

Fermilab

Efficient Floating-Point Arithmetic on Fault-Tolerant Quantum Computers

FERMILAB-PUB-25-0270-SQMS

arXiv:2510.20145

This manuscript has been authored by Fermi Forward Discovery Group, LLC
under Contract No. 89243024CSC000002 with the U.S. Department of Energy,
Office of Science, Office of High Energy Physics.

Efficient Floating-Point Arithmetic on Fault-Tolerant Quantum Computers

José E. Cruz Serrallés,^{1,*} Oluwadara Ogunkoya,^{2,3,†} Doğa Murat Kürkçüoğlu,^{2,3,‡} Nicholas Bornman,^{2,3} Norm M. Tubman,^{2,4} Anna Grassellino,^{2,4} Silvia Zorzetti,^{2,3} and Riccardo Lattanzi¹

¹*Center for Biomedical Imaging, Department of Radiology, New York University Grossman School of Medicine, New York, New York, 10016, USA*

²*Superconducting and Quantum Materials System Center (SQMS), Batavia, Illinois, 60510, USA*

³*Fermi National Accelerator Laboratory, Batavia, Illinois, 60510, USA*

⁴*NASA Ames Research Center, Moffett Field, California 94035, USA*

(Dated: October 24, 2025)

We propose a novel floating-point encoding scheme that builds on prior work involving fixed-point encodings. We encode floating-point numbers using Two's Complement fixed-point mantissas and Two's Complement integral exponents. We used our proposed approach to develop quantum algorithms for fundamental arithmetic operations, such as bit-shifting, reciprocation, multiplication, and addition. We prototyped and investigated the performance of the floating-point encoding scheme on quantum computer simulations by performing reciprocation on randomly drawn inputs and by solving first-order ordinary differential equations, while varying the number of qubits in the encoding. We observed rapid convergence to the exact solutions as we increased the number of qubits and a significant reduction in the number of ancilla qubits required for reciprocation when compared with similar approaches.

I. INTRODUCTION

IEEE-754 floating-point arithmetic is the foundation of numerical computing, ensuring accuracy, consistency, and efficiency across a wide range of classical CPU architectures and integrated circuits [1]. Whether in high-performance computing, mobile devices, or specialized hardware like graphics processing units (GPUs) and application-specific integrated circuits (ASICs), the IEEE-754 standard plays a crucial role in enabling reliable floating-point computations universally. Floating-point numbers offer fixed relative precision [2], as opposed to fixed-point numbers that offer fixed absolute precision [3]. By adjusting the exponent and mantissa within corresponding limits, floating-point numbers allow for a wider range of values and precisions than fixed-point numbers [2]. The latter are represented using integer primitives with an implicit decimal or binary point, so fixed-point arithmetic is often used in applications where deterministic precision and lower computational overhead are necessary, such as digital signal processing and financial calculations [3]. However, most classical computing architectures do not provide native hardware support for fixed-point arithmetic. Instead, fixed-point operations are typically emulated using integer arithmetic, requiring additional software-based scaling and rounding techniques to achieve the desired level of numerical accuracy [3].

The challenge of performing arithmetic operations on quantum computers lies in the precise encoding of numerical data into qubits, along with the design of optimized quantum circuits capable of efficiently executing

addition, multiplication, division, and other mathematical operations. Researchers have investigated various methods for implementing both fixed-point and floating-point arithmetic [4–20]. These approaches were designed to facilitate efficient numerical computations, which are crucial for numerous applications. However, despite the development of theoretical frameworks and preliminary implementations, the majority of these methods have not been thoroughly explored nor widely adopted. A significant number of these approaches rely on brute-force techniques, using hardware description languages (HDLs) such as Verilog or VHDL to construct circuit descriptions [9, 21]. While HDLs provide a straightforward way to model quantum arithmetic, these languages were designed with classical logical synthesis in mind, requiring the use of irreversible operations, such as logical AND, that have no direct unitary analogue on quantum computers. As a result, implementing a single irreversible operation on quantum computers would require its own set of ancillas, which quickly becomes impractical as the complexity of the quantum circuit increases, requiring a large number of irreversible operations.

In addition to techniques based on HDLs, other approaches for floating-point arithmetic include Quantum Fourier Transform (QFT) and Clifford+T gate-based arithmetic design. A comprehensive review of previous work based on QFT can be found in [22, 23]. QFT-based approaches utilize Hadamard gates and controlled rotation gates, which include both Clifford and non-Clifford gates. Much like how the Discrete Fourier Transform diagonalizes the convolution of two sequences into multiplication in the frequency domain, the Quantum Fourier Transform operates instead over the binary states of qubits and allows one to perform arithmetic such as addition and multiplication by diagonalizing the respective operations [24]. In contrast, the Clifford+T-based arithmetic design relies on Clifford gates along with T gates,

* Jose.CruzSerralles@nyulangone.org

† ogunkoya@fnal.gov

‡ dogak@fnal.gov

which are used to decompose the Toffoli gate into a composition of one H gate, seven T gates, and six CNOT operations [25].

The primary goal of this work is to develop a floating-point arithmetic framework that minimizes the number of required ancilla qubits. To achieve this, we propose a novel QFT-based approach for efficient floating-point arithmetic that does not rely on information encoded in the wavefunction coefficients and reuses ancilla qubits by exploiting the structure of the encoding. Namely, our method strictly utilizes information embedded in the non-zero states associated with the binary strings representing a given number, whereas other approaches for quantum computation, such as amplitude encoding, phase encoding, and hybrid methods, store information in the complex weights of the wavefunction itself and differ fundamentally in their algorithmic design [6, 26–30].

II. TECHNICAL BACKGROUND

A. Classical Binary Fixed- and Floating-Point Arithmetic

Binary fixed-point and floating-point arithmetic are nearly identical in the sense that both deal with a significand that is scaled by a power-of-two exponent. In the fixed-point case, the exponent is fixed and implicit, whereas in the floating-point case, the exponent is allowed to vary and is expressed using a set of bits contained within the number, allowing one to represent a much larger set of numbers at the expense of increased algorithmic complexity when performing basic arithmetic operations. The two arithmetic operations that we focus on are addition and multiplication, as with these two operations one can implement division, exponentiation, and so on. We illustrate these differences with the following examples.

Consider subtracting the numbers $\pi/100 \approx 0.031415927$ and $\pi/128 \approx 0.024543693$. Using a 16-bit Two's Complement representation with fixed exponent 2^{-8} , the two numbers would be approximated as $8 \cdot 2^{-8} = 0.03125$ and $6 \cdot 2^{-8} = 0.0234375$, respectively, with corresponding approximate relative errors of 0.528% and 4.507%. Subtracting the two fixed-point numbers would yield $2 \cdot 2^{-8} = 0.0078125$ with a relative error of approximately 13.7%. Repeating the same exercise with 16-bit half-precision IEEE-754 floating-point numbers, the two numbers would be approximated as $0.5024 \cdot 2^{-4}$ and $0.785 \cdot 2^{-5}$, respectively, with corresponding approximate relative errors of 0.0425% and 0.0308%. Subtracting the two numbers would involve repeatedly dividing the mantissa of the second number by 2 until its exponent matches the exponent of the first input, and then subtracting the mantissas. This process of multiplying or dividing by powers of two is known as variable bit shifting. After bit shifting, the second number would have value $0.3925 \cdot 2^{-4}$. As the two numbers would have the

same exponent, we would then subtract the mantissas, resulting in $0.1099 \cdot 2^{-4}$. However, the mantissa must be in the range $[0.5, 1)$, so we would multiply the mantissa by 2^3 and subtract 3 from the exponent, resulting in the final value of $0.879 \cdot 2^{-7} \approx 0.0068664551$ with a relative error of approximately 0.084%. Table Ia summarizes this example.

As a second example, consider multiplying the numbers $\pi \approx 3.1415927$ and $1/100 = 0.01$, whose product equals approximately 0.031415297 when using 64-bit floating-point arithmetic on a classical computer. Using a 16-bit Two's Complement representation with fixed exponent 2^{-8} , the two numbers would be approximated as $804 \cdot 2^{-8} = 3.140625$ and $3 \cdot 2^{-8} = 0.01171875$, with relative errors of 0.031% and 17.19%, respectively. Multiplying the two fixed-point numbers would yield $9 \cdot 2^{-8} = 0.03515625$ with a relative error of approximately 17.2%. Repeating the same operation with 16-bit half-precision IEEE-754 floating-point numbers, the two numbers would be approximated as $0.785 \cdot 2^2$ and $0.64 \cdot 2^{-6}$, respectively. Multiplying the two numbers would involve multiplying the mantissas, adding the exponents, and adjusting the exponent if the mantissa is not in the range $[0.5, 1)$. After multiplying the mantissas and adding the exponents, we would obtain $0.502 \cdot 2^{-4} \approx 0.042458227$ with a relative error of 0.04%. No adjusting of the mantissa and exponent was necessary as 0.502 falls in the allowed interval. Table Ib summarizes this example.

In both examples, we observed that fixed-point arithmetic suffered from loss of precision when dealing with small numbers, whereas floating-point was able to represent each number with maximal precision. Fixed-point arithmetic operations required fewer steps and essentially amounted to operations on integers with an implicit exponent. Floating-point arithmetic operations, on the other hand, were algorithmically more complex but yielded considerably more precise outputs. This increased relative precision was the main motivation in pursuing floating-point arithmetic in this work.

