FRANCE HYBRID HPCQUANTUM INITIATIVE

Needs for middleware in Quantum Computing

Philippe DENIEL (philippe.deniel@cea.fr)

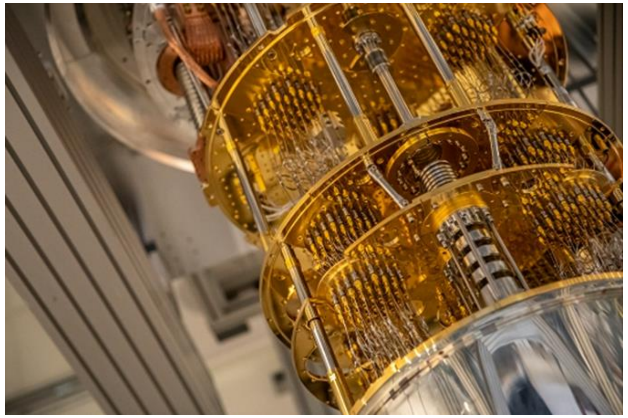
IT'S A KIND OF MAGIC…

- ❑ The end of the 19th century was an era of "science wonders", illustrated by writers such as Jules Verne

- ❑ Quantum Computing brings a new Science Wonders era
  - On some aspects, it's a kind of magic…

    "Any sufficiently advanced technology is undistinguishable from magic" **(Arthur C. Clarke)**

- ❑ When it comes to integrate QC and HPC we quickly come back to reality
  - I have a brand new QPU in my machine room, how do I make it work with the already installed HPC resources?



WHAT QC BRINGS TO HPC

- From a very high-level point of view
  - A QPU is an instrumented experience of quantum physics
  - This experience can be mapped to a mathematical problem (often a NP-C one)
  - QC makes it possible to address NP problems, inaccessible to standard HPC

- QPU is interesting in a HPC context
  - It should be seen as an accelerator to solve NP problems
  - QPU brings new compute methods, like GPUs

- QC is fundamentally to be used with HPC
  - You do NOT run an Operating System on a QPU and you NEVER will
  - QPUs are accelerators, like GPUs
  - QC is an ancillary compute technology

NUMERICAL METHODS: THE EASY COUPLING

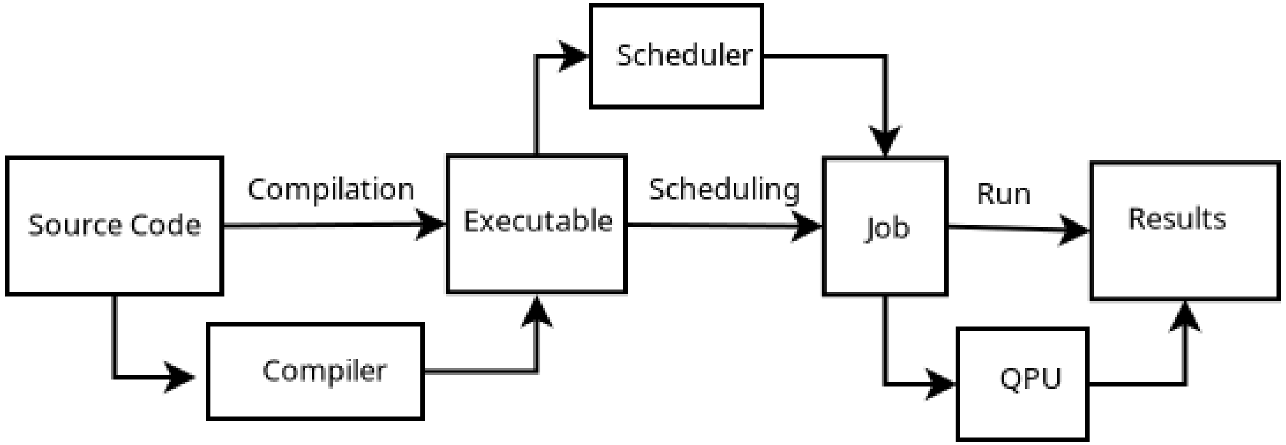
- QPUs are accelerator to address NP problems
  - NPC problems are particularly interesting
    - NP problems can be turned in NPC problems at the cost of a P transformation (doable via HPC)
    - Catalog of NPC problems exist (Karp's list): TSP, MIS, QUBO, SAT,…

- A natural approach: building libraries focused on numerical problems
  - It's natural to think about building libraries solving classical NPC problems
  - This will hide the complexity of QPU programming as well as providing QC benefits
  - This path is natural (e.g: QUBO is implemented by D-Wave, MIS is implemented by Pasqal)

