A mixed-precision quantum-classical algorithm for solving linear systems

1st Océane Koska
Université Paris-Saclay and Eviden
Orsay, France
oceane.koska@eviden.com

2nd Marc Baboulin
Université Paris-Saclay and Inria
Orsay, France
marc.baboulin@inria.fr

3rd Arnaud Gazda
Eviden Quantum Lab
Les Clayes-sous-Bois, France
arnaud.gazda@eviden.com

***Abstract*—We address the problem of solving a system of linear equations via the Quantum Singular Value Transformation (QSVT). One drawback of the QSVT algorithm is that it requires huge quantum resources if we want to achieve an acceptable accuracy. To reduce the quantum cost, we propose a hybrid quantum-classical algorithm that improves the accuracy and reduces the cost of the QSVT by adding iterative refinement in mixed-precision A first quantum solution is computed using the QSVT, in low precision, and then refined in higher precision until we get a satisfactory accuracy. For this solver, we present an error and complexity analysis, and first experiments using the quantum software stack myQLM.**

***Index Terms*—Quantum computing, Linear systems, Quantum Singular Value Transformation, Mixed-precision algorithms, Iterative refinement.**

I. Introduction

In this paper we address the linear system (LS) problem where, given a nonsingular matrix $A \in \mathbb{R}^{N\times N}$ and a vector $b \in \mathbb{R}^N$, we want to compute $x \in \mathbb{R}^N$ such that

$$Ax = b. \tag{1}$$

Solving the LS problem accurately and efficiently is a fundamental problem in computational science for which there exist many algorithms and software libraries on classical processors [2], [16]. Numerical methods based on factorization or iterative algorithms have reached a high level of maturity on classical computers in particular with the impressive performance of 1 Exaflop/s achieved for solving a dense linear system by Gaussian elimination (Linpack benchmark [33]). The perspective of having operational quantum computers in a near future has motivated the development of quantum algorithms for the LS problem, with the promise of either obtaining faster solution or solving problems that are currently intractable with classical supercomputers. With algorithms such as Harrow-Hassidim-Lloyd (HHL) [18], the Quantum Singular Value Transformation (QSVT) [15], or the Variational Quantum Linear Solver (VQLS) [6], quantum algorithms for linear systems offer the potential for exponential speedups under specific conditions.

However, these quantum algorithms currently have limitations in terms of matrix conditioning and solution accuracy. These can be mitigated by using a combination of both classical computations (running on a Central Processing Unit - CPU) and quantum computations (running on a Quantum Processing Unit - QPU). Recent studies have proposed architectural designs for the integration of quantum devices into High-Performance Computing (HPC) systems, to improve the computation accuracy and to reduce the latency [5], [22]. This type of architectures will be beneficial for implementations that require data transfer between CPU and QPU. At the algorithm level, the techniques that could be targeted to improve linear system solvers could for instance include preconditioning methods or iterative refinement.

Iterative refinement has been recently studied for the HHL algorithm [36], [39] or for optimization problems [31]. In this paper we consider the LS problem solved via the QSVT method which offers several advantages: the matrix does not need to be Hermitian and even not square (in this case we solve a least squares problem), the QSVT method can exploit efficient block-encoding techniques and appears to be well suited for Large Scale Quantum (LSQ) architectures. Note that our work is carried out in an LSQ context and not NISQ (Noisy Intermediate-Scale Quantum) due to the excessive depth of quantum circuits for the QSVT algorithm. The quantum circuit for the QSVT requires a polynomial approximation that is very expensive if we want to achieve acceptable accuracy (typically better than 10^{-5} [32]). This has motivated our choice for developing a mixed-precision algorithm which combines the speed and limited precision of the QSVT with iterative refinement in higher precision. Mixed-precision techniques are widely used in HPC algorithms [1]. They have been initially motivated by the emergence of GPU accelerators that provide high performance when computing in lower precision arithmetic, for instance using tensor core units [3]. For hybrid CPU/GPU architectures we can achieve high performance of the solver while preserving the accuracy of the higher precision and we investigate how similar techniques can be applied with the QSVT method. Similarly to the CPU/GPU case, our iterative refinement is also hybrid since it uses a classical processor (to compute the residual and solution update) and a quantum processor (for the QSVT solver). We adapt the iterative refinement to the case where the LS problem is solved via the QSVT and we present an error and complexity analysis to evaluate the quantum ans classical costs of our algorithm. We also show that our approach provides a significant advantage in complexity compared to directly solving the LS problem with QSVT in higher precision.

The paper is organized as follows: in Section II, we recall some results on QSVT and classical mixed-precision iterative refinement applied to LS problems. Then in Section III, we describe a hybrid algorithm for LS problems using iterative refinement based on the QSVT. We also provide a convergence and complexity analysis. In Section IV, we present experimental results on random matrices with various condition numbers. Finally, concluding remarks are given in Section V.

Notation: Unless otherwise stated, $\|.\|$ denotes the Euclidean norm for vectors and the spectral norm for matrices.

