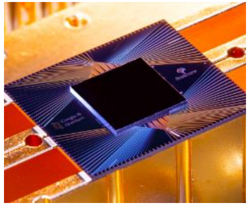
International Conference on Quantum Technologies for High-Energy Physics

Quantum vs classical: assessing quantum advantage

Thomas Ayral
Atos Quantum Laboratory

Atos

Quantum advantages



Google "supremacy"

Sample bitstrings from a ***random*** circuit

200s (quantum) vs 10,000 years (classical) !!

… with total fidelity 0.2%!

1. Simulating large circuits with a finite fidelity

… vs "practical" quantum advantage

No killer app yet!

How to measure practical advantage?

2. An application-centric benchmark: the Q-score

Atos

1. Simulating large circuits with a finite fidelity

Classical simulation of quantum circuits

From a tensor-network perspective

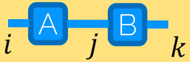
Goal: compute $P_U(x) = |\langle x|\Psi\rangle|^2 = |\langle x|U|0\rangle|^2$

Tensor network primer:

Matrix: 

Vector: 

Tensor: 

Matrix-matrix: 

$= A_{ij}B_{jk}$

Contraction: 

$= C_{ik}$

Classical simulation of quantum circuits

From a tensor-network perspective

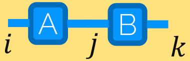
Goal: compute $P_U(x) = |\langle x|\Psi\rangle|^2 = |\langle x|U|0\rangle|^2$

Tensor network primer:

Matrix: 

Vector: 

Tensor: 

Matrix-matrix: 

$= A_{ij}B_{jk}$

Contraction: 

$= C_{ik}$

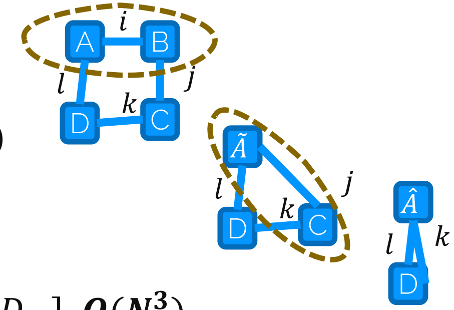
Order matters:

- Naive contraction:

$s = \sum_{ijkl} A_{li}B_{ij}C_{jk}D_{kl}$, $\boldsymbol{O(N^4)}$

- Clever contraction:

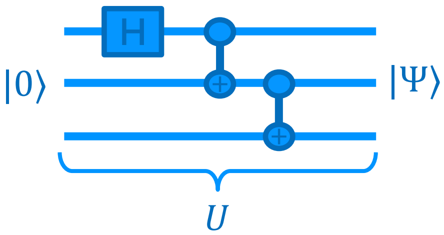
$s = \sum_l [\sum_k \{\sum_j (\sum_i A_{li}B_{ij})\, C_{jk}\}\, D_{kl}]$, $\boldsymbol{O(N^3)}$



Atos

Classical simulation of quantum circuits

From a tensor-network perspective



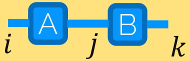
Goal: compute $P_U(x) = |\langle x|\Psi\rangle|^2 = |\langle x|U|0\rangle|^2$

Tensor network primer:

Matrix: 

Vector: 

Tensor: 

Matrix-matrix: 

$= A_{ij}B_{jk}$

Contraction: 

$= C_{ik}$

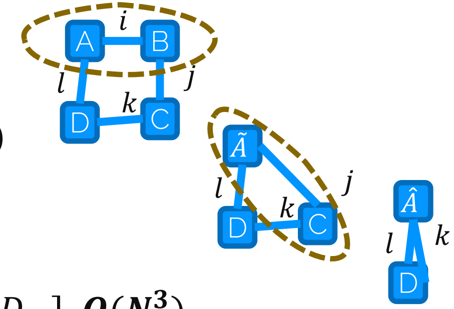
Order matters:

- Naive contraction:

$s = \sum_{ijkl} A_{li}B_{ij}C_{jk}D_{kl}$, $\boldsymbol{O}(\boldsymbol{N}^4)$

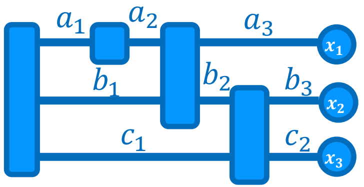
- Clever contraction:

$s = \sum_l[\sum_k\{\sum_j(\sum_i A_{li}B_{ij})\, C_{jk}\}\, D_{kl}]$, $\boldsymbol{O}(\boldsymbol{N}^3)$



Computing $\langle x|U|0\rangle$

- Corresponding tensor network



$x = (x_1, x_2, x_3)$

- Find $\langle x|U|0\rangle$: contract the tensor network.

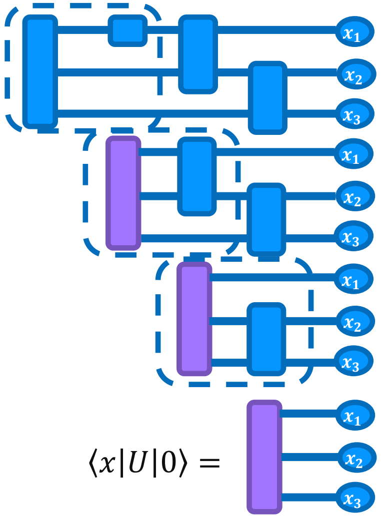
Note: storage cost

- 3 qubits: 2x2x2 = 8
- n qubits: 2^n!

Three main classical simulation methods

… as three contraction strategies!

