TERATEC SEMINAR, NOV 13, 2024

Bridging the gap between High Performance (scientific) Computing and Quantum Computing

Marc Baboulin, Inria team QuaCS
Collaboration with EVIDEN

3 reasons for **HPC not to be confident** with **Quantum**

01

Subject to errors, probabilistic results

02

Too many technologies

03

No standard in programming

3 reasons for **HPC to be enthousiast with Quantum**

01

Speedup

Some theoretical advantage for VQE, HHL, QSVT...

02

Hybridization

Heterogeneity is now commonly addressed in HPC.

03

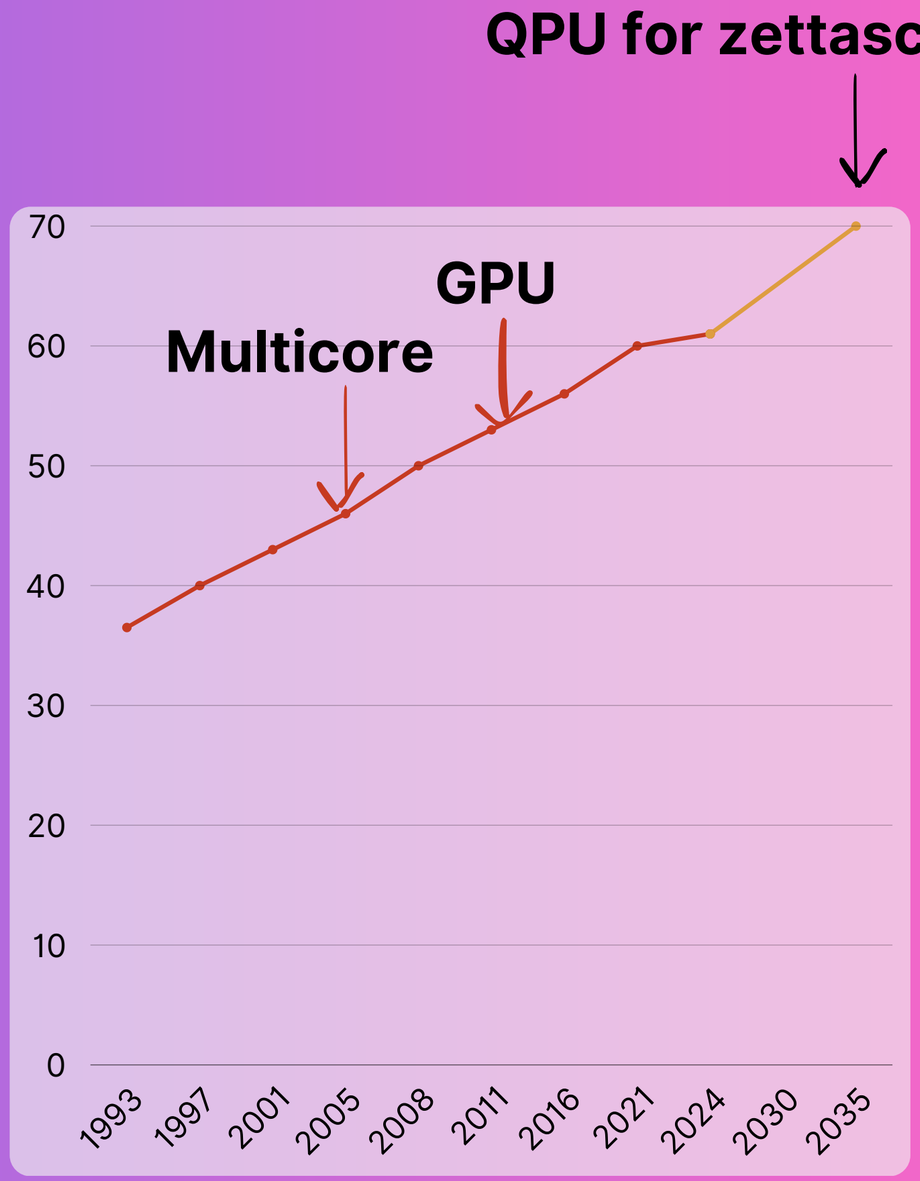
Energy efficiency

23MW for the exascale machine vs 10^4 times less for Google Sycamore.