II. Background

A. Quantum Singular Value Transformation (QSVT)

1) Block-encoding of matrices: Since quantum algorithms can only handle unitary matrices, the block-encoding technique consists in embedding a non-unitary matrix into a unitary one [9], [10]. Namely a general matrix $A \in \mathbb{C}^{N\times N}$ (with $N = 2^n$) is encoded into a unitary matrix U as

$$U = \begin{bmatrix} A & \cdot \\ \cdot & \cdot \end{bmatrix}.$$

The matrix A can be expressed from the unitary U using two projectors $\tilde{\Pi}$ and Π with

$$A = \tilde{\Pi} U \Pi.$$

When we apply U to $|0\rangle_a |\psi\rangle_d$, where $|0\rangle_a$ corresponds to the ancilla qubits and $|\psi\rangle_d$ corresponds to the data qubits, we get

$$U\left(|0\rangle_a |\psi\rangle_d\right) = |0\rangle_a A |\psi\rangle_d + \cdots .$$

The literature provides several block-encoding methods to encode an arbitrary matrix into a unitary. The Linear Combination of Unitaries (LCU) method is a versatile approach to encode matrices [12]. It consists in representing the matrix A as a weighted sum of unitary operators $A = \sum_j \alpha_j U_j$.This method relies on the state preparation of the coefficients α_j using ancillary qubits and controlled operations. This method can be used for general matrices [25]. The Fast Approximate Block Encoding (FABLE) algorithm [10] provides an efficient way to construct an approximation of block-encoding by eliminating the negligible terms and removing useless controls in the LCU approach. This algorithm achieves a complexity bounded by $\mathcal{O}(4^n)$ gates for general and unstructured matrices. However it is also possible to take advantage of the sparsity and the structure of some matrices to reach a complexity in $\mathcal{O}(poly(n))$ [9].

2) QSVT definition: The Quantum Singular Value Transformation (QSVT) is a matrix function that operates on the singular values of a matrix using a quantum computer [15]. Given a matrix $A \in \mathbb{C}^{N\times N}$ with the Singular Value Decomposition (SVD)

$$A = W\Sigma V^\dagger$$

($\Sigma = diag(\sigma_1, \ldots, \sigma_N)$ and W, V unitary matrices), given a polynomial P of degree d, then the QSVT of A using P is expressed by

$$W\Sigma V^\dagger = \begin{cases} WP(\Sigma)V^\dagger \text{ if } d \text{ is odd}, \\ VP(\Sigma)V^\dagger \text{ if } d \text{ is even}. \end{cases}$$

However, the QSVT imposes some constraints on the polynomial used in the transformation [15], [29]:

- P has parity-$(d \bmod 2)$,
- $\forall x \in [-1, 1]$, $|P(x)| \leq 1$,
- $\forall x \in (-\infty, -1] \cup [1, \infty)$, $|P(x)| \geq 1$,
- if d is even, then $\forall x \in \mathbb{R}$, $P(ix)P^*(ix) \geq 1$.

3) QSVT implementation: If the condition above are respected, there exists a vector of phases $\Phi = \{\phi_1, \ldots, \phi_d\} \in \mathbb{R}^d$ such that we can define an alternating phase modulation sequence operator U_Φ such that:
if d is odd,

$$U_\Phi = e^{i\phi_1(2\tilde{\Pi}-I)} U \prod_{j=1}^{(d-1)/2} \left(e^{i\phi_{2j}(2\Pi-I)} U^\dagger e^{i\phi_{2j+1}(2\tilde{\Pi}-I)} U \right) \tag{2}$$

and if d is even,

$$U_\Phi = \prod_{j=1}^{d/2} \left(e^{i\phi_{2j-1}(2\Pi-I)} U^\dagger e^{i\phi_{2j}(2\tilde{\Pi}-I)} U \right). \tag{3}$$

This operator can be used to apply the QSVT in a quantum computer, according to the polynomial P, to the matrix A

$$QSVT^P(A) = \begin{cases} \tilde{\Pi} U_\Phi \Pi, \text{if } d \text{ is odd} \\ \Pi U_\Phi \Pi, \text{if } d \text{ is even} \end{cases}$$

Remark 1. *For a problem of size 2^n with a polynomial of degree d, then it comes from [15]) that the alternating phase modulation sequence described in Equations (2) and (3) can be efficiently implemented using*

- *n data qubits,*
- *a single ancilla qubit,*
- *d calls to the block-encoding U and $U^\dagger$,*
- *d calls to the operators of the form $e^{i\phi(2\Pi-I)}$ and $e^{i\phi(2\tilde{\Pi}-I)}$.*

Then the circuit's depth and complexity scale logarithmically with the problem dimension and linearly with the degree of the polynomial.

4) Solving linear systems with QSVT: To solve LS problems via the QSVT we need to find an odd-degree polynomial approximation of the inverse function that fits the requirements given in Section II-A2. By applying this polynomial P for the QSVT of the matrix $A^\dagger$ we get:

$$QSVT^P(A^\dagger) = VP(\Sigma)W^\dagger \approx V\Sigma^{-1}W^\dagger = A^{-1},$$

where $\Sigma^{-1} = diag(\sigma_1^{-1}, \ldots, \sigma_N^{-1})$.

In practice we want to find an $\frac{\epsilon}{2\kappa}$-approximation to $\frac{1}{2\kappa}\frac{1}{x}$, where κ is the condition number of the matrix to be inverted, and ϵ the error on $[-1, 1] \setminus [\frac{-1}{\kappa}, \frac{1}{\kappa}]$. The inverse function is

difficult to approximate with a polynomial (not continuous, infinite values in 0...). We need to find an odd function approximating the inverse function on $[-1,1] \setminus [\frac{-1}{\kappa}, \frac{1}{\kappa}]$ that would be easier to approximate using a polynomial. A function that can be used is

$$f_{\epsilon,\kappa}(x) = \frac{1-(1-x^2)^b}{x},$$

where $b(\epsilon,\kappa) = \lceil \kappa^2 log(\kappa/\epsilon) \rceil$ [15].

Then this function can be ϵ-approximated by the polynomial

$$P^{1/x}_{2\epsilon,\kappa}(x) = 4\sum_{j=0}^{D}(-1)^j \left[2^{-2b} \sum_{i=j+1}^{b} \binom{b+i}{2b} \right] T_{2j+1}(x), \tag{4}$$

where $T_i(x)$ is the Chebyshev polynomial of first kind of order i, and $D(\epsilon,\kappa) = \lceil \sqrt{b(\epsilon,\kappa)\, log(4b(\epsilon,\kappa)/\epsilon)} \rceil$ [30]. Using Chebyshev polynomials to perform the polynomial approximation (instead of expressing the approximation in the canonical basis) highly reduces the impact of Runge's phenomenon when working with high degree polynomials [35].