1. Schrödinger



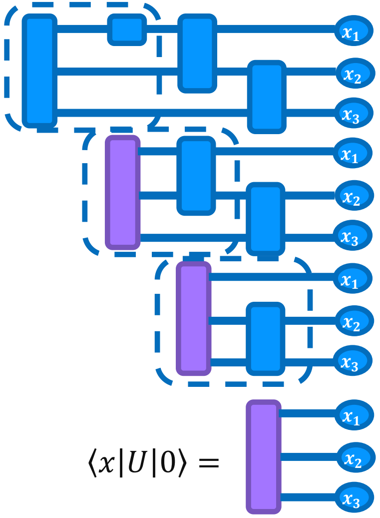
- **CPU cost**: $N_{\text{gates}} \exp(n)$ (width)
- **Storage cost**: $\exp(n)$

All amplitudes at once: "strong"

Three main classical simulation methods

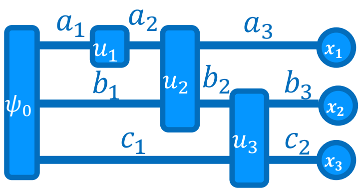
… as three contraction strategies!

1. Schrödinger



$\langle x|U|0\rangle =$

2. Feynman (sum over paths)



$$\langle x|U|0\rangle = \underbrace{\sum_{a_1,a_2,\ldots c_1,c_2}}_{\text{paths}} [\psi_0]_{a_1 b_1 c_1}[u_1]_{a_1 a_2} [u_2]_{a_2 b_1, a_3 b_2}\,[u_3]_{b_2 c_1, b_3 c_2}\, \delta_{a_3 x_1}\delta_{b_3 x_2}\delta_{c_2 x_3}$$

width

- **CPU cost**: $N_{\text{gates}}\exp(n)$
- **Storage cost**: $\exp(n)$

All amplitudes at once: "strong"

depth

- **CPU**: $\propto N_{\text{paths}} \sim \exp(N_{\text{gates}})$
- **Storage**: const.

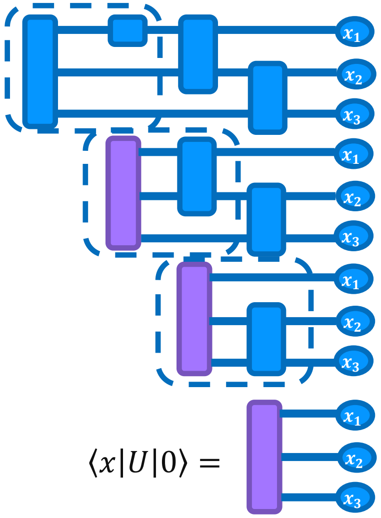
One amplitude: "closed"

Three main classical simulation methods

… as three contraction strategies!

Markov & Shi '08

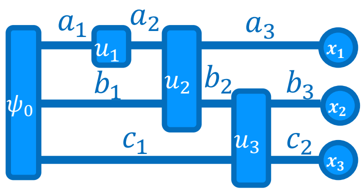
1. Schrödinger



- **CPU cost**: $N_{\mathrm{gates}} \exp(n)$ (width)
- **Storage cost**: $\exp(n)$

All amplitudes at once: "strong"

2. Feynman (sum over paths)

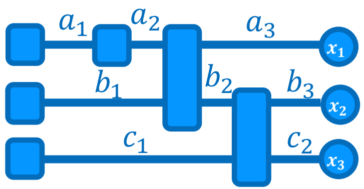


$$\langle x|U|0\rangle = \underbrace{\sum_{a_1,a_2,\dots c_1,c_2}}_{\text{paths}} [\psi_0]_{a_1 b_1 c_1} [u_1]_{a_1 a_2} [u_2]_{a_2 b_1, a_3 b_2} [u_3]_{b_2 c_1, b_3 c_2} \delta_{a_3 x_1} \delta_{b_3 x_2} \delta_{c_2 x_3}$$

- **CPU**: $\propto N_{\mathrm{paths}} \sim \exp(N_{\mathrm{gates}})$ (depth)
- **Storage**: const.

One amplitude: "closed"

3. "Tensor network"



1. Find (close to) optimal contraction strategy (NP hard problem!)
2. Contract (GPUs, TPUs…)

(min(depth, width))

- **CPU** : $\exp(\mathrm{Treewidth})$
- **Storage**: $\exp(\mathrm{Treewidth})$

One amplitude: "closed"

Beating the exponential with a finite fidelity

Matrix Product States (MPS)

See e.g
Schollwöck '11

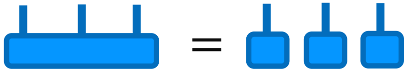
- Previous attempts:

Surrender fidelity by summing fewer Feynman paths.

Idea: use key quantum property: entanglement

- Trivial case: Product states

$$[\psi_0]_{a_1 b_1 c_1} = [\psi_0^1]_{a_1} \, [\psi_0^2]_{b_1} \, [\psi_0^3]_{c_1}$$



Atos

Beating the exponential with a finite fidelity

Matrix Product States (MPS)

See e.g
Schollwöck '11

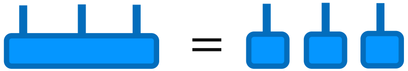
- Previous attempts:

Surrender fidelity by summing fewer Feynman paths.

Idea: use key quantum property: entanglement

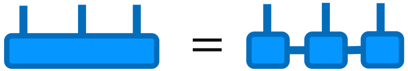
- Trivial case: Product states

$$[\psi_0]_{a_1 b_1 c_1} = [\psi_0^1]_{a_1} [\psi_0^2]_{b_1} [\psi_0^3]_{c_1}$$



- Example: an entangled state:

$$[\psi_0]_{a_1 b_1 c_1} = \frac{1}{\sqrt{2}}[\psi_0^1]_{a_1} [\psi_0^2]_{b_1} [\psi_0^3]_{c_1} + \frac{1}{\sqrt{2}} [\chi_0^1]_{a_1} [\chi_0^2]_{b_1} [\chi_0^3]_{c_1}$$



= "Matrix product state".

Atos

Beating the exponential with a finite fidelity

Matrix Product States (MPS)

See e.g
Schollwöck '11

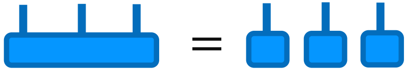
- Previous attempts:

Surrender fidelity by summing fewer Feynman paths.