B. Quantum Fixed-Point Arithmetic

We recently introduced a suite of quantum algorithms for performing fixed-point arithmetic—addition, multiplication, and division—within the framework of gate-level quantum computation, using a QFT-based approach [24]. These algorithms leverage the encoding of quantum states as eigenstates of the discretized position operator, enabling efficient numerical operations within the quantum domain. In this representation, we denoted qubits containing a superposition of fixed-point values using bra-ket notation. We defined a quantum register $|a\rangle$ containing fixed-point values with n total qubits, and

Type	x	y	$x - y$	Type	x	y	$x \times y$
Analytical	$\pi/100$	$\pi/128$	$7\pi/3200$	Analytical	π	$1/100$	$\pi/100$
64-bit floating	0.03142	0.02454	0.006872	64-bit floating	3.141593	0.01000	0.031416
16-bit fixed	0.03125	0.02344	0.007813	16-bit fixed	3.140625	0.01172	0.035156
16-bit floating	0.03140	0.02454	0.006866	16-bit floating	3.140625	0.010002	0.031403
Fixed rel. error	0.528%	4.51%	13.7%	Fixed rel. error	0.03080%	17.19%	13.7%
Float. rel. error	0.0425%	0.0308%	0.084%	Float. rel. error	0.03080%	0.02136%	0.0425%

(a) Subtraction

(b) Multiplication

TABLE I: Summary of classical arithmetic examples.

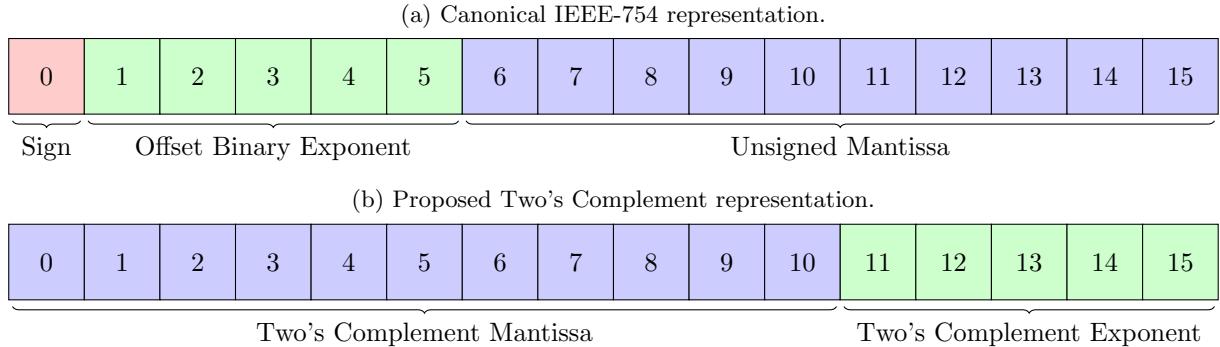


FIG. 1: Comparison between canonical IEEE-754 encoding (a) and floating-point encoding in this work (b).

f qubits after the binary point as follows.

$$|a\rangle = |a_{n-1}a_{n-2}\dots a_f.a_{f-1}\dots a_0\rangle \equiv \left| \sum_{k=0}^{n-1} a_k 2^{k-f} \right\rangle \quad (1)$$

We referred to such a register as a (n, f) unsigned fixed-point register. Note that constant fixed-point numbers (including integers) are denoted without the ket $|\cdot\rangle$. Since Two's Complement signed fixed-point arithmetic is equivalent to unsigned fixed-point arithmetic, we refer to signed fixed-point registers using the same notation, and recycle the same operations as in the unsigned case. Given a signed fixed-point register $|a\rangle$, the definition is nearly identical to that of Eq. 1, with a slight modification to include negative numbers, which is detailed in the following equation.

$$|a\rangle \equiv \left| -a_{n-1} 2^{n-1-f} + \sum_{k=0}^{n-2} a_k 2^{k-f} \right\rangle \quad (2)$$

The addition algorithm was designed in a general form to support both signed and unsigned integers. This circuit remains relatively compact due to its linear structure and modest gate requirements. We denoted the in-place addition operation involving fixed-point numbers as $\text{Add}(\cdot)$. For example,

$$\text{Add}(|a\rangle, c): |a\rangle \rightarrow |a + c\rangle \quad (3)$$

denotes the in-place addition of the fixed-point register with state $|a\rangle$ and a fixed-point constant c , resulting in a

new state $|a + c\rangle$. Similarly,

$$\text{Add}(|a\rangle, |b\rangle): |a, b\rangle \rightarrow |a + b, b\rangle \quad (4)$$

denotes the in-place addition of fixed-point registers with states $|a\rangle$ and $|b\rangle$, ending in respective states $|a + b\rangle$ and $|b\rangle$.

For multiplication, we extended previous techniques, notably those presented in [4], by computing the full product prior to rounding, achieving an exact computation of the product when compared with classical fixed-point products. However, this exactness comes at the cost of increased resource usage, particularly in the form of additional ancilla qubits. Note that these ancillae are not wasted, as they are strategically re-used across different steps of the computation to optimize resource efficiency. The multiplication algorithm employs a fused three-register architecture, involving doubly controlled rotation gates, which considerably increases the circuit's overall gate count compared to the addition operation.

We denoted the in-place fused multiplication-addition (FMA) of three fixed-point numbers as $\text{FMA}(\cdot)$. For example,

$$\text{FMA}(|a\rangle, |b\rangle, |c\rangle): |a, b, c\rangle \rightarrow |a + b \cdot c, b, c\rangle \quad (5)$$

denotes the in-place fused multiplication-addition of fixed-point registers with states $|a\rangle$, $|b\rangle$, and $|c\rangle$, resulting in respective states $|a + b \cdot c\rangle$, $|b\rangle$, and $|c\rangle$.

To implement division, we introduced a reciprocal operation based on Newton-Raphson's root-finding method.

While the resulting algorithm is inherently approximate rather than exact, it exhibits strong convergence properties, especially when initialized with a well-chosen starting point. This makes the method practically viable for a range of fixed-point applications, despite the iterative nature of the procedure. We also implemented a basic Negate(\cdot) gate that computes the two's complement – effectively the negative of each element in a superposition of integers. This was accomplished by first applying an \hat{X} gate to each qubit to perform bitwise inversion, followed by an increment operation using the Add(\cdot) gate to complete the negation. We include algorithmic description of the negation operation in Alg. 2.

Together, these algorithms represent a foundational step toward building scalable arithmetic components for quantum processors, particularly those that operate on fixed-point representations within the QFT paradigm. We implemented the above operations using the Clifford gates, namely basic Pauli $\hat{X}, \hat{Y}, \hat{Z}$ gates, Hadamard \hat{H} gates, and z -rotation $\hat{R}(\theta)$ gates, along with their controlled and doubly-controlled variants. These gates admit the following basic definitions.

$$\begin{aligned}\hat{X} &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, & \hat{Y} &= \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, & \hat{Z} &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \\ \hat{H} &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, & \hat{R}(\theta) &= \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}\end{aligned}\quad (6)$$

We use the above-described fixed-point addition and multiplication operations extensively in our proposed implementation of floating-point arithmetic, as the primitive operations used when performing arithmetic on the mantissa and exponent parts of the floating-point numbers.

III. ALGORITHM

A. Notational Conventions

We denote the adjoint of an operator with superscripted \dagger . For example, Add^\dagger denotes the adjoint of the addition operator, which results in subtraction of the arguments. In the rest of this work, we make extensive use of singly and doubly controlled gates. We denote these gates by prepending C to the name of the gate, as is common in the literature. For example, the Toffoli gate

$$CC\hat{X}|a, b, c\rangle \quad (7)$$

denotes applying an \hat{X} to $|c\rangle$ gate that is doubly controlled by qubits $|a\rangle$ and $|b\rangle$. In addition, we add a line over a qubit to denote applying an \hat{X} gate to the qubit, then performing a controlled gate with the qubit, and then applying another \hat{X} gate to flip the qubit back to its original state. For example,

$$CC\hat{X}|a, \bar{b}, c\rangle \quad (8)$$

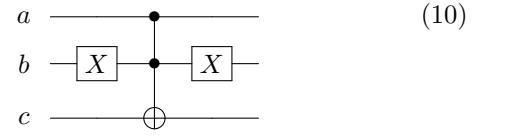
denotes applying an \hat{X} gate to $|b\rangle$, applying an \hat{X} gate to $|c\rangle$ that is double controlled by qubits $|a\rangle$ and $|b\rangle$,

and then applying an \hat{X} gate to $|b\rangle$ to flip it back to its original state. This notation is meant to resemble the logical negation operation that is common in digital circuit design because this operation effectively amounts to controlling by the inverse of the corresponding qubit.

Throughout this work, we omit the name of a gate in place of its arithmetic operation, such as $|a + b\rangle$ instead of $\text{Add}(|a\rangle, |b\rangle)$. We also omit the ‘ C s’ denoting controlled variants and instead append $c.b.$, which is shorthand for *controlled by*, followed by the qubits by which we control the gate. Additionally, we will denote controlling by the logical inverse using the same convention, with the line over the qubit to be inverted and controlled by. For example, the statement

$$\hat{X}|c\rangle \text{ c.b. } |a, \bar{b}\rangle \quad (9)$$

is equivalent to (8). The quantum circuit describing (8) (hence, (9)) is the following.



B. Resetting Ancilla Qubits

One problem when implementing floating-point operations on quantum hardware is the reliance on ancilla qubits that serve as “scratch space” and whose values we discard after an operation. The issue with ancilla qubits arises from the constraint that the ancillas start in the $|0\rangle$ or ground state. However, once an operation is performed, these ancilla qubits are not necessarily in the ground state and are often entangled with other qubits. For example, let us consider a three-qubit system, with the least significant (rightmost) qubit as an ancilla qubit. The system starts with an initial state

$$|\psi_0\rangle = |000\rangle. \quad (11)$$

Applying a Hadamard gate to the most significant qubit would yield the following state.