- Is it enough ?
  - It just offer an algorithmic coupling
  - The system related aspects are not addressed by this approach

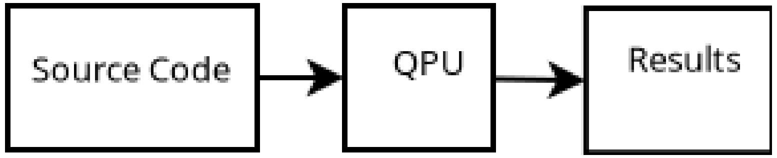
SYSTEM INTEGRATION: WHAT WE NEED

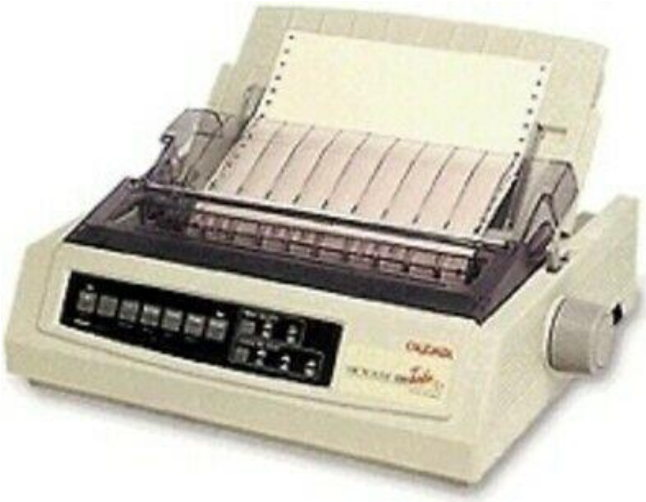
- Standard HPC relies on well known and well defined concepts
  - Source code: human-readable text describing the program as a set of instructions
  - **executable** : a set of instructions to be executed on the CPU
  - **compilation**: operation that turn a source code into an executable
  - **job**: running the executable once or several times to perform computation
  - **scheduling**: optimizing the use of compute resources to run as many jobs as possible
- From a very high-level point of view, HPC/QC integration should look like this



WHAT WE HAVE: THE QUANTUM PRINTER

- What is generally provided is
  - A python framework able to submit computation to the QPU
  - Most of the time, that's all we have





- We know this model well: this is how a USB attached printer works!
  - Today's QPU are just “Quantum Printer”

- **This is not enough!! ;-)**

NEED FOR STANDARDS

❑ HPC interfaces are standardized

- Protocols and paradigms exist, each is well described by documents (usually RFC from the IETF)

❑ No established standard currently exists in QC

- The word “qubit” itself is quite polymorphic and has different meaning across vendors
- Standardization international groups are working on this matter

❑ In order to perform HPC/QC integration, you’ll need to forge your own weapons

- Define a common QC vocabulary
- Define a software stack, compatible with HPC integration

❑ Performances

- Linpack has defaults, build it is a strong backbone in the HPC community via the Top500
- QC has nothing like Linpac… what should QC benchmarks be? (BACQ)

SCHEDULING / ACCOUNTING / AUTHENTICATION

❑ Scheduler point of view: a QC job is actually a mix of HPC and QC steps

- The HPC bootstraps the simulation
- This HPC steps do “rifles” of QC jobs (such as QAOA)
- The QC steps use the QPU is an exclusive mode (the QPU currently can’t be shared among users)
- QC results are processed by HPC
- The scheduler has to handle both HPC and QC resources (hybrid scheduling)

❑ Scheduling in not the only required feature

- Running jobs consume QPU time to be **accounted**, and later billed to the right users
- Billing the right user means that is has been fully **authenticated**
- In order to run a QC job, the accesses to the QPU have to handle those two aspects
- Most of the available QC programming framework (usually Python libraries) do not handle them

- Defining a job means converting a QC source code to a QC executable
  - The job is to be compiled from the source before scheduling it
  - It is necessary to define a QC executable format, agnostic to any QPU technology
    - QIR and QASM/OpenQASM are good candidates

- Compiling to QPU is more complicated than compiling to CPU
  - Each technology comes with its contraints
    - Possible entanglements between qubits
    - Available quantum gates
  - How do you express the constraints related to the technology?
    - Width and depth of the circuit: number of available gates and qubits
    - Compilation has to implement on-the-fly transpilation