Idea: use key quantum property: entanglement

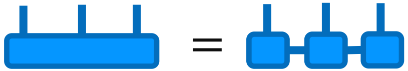
- Trivial case: Product states

$$[\psi_0]_{a_1 b_1 c_1} = [\psi_0^1]_{a_1}\,[\psi_0^2]_{b_1}\,[\psi_0^3]_{c_1}$$



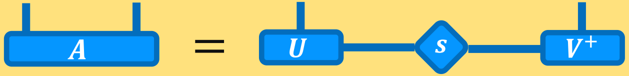
- Example: an entangled state:

$$[\psi_0]_{a_1 b_1 c_1} = \frac{1}{\sqrt{2}}[\psi_0^1]_{a_1}\,[\psi_0^2]_{b_1}\,[\psi_0^3]_{c_1} + \frac{1}{\sqrt{2}}[\chi_0^1]_{a_1}\,[\chi_0^2]_{b_1}\,[\chi_0^3]_{c_1}$$

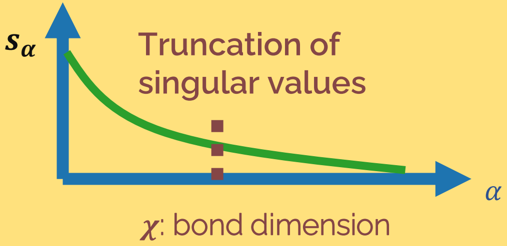


= "Matrix product state".

Compressing any state? Singular value decomposition (SVD)



$$A_{ij} = U_{i\alpha} s_\alpha V^+_{\alpha j}$$



Key result:

$$f = \left|\langle \psi | \psi_{\text{compressed}} \rangle\right|^2 = \sum_{\alpha<\chi} s_\alpha^2$$

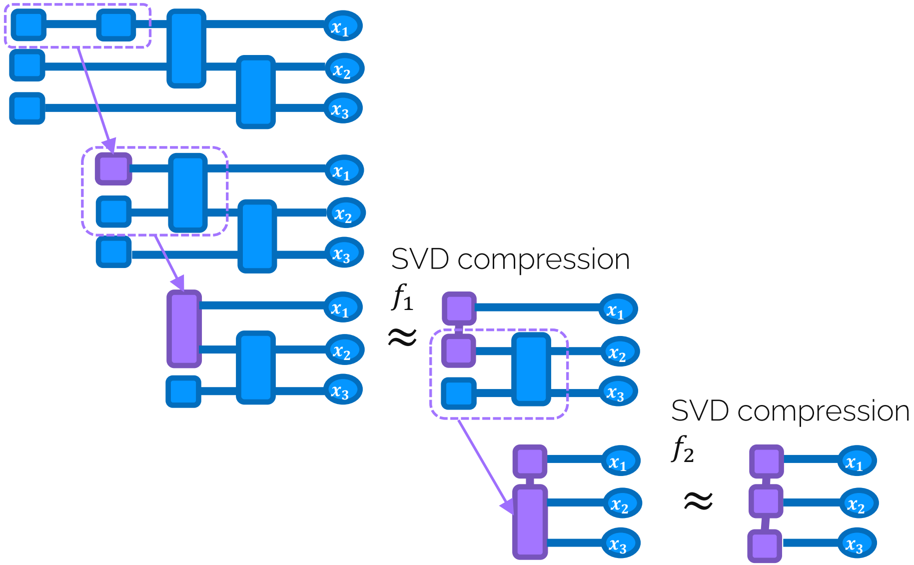
Atos

A first step towards reproducing the experiment

with “grouped” Matrix Product States

Zhou, Stoudenmire, Waintal PRX 20

Algorithm to compute $\langle x|U|0\rangle$: Vidal ‘04



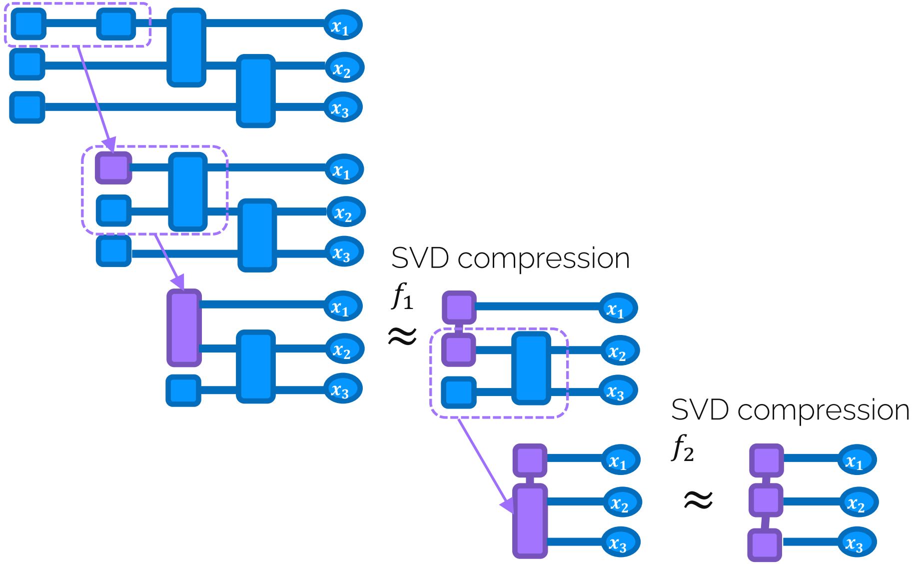
- Final fidelity: $F = f_1 f_2$
- Works… but not enough to reproduce Google

A first step towards reproducing the experiment

with “grouped” Matrix Product States

Zhou, Stoudenmire, Waintal PRX 20

Algorithm to compute $\langle x|U|0\rangle$: Vidal ‘04



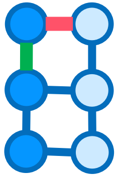
- Final fidelity: $F = f_1 f_2$
- Works… but not enough to reproduce Google

Improvement: Schrödinger + MPS

- “Group tensors together”

Useful for 2D qubit grids:

- Vertical gates are “exact”
- Horizontal gates are “compressed”



Improves fidelity… but still not enough!