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |00\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |100\rangle) \quad (12)$$

Then, if we perform an operation (e.g., multiplication) that entangles the ancilla register with other qubit registers such that we obtain the following state:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|001\rangle + |110\rangle) \quad (13)$$

The ancilla qubit is hence in a uniformly superposed state, with its value dependent on the rest of the superposition. Measuring the ancilla qubit in order to reset the qubit would collapse the superposition to one of the

two states with probability $1/2$, hence destroying the information stored in the entangled registers. This would defeat the purpose of using a quantum computer for basic operations, since it would prevent taking advantage of quantum parallelism when performing computations.

Following conventional approaches, there would be two alternatives to overcome this problem: to reverse the computation involving the ancilla qubits, or to simply discard the entangled ancilla qubits and use new ancilla qubits that are known to be at ground state. While reversing the computation is possible, in principle, for all possible unitary operations on a quantum computer, doing so can prove difficult if one wants to preserve the non-ancilla state after applying one of these transformations. Also, this approach would add immensely to the overall gate cost. On the other hand, having an endless pool of ancilla qubits is practical only for small, simple problems. For example, as we showed in [24], each multiplication of fixed-point registers with F qubits after the decimal point would require F new ancilla qubits, which would be completely impractical for problems involving many multiplications.

In light of these limitations of conventional approaches, in this work, we propose an approach in which we apply a Hadamard gate to each ancilla qubit prior to measurement. This effectively creates two trajectories for storing the main qubits' information, which is useful for decoupling the ancilla qubits from the rest of the wavefunction. Note that this is similar to the separating trajectories observed when a coupled cavity-transmon system is driven by a microwave field in the presence of a dispersive interaction, with the transmon initially prepared as a state on the equator of its Bloch sphere [31]. Following our proposed approach, if we measure a value of 1 , we apply an \hat{X} gate to force the ancilla qubit to its ground state $|0\rangle$. To see this in practice, we continue with the example of Eqs. (11) to (13). We first apply a Hadamard gate to the ancilla qubit in eqn (13), resulting in the following state:

$$\begin{aligned} |\psi_3\rangle &= \frac{1}{\sqrt{2}}|00\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &\quad + \frac{1}{\sqrt{2}}|11\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \end{aligned} \quad (14)$$

The ancilla qubit is now in a superposed state that is independent of the states in the original superposition, which allows us to factorize and rewrite the state as follows.

$$\begin{aligned} |\psi_3\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \otimes \frac{1}{\sqrt{2}}|0\rangle \\ &\quad - \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \otimes \frac{1}{\sqrt{2}}|1\rangle \end{aligned} \quad (15)$$

Therefore, no matter which value we obtain when measuring and resetting the ancilla qubit, we always preserve the original states along with the desired entangled outputs, while possibly picking up a relative phase factor of $e^{i\pi} = -1$. At this point, we would measure the ancilla qubit, resulting in a value of $|0\rangle$ with probability

```

Registers: input/output  $|q\rangle = |q_{n-1} \dots q_0\rangle$ ; input  $|s\rangle = |s_{m-1} \dots s_0\rangle$ ; ancillae  $|a\rangle = |a_3 a_2 a_1 a_0\rangle$ .
1:  $|a_3\rangle \leftarrow |q_{n-1}\rangle$                                  $\triangleright$  Copy (store sign bit)
2:  $|q\rangle \leftarrow |-q\rangle$  c.b.  $|a_3\rangle$                        $\triangleright$  CNegate (abs. value)
3: for  $k \leftarrow (m-2)$  to  $0$  do
4:    $d \leftarrow 2^k$ 
5:    $|a_1\rangle \leftarrow \hat{X}|0\rangle$  c.b.  $|s_k, \overline{s_{n-1}}\rangle$        $\triangleright CC\hat{X} \circ$  Reset
6:   if  $d < n$  then
7:     for  $l \leftarrow 0$  to  $(n-d-1)$  do
8:        $|q_l, q_{l+d}\rangle \leftarrow |q_{l+d}, q_l\rangle$  c.b.  $|a_1\rangle$        $\triangleright CSwap$ 
9:     end for
10:   end if
11:   for  $l \leftarrow \max(n-d, 0)$  to  $n-1$  do
12:      $|a_0\rangle \leftarrow |q_l\rangle$                                  $\triangleright$  Copy
13:      $|q_l\rangle \leftarrow \hat{X}|q_l\rangle$  c.b.  $|a_0, a_1\rangle$            $\triangleright CC\hat{X}$ 
14:   end for
15: end for
16:  $|a_2\rangle \leftarrow |s_{n-1}\rangle$                                  $\triangleright$  Copy (store sign bit)
17:  $|s\rangle \leftarrow |-s\rangle$                                       $\triangleright$  Negate
18: for  $k \leftarrow (m-1)$  to  $0$  do
19:    $d \leftarrow 2^k$ 
20:    $|a_1\rangle \leftarrow \hat{X}|0\rangle$  c.b.  $|s_k, a_2\rangle$            $\triangleright CC\hat{X} \circ$  Reset
21:   if  $d < n$  then
22:     for  $l \leftarrow (n-1)$  to  $d$  do
23:        $|q_l, q_{l-d}\rangle \leftarrow |q_{l-d}, q_l\rangle$  c.b.  $|a_1\rangle$        $\triangleright CSwap$ 
24:     end for
25:   end if
26:   for  $l \leftarrow \min(d-1, n-1)$  to  $0$  do
27:      $|a_0\rangle \leftarrow |q_l\rangle$                                  $\triangleright$  Copy
28:      $|q_l\rangle \leftarrow \hat{X}|q_l\rangle$  c.b.  $|a_0, a_1\rangle$            $\triangleright CC\hat{X}$ 
29:   end for
30: end for
31:  $|s\rangle \leftarrow |-s\rangle$                                       $\triangleright$  Negate (undo neg.)
32:  $|q\rangle \leftarrow |-q\rangle$  c.b.  $|a_1\rangle$                        $\triangleright$  CNegate (undo abs.)

```

ALG. 1: Proposed algorithm for in-place bit shifting of $|q\rangle$ by $|s\rangle$.

$1/2$, requiring no further action, or in a value of $|1\rangle$ with probability $1/2$, requiring the application of an \hat{X} gate to set the qubit to $|0\rangle$. We would then obtain one of the following final states, with the minus sign corresponding to measuring $|1\rangle$ in the ancilla qubit.

$$|\psi_4\rangle = \frac{1}{\sqrt{2}}(|110\rangle \pm |000\rangle) \quad (16)$$

The ancilla qubit is now in the ground state, as desired, and the rest of the superposition remains intact. This process can then be generalized to any number of ancilla qubits, wherein each ancilla qubit would be reset individually, either sequentially or in parallel.

C. Bit Shifting

In order to implement floating-point arithmetic on quantum computers, we require an operator $\text{Shift}(\cdot, n)$ that shifts its input to the right by n places when n is positive, shifts to the left by $-n$ places when n is negative, and leaves the input unchanged when n is 0. When

ALG. 2: Proposed algorithm for the negation of Two's Complement integral registers. Negation of fixed-point registers is equivalent to negating the register as if it were integral.

the input is unsigned, we introduce 0 values as we shift the register. When the input is signed, the behavior is nearly identical, with the only difference being that a shift to the right fills with values equal to the most significant qubit prior to shifting. For example, if we want to shift a quantum n -bit string representing an unsigned number $|a\rangle = |a_{n-1}a_{n-2}\dots a_0\rangle$ to the right by 1 place, then $\text{Shift}(\cdot, +1)$ must carry out the following operation.

$$\text{Shift}(|a\rangle, +1): |a_{n-1} \dots a_0\rangle \rightarrow |0 \ a_{n-1} \dots a_1\rangle \quad (17)$$

Similarly, a shift to the left by 1 using $\text{Shift}(\cdot, -1)$ should carry out the following operation.

$$\text{Shift}(|a\rangle, -1): |a_{n-1} \dots a_0\rangle \rightarrow |a_{n-2} \dots a_0\ 0\rangle \quad (18)$$

Finally, when the input is signed and encoded using a Two's Complement representation, then Shift($\cdot, +1$) should carry out the following operation.

$$\text{Shift}(|a\rangle, +1): |a_{n-1} \dots a_0\rangle \rightarrow |a_{n-1}a_{n-1} \dots a_1\rangle \quad (19)$$

Our approach to implement the $\text{Shift}(\cdot, n)$ operation is detailed in Alg. 1. Note that the shifting works when the shift is a superposition of integers $|n\rangle$. To summarize how this approach works, we first note that because a shift is akin to multiplication by 2^n , if we express $n = \sum_{k=0}^{m-1} n_k 2^k$, then, due to the properties of exponentiation, we can decompose the shift by n into shifts by powers of 2, as can be seen in the following equation.

$$2^n = 2^{\sum_{k=0}^{m-1} n_k 2^k} = \prod_{k=0}^{m-1} 2^{n_k 2^k} \quad (20)$$

Then, for each power of 2, we swap the qubits from most significant to least significant if n contains that power of 2. This is achieved with controlled swaps. We take special care to account for negative inputs when the output is signed, resulting in a second pass on the negated input. The copies to ancilla qubits and negations, controlled or uncontrolled, are merely housekeeping steps to ensure that the correct fill value is used. Note that this is meant to be a proof-of-concept and more efficient ways of performing shifts over superpositions of shift values could be found.

D. Setting Exponent to Zero

Another requirement for implementing floating-point arithmetic is setting the exponent to $|0\rangle$ when the man-

Registers: input/output $|q\rangle = |q^e, q^m\rangle$ where exponent $|q^e\rangle = |q_{e-1}^e \dots q_0^e\rangle$ and mantissa $|q^m\rangle = |q_{m-1}^m \dots q_0^m\rangle$; ancillae $|a\rangle = |a_1 a_0\rangle$.