A quantum view of the Top500

Performance development over time “converted” to error free qubits :)

On the road to zettascale, Quantum can be a **game changer**



LINPACK benchmark (Ax=b)

Some research interests in QC/HPC

HPC for QC

- Quantum circuit synthesis
- Matrix decomposition
- Quantum simulation

QC for HPC

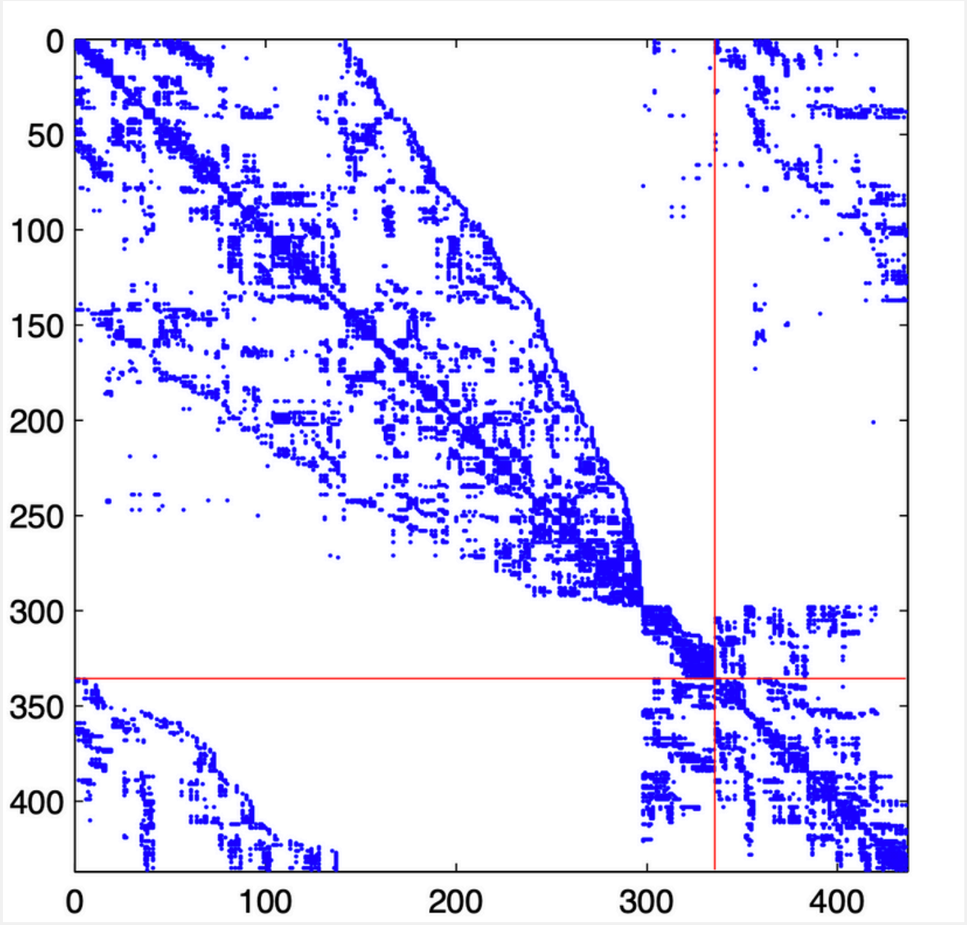
- Linear algebra
- Partial Differential Equations
- AI/Clustering

Outline

What do we need as bricks for quantum scientific computing ?

- Matrix decompositions
- Encoding matrices into unitaries
- Exploiting structures (sparsity, PDEs...)
- Quantum Singular Value Transformation
- Linear system solution
- Iterative refinement

Scientific computing plays with matrices



Matrix decomposition

We decompose a matrix in an appropriate basis in order to encode this matrix in a quantum memory.

Pauli matrices

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Pauli operator basis

$$\mathcal{P}_n = \left\{ \bigotimes_n M_n, \;\; M_n \in \{I, X, Y, Z\} \right\}$$

Decomposition in the Pauli basis

$$\text{For } A \in \mathbb{C}^{2^n \times 2^n}, \; A = \sum_{P_i \in \mathcal{P}_n} \alpha_i P_i, \; \text{with } \alpha_i \in \mathbb{C}$$

If A is Hermitian, the α_i's are real numbers

At most 4^ncoefficients

Pauli decomposition

Straightforward method

$$\text{If } A \in \mathbb{C}^{2^n \times 2^n},\ \alpha_{M_1 M_2 \ldots M_n} = \frac{1}{2^n} Tr\left(\left(\bigotimes_{i=1}^{n} M_i\right) A\right), \text{where } M_i \in \{I, X, Y, Z\}.$$

For each coefficient we have $n-1$ tensor products $(\mathcal{O}(2^n))$ and the trace of the product $(\mathcal{O}(2^n))$ and exploiting the specific structures of Pauli matrices $\longrightarrow \mathcal{O}(8^n)$ flops.

Existing implementations

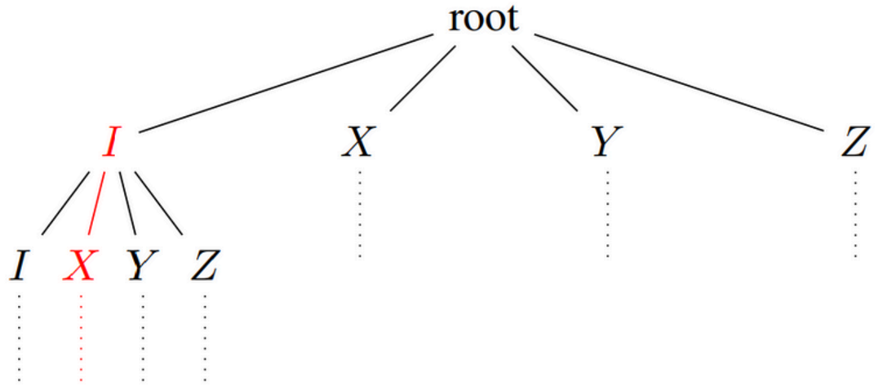
[Pesce, Stevenson. Pauli spin matrix decomposition of real symmetric matrices, 2021]