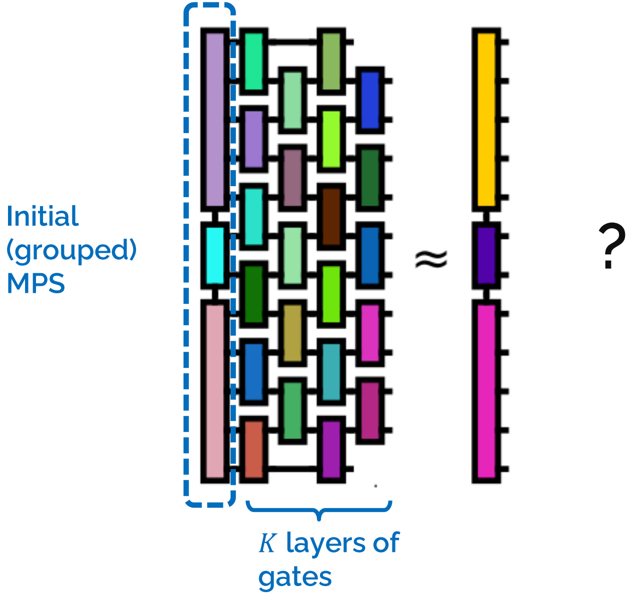
This work: a triply hybrid strategy

MPS + Schrödinger + tensor networks via a Density Matrix Renormalization Group method

Previous approach: apply 1 gate and compress

Here: apply several layers of gates,

… and find "optimal" MPS:



TA, Louvet, Zhou, Lambert, Stoudenmire, Waintal, 2207.05612

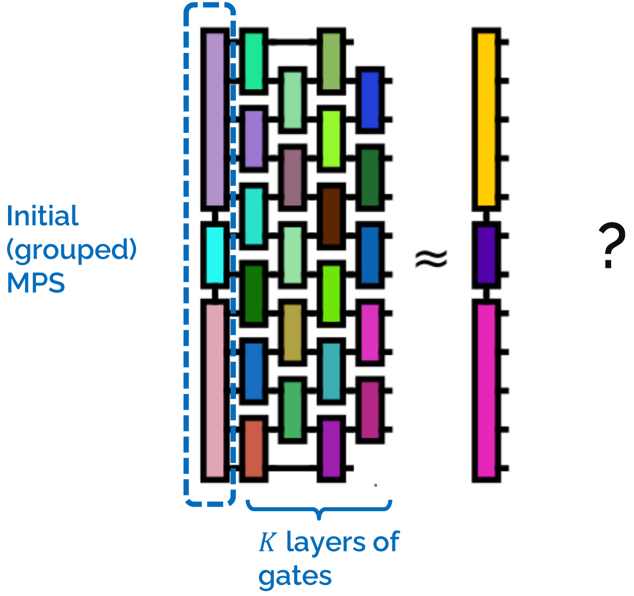
This work: a triply hybrid strategy

MPS + Schrödinger + tensor networks via a Density Matrix Renormalization Group method

Previous approach: apply 1 gate and compress

Here: apply several layers of gates,

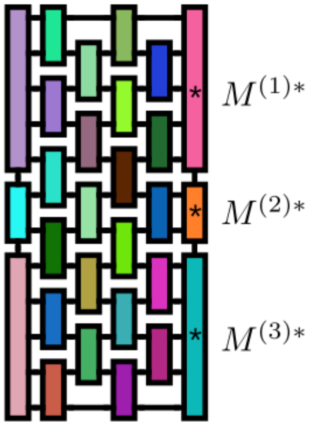
… and find "optimal" MPS:



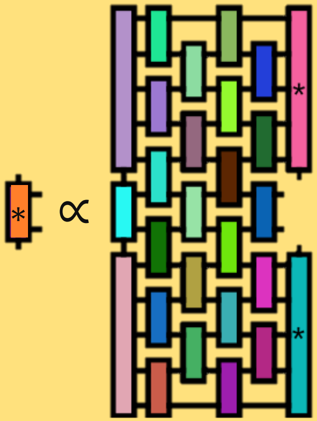
How to find optimal MPS?

- DMRG: find MPS with maximal overlap
- Tensor-by-tensor optimization: n_s "sweeps"

Overlap:



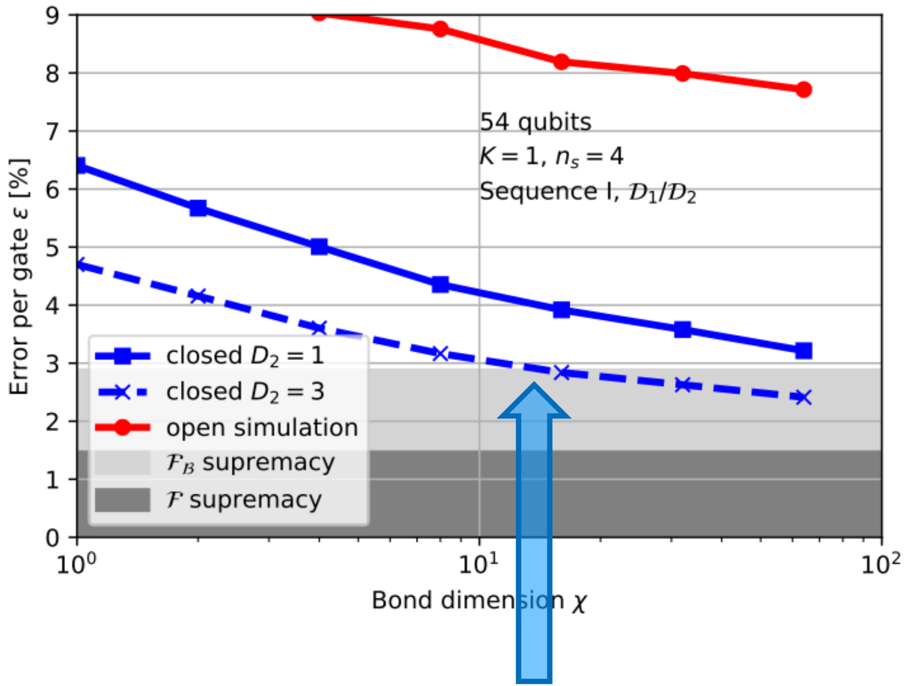
Best $M^{(2)}$ tensor:



A tensor network!