- 1: $|a\rangle \leftarrow |0\rangle$ ▷ Reset
- 2: **for** $k \leftarrow (m-1)$ **to** 0 **do**
- 3: $|a_1\rangle \leftarrow \hat{X}|0\rangle$ c.b. $|q_k^m, \overline{a_0}\rangle$ ▷ $CC\hat{X} \circ$ Reset
- 4: $|a_0\rangle \leftarrow \hat{X}|a_0\rangle$ c.b. $|q_k^m, a_1\rangle$ ▷ $CC\hat{X}$
- 5: **end for**
- 6: **for** $k \leftarrow (e-1)$ **to** 0 **do**
- 7: $|a_1\rangle \leftarrow \hat{X}|0\rangle$ c.b. $|q_k^e, \overline{a_0}\rangle$ ▷ $CC\hat{X} \circ$ Reset
- 8: $|q_k^e\rangle \leftarrow \hat{X}|q_k^e\rangle$ c.b. $|a_1\rangle$ ▷ $CC\hat{X}$
- 9: **end for**

ALG. 3: ZeroExp: Proposed algorithm for setting exponent to $|0\rangle$ when mantissa is zero.

tissa is equal to $|0\rangle$. This procedure requires two ancilla qubits $|a_1 a_0\rangle$. We use ancilla $|a_0\rangle$ to keep track of whether the mantissa is non-zero and ancilla $|a_1\rangle$ for temporary calculations. We determine whether the mantissa is non-zero by scanning the mantissa qubits, whereby at each step, we reset $|a_1\rangle$, invert $|a_1\rangle$ if $|a_0\rangle$ is zero and the mantissa qubit is non-zero, and invert $|a_0\rangle$ if $|a_1\rangle$ is active and the mantissa qubit is also active. We then scan the exponent qubits, whereby at each step, we reset $|a_1\rangle$, invert $|a_1\rangle$ if the exponent qubit is active and $|a_0\rangle$ is inactive, and then we invert the exponent qubit if $|a_1\rangle$ is active. Algorithm 3 describes this procedure in more detail.

E. Floating-Point Encoding

We choose to represent superpositions of floating point values by separating each floating point number into a Two's Complement signed integer exponent and a Two's Complement $(f + 1, f)$ signed fixed-point mantissa, denoted with superscripts e and m , respectively. For example, given a prescribed exponent width of 5 and a mantissa width of 11 for register $|a\rangle$,

$$|a\rangle = |a^e a^m\rangle = |a_{15} a_{14} \dots a_{11} a_{10} \dots a_0\rangle, \quad (21)$$

where exponent $|a^e\rangle = |a_{15} \dots a_{11}\rangle$ and mantissa $|a^m\rangle = |a_{10} \dots a_0\rangle$. Note that this encoding does not exactly mimic IEEE-754 and was chosen because it builds on our previous work on Two's Complement signed fixed-point encodings [24]. Fig. 1 illustrates the difference between the IEEE-754 convention and our similar encoding. Extending our work to One's Complement numbers is a trivial exercise and would result in operations with similar complexity, meaning that our approach could be used to exactly mimic IEEE-754.

F. Floating-Point Multiplication

For convenience, multiplication is not done in-place as for the case of fixed-point numbers [24] due to the oth-

Registers: inputs $|q\rangle = |q^e, q^m\rangle = |q_{e-1}^e \dots q_0^e, q_{m-1}^m \dots q_0^m\rangle$ and $|r\rangle = |r^e, r^m\rangle$; output $|s^e, s^m\rangle$; and ancillae $|a\rangle = |a_{n-1}a_{n-2} \dots a_0\rangle$, where $n = \max(m, 7)$.

Let $|s^{m,m-1}\rangle := |s^m a_{n-1} \dots a_{n-m+1}\rangle$, $|s^{m,1}\rangle := |s^m a_{n-1}\rangle$, and $|a^s\rangle := |a_{n-4}a_{n-5}a_{n-6}a_{n-7}\rangle$.

- 1: $|s^{m,m-1}\rangle \leftarrow |s^{m,m-1} + q^m \cdot r^m\rangle$ \triangleright FMA
- 2: $|a_{n-2}\rangle \leftarrow |s_0^m\rangle$ \triangleright Copy
- 3: $|s^{m,1}\rangle \leftarrow |-s^{m,1}\rangle$ c.b. $|a_{n-2}\rangle$ \triangleright C Negate (abs. value)
- 4: $|a_{n-3}\rangle \leftarrow |a_{n-2}\rangle$ \triangleright Copy
- 5: $|a_{n-3}\rangle \leftarrow \hat{X}|a_{n-3}\rangle$
- 6: $|s^{m,1}\rangle \leftarrow \text{Shift}(|s^{m,1}\rangle, |a_{n-3}\rangle, |a^s\rangle)$ \triangleright Shift if $|q^m| < 0.5$
- 7: $|s^e\rangle \leftarrow |s^e + a_{n-3}\rangle$ \triangleright Adjust q^e with Add
- 8: $|s^{m,1}\rangle \leftarrow |-s^{m,1}\rangle$ c.b. $|a_{n-2}\rangle$ \triangleright C Negate (undo abs. value)
- 9: $|a_{n-1}a_{n-2}\rangle \leftarrow |0\rangle$ \triangleright Reset
- Let** $|s^{1,e}\rangle := |a_{n-1}s^e\rangle$.
- 10: $|a_{n-1}\rangle \leftarrow \hat{X}|a_{n-1}\rangle$ c.b. $|s_0^e\rangle$ \triangleright $C\hat{X}$ (extend sign)
- 11: $|s^{1,e}\rangle \leftarrow |s^{1,e} + q^e\rangle$ \triangleright Add
- 12: $|s^{1,e}\rangle \leftarrow |s^{1,e} + r^e\rangle$ \triangleright Add
- 13: $|a_{n-2}\rangle \leftarrow \hat{X}|a_{n-2}\rangle$ c.b. $|a_{n-1}, \overline{s_0^e}\rangle$ \triangleright $CC\hat{X}$
- 14: **for** $k \leftarrow (m-1)$ **to** 0 **do** \triangleright Underflow correction
- 15: $|a_{n-1}\rangle \leftarrow |s_k^m\rangle$ \triangleright Copy
- 16: $|s_k^m\rangle \leftarrow \hat{X}|s_k^m\rangle$ c.b. $|a_{n-1}, a_{n-2}\rangle$ \triangleright $CC\hat{X}$
- 17: **end for**
- 18: $|s\rangle \leftarrow \text{ZeroExp}(|s\rangle, |a^s\rangle)$ \triangleright Zero exponent if mantissa is 0

ALG. 4: Proposed algorithm for multiplication of floating-point registers.

erwise required Shift operations. We denote the multiplication operation as $\text{Mult}(\cdot)$. We denote the mantissa and exponent of a number using superscripts m and e , respectively. Given two input numbers $a = a^m \cdot 2^{a^e}$ and $b = b^m \cdot 2^{b^e}$, the output number $c = c^m \cdot 2^{c^e}$ set to the product of a and b is given by

$$c^m \cdot 2^{c^e} = \text{Mult}(a, b) = a^m \cdot b^m \cdot 2^{a^e + b^e}. \quad (22)$$

Therefore, $c^m = a^m \cdot b^m$ and $c^e = a^e + b^e$. We use the fixed-point FMA operation to compute the mantissa and the Add operation to compute the exponent. We prepend an ancilla qubit to c^m to allow for overflow in the mantissa. If overflow occurs, then we shift the mantissa by 1 to the right using the Shift operation, and then add 1 to the exponent c^e , which is done by controlling using a copy of the overflow qubit in another ancilla qubit. We also prepend an ancilla qubit to the exponent to allow for underflow, in which case the output is set to 0. Finally, we set the exponent to 0 if the mantissa is 0 using the ZeroExp gate (Alg. 3). We present a full algorithmic description of the multiplication operation in Algorithm 4.

G. Floating-Point Addition

Floating-point addition is considerably more difficult to implement than multiplication because of possible cancellations in the mantissas, and because the inputs must

Registers: inputs $|q\rangle = |q^e, q^m\rangle = |q_{e-1}^e \dots q_0^e, q_{m-1}^m \dots q_0^m\rangle$ and $|r\rangle = |r^e, r^m\rangle$; output $|s^e, s^m\rangle$; and ancillae $|a\rangle = |a_7a_6 \dots a_0\rangle$.

Let $|s^{m,1}\rangle := |s^m a_2\rangle$, $|s^{m,2}\rangle := |a_1 s^{m,1}\rangle$, and $|a'\rangle := |a_6 a_5 a_4 a_3\rangle$.