[Romero, Santos-Suarez. Compute tensor products of Pauli matrices efficiently, 2023]

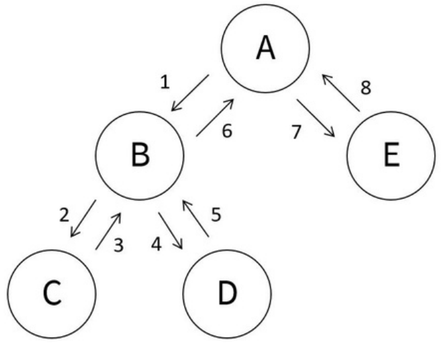
[Hantzko, Binkowski, Gupta. Tensorized Pauli decomposition algorithm, 2024]

All of them are serial and in python.

Fast Pauli decomposition [Koska, MB, Gazda, ISC 2024]

- Exploit the similarity of information (structure, values) from one Pauli operator to another to reduce the number of elementary operations.
- 

Pauli tree



In-order tree exploration

- We update 2 vectors of $\mathbb{C}^{2^n}$, one for column indices and one for the nonzero values.
- Time complexity: $\frac{9}{7} + \frac{5}{7}8^n$, memory: $\mathcal{O}(2^n)$.

Fast Pauli decomposition

Exploiting matrix structure

Order of magnitude for cost of Pauli decomposition:

	#coeff	flops
general	4^n	8^n
diagonal	2^n	4^n
tridiagonal	$n2^n$	$n4^n$
band-diagonal*	$(sn - c(s))2^n$	$sn4^n$