However, this polynomial does not fit the conditions of the QSVT, because it is not necessarily bounded in magnitude by 1 for $x \in \left[\frac{-1}{2\kappa}, \frac{1}{2\kappa}\right]$. To enforce the magnitude to be bounded, we need to multiply this polynomial by another one. A polynomial that approximates a rectangular function will fit these constraints [30].

One of the main challenges in solving linear systems using QSVT lies in accurately approximating the inverse function with a polynomial, without drastically increasing the polynomial's degree. This constraint impacts the achievable accuracy in linear system solving and/or limits the maximum condition number that can be addressed. Current state-of-the-art methods achieve a precision of 10^{-5} for condition numbers as large as 10^6 [32]. A summary of error analysis for the polynomial approximation and the resulting forward error on the LS solution can be found in [28, p. 125].

B. Mixed-precision iterative refinement for linear systems

Iterative refinement [19], [38] is a well-known technique that improves a computed solution $\tilde{x}$ to $Ax = b$ (step 0) by performing the following steps:

1) compute the residual $r = b - A\tilde{x}$,
2) solve $Ae = r$ to obtain the correction vector e,
3) update $\tilde{x} \leftarrow \tilde{x} + e$ which gives a $\tilde{x}$ "closer" to the exact solution x.

This process can then be repeated until we obtain a satisfying $\tilde{x}$. Using the same precision arithmetic throughout the process (*fixed-precision* iterative refinement) is classically used to improve the accuracy of a computed solution or added to ameliorate a potentially instable solver [19, p. 232]. However, with the emergence of processors that propose much faster computation using lower precision arithmetic, it became attractive to combine different precisions in order to exploit the high performance provided by some processors or computational units (e.g., tensor cores for NVIDIA GPUs), resulting in so-called *mixed-precision* iterative refinement algorithms.

When solving linear systems, mixed-precision is of interest when the refinement (computed in higher precision) is cheap compared to the computation of the first $\tilde{x}$ (computed in lower precision). In mixed-precision iterative refinement for general matrices, the most expensive tasks are the initial solution of $Ax = b$ (step 0) and the successive solves of $Ae = r$ (step 2) which are then achieved in lower precision. On the other hand the residual and the updated $\tilde{x}$ (steps 1 and 3) are computed in higher precision. Algorithm 1 gives the precisions that are used for each task of the algorithm. When the solves are performed using LU factorization, then we can use the L and U factors produced in step 0 to achieve step 2, which still reduces the computational cost of the overall process.

A general framework that describes mixed-precision algorithms for linear systems can be found in [11]. A special case can be derived that corresponds to a common use in heterogeneous computing (see e.g., [4]) where we have a working (high) precision u (e.g., double precision, $u = 10^{-16}$) and a low precision u_l (e.g., single precision, $u = 10^{-8}$). In this situation the most expensive part is performed at precision u_l on an accelerator like a GPU to take advantage of low precision arithmetic while the other steps remain executed on the CPU.

Algorithm 1 Mixed-precision linear system solution using 2 precisions.

Input: $A \in \mathbb{R}^{N\times N}, b \in \mathbb{R}^N$ stored at precision u with $u \ll u_l$.
Compute a solution x_0 to $Ax = b$ at precision u_l.
while desired accuracy not reached **do**
  Compute $r_i = b - Ax_i$ at precision u.
  Solve $Ae_i = r_i$ at precision u_l.
  Update $x_{i+1} = x_i + e_i$ at precision u.
end while

The limiting accuracy does not depend on u_l in the system solving $Ae_i = r_i$, it only depends on the choice for u [20]. Therefore we can work with $u \ll u_l$ and still get an accurate result if u is well chosen. Traditionally, we have $u = u_l^2$ to get a limiting accuracy of order u. Note that a large value of u_l accelerates each iteration but requires more iterations.

III. Iterative refinement for QSVT-based linear system solution

A. Algorithm

Suppose we can solve $Ax = b$ via the QSVT method with a (low) accuracy ϵ_l, i.e., that can produce a solution $\tilde{x}$ such that $\|x - \tilde{x}\| \leq \epsilon_l \|x\|$. Following [28, p. 126], to obtain an accuracy (relative error) of order ϵ_l on the non-normalized solution $\tilde{x}$, we need to approximate the inverse function on $[-1,-1/\kappa] \cup [1/\kappa, 1]$ with an error $\epsilon' = \mathcal{O}(\epsilon_l/\kappa)$.

We want to improve the quality of the solution given by the QSVT described in Section:II-B by refining it in higher

precision u as presented in Algorithm 2 until we achieve an accuracy ϵ. In this algorithm we will use two different types of processors: the solving phases via the QSVT will be achieved on a quantum processor (QPU) while the residual r_i and the correction to the solution x_{i+1} will be computed on a classical processor (CPU). Note that in practice the QSVT routine, like most quantum algorithms, is by nature hybrid and not fully executed on the QPU. Indeed some tasks are performed on the CPU (matrix decomposition before block-encoding, preprocessing of the state preparation, computation of angles for the polynomial approximation, post-processing after measurement). Note also that the hybrid scheme of Algorithm 2 requires to store A and b in a precision at least u on the CPU.