- 1: $|a\rangle \leftarrow |0\rangle$ \triangleright Reset
- 2: $|s^e\rangle \leftarrow |q^e\rangle$ \triangleright Copy
- 3: $|s^e\rangle \leftarrow |s^e - r^e\rangle$ \triangleright Add^\dagger
- 4: $|a_0\rangle \leftarrow \hat{X}|a_0\rangle$ c.b. $|s_0^e\rangle$ \triangleright $C\hat{X}$
- 5: $|s^m\rangle \leftarrow |0\rangle$ \triangleright Reset
- 6: **for** $k \leftarrow (m-1)$ **to** 0 **do**
- 7: $|s_k^m\rangle \leftarrow \hat{X}|s_k^m\rangle$ c.b. $|a_0, q_k^m\rangle$ \triangleright $CC\hat{X}$
- 8: **end for**
- 9: $|a_0\rangle \leftarrow \hat{X}|a_0\rangle$
- 10: **for** $k \leftarrow (m-1)$ **to** 0 **do**
- 11: $|s_k^m\rangle \leftarrow \hat{X}|s_k^m\rangle$ c.b. $|a_0, r_k^m\rangle$ \triangleright $CC\hat{X}$
- 12: **end for**
- 13: $|s^e\rangle \leftarrow |-s^e\rangle$ c.b. $|\overline{a_0}\rangle$ \triangleright C Negate
- 14: $|s^{m,1}\rangle \leftarrow \text{Shift}(|s^{m,1}\rangle, |s^e\rangle, |a'\rangle)$ \triangleright Alg. 1
- 15: $|a_1\rangle \leftarrow \hat{X}|a_1\rangle$ c.b. $|s_0^m\rangle$ \triangleright $C\hat{X}$
- 16: $|s^{m,2}\rangle \leftarrow |s^{m,2} + q^m\rangle$ c.b. $|a_0\rangle$ \triangleright $C\text{Add}$
- 17: $|s^{m,2}\rangle \leftarrow |s^{m,2} + r^m\rangle$ c.b. $|\overline{a_0}\rangle$ \triangleright $C\text{Add}$
- 18: $|a_7\rangle \leftarrow |a_1\rangle$ \triangleright Copy
- 19: $|s^{m,2}\rangle \leftarrow |-s^{m,2}\rangle$ c.b. $|a_7\rangle$ \triangleright Negate
- 20: $|s^e\rangle \leftarrow |1\rangle$ \triangleright Copy
- 21: $|a'\rangle \leftarrow |0\rangle$ \triangleright Reset
- 22: **for** $k \leftarrow m$ **to** 0 **do**
- 23: $|s^e\rangle \leftarrow |s^e + 1\rangle$ c.b. $|\overline{s_k^{m,1}}, \overline{a_7}\rangle$ \triangleright Copy
- 24: $|a_6\rangle \leftarrow |a_7\rangle$
- 25: $|a_7\rangle \leftarrow \hat{X}|a_7\rangle$ c.b. $|s_k^{m,1}, \overline{a_6}\rangle$
- 26: **end for**
- 27: $|s^{m,2}\rangle \leftarrow \text{Shift}(|s^{m,2}\rangle, s^e, a')$ \triangleright Alg. 1
- 28: $|s^m\rangle \leftarrow |-s^m\rangle$ c.b. $|a_7\rangle$ \triangleright C Negate
- 29: $|s^e\rangle \leftarrow |s^e + r^e\rangle$ c.b. $|a_0\rangle$ \triangleright $C\text{Add}$
- 30: $|s^e\rangle \leftarrow |s^e + q^e\rangle$ c.b. $|\overline{a_0}\rangle$ \triangleright $C\text{Add}$
- 31: $|s\rangle \leftarrow \text{ZeroExp}(|s\rangle, |a\rangle)$ \triangleright Alg. 3

ALG. 5: Proposed algorithm for addition of floating-point registers.

be shifted prior to addition to ensure that the addition is performed under the same exponent value. Let us assume that we want to add two floating point registers $|a\rangle = |a^e a^m\rangle$ and $|b\rangle = |b^e b^m\rangle$ and to store the output in register $|c\rangle = |c^e c^m\rangle$. We start by setting the output register $|c\rangle$ to the ground state using the reset approach presented in this work and copy $|a^e\rangle$ to $|c^e\rangle$ using controlled \hat{X} gates. We then subtract the exponents $|c^e\rangle$ and $|b^e\rangle$ using $\text{Add}^\dagger(|c^e\rangle, |b^e\rangle)$. We then use this value to copy the appropriate input mantissa to the output mantissa using doubly controlled \hat{X} gates. If the difference is negative, then we copy the second input; otherwise, we copy the first input. We then set the output exponent to its absolute value by using controlled negation, and shift the output mantissa to the right by using $\text{Shift}(|c^m\rangle, |c^e\rangle)$. After shifting, we add the two mantissas with the operation $\text{Add}(|c^m\rangle, |b^m\rangle)$. Then, we scan the mantissa from most to least significant qubit, subtracting 1 from an ancilla integer counter for every zero qubit until the first

non-zero qubit is encountered. Because this operation must work for all possible combinations of inputs, we implement the scanning procedure by controlling with an ancilla qubit that indicates whether a non-zero qubit has been encountered. Using this ancilla counter, we shift the output mantissa as appropriate and then add this counter to the exponent of the output, which yields the correct value in the output register.

H. Higher-Order Floating-Point Operations

We can combine addition and multiplication to implement other operations. For example, we can use the same Newton's Method approach that we used for fixed-point division in [24] to carry out floating-point division. We can also implement exponentiation for small arguments by expressing the exponential function in terms of its Taylor series, and then factorizing the polynomial approximation using Horner's Method for Polynomial Evaluation, as summarized in the following equation for polynomial approximation order N .

$$\begin{aligned} \exp(x) &\approx \sum_{k=0}^N \frac{x^k}{k!} \\ &\approx 1 + x \cdot \left(1 + \frac{x}{2} \cdot \left(1 + \dots \left(1 + \frac{x}{N}\right)\right)\right) \end{aligned} \quad (23)$$

For sufficiently high N (on the order of 10-12), this approximation is accurate to the precision of the floating-point representation. This technique could be extended to larger arguments by storing a look-up table of values (classically) for each power of 2 in a range. We can also extend this approach to the trigonometric functions $\sin(x)$ and $\cos(x)$, and to variants $\sin(\pi x)$ and $\cos(\pi x)$. Once we implement these functions, we can derive the inverse trigonometric functions $\arcsin(x)$ and $\arccos(x)$, other trigonometric functions like $\tan(x)$, the natural logarithm $\log(x)$, and so on.

IV. METHODS

As a proof-of-concept, we used our floating-point framework to replicate the analysis of the fixed-point reciprocal operation that was presented in [24], all in simulation on a classical computer (Apple M2 SoC with 8 cores and 24 GiB RAM, 2022). The reciprocal operation approximates the reciprocal using Newton's Method for root finding. The main difference from the fixed-point case is in how we obtain the initial guess: Given an input, we simply set the mantissa to ± 1 and negate the input's exponent. We carried out the same analysis for 10-, 12-, 14-, 16- and 18-qubit floating point registers.

Using our proposed floating-point encoding scheme, we also replicated the numerical solution of the system of ordinary differential equations that we had previously sim-

ulated using fixed-point registers in [24]:

$$\frac{d\mathbf{u}}{dt} = \frac{d}{dt} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} \quad (24)$$

with initial condition $\mathbf{u}(0^+)^T = \mathbf{u}_0^T = [0 \ -1]$. The analytical solution of this system is given by

$$\mathbf{u}(t) = - \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix} \Theta(t), \quad (25)$$

where $\Theta(t)$ is the Heaviside step function. We solve the system using 14-, 16-, 18-, and 20-qubit floating-point registers. For each register width, we employed time steps of 2^{-2} s, 2^{-3} s, 2^{-4} s, and 2^{-5} s for the evolution of the system using the trapezoidal rule for numerical integration. For each time step, we simulated the system for approximately 2π s, corresponding to the period of the analytical solution in Eq. (25). For time step Δt , we approximated the analytic solution of the system with the following explicit trapezoidal rule update step.

$$\mathbf{u}_{k+1} \leftarrow \frac{1}{1 + \frac{\Delta t^2}{4}} \begin{bmatrix} 1 - \frac{\Delta t^2}{4} & \Delta t \\ -\Delta t & 1 - \frac{\Delta t^2}{4} \end{bmatrix} \mathbf{u}_k, \quad (26)$$

In order to assess the performance of our proposed methods, we counted the total number of Clifford and rotation operations when simulating these circuits. Note that we performed this without decomposing Toffoli gates as is done to evaluate Clifford+T gate-based approaches. In those cases, authors often use T gate count and depth as metrics for the complexity of quantum circuits, in addition to Toffoli depth, Toffoli count, and qubit count. For QFT-based approaches like ours, common performance metrics instead include QFT gate count and non-Clifford (rotation) gate count, excluding those used in the QFT routine. In some instances, the evaluation metrics used for QFT-based designs can be related to those of Clifford+T-based designs [32]. Here, we chose to report the raw count of elementary gates and their controlled variants because counting only QFT operations can obscure the true complexity of the circuit, since an n -qubit QFT circuit requires $n(n-1) \div 2$ non-Clifford controlled phase gates.

V. RESULTS

Fig. 2 shows the signed relative error distributions of the output of the Recip operation for different register widths and for 100 samples from a Gaussian distribution with mean of 0 and standard deviation of 5. We discarded samples from the Gaussian distribution whose reciprocals are not representable, such as $1/0$. We observed an exponential decay in the error as we increased the register widths, which matched our expectations. Fig. 3 illustrates the gate resources required for implementing the reciprocal operation across various qubit register sizes.

Qubits		Depth	1-Qubit Operations				2-Qubit Operations			3-Qubit Operations		
$ a\rangle$	$ \psi\rangle$		H	X	P_z	Reset	CX	CP_z	Swap	CCX	CCP_z	CSwap
8	40	21 716	8 777	3 572	404	3 119	2 575	13 040	2 704	1 900	2 830	840
10	48	33 686	10 645	4 474	424	3 505	3 139	21 100	3 468	2 280	7 420	1 360
12	56	46 728	13 197	5 889	515	4 551	4 135	29 739	3 930	3 230	11 430	1 660
14	65	67 958	15 206	6 951	535	4 999	4 699	42 091	4 694	3 610	22 480	2 340
16	75	96 497	17 276	8 013	555	5 507	5 263	57 091	5 458	3 990	39 410	3 020
18	84	121 843	20 569	9 908	646	7 054	6 699	71 046	6 748	5 380	50 920	3 360
20	93	150 860	24 102	11 963	737	8 761	8 295	87 149	7 250	6 930	64 450	3 700

TABLE II: Summary of Recip resource utilization for every register width. The first, second, and third columns list the number of qubits per register, the total number of qubits in the wavefunction, and the circuit depth, respectively. The remaining columns list the number of operations, in order of increasing gate width.