$*\text{Bandwidth } = 2s + 1$

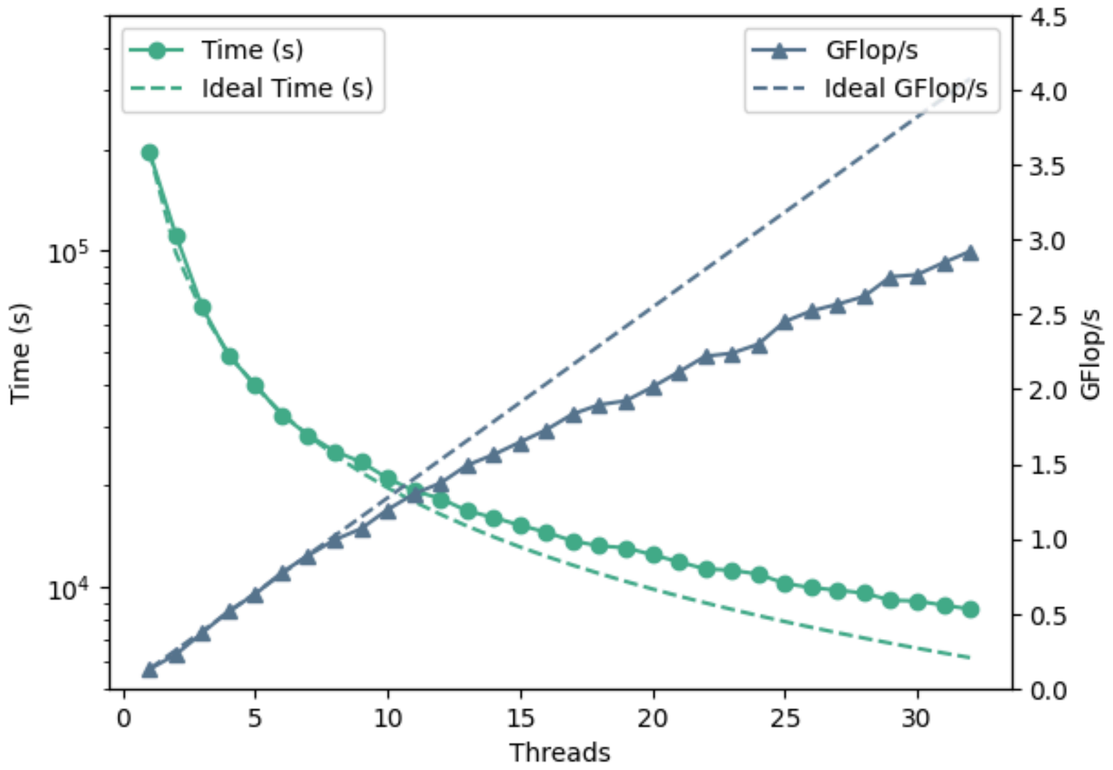
Fast Pauli decomposition

Combinations of Pauli decompositions

Let $A = \sum_j \alpha_j P_j,$ and $B = \sum_j \beta_j P_j$

- ✓ Direct sum: $A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} = I \otimes \sum_j \frac{\alpha_j + \beta_j}{2} P_j + Z \otimes \sum_j \frac{\alpha_j - \beta_j}{2} Pj.$
- ✓ Block-diagonal: $\mathcal{A}_N = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_N \end{bmatrix} = \mathcal{A}_N = \Big((A_1 \oplus A_2) \oplus (A_3 \oplus A_4)\Big) \oplus \Big((A_5 \oplus A_6) \oplus (A_7 \oplus A_8)\Big)\ (N = 8)$
- ✓ Linear combination: $\mu A + B = \sum_j (\mu\alpha_j + \beta_j) P_j.$
- ✓ Matrix multiplication: $A \times B = \sum_{j,k} \alpha_j \beta_q (P_j \times P_q)$
- ✓ Hermitian matrix augmentation: $\begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix} = X \otimes \sum_{P_j \in \mathcal{P}_n} a_j P_j + Y \otimes \sum_{P_j \in \mathcal{P}_n} b_j P_j,\ \text{with}\ \alpha_j = a_j + i b_j$

Parallel performance



Strong scaling (15 qubits)

- **Multi-threaded code:** The Pauli tree is split into forests of subtrees and each thread handles an independent part of the tree (no communication).
  No errors due to the parallelization.
- We don't need to store the input matrix (can be guessed via a function for instance).
- Memory footprint: 2^n

Matrix size: 32768 (complex), Pauli trees in forest: 256.

Encoding matrices in quantum computers

Quantum computers only handle unitary matrices

Block-encoding (BE)

We want to encode $A \in \mathbb{C}^{2^n} \times \mathbb{C}^{2^n}$ in a unitary

$$U_A = \begin{bmatrix} A & . \\ . & . \end{bmatrix}.$$

Suppose such $U_A \in \mathbb{C}^{2^{n+1}} \times \mathbb{C}^{2^{n+1}}$ exits (one ancilla)

then $U_A|0\rangle|x\rangle = \begin{bmatrix} Ax \\ . \end{bmatrix} = |0\rangle A|x\rangle + |1\rangle|\psi\rangle$ and

$\frac{Ax}{\|Ax\|}$ can be obtained upon measurement of qubit 0.

Can be extended to m ancilla qubits with the relation

$$A = (\langle 0^m| \otimes I_N) U_A (|0^m\rangle \otimes I_N),$$

and measurement of the m ancillas

This requires that $\|A\|_2 \leq 1,$ or need for scaling.

Matrix block-encoding

Approximate BE

An (α, m, ϵ) block-encoding of A is defined by

$$\|A - \alpha(\langle 0^m| \otimes I_N) U_A (|0^m\rangle \otimes I_N)\|_2 \leq \epsilon \text{ with } \alpha, \epsilon \in \mathbb{R}_+.$$

Examples

- For $A = W\Sigma V^\dagger$ with $\|A\|_2 \leq 1$ then we have

$$U_A = \begin{bmatrix} A & W\sqrt{I_N - \Sigma^2} \\ \sqrt{I_N - \Sigma^2}V^\dagger & -\Sigma \end{bmatrix}.$$

- From random circuit U_A (Haar distribution) then $A = (\langle 0| \otimes I) U_A (|0\rangle \otimes I)$ can be seen as the equivalent of a dense random matrix.

See [Dong and Lin, 2021] for an application to linear system benchmarking similar to LINPACK.

Block encoding via Pauli decomposition

- Consider $A \in \mathbb{C}^{2^n \times 2^n}$ Hermitian with $M = 2^m$ Pauli operators: $A = \sum_{i=0}^{M-1} \alpha_i V_i,\ \alpha_i \in \mathbb{R}$

  (if A not hermitian we can use the augmented matrix $\tilde{A} = \begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix}$).

- To apply the matrix A to an n-qubit quantum state $|\psi\rangle_d$ we use the LCU method:

1. Allocate m ancilla qubits and prepare state $|\alpha\rangle_a = \frac{1}{\sqrt{\sum_i |\alpha_i|}} \sum_i \sqrt{|\alpha_i|} |i\rangle_a$

