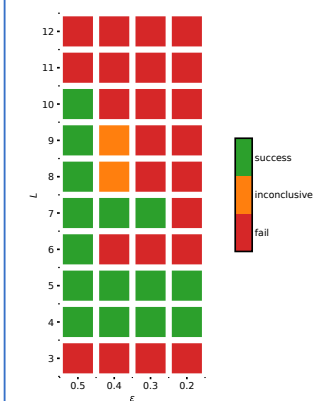
measured bitstrings → correlations

exact values of correlations at t_*

t_* = time needed for correlations to propagate to distance $\sim L/2$

compare: **pass** or **fail**? tolerance ϵ

tests passed up to size L
⇒ **score** $= L$



Implementation timeline

Apr.–Aug. 2024 first simulations and developments of the protocol

Sep.–Dec. 2024 first contacts with Pasqal to discuss the protocol and implementation on Ruby

Nov.–Jan. 2025 prototyping on the QLM40 emulator (test of the complete pipeline, from job submission to postprocessing)

May 2025 first runs of the protocol on Ruby (**world first** use-case of commercial neutral-atom QPU)

Jun.–Oct. 2025 better (but still partial) understanding of the different sources of noise on Ruby

Nov. 2025 inauguration of Ruby and first independent runs

Jan. 2026 preprint on arXiv [2601.03461, CEA-IPHT/Bull]

Theory vs. hardware reality: examples

- Qubit connectivity
  - Theory: arbitrary all-to-all interactions
  - Reality: geometry-dependent, $1/r^6$ decay
- Pulse sequences
  - Theory: arbitrary Rabi frequency and detuning profiles
  - Reality: hardware constraints on max amplitude, rise/fall times, discretization of time steps
- State preparation
  - Theory: perfect ground state or arbitrary initial state
  - Reality: non-adiabatic preparation, imperfect fidelity, residual excitations, preparation time
- Readout
  - Theory: projective measurement in any basis
  - Reality: fixed measurement basis, limited repetitions, dead time

Challenges as a first user

- Getting accustomed to national HPC environment (including administrative aspects)
  $\rightarrow$ large potential barrier for condensed matter and theoretical physicists (used to simulations on laptop / lab cluster)
- Dealing with two different job submission processes at TGCC (usual HPC scheduler + Qaptiva's quantum stack scheduler)
- No job queue for Ruby at TGCC
- Absence of maintenance period information and downtimes
- Day-to-day performance and calibration deviations on Ruby
- Numerous differences between the theoretical idealized protocol, and the actual implementation on the machine
- Exact noise model for Ruby not yet available

Key insights after 18 months

- Ruby's calibration drifts on a daily timescale
  $\rightarrow$ protocols must be robust to parameter variations
- The "no queue" system means timing matters
  $\rightarrow$ submit during low-usage windows for faster turnaround
- Noise is not just depolarizing
  $\rightarrow$ many different sources, hard to mitigate
- Emulator results $\neq$ QPU results even for small systems
  $\rightarrow$ validation pipeline needed before using hardware data
- Post-processing is 50% of the work
  $\rightarrow$ raw results need significant cleaning and analysis

Outline: 6. Conclusion

Introduction

Onboarding

Running quantum computations

TGCC QPUs and Bull's ecosystem

Real-world research setup

Conclusion

Status summary

- TGCC provides unique access to quantum resources
  - hybrid architecture: HPC + quantum SaaS
  - multiple machines (QLM40, Ruby, Lucy) for different use cases
- Getting started is possible but challenging
  - administrative and technical barriers for newcomers
  - documentation exists but is scattered and sometimes outdated
  - hotline support is essential and helpful
- Running research on quantum hardware requires patience and expertise
  - 18 months from protocol design to first hardware results
  - significant gap between theory and implementation
  - noise characterization and mitigation are ongoing challenges

Main takes

- France has the chances to host some of the most advanced quantum providers in the world.
- The quantum computing ecosystem at TGCC is powerful but complex. Success requires both technical skill and institutional knowledge.
- What I find remarkable: as a theoretical physicist who is mostly ignorant about lasers and other technical stuff, I can design the protocol of a quantum experiment on my laptop and run it on a device hundreds of kilometers away. This new research instrument alone is a revolution.