Remark 2. *One particularity of using iterative refinement in quantum algorithms is that we deal with quantum states. Therefore before solving $Ax = b$, we need to normalize b as*

$$A\frac{x}{\|b\|} = \frac{b}{\|b\|}.$$

The sampling at the end of the QSVT will provide $\eta = \frac{x}{\|x\|}$ and we are able to recover $\|x\|$ by solving the minimization problem

$$argmin_{\mu\in\mathbb{R}}|A(x+\mu\eta) - b|.$$

This phase occurs for each call to the QSVT routine and is performed on the CPU device.

The **stopping criterion** for our iterative refinement will be based on the scaled residual defined by $\omega = \frac{\|b-A\tilde{x}\|}{\|b\|}$. We aim at finding $\tilde{x}$ such that $\omega \leq \epsilon$. When κ is not too large, ω classically provides close bounds for the relative error since we have (see, e.g., [26, p. 68])

$$\frac{\omega}{\kappa} \leq \frac{\|x-\tilde{x}\|}{\|x\|} \leq \kappa\omega. \tag{5}$$

Moreover ω is independent to scaling of Ax and b by a same coefficient, which will occur because quantum algorithms require b to be normalized (see Remark 2).

Algorithm 2 Iterative refinement for QSVT-based linear system solution.

Input: A, b, QSVT accuracy ϵ_l, targeted accuracy ϵ at precision u.
Compute $x_0 = A^{-1}b$ at accuracy ϵ_l using QSVT (QPU).
while accuracy ϵ is not reached on x_i **do**
  Compute $r_i = b - Ax_i$ in high precision u (CPU).
  Compute $e_i = A^{-1}r_i$ at accuracy ϵ_l with QSVT (QPU).
  Update $x_{i+1} = x_i + e_i$ in high precision u (CPU).
end while

To compute x_0 we first need to generate all the quantum circuits/routines that will be executed on the QPU. This generation is called quantum circuit synthesis, which corresponds in classical computing to the compilation phase, and is executed on a classical computer. Computing x_0 requires the use of 3 quantum routines:

- State preparation implementation for the normalized value of b,
- Block-encoding of $A^\dagger$,
- QSVT routine implementing A^{-1} (relying on the block encoding of $A^\dagger$).

Quantum routines, once compiled and transferred to the QPU, do not have to be redefined. In particular, the concept of "linker-loader", widely used in classical computing, is part of the architecture for integrating QPUs into HPC resources (see, e.g., Figure 4 in [5]). Thus, each iteration of Algorithm 2 requires to transfer a reduced amount of data compared to computing x_0 (since only r_i needs to be encoded and transferred to the QPU).

Remark 3. *We point out that the result is obtained through sampling/measurement. Consequently, this hybrid algorithm relies on the "collapse" of the quantum solution and cannot be used in a subsequent quantum algorithm, except if we re-encode the solution in the quantum computer via state preparation.*

B. Convergence and accuracy

In this section we compute the scaled residual obtained after each iteration of the refinement and a bound on the iteration count. In our demonstrations we omit the effect of rounding errors but the high precision u (unit roundoff) used for CPU computations should be chosen accordingly to the target precision ϵ, a safe choice being $u = \theta\epsilon$ with $\theta \leq 1$.

Theorem III.1. *Suppose the QSVT can solve $Ax = b$ with low accuracy ϵ_l with $\epsilon_l\kappa < 1$ and that we apply mixed-precision iterative refinement as given in Algorithm 2 with high precision u when computing the residual and solution update. Then after i iterations we have $\|r_i\| \leq (\epsilon_l\kappa)^{i+1}\|b\|$. The number of iterations to obtain a solution $\tilde{x}$ such that $\frac{\|b-A\tilde{x}\|}{\|b\|} \leq \epsilon$ is bounded by $\lceil \log(\epsilon)/\log(\epsilon_l\kappa)\rceil$.*

Proof. We first compute x_0 with the QSVT with $\|x - x_0\| \leq \epsilon_l\|x_0\|$. Then using the left part of Equation (5) we have

$$\|r_0\| = \|b - Ax_0\| \leq \epsilon_l\kappa\|b\|.$$

Then for the first iteration we have

$$\|r_1\| = \|b - Ax_1\| = \|b - A(x_0 + e_0)\| = \|r_0 - Ae_0\|.$$

Using again Equation (5) to the linear system $Ae = r_0$ we get

$$\|r_1\| = \|r_0 - Ae_0\| \leq \epsilon_l\kappa\|r_0\| \leq (\epsilon_l\kappa)^2\|b\|.$$

Then by straightforward recurrence on i we obtain

$$\forall i,\ \|r_i\| \leq (\epsilon_l\kappa)^{i+1}\|b\|.$$

The desired final error ϵ will be obtained when we will have

$$\frac{\|b - Ax_i\|}{\|b\|} \leq (\epsilon_l\kappa)^{i+1} \leq \epsilon,$$

which yields $i \leq \log(\epsilon)/\log(\epsilon_l\kappa) - 1$ and thus $\lceil \log(\epsilon)/\log(\epsilon_l\kappa)\rceil$ will be an upper bound for the iterative refinement process.

□

Theorem III.1 states that the convergence of Algorithm 2 is ensured as long as $\epsilon_l \kappa < 1$. The scaled residual contracts by a factor $\epsilon_l \kappa$ at each iteration until it reaches a maximum value of $\mathcal{O}(\epsilon)$ after at most $i_{max} = \lceil \log(\epsilon)/\log(\epsilon_l \kappa) \rceil$ iterations. The quantity i_{max} will be used in the following section to evaluate the complexity of the linear system solver.

C. Complexity analysis

Since Algorithm 2 is hybrid, the resulting complexity includes quantum and classical costs which are presented in this section. We also mention the data communication between the 2 devices.