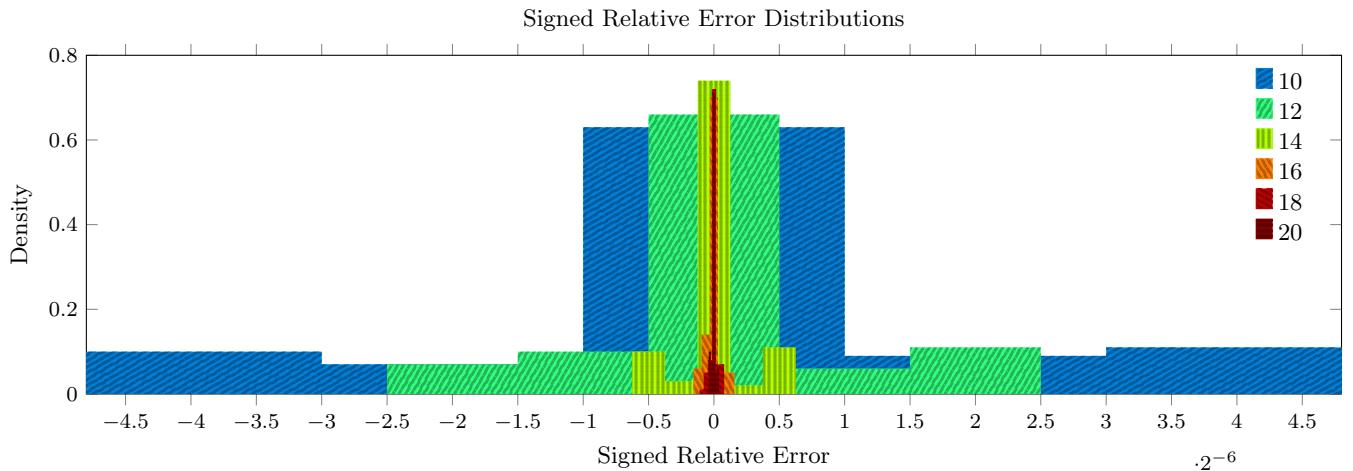


FIG. 2: Distribution of signed error of output of reciprocal gate when compared with expected output. We assessed performance using register widths of 10, 12, 14, 16, 18, and 20 qubits, with corresponding mantissa widths of 6, 7, 9, 11, 12, and 13 and exponent widths 4, 5, 5, 5, 6, and 7. Each reciprocal gate was tested with 100 samples from a normal distribution $\mathcal{N}(0, 5)$ and for 10 Newton iterations. We discarded samples that did not have a representable reciprocal, such as 0.

The total number of 1-, 2-, and 3-qubit gates increased approximately linearly with respect to the width of the register. The proportion by operation type also varies: there are relatively fewer Reset gates in the 1-qubit category, fewer CNOT gates in the 2-qubit category, and fewer CSWAP gates in the 3-qubit category. Among these, the number of Toffoli (CCX) gates offers a more realistic and informative measure of circuit complexity, as further elaborated in the discussion.

Fig. 4 shows the results of the numerical experiment, with each column corresponding to a different time step. The top, middle, and bottom rows show the evolution of component u_1 , the evolution of component u_2 , and the relative error in the ℓ_2 sense, respectively, for the prescribed register widths. The relative error in each case exhibits two regimes: in the first regime, limited precision in the mantissa dominates, whereas in the second

regime, the trapezoidal rule approximation error dominates. Fig. 5 shows the total number of operations used in each ODE simulation. We observed a roughly linear relationship between the register widths and the number of operations, except for rotations P_z , which increased quadratically due to our use of QFT-based fixed-point arithmetic. As the time step is halved, we observe a corresponding doubling in the number of operations, as expected.

VI. DISCUSSION

While fixed-point arithmetic is often simpler to implement on quantum hardware than floating-point arithmetic, the latter provides several key advantages, including a wider dynamic range, improved control over relative

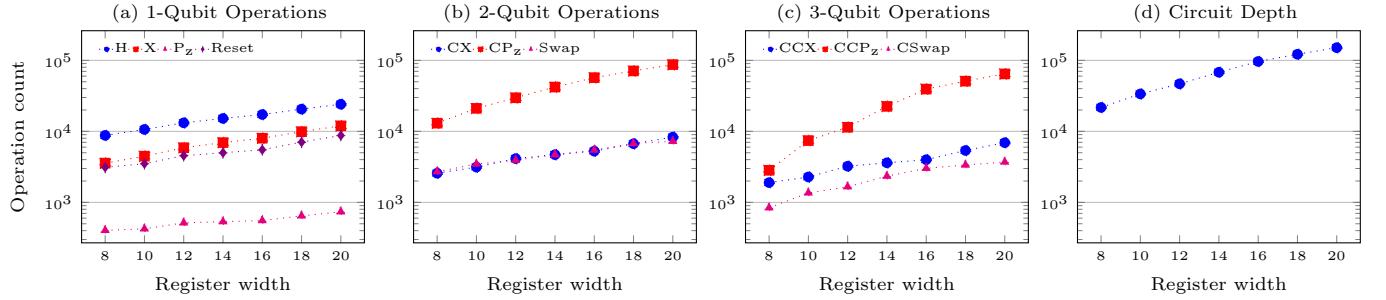


FIG. 3: Number of operations per Recip operation as a function of floating-point register width. Figs. (a)-(c) shows counts of 1-, 2-, and 3-qubit operations, respectively, as a function of register width, while Fig. (d) shows the estimate for the overall circuit depth growth as a function of register width.

error, and natural compatibility with QFT-based circuit components. These benefits make floating-point encoding particularly appealing for quantum algorithms that must operate over a broad range of magnitudes and require high precision. Additionally, floating-point designs often result in shallower circuits with fewer qubits, which is an important consideration for both near-term quantum devices and future fault-tolerant systems.

In this work, we developed a floating-point arithmetic framework that minimizes the number of required ancilla qubits. We accomplished this by leveraging our previously proposed arithmetic primitives [24] and introducing an ancilla reuse strategy to optimize quantum resource efficiency. In the context of division or reciprocation, our approach used 13 ancilla qubits for 20-qubit reciprocation. A 32-qubit single-precision example would require 23 ancilla qubits, a dramatic reduction from the thousands of qubits that were required with approaches proposed in previous work [13, 16].

Implementing floating-point multiplication was relatively straightforward, as it directly reused fixed-point addition and multiplication subroutines that we had previously developed. Floating-point addition, however, required additional circuit complexity. In fact, the mantissas must be aligned by shifting to a common exponent before addition, which introduced extra gate depth and an increased number of ancilla qubits. Our proposed floating-point encoding also produced significant improvements in numerical accuracy when compared with our fixed-point encoding. For example, we observed a mean error reduction from 2^{-4} to 2^{-10} in our reciprocal computation benchmarks, compared to our previous results for fixed-point arithmetic [24]. When solving the differential equation using fixed-point arithmetic [24], for time step $\Delta t = 2^{-4}$, the smallest relative error achieved was approximately 2^{-6} using fractional bits $f = 12$, corresponding to a register width of $2f + 1 = 25$ qubits. In contrast, as shown in Fig. 4k, the floating-point representation achieved a lower relative error of 2^{-8} using only ~ 20 qubits per register, demonstrating better precision with fewer resources.

Our approach has limitations that prevent its use on present-day quantum hardware. In particular, our approach depends on the use of doubly controlled single-qubit gates and singly controlled Swap gates, which would be difficult to implement efficiently on current quantum hardware. We anticipate, however, that as quantum architectures continue to evolve, these three-qubit operations will become more feasible, making our design increasingly practical. Furthermore, our current implementation does not account for the impact of noise or incorporate any form of error correction or mitigation. That said, given the structural similarity of our floating-point representation to classical formats, we expect that conventional error correction techniques will be able to be adapted with minimal modification to suit our quantum framework.

VII. CONCLUSION

We implemented novel quantum algorithms for efficient floating-point arithmetic and fixed-point arithmetic that optimizes resource to be more easily implemented on quantum hardware. Our framework could be used to exploit quantum parallelism to solve optimization problems in scientific computing in general. For example, it could enable one to develop a quantum algorithm to simulate (non-linear) ordinary differential equations like the Bloch Equation for all possible physical parameter combinations at once. Such a quantum advantage could be exploited to optimize hard-to-solve cost functions, such as pulse sequence optimization in magnetic resonance imaging (MRI) [33]. Another possible application is uncertainty quantification of design parameters using Monte Carlo sampling. Essentially, any problem in scientific computing that can be easily parallelized but whose parallelization can be expensive in terms of memory would be an ideal candidate for our floating-point approach using quantum computers.