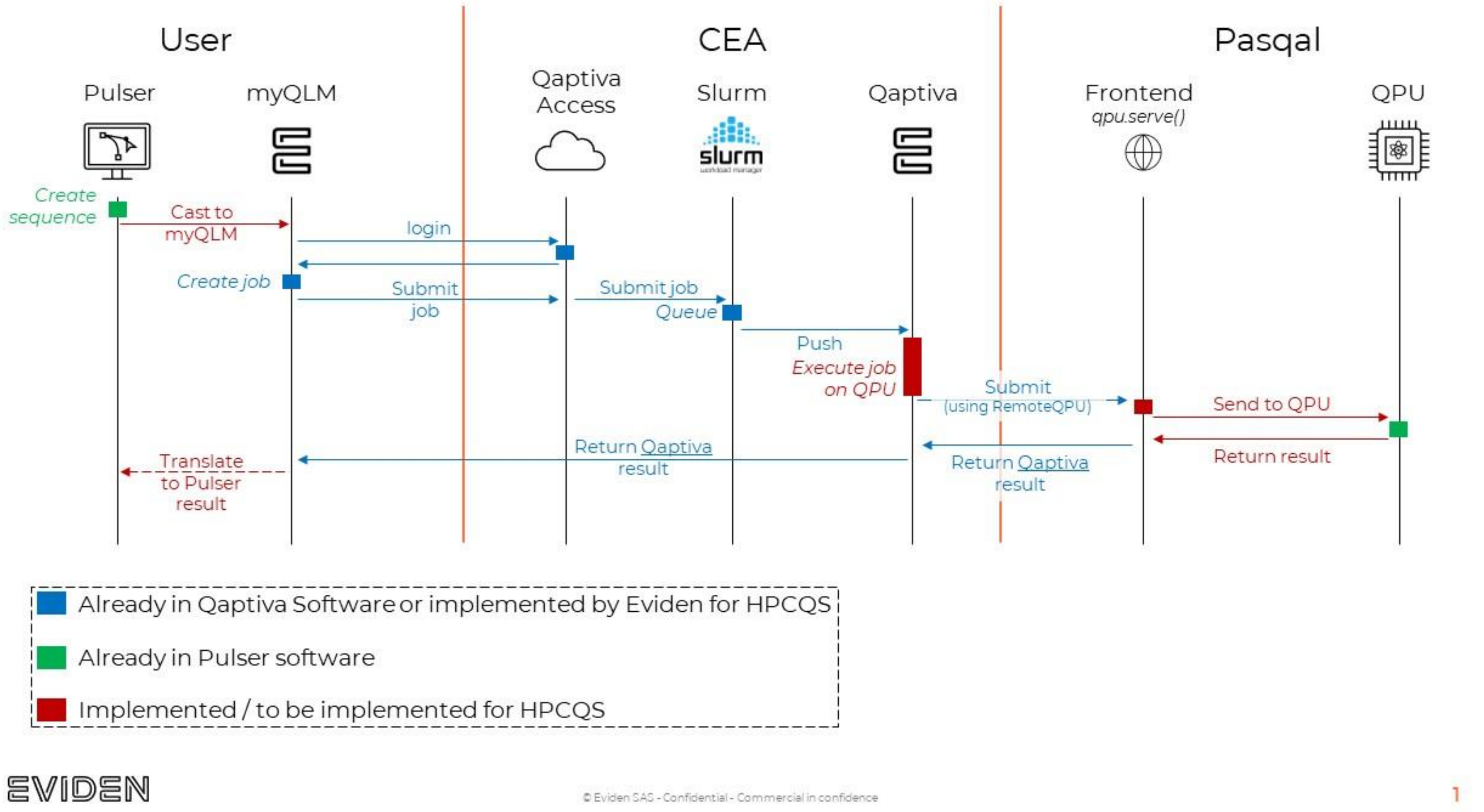
HPC/QS IMPLEMENTATION 1/2

- In the scope of the HPC/QS project, a Pasqal QPU was acquired
  - The QPU is currently under installation at the TGCC facility

- Qaptiva is used as a proxy to the Pasqal system
  - Users connect to Qaptiva, it handles the authentication and accounting features
  - Qaptiva schedule the jobs and send them to the QPU via the QLM/interop protocol
  - QLM/Qaptiva (without the simulation engines) helps building the first bricks of a middleware, providing a generic and efficient “quantum proxy”

- QC program steps are currently written in Python
  - The program is initially encoded using the Pulser framework from Pasqal
  - The Pulser code is wrapped in QLM syntax via the Pulser-myQLM binding module