Atos

Closing the supremacy gap



Better XEB error rate than Google Sycamore

2. An application-centric benchmark: The Q-score

Atos

Relevant criteria for a benchmarking protocol
A HPC-driven wish list

- **Must quantify the “usefulness” of a processor**

Solution of a concrete problem

- hard…
- … for the best classical algorithm

Relevant criteria for a benchmarking protocol
A HPC-driven wish list

- **Must quantify the "usefulness" of a processor**

- **Must be scalable**

Solution of a concrete problem

- hard…
- … for the best classical algorithm

Computable for systems of 100, 1000, etc "qubits"

Relevant criteria for a benchmarking protocol
A HPC-driven wish list

- **Must quantify the “usefulness” of a processor**

Solution of a concrete problem

- hard…
- … for the best classical algorithm

- **Must be scalable**

Computable for systems of 100, 1000, etc “qubits”

- **Must not be platform-specific**

Do not unduly favor

- A platform
- A computation paradigm (gate-based, analog…)
- etc.

Atos

Existing characterization protocols

Gate-level

- Gate error rates
- Readout error rates
- Coherence times

Gate tomography

~~Usefulness~~

~~Scalability~~

~~Agnosticity~~

Existing characterization protocols

Gate-level

- Gate error rates
- Readout error rates
- Coherence times

Gate tomography

Circuit-level

Ability to generate "nonclassical" distributions:

- Google: **cross-entropy benchmarking**
- IBM: **quantum volume**

(random circuits)

Existing characterization protocols

Gate-level

- Gate error rates
- Readout error rates
- Coherence times

Gate tomography

Circuit-level

Ability to generate "nonclassical" distributions:

- Google: **cross-entropy benchmarking**
- IBM: **quantum volume**

(random circuits)

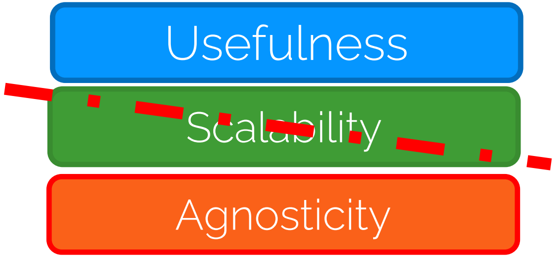
Application-level

Solving linear systems

Dong & Lin (2020)

Computing the GS energy of the 1D Hubbard model

Dallaire-Demers et al (2020)

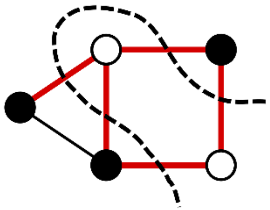


Our proposal: the Q-score protocol

Martiel, TA, Allouche (Transactions in Quantum Engineering, 2102.12973)

Problem to be solved : MaxCut

Find the set of vertices that maximizes the number of outgoing edges



- **Hard to approximate**

(and used in many application domains)

- **Quantum formulation:**

Ising Hamiltonian:

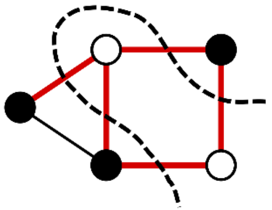
$$H = \sum_{i,j \in E} \sigma_z^{(i)} \sigma_z^{(j)} + const.$$

Our proposal: the Q-score protocol

Martiel, TA, Allouche (Transactions in Quantum Engineering, 2102.12973)

Problem to be solved : MaxCut

Find the set of vertices that maximizes the number of outgoing edges



- **Hard to approximate**

(and used in many application domains)

- **Quantum formulation:**

Ising Hamiltonian:

$$H = \sum_{i,j \in E} \sigma_z^{(i)} \sigma_z^{(j)} + const.$$

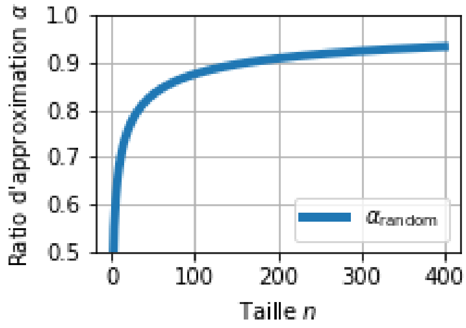
Reference point: classical state-of-the-art

- *Average* optimal cost:

$$C(S_0) = \frac{n^2}{8} + 0.18 n^{3/2}$$

- Random algorithm:

$$C_{\text{random}}(S) = \frac{n^2}{8}$$

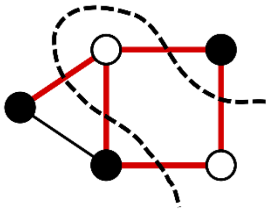


Our proposal: the Q-score protocol

Martiel, TA, Allouche
(Transactions in Quantum Engineering, 2102.12973)

Problem to be solved : MaxCut

Find the set of vertices that maximizes the number of outgoing edges



- **Hard to approximate**

(and used in many application domains)