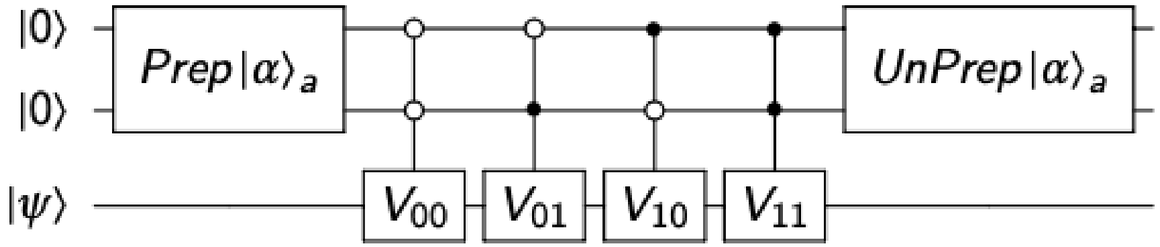
2. Apply V_i to the data state $|\psi\rangle_d$, controlled by the ancilla qubits in state $|i\rangle_a$

$$|i\rangle_a |\psi\rangle_d \to |i\rangle_a V_i |\psi\rangle_d$$

3. Unprepare step 1.

BE circuit from Pauli factors

Circuit



Circuit for m=2

Resulting state

$$\frac{1}{\|\alpha\|_1}|00\rangle A|\psi\rangle + \cdots = U_A(|00\rangle|\psi\rangle)$$

$$\text{with } U_A = \frac{1}{\|\alpha\|_1}\begin{bmatrix} A & \cdot \\ \cdot & \cdot \end{bmatrix}.$$

Comments on complexity

- NISQ: BE scales logarithmically with matrix dimension but circuit depth implies low fidelity due to successive errors.
- LSQ: complexity in T gates.

$$\text{T-count: } \mathcal{O}(2^m(nm + polylog(1/\epsilon)))$$

$$\text{Time complexity } \propto \text{ T-count}$$

- If the number of ancillas is small, the main cost is the Pauli decomposition (exponential in n), which requires HPC resources.

Block-encoding for PDE matrices

We can exploit some specific structures of PDE matrices

[Ty, Vilmart et al., 2024]

Example: Poisson equation

$$\text{Solving } u''(x) = f,\ \forall x \in [0,1]$$

$$\text{Boundary conditions} : u(0) = u(1) = 0$$

Finite difference method

$$-\frac{1}{h^2}\underbrace{\begin{pmatrix} 2 & -1 & & 0 \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{pmatrix}}_{\mathbf{L}\in\mathcal{M}_{N\times N}} \underbrace{\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix}}_{|u\rangle} = \underbrace{\begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}}_{|f\rangle}$$

Obtaining the circuit for FD matrix

The tridiagonal matrix is decomposed into a sum of **2 block diagonal matrices** (potentially shifted) such that each is efficiently implementable using quantum operators. Can be generalized to band matrices.

Splitting

$$\mathbf{L} = \underbrace{\begin{pmatrix} 1 & -1 & & & & & \\ -1 & 1 & & & & & \\ & & \ddots & & & & \\ & & & 1 & -1 & & \\ & & & -1 & 1 & & \\ & & & & & 1 & -1 \\ & & & & & -1 & 1 \end{pmatrix}}_{\mathbf{L_0}} + \underbrace{\begin{pmatrix} 1 & & & & & & 0 \\ & 1 & -1 & & & & \\ & -1 & 1 & & & & \\ & & & \ddots & & & \\ & & & & 1 & -1 & \\ & & & & -1 & 1 & \\ 0 & & & & & & 1 \end{pmatrix}}_{\mathbf{L_1}}$$

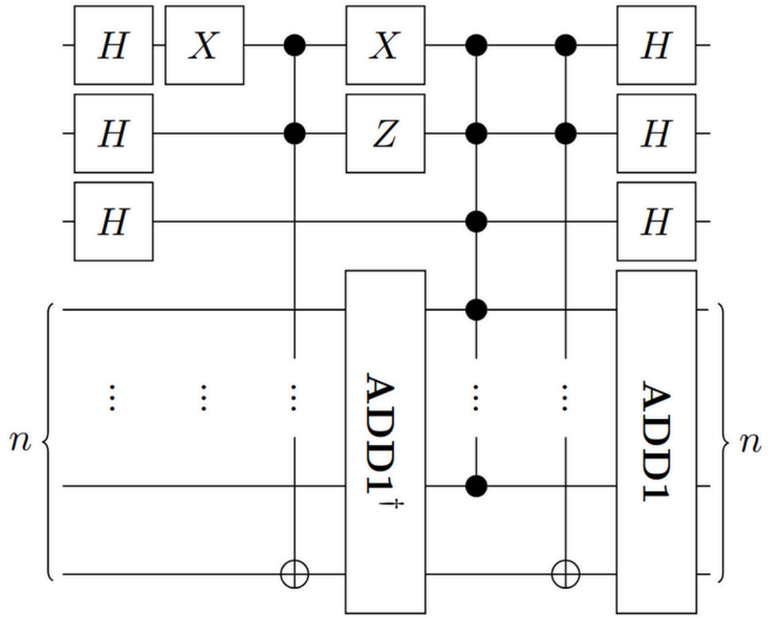
``Block diagonalization”

$$\mathbf{L} = L_0 + L_1 = B_0 + P_1 B_1 P_1^{-1}$$

with $B_0 = I^{\otimes(n-1)} \otimes (I - X)$ and $B_1 = I^{\otimes n} + \frac{1}{2} P_1((I^{\otimes(n-1)} + C^{n-2}Z) \otimes X) P_1^{-1}$, P_1 permutation.