1) Quantum cost: The quantum complexity of the QSVT (denoted by $\mathcal{C}_{QSVT}$ and detailed in Remark 1) mainly relies on the cost $\mathcal{B}$ of the block-encoding circuit used to encode $A^\dagger$, which depends on the chosen block-encoding method (see Section II-A1). Then the quantum complexity of the mixed-precision QSVT solver depends on the 3 following components:

- The **number of calls to the solver** is evaluated using the upper bound provided in Theorem III.1 for the mixed-precision solver.
- The **complexity of the QSVT** $\mathcal{C}_{QSVT}$, which corresponds to the cost of the multiple calls to the block-encoding. We recall that the number of calls to the block-encoding is given by the degree of the polynomial as explained in Remark 1.
- The **number of samples** necessary to achieved a targeted error ϵ is $\mathcal{O}(1/\epsilon^2)$.

Then we have

$$\text{Total complexity} \;=\; \#\text{solves} \;\times\; \mathcal{C}_{QSVT} \;\times\; \#\text{samples}.$$

In Table I, we compare the complexity when using directly QSVT in high precision (one call to the QSVT and $\epsilon_l = \epsilon$) and when using iterative refinement in mixed-precision.

	QSVT only	QSVT with iterative refinement
# solves	1	$\leq \left\lceil \frac{log(\epsilon)}{log(\kappa\ \epsilon_l)} \right\rceil$
$\mathcal{C}_{QSVT}$	$\mathcal{O}\left(\mathcal{B}\ \kappa\ log(\kappa/\epsilon)\right)$	$\mathcal{O}\left(\mathcal{B}\ \kappa\ log(\kappa/\epsilon_l)\right)$
# samples	$\mathcal{O}(1/\epsilon^2)$	$\mathcal{O}(1/\epsilon_l^2)$
Total	$\mathcal{O}\left(\frac{\mathcal{B}\kappa}{\epsilon^2}\ log(\kappa/\epsilon)\right)$	$\mathcal{O}\left(\left\lceil \frac{log(\epsilon)}{\log(\kappa\ \epsilon_l)} \right\rceil \frac{\mathcal{B}\kappa}{\epsilon_l^2}\ log(\kappa/\epsilon_l)\right)$

TABLE I: Quantum cost for QSVT-based LS solution with and without iterative refinement.

Using iterative refinement jointly with QSVT to solve linear systems provides some advantages. First, working with a lower precision decreases the number of samples needed to reach this precision. Actually, to get a precision ϵ we need $\mathcal{O}(1/\epsilon^2)$ samples, and then we need to run the quantum circuit $\mathcal{O}(1/\epsilon^2)$ times. Then, something more specific to the QSVT is that we can adapt the precision of the polynomial approximation of the inverse function to the accuracy ϵ_l used in the iteration. Reducing this precision also reduces the degree of the polynomial and the resulting number of calls to the block-encoding U of $A^\dagger$ (and $U^\dagger$). Finally we will verify in our experiments in Section IV that, for our experimental values of ϵ, ϵ_l and κ (with $\epsilon < \epsilon_l < 1/\kappa$) and using less than $\lceil \log(\epsilon)/\log(\epsilon_l \kappa) \rceil$ iterations, the quantum cost is smaller when using iterative refinement.

2) Classical cost: First, the QSVT (with or without iterative refinement) requires to pre-process (on a classical computer) the input matrix to create its block-encoding circuit [17], [25]. This task can be computationally expensive especially for dense and unstructured matrices (e.g., $\mathcal{O}(n4^n)$ flops using [17]). In both columns of Table I this pre-processing step is performed only once (for the iterative approach the result is reused by the QPU throughout the iterations). Another initial computation consists in finding the phases Φ for the QSVT, using algorithm such as [13], [32]. The computational cost of this task can scale linearly with the condition number κ (see [32]). An additional cost concerns the state preparation of the right-hand sides with the generation of the corresponding circuits. For instance the algorithm provided in [23] relies on a tree that needs to be classically computed and can be performed in $\mathcal{O}(N)$ flops. Then, for the quantum iterative refinement method we need to perform some processing on the CPU before and after each solve. These tasks consist in normalizing the residual before the solve, then de-normalizing the result sent back from the QPU at each iteration (see Remark 2) and finding the associated residual. The normalization can be performed in $\mathcal{O}(N)$ flops and finding the residual is a matrix-vector operation ($\mathcal{O}(N^2)$ flops). The de-normalization step can be performed for example using the Brent's method [7] (in the worst case scenario the complexity is $\mathcal{O}(log(1/\epsilon))$).

3) Remarks on data communication: Our algorithm involves data communication between the CPU and the QPU. At the beginning we send the circuit representation of the block-encoding of $A^\dagger$ (denoted as BE($A^\dagger$)) from the CPU to the QPU. This communication cost will depend on the matrix A but also on the method used to perform the block-encoding and that determines the circuit's size (see Section II-A1). This data transfer has to be performed only once because the matrix A remains unchanged during the whole refinement process. Another transfer concerns the vector of phases Φ used in the QSVT circuit. This vector is of size d, where d is the degree of the polynomial approximating the inverse function in Equation (4). Then at each solve step, we need to transfer the right-hand sides (b and the residuals) from the CPU to the QPU. This is performed by sending the corresponding circuit description for state preparation (denoted as SP(b), SP(r_i), etc). The size of these circuits depends also on the method used to perform the state preparation. Moreover, after the solve phase, we need to transfer from the QPU to the CPU the sampled solution, which is a vector of size $N = 2^n$. These data transfers between CPU and QPU are depicted in Figure 1.