- **Quantum formulation:**

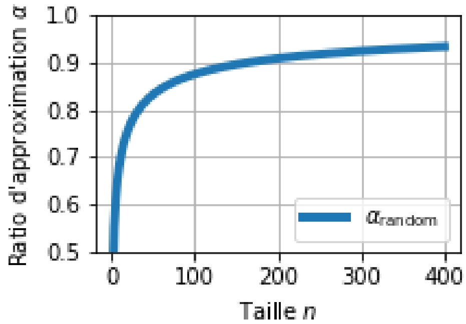
Ising Hamiltonian:

$$H = \sum_{i,j \in E} \sigma_z^{(i)} \sigma_z^{(j)} + const.$$

Reference point: classical state-of-the-art

- *Average* optimal cost:

$$C(S_0) = \frac{n^2}{8} + 0.18 n^{3/2}$$



- Random algorithm:

$$C_{\text{random}}(S) = \frac{n^2}{8}$$

- “Above random” approximation ratio:

$$\beta(S) = \frac{C(S) - n^2/8}{0.18 n^{3/2}}$$

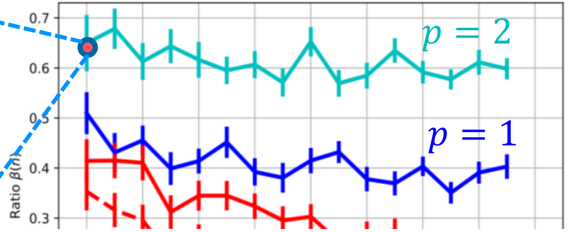
  - $\beta_{\text{random}}(S) = 0$
  - $\beta_{\text{optimal}}(S) = 1$

The Q-score protocol in practice

Martiel, TA, Allouche (TQE, 2102.12973)

a. For a size-n graph G:
   i. Execute an algorithm to find a solution S
   ii. Compute cost $C_G(S)$

b. Average costs: $C_n = \langle C_G(S) \rangle_G$ and compute $\beta(n)$



Graph size $\boldsymbol{n}$

Without decoherence

The longer the preparation circuit ($p = 1 \rightarrow 2$), the higher the quality (constant w.r.t size)

The Q-score protocol in practice

Martiel, TA, Allouche (TQE, 2102.12973)

a. For a size-n graph G:
   i. Execute an algorithm to find a solution S
   ii. Compute cost $C_G(S)$
b. Average costs: $C_n = \langle C_G(S) \rangle_G$ and compute $\beta(n)$

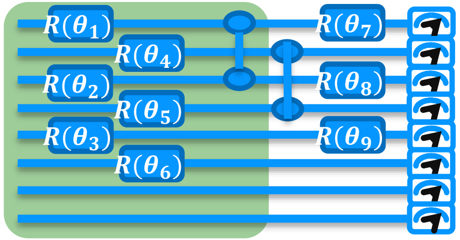
Quantum algorithm:

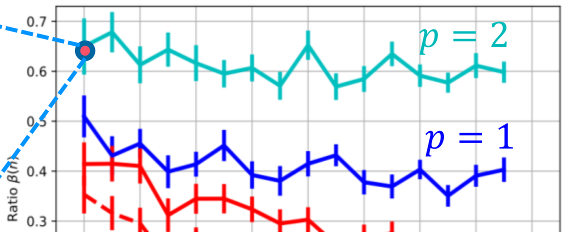
User's choice (gate-based, analog).

Here, variational algorithm ("QAOA")

$|\psi\rangle = U_{\vec{\theta}^*}|0\rangle$ with $\vec{\theta}^*$ minimizing $\langle \psi_{\vec{\theta}} | H | \psi_{\vec{\theta}} \rangle$

Preparation of $U_{\vec{\theta}}|0\rangle$





Graph size n

Without decoherence

The longer the preparation circuit ($p = 1 \rightarrow 2$), the higher the quality (constant w.r.t size)

The Q-score protocol in practice

Martiel, TA, Allouche (TQE, 2102.12973)

a. For a size-n graph G:
   i. Execute an algorithm to find a solution S
   ii. Compute cost $C_G(S)$
b. Average costs: $C_n = \langle C_G(S) \rangle_G$ and compute $\beta(n)$

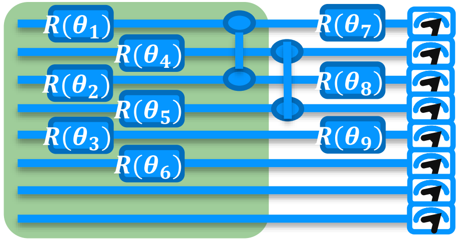
Quantum algorithm:

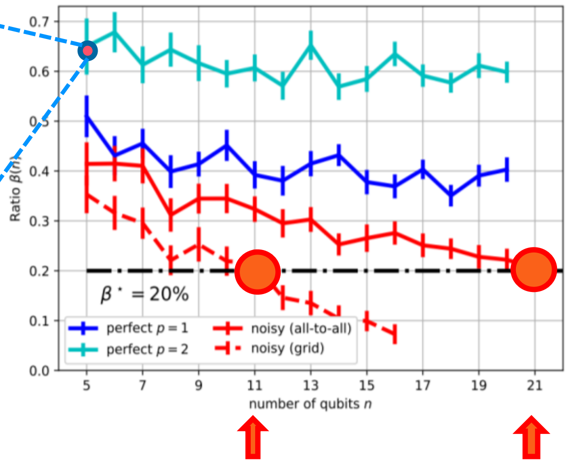
User's choice (gate-based, analog).

Here, variational algorithm ("QAOA")

$|\psi\rangle = U_{\vec{\theta}^*}|0\rangle$ with $\vec{\theta}^*$ minimizing $\langle\psi_{\vec{\theta}}|H|\psi_{\vec{\theta}}\rangle$

Preparation of $U_{\vec{\theta}}|0\rangle$





Without decoherence

The longer the preparation circuit ($p = 1 \rightarrow 2$), the higher the quality (constant w.r.t size)

With (simulated) decoherence

(here, depolarizing noise)

- Quality decreases with size
- Qubit connectivity plays a role **(compilation)**

The Q-score protocol in practice

Martiel, TA, Allouche (TQE, 2102.12973)

a. For a size-n graph G:
   i. Execute an algorithm to find a solution S
   ii. Compute cost $C_G(S)$
b. Average costs: $C_n = \langle C_G(S) \rangle_G$ and compute $\beta(n)$