Obtaining the circuit for FD matrix

Circuit



Complexity

- ✓ $\#\text{qubits} : n + 3 \text{ with } n = \log_2(N)$
- ✓ $\text{depth} : \mathcal{O}(\log_2(n))$

Block-encoding of sparse matrices

[Camps et al., 2023]

Example

$$A = \begin{pmatrix} 0 & b & 0 & 0 \\ a & 0 & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{pmatrix}$$

Classical CSC format:

values: a, b, c, d

row indices: $1, 0, 2, 3$

col. pointers: $0, 1, 2, 3$

Encoding nonzeros

$$O_A : (i, j) \mapsto a_{ij}$$

$$\{(1,0) \to \mathbf{a}; (0,1) \to \mathbf{b}; (2,2) \to \mathbf{c}; (3,3) \to \mathbf{d}\}$$

Selecting nonzeros

Selecting (i, j) such that $a_{ij} \neq 0$:

$c : (j, l) \mapsto i,$ where i is the row index that corresponds

to the l-th nonzero entry in the j-th column

$$\{(0,0) \to \mathbf{1}; (1,0) \to \mathbf{0}; (2,0) \to \mathbf{2}; (3,0) \to \mathbf{3}\}$$

Block-encoding for sparse matrices

Oracles for querying matrix entries

- **General access to A:**

$$\text{With } \|A\|_{max} \leq 1,\ O_A|0\rangle|i\rangle|j\rangle = \left(a_{i,j}|0\rangle + \sqrt{1-|a_{i,j}|^2}|1\rangle\right)|i\rangle|j\rangle$$

- **Nonzero entries of A:**

$$O_A|0\rangle|l\rangle|j\rangle = \left(a_{c(j,l),j}|0\rangle + \sqrt{1-|a_{c(j,l),j}|^2}|1\rangle\right)|l\rangle|j\rangle$$

- **Selecting indices for nonzeros:**

$$O_c|l\rangle|j\rangle = |l\rangle|c(j,l)\rangle$$

(We must ensure reversibility of O_c)

Block-encoding

After construction of O_A and O_c(which depend on structure of A),

A is encoded in $U_A = (I_2 \otimes H^{\otimes m} \otimes I_N)(I_2 \otimes O_c)O_A(I_2 \otimes H^{\otimes m} \otimes I_N)$

Summary for matrices

Dense matrices

Generic algorithm, circuit automatically generated.

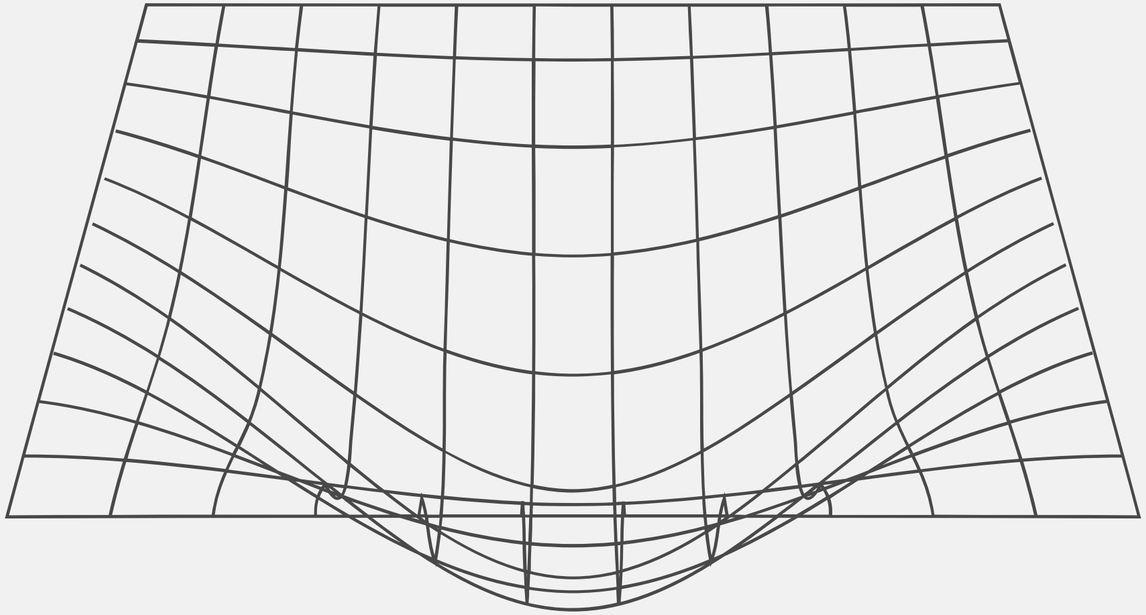
Expensive to achieve on a NISQ computer (+ classical cost).

Sparse matrices

Low number of qubits and CNOT gates, adapted to NISQ.

Oracles depend on the matrix structure, hand-designed.

