arXiv:1909.11719v1 [quant-ph] 25 Sep 2019

Understanding Quantum Control Processor Capabilities and Limitations through Circuit Characterization

Anastasiia Butko, George Michelogiannakis, Samuel Williams,
Costin Iancu, David Donofrio, John Shalf, Jonathan Carter, Irfan Siddiqi
Lawrence Berkeley National Laboratory
(abutko, mihelog, swwilliams, cciancu, ddonofrio, jshalf, jcarter, iasiddiqi)@lbl.gov

Abstract

Building usable quantum computers hinges on building a classical control hardware pipeline that is scalable, extensible, and provides real time response. The control processor part of this pipeline provides functionality to map between the high-level quantum programming languages and low-level pulse generation using Arbitrary Waveform Generators. In this paper, we discuss design alternatives with an emphasis on supporting intermediate-scale quantum devices, with $O(10^2)$ qubits. We introduce a methodology to assess the efficacy of a quantum ISA to encode quantum circuits. We use this methodology to evaluate several design points: RISC-like, vectors, and VLIW-like. We propose two quantum extensions to the broadly used open RISC-V ISA. Given the rapid rate of change in the quantum hardware pipeline, our open-source implementation provides a good starting point for design space experimentation, while our metrics can be independently used to guide design decisions.

1. Introduction

Quantum computing has the potential to deliver transformational science results for certain types of calculations [27] [34] [1]. Inspired by this potential and with the rising uncertainty of the classical computing performance scaling [26], quantum field gained an impressive boost of research and engineering efforts resulting in a growing number of quantum accelerators of different kinds [18, 22, 16, 25, 8, 17]. These are controlled by an ad-hoc combination of classical control electronics. The requirements for classical control electronics vary almost beyond recognition when going from one implementation technology to another [8, 3]. Yet one concern remains the same across all of them - control electronics has to meet stringent real time and sensitivity constraints that will dramatically grow with the increasing size of quantum accelerators.

The functionality of the classical control hardware can be partitioned as described in Figure 1. A "CPU Host" compiles the high-level quantum program into a sequence of instructions for a "Quantum Control Processor", QCP for short. The QCP translates these instructions into commands for an Arbitrary Waveform Generator (AWG). In turn the AWG controls the execution of the Quantum Processing Unit (QPU). Quantum operations (gates) are controlled by AWG pulses. Efforts to implement this pipeline have already started. Fu et al [13]

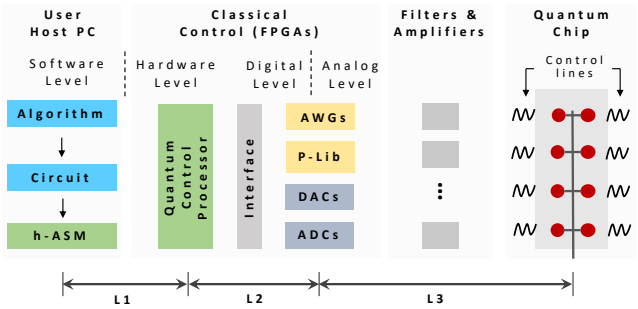


Figure 1: A high level view of a quantum system.

describe the functional requirements and timing constraints for an architecture similar to the system organization in Figure 1. They also propose a QCP design using a specialized Instruction Set Architecture (ISA) [12] to control a QPU with superconducting qubits. AWG hardware vendors (Tektronix, Keysight, Zurich Instruments) have started adding higher level processing capabilities to their devices for optimization and tighter timing constraints.

Overall, while many efforts are underway, little exists in terms of a regimented methodology required to define the ISA based on quantum accelerator requirements and circuit characteristics. The design criteria for conventional cores focus on metrics that are largely orthogonal to the requirements of large-scale quantum control systems. This work takes the first necessary steps toward evaluating the unique characteristics of the quantum computing control systems that will dictate the directions for future efforts in its architecture design and integration. This paper explores the QCP design space and its interfaces to the surrounding layers (high-level programming languages and AWGs) from a pragmatic functional and operational point of view. Our contributions are:

- We provide a methodology for the exploration of the ISA design space and argue that one design metric is the ability to efficiently encode quantum circuits. A quantum circuit is a description of the application in space (qubits) and time (QPU cycles) of quantum gates (operations). An optimality criteria is to develop encoding that minimize the number of instructions required to represent the application of gates to all qubits in one QPU cycle. This is likely to minimize the QCP cycles per QPU "cycles" (time stamp). The key observation is that the ratio between the number of addressable

qubits and the number of native gates supported by the QPU is a suitable ISA design guide. In Section 4 we introduce metrics that capture the "diversity" of gates applied by a quantum circuit in a "QPU cycle". We use synthetic circuits, as well as proper algorithms to derive a sample and examine four designs: basic RISC, eQASM [12] and the proposed QUASAR and quantum vector (qV) extensions.

- We propose and implement the QUAntum instruction Set ARchitecture (QUASAR) extension to the widely used open source RISC-V [31] processor. Instructions are derived to support the native gate set for superconducting qubits. We also demonstrate the ability to meet real-time gate cycle requirements for QPU control.
- Finally, we summarize the capabilities and limitations of the ISAs based on their encoding efficiency, generated program size and potential to satisfy timing constrains.