Δt	Qubits		Depth	1-Qubit Operations				2-Qubit Operations			3-Qubit Operations		
	$ a\rangle$	$ \psi\rangle$		H	X	P_z	Reset	CX	CP_z	Swap	CCX	CCP_z	CSwap
2^{-2} s	8	48	68 436	27 872	10 195	1 560	9 160	8 684	46 696	8 424	6 032	3 640	2 392
	10	58	97 766	34 008	12 691	1 560	10 358	10 712	79 560	10 920	7 384	5 408	3 848
	12	68	129 123	41 600	16 539	1 872	13 220	13 676	111 384	12 792	10 088	7 384	4 680
	14	79	169 373	48 100	19 451	1 872	14 574	15 704	162 552	15 288	11 440	9 880	6 552
	16	91	215 551	54 756	22 363	1 872	16 084	17 732	225 368	17 784	12 792	12 792	8 424
	18	102	262 976	64 272	27 459	2 184	20 246	21 840	277 316	21 216	16 640	15 704	9 360
	20	113	315 601	74 412	32 971	2 496	24 824	26 364	336 388	23 192	20 904	18 928	10 296
2^{-3} s	8	48	134 236	54 672	19 995	3 060	17 960	17 034	91 596	16 524	11 832	7 140	4 692
	10	58	191 766	66 708	24 891	3 060	20 308	21 012	156 060	21 420	14 484	10 608	7 548
	12	68	253 273	81 600	32 439	3 672	25 920	26 826	218 484	25 092	19 788	14 484	9 180
	14	79	332 223	94 350	38 151	3 672	28 574	30 804	318 852	29 988	22 440	19 380	12 852
	16	91	422 801	107 406	43 863	3 672	31 534	34 782	442 068	34 884	25 092	25 092	16 524
	18	102	515 826	126 072	53 859	4 284	39 696	42 840	543 966	41 616	32 640	30 804	18 360
	20	113	619 051	145 962	64 671	4 896	48 674	51 714	659 838	45 492	41 004	37 128	20 196
2^{-4} s	8	48	265 836	108 272	39 595	6 060	35 560	33 734	181 396	32 724	23 432	14 140	9 292
	10	58	379 766	132 108	49 291	6 060	40 208	41 612	309 060	42 420	28 684	21 008	14 948
	12	68	501 573	161 600	64 239	7 272	51 320	53 126	432 684	49 692	39 188	28 684	18 180
	14	79	657 923	186 850	75 551	7 272	56 574	61 004	631 452	59 388	44 440	38 380	25 452
	16	91	837 301	212 706	86 863	7 272	62 434	68 882	875 468	69 084	49 692	49 692	32 724
	18	102	1 021 526	249 672	106 659	8 484	78 596	84 840	1 077 266	82 416	64 640	61 004	36 360
	20	113	1 225 951	289 062	128 071	9 696	96 374	102 414	1 306 738	90 092	81 204	73 528	39 996
2^{-5} s	8	48	531 668	216 544	79 187	12 120	71 112	67 468	362 792	65 448	46 864	28 280	18 584
	10	58	759 526	264 216	98 579	12 120	80 406	83 224	618 120	84 840	57 368	42 016	29 896
	12	68	1 003 139	323 200	128 475	14 544	102 628	106 252	865 368	99 384	78 376	57 368	36 360
	14	79	1 315 837	373 700	151 099	14 544	113 134	122 008	1 262 904	118 776	88 880	76 760	50 904
	16	91	1 674 591	425 412	173 723	14 544	124 852	137 764	1 750 936	138 168	99 384	99 384	65 448
	18	102	2 043 040	499 344	213 315	16 968	157 174	169 680	2 154 532	164 832	129 280	122 008	72 720
	20	113	2 451 889	578 124	256 139	19 392	192 728	204 828	2 613 476	180 184	162 408	147 056	79 992

TABLE III: Summary of resource utilization for every ODE simulation. The first, second, and third columns list the time integration step size, the number of qubits per register, and the total number of qubits in the wavefunction, respectively. The remaining columns list the number of operations, in order of increasing gate width.

ACKNOWLEDGEMENTS

This work was supported by the U.S. Department of Energy, Office of Science, National Quantum Informa-

tion Science Research Centers, Superconducting Quantum Materials and Systems Center (SQMS), under Contract No. 89243024CSC000002. Fermilab is managed by FermiForward Discovery Group, LCC, acting under Contract No. 89243024CSC000002.

[1] “Ieee standard for binary floating-point arithmetic,” *ANSI/IEEE Std 754-1985*, pp. 1–20, 1985.

[2] D. Goldberg, “What every computer scientist should know about floating-point arithmetic,” *ACM computing surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[3] R. Yates, “Fixed-point arithmetic: An introduction,” *Digital Signal Labs*, vol. 81, no. 83, p. 198, 2009.

[4] B. Zanger, C. B. Mendl, M. Schulz, and M. Schreiber, “Quantum algorithms for solving ordinary differential equations via classical integration methods,” *Quantum*, vol. 5, p. 502, 2021.

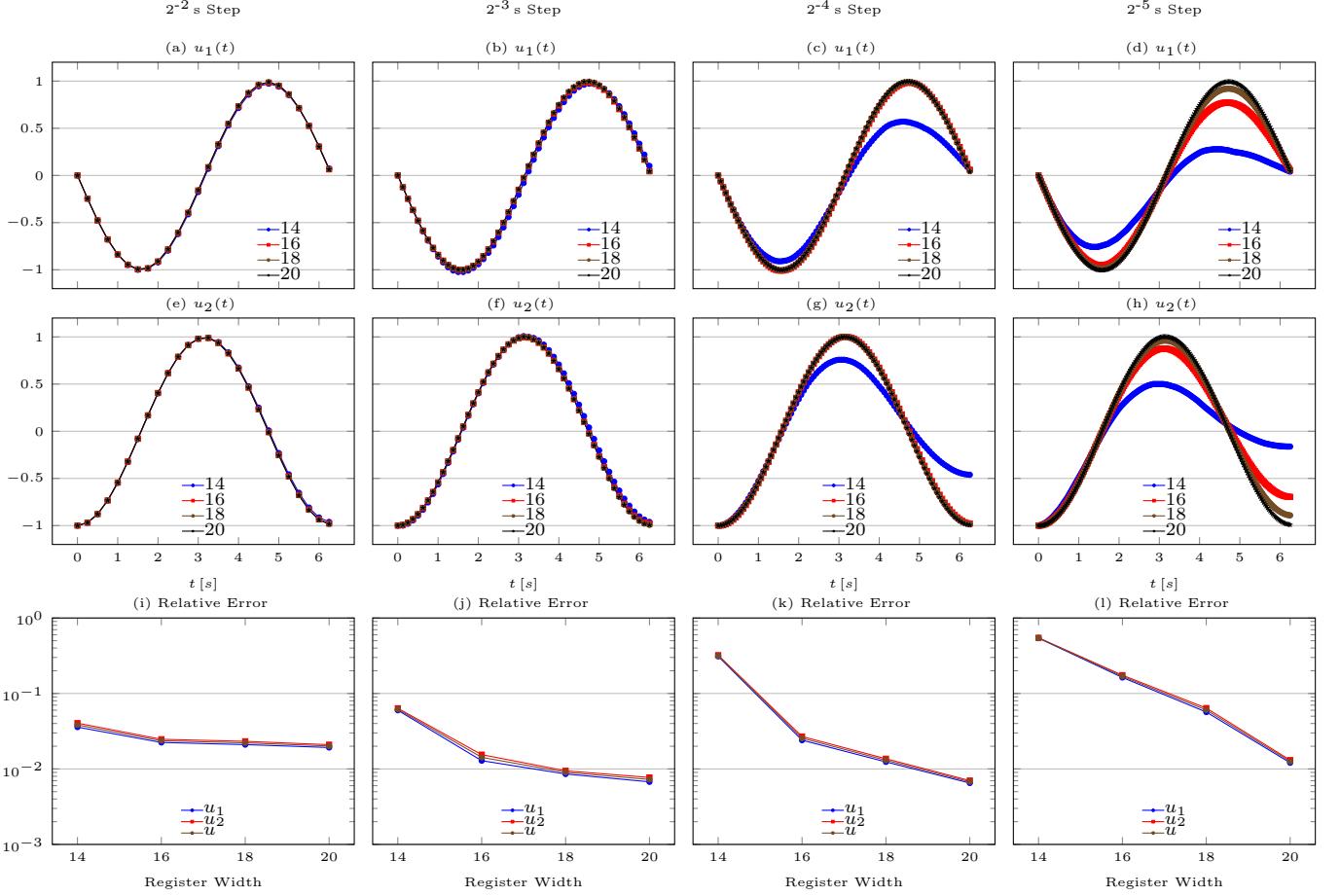


FIG. 4: Results from the solution of the tested system of ordinary differential equations for 14-, 16-, 18-, and 20-qubit registers. Each column in the figure corresponds to a different timestep Δt used when integrating the system. The top and middle rows show the evolution of u_1 and u_2 , respectively, for different Δt . Each data point is represented using a circular marker, resulting in a thicker appearance as Δt is refined. The bottom row shows the relative error with respect to the analytical solution as a function of register widths, for different Δt .