Then we need **linear solvers**



Linear system solvers

$$Ax=b$$

- **Harrow-Hassidim-Lloyd (HHL)**

  Eigenvalue decomposition and phase estimation.

- **Variation Quantum Linear Solver (VQLS)**

  Optimize a cost function representing the solution. More suited for NISQ architectures.

- **Quantum Singular Value Transformation (QSVT)**

  Transform the singular values to their inverse.

QSVT

Quantum Singular Value Transformation

Principle [Gilyen et al., 2019]

If $W\Sigma V^\dagger$ is the SVD of a matrix $A \in \mathbb{C}^{N\times N}$ and p is a polynomial of degree d with some constraints:

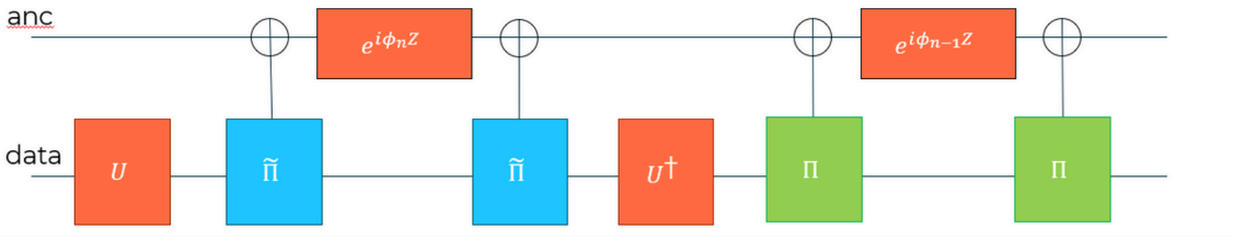
$$QSVT^p(A) = \begin{cases} Wp(\Sigma)V^\dagger, & \text{if } d \text{ is odd} \\ Vp(\Sigma)V^\dagger, & \text{if } d \text{ is even} \end{cases}$$

$$\text{with } p(\Sigma) = diag(p(\sigma_1), \ldots, p(\sigma_N))$$

Can be applied to linear system inversion, Hamiltonian simulation, Grover...

Quantum implementation

From the block encoding $A = \tilde{\Pi}U\Pi$ we define an operator used to obtain the QSVT



QSVT for linear systems

Idea

$$\text{If } A = W\Sigma V^\dagger, \text{ then } A^\dagger = V\Sigma W^\dagger \text{ and } A^{-1} = V\Sigma^{-1}W^\dagger$$

If p approximates the function $x \to x^{-1}$ then

$QSVT^p(A^\dagger) = Vp(\Sigma)W^\dagger$ approximates A^{-1}

Polynomial approximation of 1/x

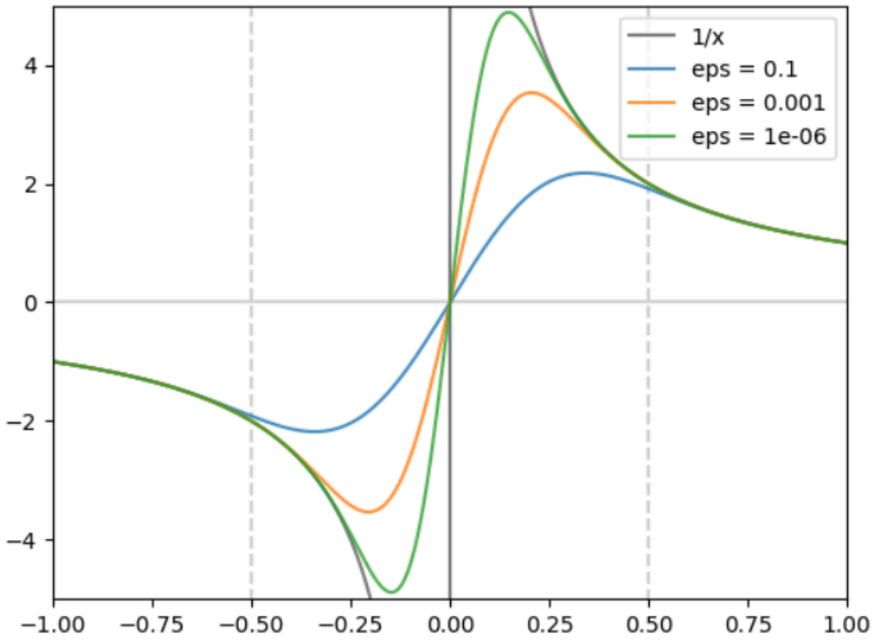
The inverse function is approximated by an odd polynomial on $[-1, -1/\kappa] \cup [1/\kappa, 1]$,

$$\text{starting from } f_{\epsilon,\kappa}(x) = \frac{1-(1-x^2)^b}{x}, \text{ with } b(\epsilon,\kappa) = \lceil \kappa^2 log(\kappa/\epsilon) \rceil.$$

QSVT for linear systems

Algorithm

1. State preparation of right-hand side $|b\rangle$
2. Block-encoding of $A^\dagger$
3. Polynomial approximation of 1/x
4. Convert polynomial to angles
5. Create quantum circuit
6. Run the circuit and post-processing



Polynomial approximation of 1/x ($\kappa = 2$).