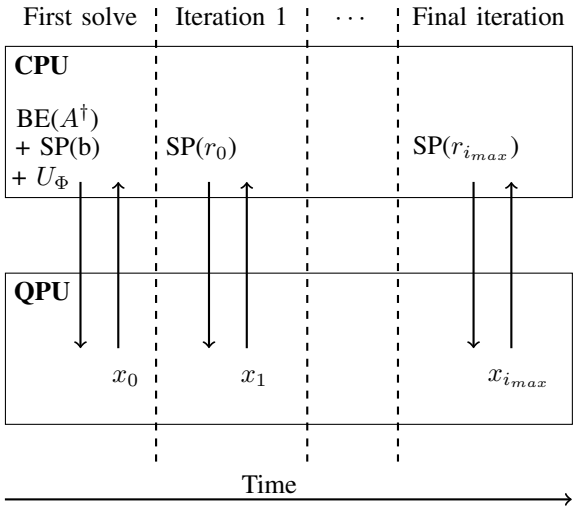

Fig. 1: CPU-QPU communication scheme for Algorithm 2 (BE=block-encoding, SP=state preparation).

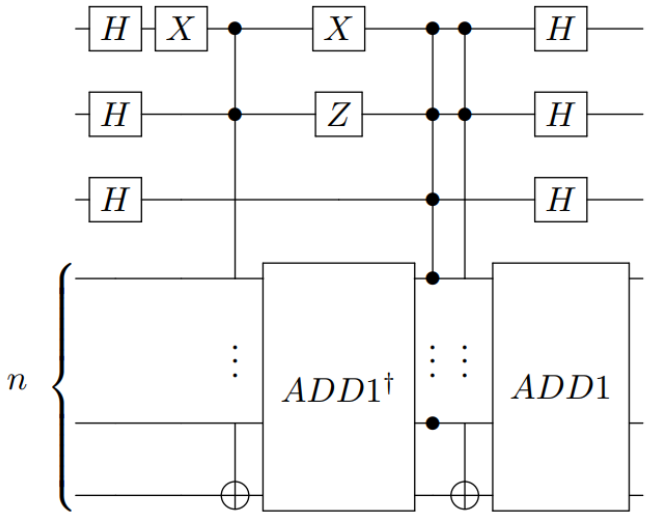

Fig. 2: Circuit for the block-encoding of the tridiagonal matrix in Equation (7).

4) Practical example: Let us consider the one-dimensional Poisson equation

$$\forall x \in (0,1), -u''(x) = f(x), \tag{6}$$

with the Dirichlet boundary conditions $u(0) = u(1) = 0$. This problem can be solved using the finite difference method by discretizing Equation (6) with a step $h = 1/(N+1)$ and $u_j = jh$, $f_j = f(jh)$.

The solution can be obtained by solving the linear system provided in Equation 7.

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ 0 & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_N \end{pmatrix}. \tag{7}$$

a) Block-encoding: For the matrix given in Equation (7) (with $N = 2^n$) we use the block-encoding technique provided in [37] resulting in the circuit given in Figure 2.

b) Complexity: In this section we detail the quantum and classical complexities of our hybrid algorithm when solving Equation (7). The classical cost is expressed in floating-point operations (flops) and the quantum cost is expressed in number of T-gates, because the depth of the circuit requires to use a fault-tolerant quantum computer [21]. The subroutines used in our algorithm are:

- Tree-based state preparation (SP) described in [23],
- Block-encoding (BE) provided in [37],
- QSVT quantum circuit (U_Φ...) proposed in [15],
- QSVT phases (Φ) computed as in algorithm given in [32],
- Solution via Brent's method [7] (de-normalization, see Remark 2).

In Table II we summarize the classical and quantum costs for the first solve (First) and at each iteration (Iter). The block-encoding of the tridiagonal matrix is predetermined (expressed analytically in [37]) and requires no classical cost. We refer to [24], [34] for the decomposition in T gates of the multi-controlled Toffoli gates and adders for the different quantum circuits of our implementation.

		Classical	Quantum
First	SP	$\mathcal{O}(2^n)$	$\mathcal{O}(polylog(n))$
	BE	-	$\mathcal{O}(n\kappa\log(\kappa/\epsilon_l))$
	QSVT (Φ, U_Φ...)	$\mathcal{O}(\kappa)$	$\mathcal{O}(n\kappa\log(\kappa/\epsilon_l))$
	Solution	$\mathcal{O}(4^n + \log(1/\epsilon))$	-
Iter	SP	$\mathcal{O}(2^n)$	$\mathcal{O}(polylog(n))$
	BE	-	$\mathcal{O}(n\kappa\log(\kappa/\epsilon_l))$
	QSVT (Φ, U_Φ...)	-	$\mathcal{O}(n\kappa\log(\kappa/\epsilon_l))$
	Solution	$\mathcal{O}(4^n + \log(1/\epsilon))$	-

TABLE II: Complexity for solving the Poisson equation with mixed precision iterative refinement.

We point out that this use case is given as an example to illustrate the complexity breakdown of our algorithm. Current classical solvers are efficient at solving this type of linear systems (in $\mathcal{O}(N)$ flops [8]). Moreover the condition number κ exhibits a rapid increase with the problem size (in $\mathcal{O}(N^2)$ with no preconditioning [27]) which makes this linear system solution very expensive for large matrices using QSVT, given the current state of the art.

IV. Numerical experiments

A. Experimental framework

We have implemented the QSVT solver and the iterative refinement using Python and the *myQLM* simulator [14]. Our implementation relies on the following components:

- State preparation using the tree-based method described in [23],

- Polynomial approximation of the inverse function as detailed in [15],
- Algorithm for determining the QSVT angles: for small condition numbers ($\kappa < 100$), we perform the numerical computation of the angles, using the symmetric QSP approach from [13]. For bigger condition numbers, we use the estimation approach provided in [32],
- Implementation of the QSVT implemented with myQLM, following [15],
- Iterative refinement in Python, using numpy and scipy libraries.