- [5] M. Nachtigal, H. Thapliyal, and N. Ranganathan, “Design of a reversible floating-point adder architecture,” in *2011 11th IEEE International Conference on Nanotechnology*. IEEE, 2011, pp. 451–456.
- [6] N. Wiebe and V. Kliuchnikov, “Floating point representations in quantum circuit synthesis,” *New Journal of Physics*, vol. 15, no. 9, p. 093041, 2013.
- [7] X. Peng, Q. Xu, T. Kato, Y. Yamanashi, N. Yoshikawa, A. Fujimaki, N. Takagi, K. Takagi, and M. Hidaka, “High-speed demonstration of bit-serial floating-point adders and multipliers using single-flux-quantum circuits,” *IEEE Transactions on Applied Superconductivity*, vol. 25, no. 3, pp. 1–6, 2014.
- [8] J. Jain and R. Agrawal, “Design and development of efficient reversible floating point arithmetic unit,” in *2015 Fifth International Conference on Communication Systems and Network Technologies*. IEEE, 2015, pp. 811–815.
- [9] T. Haener, M. Soeken, M. Roetteler, and K. M. Svore, “Quantum circuits for floating-point arithmetic,” in *International conference on reversible computation*. Springer, 2018, pp. 162–174.
- [10] A. Sanada, Y. Yamanashi, and N. Yoshikawa, “Study on single flux quantum floating-point divider based on goldschmidt’s algorithm,” *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 5, pp. 1–4, 2019.
- [11] R. Zhang, M. Xu, and D. Lu, “A generalized floating-point quantum representation of 2-d data and their applications,” *Quantum Information Processing*, vol. 19, no. 11, p. 390, 2020.
- [12] M. L. Rogers and R. L. Singleton Jr, “Floating-point calculations on a quantum annealer: Division and matrix inversion,” *Frontiers in Physics*, vol. 8, p. 265, 2020.
- [13] S. Gayathri, R. Kumar, S. Dhanalakshmi, G. Dooly, and D. B. Duraibabu, “T-count optimized quantum circuit designs for single-precision floating-point division,” *Electronics*, vol. 10, no. 6, p. 703, 2021.
- [14] M. K. Bhaskar, S. Hadfield, A. Papageorgiou, and I. Petras, “Quantum algorithms and circuits for scientific computing,” *arXiv preprint arXiv:1511.08253*, 2015.
- [15] Y. Cao, A. Papageorgiou, I. Petras, J. Traub, and S. Kais, “Quantum algorithm and circuit design solving the pois-

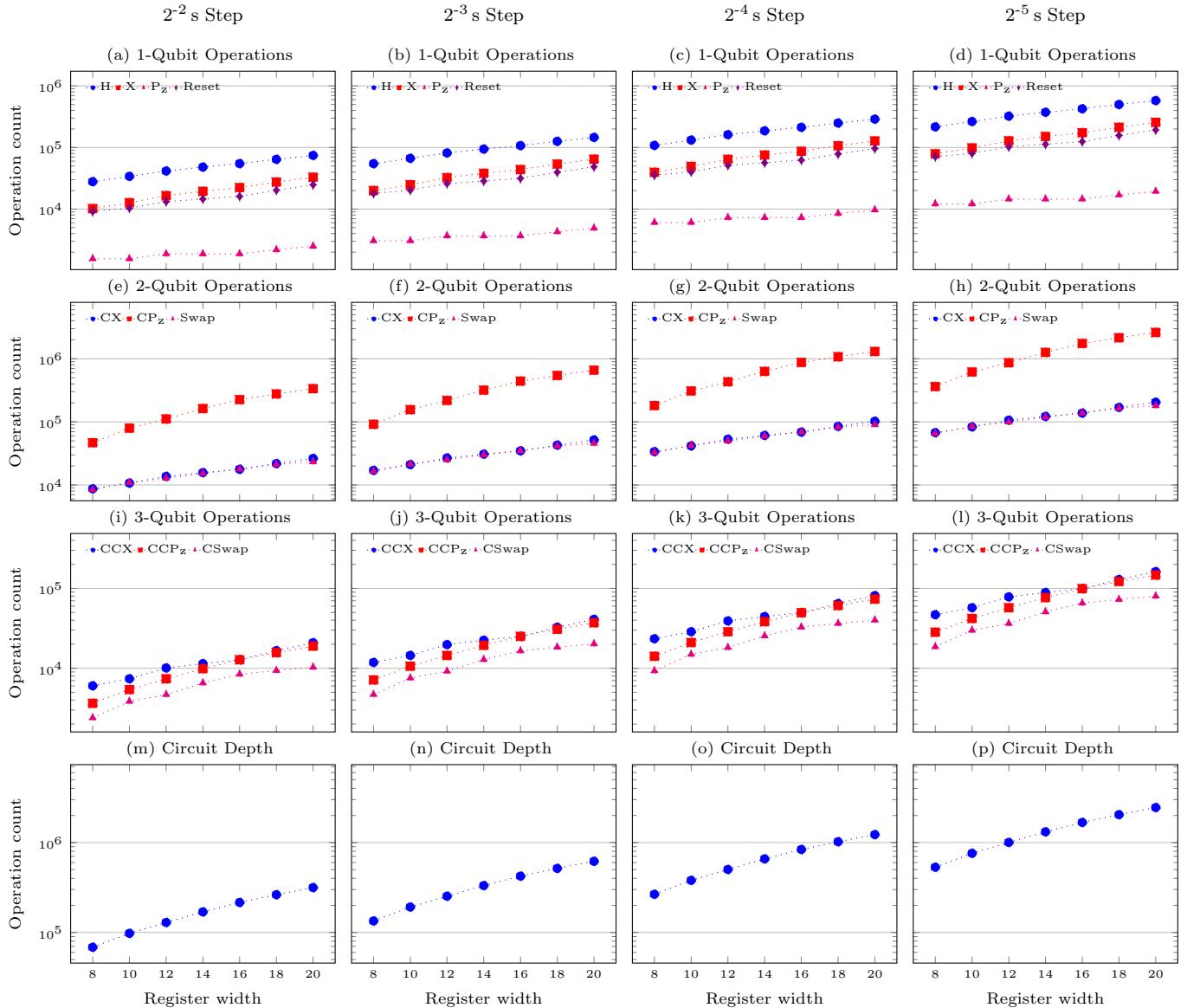


FIG. 5: Number of operations per simulation of the ordinary differential equation system of Eq. (24). We simulated the system for one period, while varying the step from 2^{-2} s to 2^{-5} s. Figs. (a)-(d) show count of 1-qubit operations for each step size, as a function of register width. Similarly, Figs. (e)-(h) and Figs. (i)-(l) show counts of 2- and 3-qubit operations, respectively, for each step size and as a function of register width. Figs. (m)-(p) show the overall circuit depth for each step size and as a function of register width.

son equation,” *New Journal of Physics*, vol. 15, no. 1, p. 013021, 2013.

[16] S. Gayathri, R. Kumar, S. Dhanalakshmi, and B. K. Kaushik, “T-count optimized quantum circuit for floating point addition and multiplication,” *Quantum Information Processing*, vol. 20, no. 11, p. 378, 2021.

[17] R. Kumar, M. Haghparast, and S. Dhanalakshmi, “A novel and efficient square root computation quantum circuit for floating-point standard,” *International Journal of Theoretical Physics*, vol. 61, no. 9, p. 234, 2022.

[18] R. Seidel, N. Tcholtchev, S. Bock, C. K.-U. Becker, and M. Hauswirth, “Efficient floating point arithmetic for quantum computers,” *IEEE Access*, vol. 10, pp. 72 400–72 415, 2022.

[19] S. Zhao, H. Li, G. Li, and X. Tang, “The implementation of the enhanced quantum floating-point adder,” *Modern Physics Letters A*, vol. 37, no. 26, p. 2250169, 2022.

[20] R. Steijl, “Floating-point arithmetic with consistent rounding on a quantum computer,” in *Quantum Information Science*, R. Steijl, Ed. Rijeka: IntechOpen, 2024, ch. 2. [Online]. Available: <https://doi.org/10.5772/intechopen.1005546>

[21] R. Das, A. Chattopadhyay, and H. Rahaman, “Optimizing quantum circuits for modular exponentiation,” in

2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID). IEEE, 2019, pp. 407–412.

[22] L. Ruiz-Perez and J. C. Garcia-Escartin, “Quantum arithmetic with the quantum fourier transform,” *Quantum Information Processing*, vol. 16, pp. 1–14, 2017.

[23] P. Atchade-Adelomou and S. Gonzalez, “Efficient quantum modular arithmetics for the isq era,” *arXiv preprint arXiv:2311.08555*, 2023.

[24] J. E. Cruz Serrallés, O. Ogunkoya, D. M. Kürkçüoğlu, N. Bornman, N. M. Tubman, S. Zorzetti, and R. Lattanzi, “A quantum approach for implementing fixed-point arithmetic in solving ordinary differential equations,” in 2024 IEEE International Conference on Quantum Computing and Engineering (QCE), vol. 1. IEEE, 2024, pp. 50–57.

[25] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, “A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 6, pp. 818–830, 2013.

[26] H.-K. Lau and M. B. Plenio, “Universal quantum computing with arbitrary continuous-variable encoding,” *Physical review letters*, vol. 117, no. 10, p. 100501, 2016.

[27] M. Weigold, J. Barzen, F. Leymann, and M. Salm, “Encoding patterns for quantum algorithms,” *IET Quantum Communication*, vol. 2, no. 4, pp. 141–152, 2021.

[28] ———, “Expanding data encoding patterns for quantum algorithms,” in 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C). IEEE, 2021, pp. 95–101.

[29] K. Korzekwa, Z. Puchała, M. Tomamichel, and K. Życzkowski, “Encoding classical information into quantum resources,” *IEEE Transactions on Information Theory*, vol. 68, no. 7, pp. 4518–4530, 2022.

[30] J. Gonzalez-Conde, T. W. Watts, P. Rodriguez-Grasa, and M. Sanz, “Efficient quantum amplitude encoding of polynomial functions,” *Quantum*, vol. 8, p. 1297, 2024.

[31] R.-S. Huang, “Strong coupling of a single photon to a superconducting qubit using circuit quantum electrodynamics,” *Nature*, vol. 431, no. 7005, pp. 162–167, 2004.

[32] A. Paler, “Quantum fourier addition simplified to toffoli addition,” *Physical Review A*, vol. 106, no. 4, p. 042444, 2022.

[33] S. P. Jordan, S. Hu, I. Rozada, D. F. McGivney, R. Boyacioglu, D. C. Jacob, S. Huang, M. Beverland, H. G. Katzgraber, M. Troyer, M. A. Griswold, and D. Ma, “Automated design of pulse sequences for magnetic resonance fingerprinting using physics-inspired optimization,” *Proceedings of the National Academy of Sciences*, vol. 118, no. 40, p. e2020516118, 2021. [Online]. Available: <https://www.pnas.org/doi/abs/10.1073/pnas.2020516118>