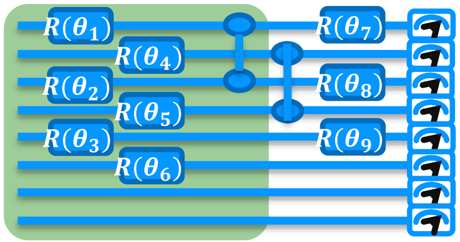
Quantum algorithm:

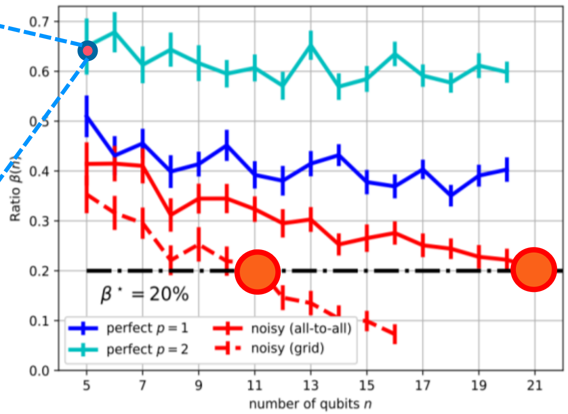
User's choice (gate-based, analog).

Here, variational algorithm ("QAOA")

$|\psi\rangle = U_{\vec{\theta}^*}|0\rangle$ with $\vec{\theta}^*$ minimizing $\langle \psi_{\vec{\theta}}|H|\psi_{\vec{\theta}}\rangle$

Preparation of $U_{\vec{\theta}}|0\rangle$





Q-score value

Without decoherence

The longer the preparation circuit ($p = 1 \rightarrow 2$), the higher the quality (constant w.r.t size)

With (simulated) decoherence

(here, depolarizing noise)

- Quality decreases with size
- Qubit connectivity plays a role (**compilation**)

Q-score : number of "useful" qubits to solve a difficult problem

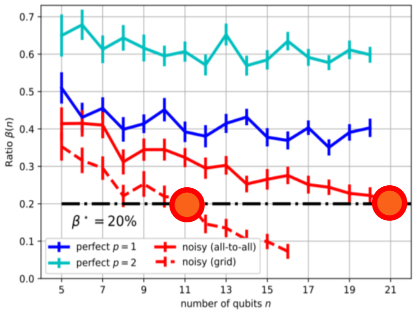
Usefulness | Scalability | Agnosticity

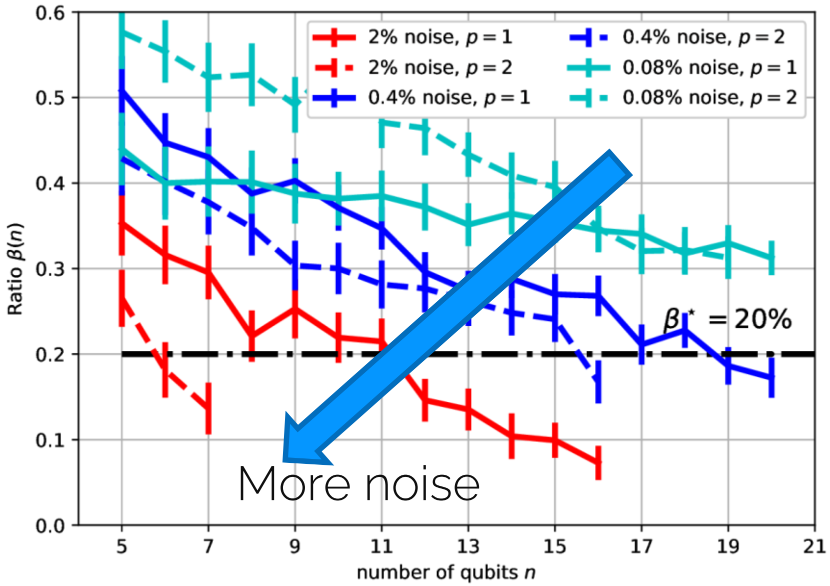
Atos

Maximizing the Q-score for NISQ

Simon Martiel, TA, Cyril Allouche (arxiv 2102.12973, Transactions in Quantum Engineering)

- QAOA example: in principle, more layers: better results. But…





Conclusions

Part I: finite fidelity classical simulation

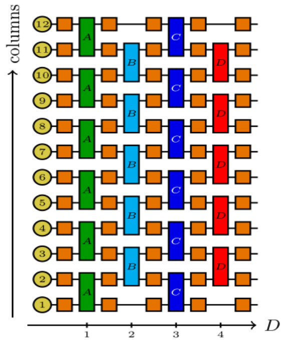
- Key to quantum advantage: increased fidelity!
- Combination of methods can beat finite-fidelity processor
- Scalable!
- Available as a QLM simulator: qat-qpeg

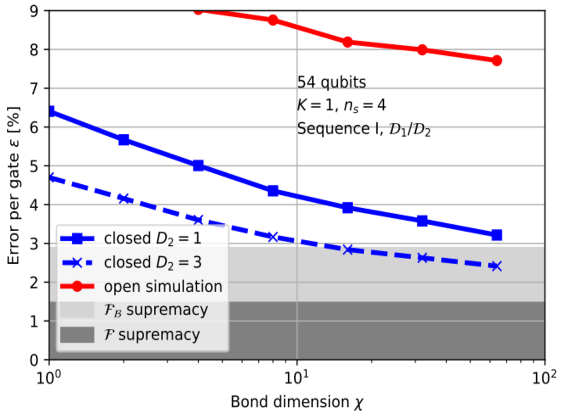
Part II: Q-score

- Application-centric
  - Can change application: why not many-body HEP problem? (qat-fermion lib on QLM)
- Hardware-agnostic
- Scalable
- Recently applied to annealing (D-wave, Rydberg atoms)