HPC/QS IMPLEMENTATION 2/2



HPC/QS IMPLEMENTATION : CODE SAMPLE

```python
import numpy as np
from pulser import Pulse, Register, Sequence
from pulser.devices import AnalogDevice
from pulser.waveforms import CustomWaveform
from pulser_simulation import QutipEmulator

from pulser_myqlm import IsingAQPU

# Pulser Sequence (find more at https://pulser.readthedocs.io/)

device = AnalogDevice
register = Register.square(2, 5, None)
seq = Sequence(register, device)
seq.declare_channel("ryd_glob", "rydberg_global")
duration = 100
seq.add(
    Pulse(
        CustomWaveform([ti / duration for ti in range(duration)]),
        CustomWaveform([1 - ti / duration for ti in range(duration)]),
        0,
    ),
    "ryd_glob",
)
seq.add(Pulse.ConstantPulse(20, 1, 0, 0), "ryd_glob")
seq.add(Pulse.ConstantPulse(20, 1, 0, np.pi / 2), "ryd_glob")

# Draw the Sequence
seq.draw(draw_phase_curve=True)

# Simulate the Sequence using Pulser
sim = QutipEmulator.from_sequence(seq, with_modulation=True)
res = sim.run().sample_final_state(2000)
print("Pulser Result obtained with pulser_simulation for 2000 samples:")
print(res, "\n")
print("Converted into MyQLM Result:")
print(IsingAQPU.convert_samples_to_result(res), "\n")
print(
    "Expressed as a dictionary of (state: probability): ",
    {
        sample.state: sample.probability
        for sample in IsingAQPU.convert_samples_to_result(res)
    },
    "\n",
)
```

```python
# Convert the Sequence to a Job
job = IsingAQPU.convert_sequence_to_job(seq, nbshots=0, modulation=True)

# Simulate the Job using pulser_simulation
aqpu = IsingAQPU.from_sequence(seq, qpu=None)
result = aqpu.submit(job)
print(
    "MyQLM Result obtained using IsingAQPU with pulser-simulation with "
    "default number of samples (2000):"
)
print(result, "\n")
print(
    "Expressed as a dictionary of (state: probability): ",
    {sample.state: sample.probability for sample in result},
    "\n",
)

print("Converted into a Pulser Result:")
print(IsingAQPU.convert_result_to_samples(result), "\n")

# Simulate the Job using AnalogQPU
try:
    from qlmaas.qpus import AnalogQPU

    analog_qpu = AnalogQPU()
    aqpu = IsingAQPU.from_sequence(seq, qpu=analog_qpu)
    results = aqpu.submit(job)
    # Display the results once they have run on AnalogQPU
    print("Results obtained with AnalogQPU: ", results.join())
    print(
        "Expressed as a dictionary of (state: probability): ",
        {sample.state: sample.probability for sample in results},
        "\n",
    )
except ImportError:
    print("Can't import AnalogQPU, check connection to Qaptiva Access.")
```

FRANCE HYBRID HPCQUANTUM INITIATIVE

FUTURE CHALLENGES

- ❑ Implement more sophisticated scheduling models
  - The current model is very close to a network spooler
    - Dedicated studies, coupling HPC and QC steps are to be done
    - Using HPC scheduler (such as Slurm) is a promising path
  - What about “embarrassingly quantum” job?
    - Can I run several jobs at the same time, splitting the qubits in several packs?
    - What are the impact (noise, unwanted entanglements) of one job to the others?
  - Future QPU may be able to be shared across multiple users: how should this be handled?

- ❑ Current programs are written in Python
  - C++ codes with explicit quantum subroutines are the next steps
  - This emphasizes the need of standardized QC compilers and QC executable formats

TOWARDS OPENSOURCE STANDARDS

- Full stack approach in HPC has proven its weaknesses
  - Generic and standardized OS and libraries, as Linux provides, was a major improvement in HPC history

- Open-sources middleware should be defined
  - Standardized API and libraries to address well known NPC problem
    - Will help in using QUBO or MIS as building blocks to create complex HPC/QC program
    - Tools to help turning NPC problems to NP problems using HPC
  - Standardized system framework to wrap a QPU from the system point of view
    - Will help in providing proxy to implement authentication / accouting / scheduler on top of QPU

- Compilers are to be standardized
  - Agreeing on an established QC executable format
  - Building a “quantum gcc” capable of handling any kind of QPU with their own constraints
  - Defining what a hybrid C++ HPC/QC syntax should be

THANK YOU

philippe.deniel@cea.fr

BACKUP SLIDES

- QC is currently focusing on "deep tech" aspects
  - How physics can be used to compute
  - Potential advantages of some technologies against other

- QC is not a standalone technology
  - It can be optimized/helped by HPC as well as it optimizes/helps HPC (e.g. QAOA)
  - QC is very good at some tasks (DFT as QFT) and quite bad at others (addition)
  - You do NOT run an Operating System on a QPU and you NEVER will

- QC is fundamentally to be used with HPC
  - QPUs are accelerators, like GPUs
  - QC is an ancillary compute technology