In the following experiments, the size of the problem was set to $N = 16$ i.e., $n = 4$ qubits. The matrix $A \in \mathbb{R}^{N\times N}$ and the vector $b \in \mathbb{R}^N$ are both randomly generated, with $\|b\| = 1$ for simplicity. We have also simulated the algorithm with the tridiagonal matrix given in Section III-C4. The obtained results are similar in terms of convergence and then they are not mentioned in the following. Note that for each simulation we limited the execution time to one hour, which enabled us to simulate problems with condition numbers of $\mathcal{O}(10^2)$.

B. Results

We present in Figure 3 the evolution of the scaled residual obtained at each iteration, until convergence. We consider a small condition number $\kappa = 10$ and three values for ϵ_l. The targeted accuracy is $\epsilon = 10^{-11}$. We observe that in all cases the bound $\lceil \log(\epsilon)/\log(\epsilon_l\kappa) \rceil$ obtained in Theorem III.1 is a sharp estimate for the iteration count. We also observe that due to the small values of κ the scaled residual is here also a good estimate of the forward error, as expected from Equation (5).

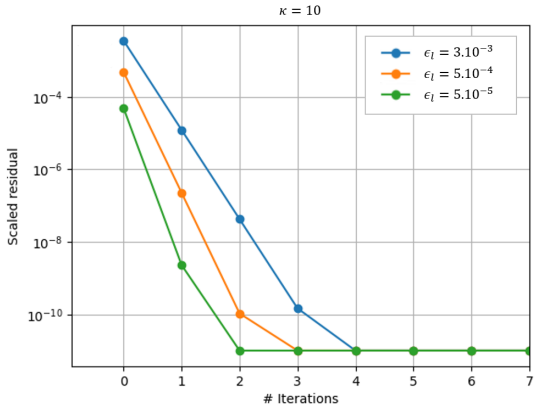

Fig. 3: Scaled residual until convergence for $\kappa = 10$, targeted accuracy $\epsilon = 10^{-11}$, and various values of ϵ_l.

In Figure 4 we present the evolution of the scaled residual at each iteration for larger condition numbers. For these experiments, the polynomial approximation and the QSVT angles are computed by the algorithm implemented in [32]. In this context, we only have control on κ since the value of ϵ_l is automatically determined by the algorithm from [32]. Our experiments show that we obtain a satisfying convergence with a number of iterations still lower than the bound from Theorem III.1.

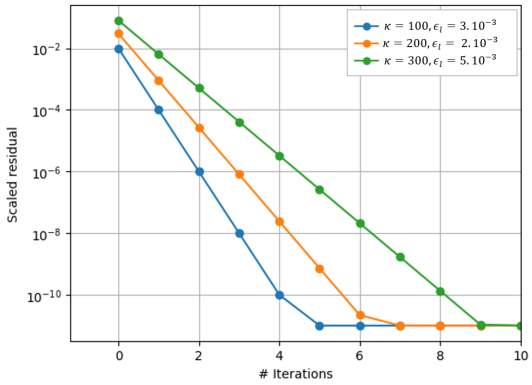

Fig. 4: Scaled residual until convergence, $\kappa = 100, 200, 300$.

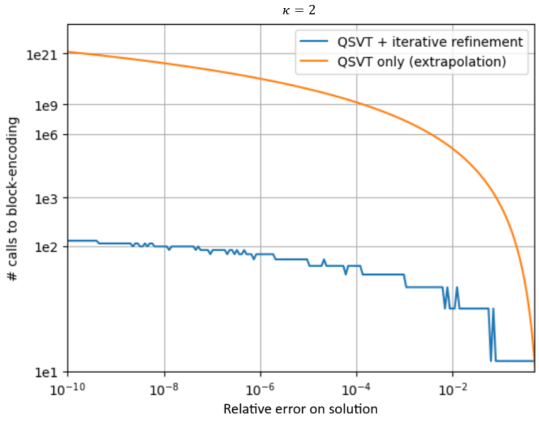

Fig. 5: Complexity in calls to block-encoding for QSVT with and without iterative refinement, $\kappa = 2$.

Then, in Figure 5, we compare the complexity of the LS solving with $\kappa = 2$ for the QSVT with and without mixed-precision iterative refinement. In this numerical experiment we consider that the main cost of the QSVT relies in the block-encoding, so the complexity is expressed as the number of calls to the block-encoding of the matrix $A^\dagger$. The results for the QSVT without iterative refinement are extrapolated from the theoretical complexity provided in Table I since the computation would be intractable for both the polynomial approximation and the simulation of the quantum circuit. On the contrary, the results for the QSVT with mixed-precision iterative refinement are obtained after running our algorithm using a quantum simulator and choosing $\epsilon_l \approx 1/\kappa$. The method with iterative refinement is clearly advantageous for $\epsilon << \epsilon_l$. Note that the two curves coincide at $\epsilon = \epsilon_l$. With larger condition numbers, the gap between the two curves would be bigger, according the complexity figures given in Table I.

V. Conclusion

We have proposed a mixed-precision hybrid CPU/QPU solver for linear systems that computes a first solution in low precision using the (expensive) QSVT method and then

refines the solution in higher precision to achieve the desired accuracy. The main advantage of this method is that it enables us to achieve satisfying accuracy by using affordable quantum resources since we can use a limited precision for the QSVT solver. This solver illustrates what a typical hybrid CPU/QPU algorithm would be in the future that would leverage the strength of each architecture.

The interest of developing such methods is to exploit existing theoretical results for mixed-precision algorithms used in classical computing. Our algorithm has been tested in simulation only which reduces the possibility of scaling to larger problems or handling ill-conditioned matrices. It requires further testing on real quantum machine when error-free hardware will be easily accessible to the scientific community.

ACKNOWLEDGEMENT

This work is part of the HQI initiative (`www.hqi.fr`) and is supported by France 2030 under the French National Research Agency award number "ANR-22-PNCQ-0002".