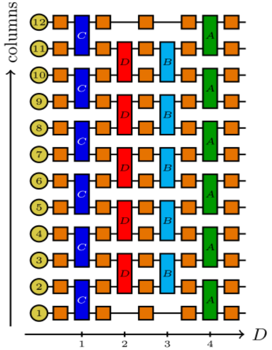
The influence of the type of circuit

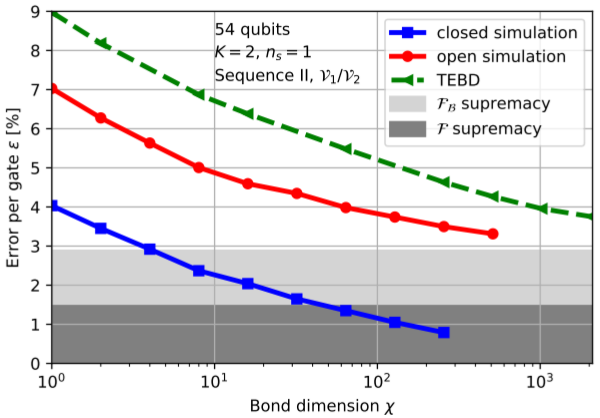
The supremacy sequency



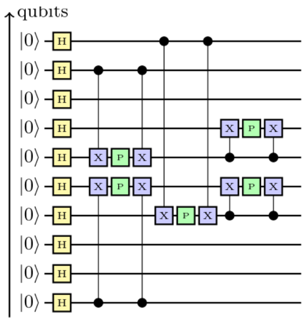


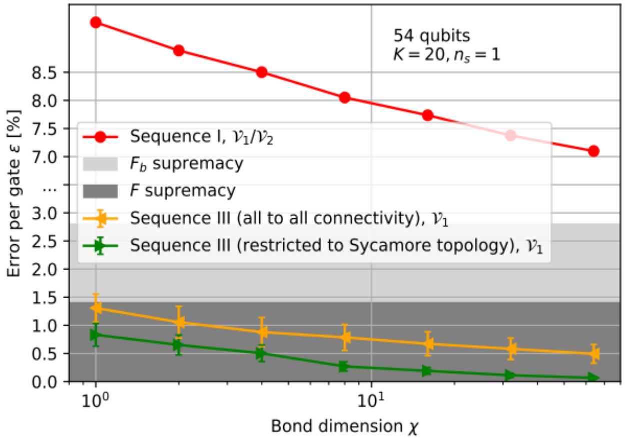
An easier sequence





A "useful" sequence

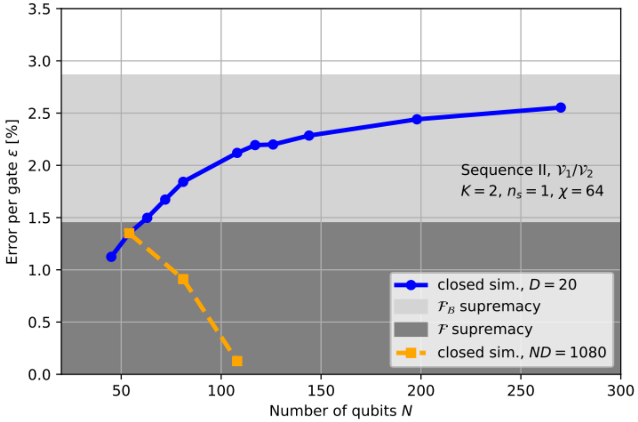




A scalable method: what happens when increasing the qubit count?

Fixed depth D:

- Error per gate increases… then stagnates:



But more gates, XEB decreases…: must increase N_{samples} to reduce variance.

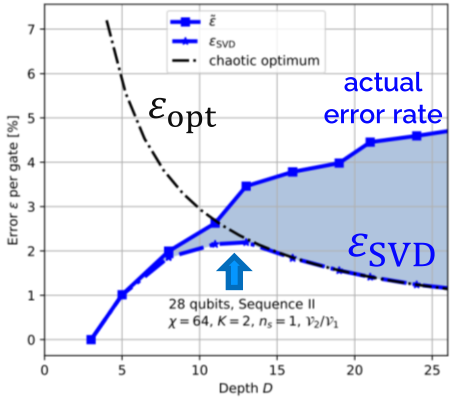
Keep nD fixed (fixed XEB: fixed experimental time!):

- better and better error per gate!

How to understand the stagnation?

Can compute "optimal" error rate after SVD compression:

$$\varepsilon_{\mathrm{opt}} = \frac{1}{D}\left(\log 2 - \frac{\log 4\chi}{2N}\right)$$



when $\varepsilon_{\mathrm{SVD}}$ reaches $\varepsilon_{\mathrm{opt}}$ [~chaotic limit], actual error rate deviates from $\varepsilon_{\mathrm{SVD}}$

Atos

Try Q-score yourself

- Github repo: https://github.com/myQLM/qscore

```python
from qat.qscore.benchmark import QScore
from qat.plugins import ScipyMinimizePlugin
from qat.qpus import get_default_qpu

# Our QPU is composed of:
# - a variational optimizer plugin
# - a QLM/myQLM default qpu (either LinAlg or
    pyLinalg)

QPU = ScipyMinimizePlugin(
    method="COBYLA",
    tol=1e-4,
    options={"maxiter": 300}
) | get_default_qpu()

benchmark = QScore(
    QPU,
    size_limit=20,  # limiting the instace sizes
    to 20
    depth=1,        # using an Ansatz depth of 1
    output="perfect.csv",
    rawdata="perfect.raw"
)
benchmark.run()
```

► ... with you own QPU (simulated/actual hardware):

```python
from qat.core.qpu import QPUHandler
from qat.core import Result


class MyQPU(QPUHandler):
    def submit_job(self, job):
        # Evaluate the job using your QPU
        # A job constains:
        # a circuit:
        circuit = job.circuit
        # possibly an observable
        observable = job.observable
        # or a list of qubits to sample:
        qubits = job.qubits

        # Results are returned in a 'Result'
    object
        return result
```

► ... with myQLM-compatible hardware (myqlm-interop)

```python
from qat.interop.qiskit import BackendToQPU
# we can select a backend from available IBMQ backends
MY_IBM_TOKEN = "..."
qpu = BackendToQPU(token=MY_IBM_TOKEN,
                   ibmq_backend="ibmq_armonk")
```