The rest of this paper is structured as follows. Section 2 provides an overview of the basic concepts of quantum computing theory, related system architecture and software stack. Section 4 introduces the requirements for quantum ISA induced by hardware and the proposed circuit characterization model. Also, we discusses the ISA design space and four alternatives for QPU control processor. Section 5 shows the results of the ISA analysis, comparative study and real algorithms use cases. This section summarizes the capabilities and limitations of the quantum ISAs. Section 7 concludes our work and proposes the potential directions for future work.

2. Background

In quantum computing, a qubit is the basic unit of quantum information. Physically, qubits are two-level quantum-mechanical systems, whose general quantum state is represented by a linear combination of two orthonormal basis states (basis vectors). The most common basis is the equivalent of the 0 and 1 values used for bits in classical information theory, respectively $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$. The generic qubit state is a superposition of the basis states, i.e. $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, with α and β complex amplitudes such as $|\alpha|^2 + |\beta|^2 = 1$.

Gates and Circuits The prevalent model of quantum computation is the circuit model introduced by [9], where information carried by qubits (wires) is modified by quantum gates that mathematically correspond to unitary operations. A complex square matrix U is **unitary** if its conjugate transpose U^* is also its inverse, i.e. $UU^* = U^*U = I$.

A set of quantum gates is *universal* if any computation (unitary transformation) can be approximated on any number of qubits to any precision when using only gates from the set. On the high-level programming side, languages provide logical gates that can operate on arbitrary qubits. On the hardware side, quantum processors expose a minimal set of native gates that constitute an universal set. Processors built from superconducting qubits usually provide a gate set consisting of single qubit rotations ($R_x(90)$, and $R_z(\theta)$) and two qubit *CNOT* gates. A *CNOT*, or controlled NOT gate, flips the target qubit *iff* the control qubit is $|1\rangle$.

Figure 4 shows an example circuit that applies single qubit and *CNOT* gates on four qubits and captures the space-time evolution of these gates. Gates are aligned in time and we refer to a time step as a QPU cycle. In the NISQ era we expect processors with $O(10^2)$ qubits but very few (3-5) native gates. Two qubit gates are hard to calibrate, and we do not expect a large increase in the types of native gates. Any arbitrary single qubit gate can be represented as $R_zR_xR_zR_xR_z$. With R_z gates as phase adjustment; this takes only two QPU cycles. Again, we do not expect large increases in the diversity of native single qubit gates. Extrapolating forward, the ratio of the number of qubits to native gates is likely only to increase. Thus, the description of gates on all qubits in one QPU cycle is likely to have regular, simple structure.

Quantum Algorithms and Their Circuits. There are very few major classes of quantum algorithms [23]: Shor's *quantum Fourier transform (QFT)* [27], Grover's algorithm for performing *quantum search* [14]. Others, such as HHL [15] usually employ a composition of basic algorithms (QFT) with state preparation. Besides proper algorithms, practitioners run randomized circuits for hardware characterization using Randomized Benchmarking [20] or Quantum Volume [4], or circuits with randomization for error mitigation using Randomized Compiling [30]. It is therefore important to evaluate hardware on the ability to execute randomized circuits for these protocols.

Gates, Pulses and AWGs High-level languages allow circuit descriptions with logical arbitrary gates applied to arbitrary qubits. Compilers translate these high-level gates into equivalent sequences of single- and two-qubit native gates, then map them onto the physical hardware. Each gate is performed by a sequence of AWG control pulses. Pulses associated with each gate are determined at calibration time and there is a one-to-one association between gates and their pulse sequences. Typical quantum gate latency is $\approx 10ns$, with the fastest gate determining the real time requirements for the QCP.

3. ISA Design Requirements

The quantum ISA design requirements are driven by two forces: physical implementation of the quantum device and the related control electronics, and quantum circuit structure. We want to physically "address" all the QPU qubits fast, while efficiently encoding quantum programs.

3.1. Induced by physical implementation

We distinguish the following set of requirements induced by the physical implementation:

1. The quantum ISA is required to support *at least the native QPU gates*. The development of the native quantum gate

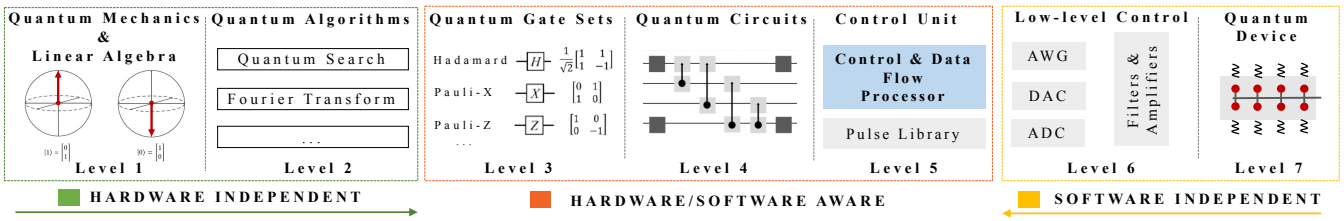


Figure 2: Quantum Abstraction Levels: languages are hardware independent, low-level hardware (e.g. AWG backends) are software/circuit independent. The QCP needs to enable efficient mapping of circuits onto the full hardware pipeline.

set is still an active research topic, but as explained, it is likely to offer only a few operations. The current set for superconducting qubits contains $R_x(90)$, arbitrary R_z rotations, two-qubit CNOT and measurement. Thus, we expect to need to represent only a small number of opcodes for the foreseeable future, encoded in a few bits.