REFERENCES

[1] A. Abdelfattah, H. Anzt, J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, and A. YarKhan. Linear algebra software for large-scale accelerated multicore computing. *Acta Numerica*, 25:1–160, 2016.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1999. 3rd edition.

[3] M. Baboulin, S. Donfack, O. Kaya, T. Mary, and M. Robeyns. Mixed precision randomized low-rank approximation with GPU tensor cores. In *Euro-Par 2024: Parallel Processing - 30th European Conference on Parallel and Distributed Processing*, volume 14803 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2024.

[4] Marc Baboulin, Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Julie Langou, Julien Langou, Piotr Luszczek, and Stanimire Tomov". Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications*, 180:2526–2533, 2009.

[5] K. Bertels, A. Sarkar, T. Hubregtsen., M. Serrao, A. A. Mouedenne, A. Yadav, A. Krol, and I. Ashraf. Quantum computer architecture: Towards full-stack quantum accelerators. In *2020 Design, Automation & Test in Europe Conference & Exhibition (2020)*. IEEE, 2020.

[6] Carlos Bravo-Prieto, Ryan LaRose, M. Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J. Coles. Variational quantum linear solver. *Quantum*, 7:1188, 2023.

[7] R. P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[8] William Briggs, Van Henson, and Steve McCormick. *A Multigrid Tutorial, 2nd Edition*. SIAM: Society for Industrial and Applied Mathematics; 2nd edition, 2000.

[9] Daan Camps, Lin Lin, Roel Van Beeumen, and Chao Yang. Explicit quantum circuits for block encodings of certain sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 45(1):801–827, 2024.

[10] Daan Camps and Roel Van Beeumen. FABLE: Fast approximate quantum circuits for block-encodings. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 104–113, 2022.

[11] Erin Carson and Nicholas J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM Journal on Scientific Computing*, 40(2):A817–A847, 2018.

[12] A. M. Childs and N. Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *Quantum Information and Computation*, 12(11 - 12):0901–0924, 2012.

[13] Yulong Dong, Lin Lin, Hongkang Ni, and Jiasu Wang. Robust iterative method for symmetric quantum signal processing in all parameter regimes. *SIAM Journal on Scientific Computing*, 46(5):A2951–A2971, 2024.

[14] Eviden Quantum Lab. myQLM: Quantum Computing Framework, 2020-2024. https://myqlm.github.io.

[15] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC '19. ACM, 2019.

[16] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2013. Fourth edition.

[17] Lukas Hantzko, Lennart Binkowski, and Sabhyata Gupta. Tensorized pauli decomposition algorithm. *Physica Scripta*, 99(8):085128, jul 2024.

[18] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), 2009.

[19] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996.

[20] Nicholas J. Higham and Theo Mary. Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31:347–414, 2022.

[21] Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[22] Travis S. Humble, Alexander McCaskey, Dmitry I. Lyakh, Meenambika Gowrishankar, Albert Frisch, and Thomas Monz. Quantum computers for high-performance computing. *IEEE Micro*, 41(5):15–23, 2021.

[23] Iordanis Kerenidis and Anupam Prakash. Quantum Recommendation Systems. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:21, 2017.

[24] Tanuj Khattar and Craig Gidney. Rise of conditionally clean ancillae for optimizing quantum circuits. *arXiv:2407.17966*, 2024.

[25] Océane Koska, Marc Baboulin, and Arnaud Gazda. A tree-approach pauli decomposition algorithm with application to quantum computing. In *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*, pages 1–11, 2024.

[26] Alan J. Laub. *Computational matrix analysis*. SIAM, Philadelphia, 2012.

[27] Byungjoon Lee and Chohong Min. Optimal preconditioners on solving the Poisson equation with Neumann boundary conditions. *Journal of Computational Physics*, 433:110189, 2021.

[28] Lin Lin. Lecture notes on quantum algorithms for scientific computation. *arXiv:2201.08309*, 2022.

[29] Guang Hao Low, Theodore J. Yoder, and Isaac L. Chuang. Methodology of resonant equiangular composite quantum gates. *Physical Review X*, 6(4), 2016.

[30] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2(4), 2021.

[31] Mohammadhossein Mohammadisiahroudi, Brandon Augustino, Pouya Sampourmahani, and Tamás Terlaky. Quantum computing inspired iterative refinement for semidefinite optimization. *Mathematical Programming*, 01 2025.

[32] I. Novikau and I. Joseph. Estimating QSVT angles for matrix inversion with large condition numbers. *Journal of Computational Physics*, 525:113767, 2025.

[33] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL - a portable implementation of the high-performance Linpack benchmark for distributed-memory computers, 2018.

[34] Maxime Remaud and Vivien Vandaele. Ancilla-free quantum adder with sublinear depth. *arXiv:2501.16802*, 2025.

[35] T. J. Rivlin. *Chebyshev Polynomials: From Approximation Theory to Algebra and Number Theory.* Dover, 2020.

[36] Yoshiyuki Saito, Xinwei Lee, Dongsheng Cai, and Nobuyoshi Asai. An iterative improvement method for HHL algorithm for solving linear system of equations. *arXiv:2108.07744*, 2021.

[37] Sunheang Ty, Renaud Vilmart, Axel TahmasebiMoradi, and Chetra Mang. Double-logarithmic depth block-encodings of simple finite difference method's matrices. *arXiv:2410.05241*, 2024.

[38] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, 1963.

[39] Muqing Zheng, Chenxu Liu, Samuel Stein, Xiangyu Li, Johannes Mülmenstädt, Yousu Chen, and Ang Li. An early investigation of the HHL quantum linear solver for scientific applications. *arXiv:2404.19067*, 2024.