Complexity

$$\text{Queries to } U_A: \ \mathcal{O}(\kappa \, log(\kappa/\epsilon))$$

$$+ \text{ classical cost}$$

Iterative refinement

for linear systems (preliminary)

Principle

Solving $Ax = b$ corresponds to zeroing $f(x) = Ax - b$.
Solution via Newton's algorithm at iteration $k+1$ is

$$x_{k+1} = x_k - f'(x_k)^{-1}.f(x_k) \Leftrightarrow x_{k+1} = x_k + A^{-1}r_k$$
$$\text{with } r_k = b - Ax_k \text{ (residual).}$$

Algorithm in mixed precision

Solve $Ax = b$ to get x_0 (low precision) $\leftarrow$ QPU

At each iteration k we do:

1. $r_k \leftarrow b - Ax_{k-1}$ (high precision) $\leftarrow$ CPU
2. Solve $Ae_k = r_k$ (low precision) $\leftarrow$ QPU
3. $x_k \leftarrow x_{k-1} + e_k$ (high precision) $\leftarrow$ CPU

until desired precision is achieved.

Iterative refinement for QSVT

Quantum specifics

Right-hand side must be normalized: $A\frac{e_k}{||r_k||} = \frac{r_k}{||r_k||}$

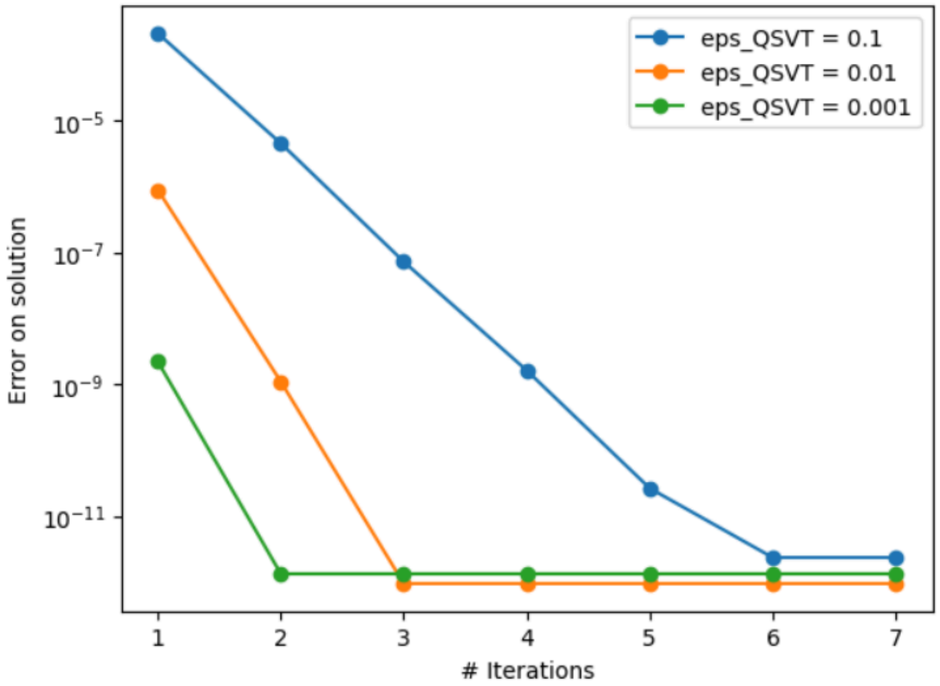
$u = \frac{e_k}{\|e_k\|}$ is recovered with $argmin_{\mu\in\mathbb{R}}|A(x+\mu u) - b|$.

Less sampling

Reduced precision at each iteration requires less samples.

Less quantum resources

Lower precision decreases the polynomial degree (then the number of RZ gates) and the number of call to BE(A).



QSVT + iterative refinement (accuracy)

low precision = eps_QSVT, $\kappa = 2$

Comments on solvers

- Quantum advantage: people expect a lot from QC for HPC but this is still a research effort that ``might'' bring advantage.
- Real cost of quantum algorithms should include: state preparation, interaction with classical machine, sampling, measurement...
  Possible classical/quantum overlapping might improve this cost.
- Always compare with the best classical counterpart and with same conditions:

  e.g., HHL: $s\kappa\mathcal{O}(\text{polylog}(N, log(1/\epsilon)))$

  CG: $s\kappa\mathcal{O}(sN, \sqrt{\kappa}log(1/\epsilon)))$
- Preconditioning is also possible.

Conclusion

What is needed to make Quantum a reality for high performance scientific computing

Accuracy

We need accurate solutions to general scientific problems, even if it is not at scale for the moment.

Community

We should promote a research community dedicated to linear algebra quantum algorithms.

Programming tools

Let’s exploit the experience we have gained from CPU/GPU hybridization. See Q-Pragma by EVIDEN.