2. The quantum ISA is required to address all qubits. This number will define how scalable the control system is. In terms of instruction specification, the number of qubits defines how many bits are reserved for qubit addressing, e.g. immediate value, register mask, indexing, etc.
3. The QCP is required to maintain gate application timings. Unlike typical program execution, where the ALU operations follow one another each clock cycle, in a quantum program, gate operations may be applied asynchronously. Inadequate gate application can lead to erroneous computation results. The time stamp at which the gate is applied needs to have a dedicated bit set and can be encoded in an absolute or a relative manner. The clock discretization defines how many bits are required and for how long the circuit can be run.
4. In addition to quantum instructions, the ISA is required to support basic control and data flow operations such as conditional branching, memory loads and stores as well as basic arithmetical-logical operations to calculate jumps, bypass register values and allow explicit bit manipulations. Since in quantum most of the computation complexity is hidden inside the qubits, the accessible data are reduced to 1's and 0's. Thus, the ISA needs to be able to efficiently operate on a bit-granularity data.

3.2. Induced by circuit structure

A quantum circuit illustrates quantum computation as a sequence of quantum gates applied on a quantum register. At any given time, none or multiple gates can be applied concurrently. The quantum ISA is required to efficiently encode the quantum circuit.

In order to quantify how the circuit structure affects the ISA design, we propose a circuit characterization model that consists of three parameters: *gate density*, *gate diversity*, and *distribution balance*. Unlike other circuit characterization models (coherence, hamiltonian, gate characterization [32]), the proposed model is designed to evaluate the efficiency of the quantum circuit encoding and its comparison across multiple ISA design alternatives.

Gate density: Depending on the algorithm, a quantum circuit may have a different number of gates applied on available qubits per single *time stamp (ts)*. *Gate density* represents these characteristics as they vary from low to high density in range from 0.0 to 1.0 and is calculated as follows:

$$\text{Gate density}_{ts} = \frac{\text{Number of gates}_{ts}}{\text{Number of qubits}_{ts}}$$

Gate diversity: This metric indicates how many *unique types* of gates are applied per chosen time stamp. The lowest level of gate diversity refers to one single type. The highest level of gate diversity implies that the number of gate types equals the total number of gates applied at a time stamp:

$$\text{Gate diversity}_{ts} = [1 \,..\, \text{Number of gates}_{ts}]$$

Distribution balance: This metric shows how many gates per each type are applied on qubits per time stamp. We specify two major categories, i.e. balanced and unbalanced distribution. The unbalanced distribution can be further characterized by the degree of asymmetry expressed with the extreme differences in the number of gates per each type. The number of possible variations grows with the number of gates and increasing gate diversity. Explicit characterization of the gate distribution is out of the scope of this paper, however some of the results touch the basis paving the way for future work.

$$\text{Distribution}_{ts} = \begin{cases} \textit{Balanced}, & \text{if equal number of gates per type} \\ \textit{Unbalanced}, & \text{otherwise} \end{cases}$$

The proposed set of parameters enables circuit characterization per each time stamp in order to determine computation patterns and evaluate the efficiency of quantum circuit encoding by different ISA designs. Section 4.2 explains in details how the proposed characterization model is used to analyze specific ISA encoding capabilities. The proposed model primarily targets static behaviour, however using the notion of the ISA implementation we demonstrate how the proposed model is used to address the dynamic behaviour of the circuit execution over time (see Section 5.3).

4. ISA Design Space

4.1. Specification Alternatives

There are many ways to encode the aforementioned quantum instruction requirements. In this section, we consider four different approaches to the ISA.

4.1.1. RISC In general purpose architectures, there are different mechanisms for processors to interface with external I/O, such as hardware interrupts, port-mapped I/O (PMIO) that uses special instructions, and memory-mapped I/O (MMIO) that assigns memory addresses to I/O devices. A hardware interrupt is device-initiated and unidirectional; it can be used only to indicate that the quantum device requires processor attention. This method is strictly limited to the quantum measurement operation.

I/O operations have historically caused performance concerns and in fact can also slow memory access in case memory and I/O operations share common buses. This is because the peripheral device is usually much slower than main memory. In some architectures, port-mapped I/O operates via a dedicated I/O bus, eliminating interference with memory accesses. However, memory-mapped I/O has been more popular because it discards the extra complexity that port I/O brings and also a CPU requires less internal logic and is thus cheaper and faster. This makes memory-mapped I/O a natural fit for Reduced Instruction Set Computer (RISC) architectures. In addition, memory-mapped I/O enables all memory operations in the ISA to be used for I/O devices, whereas in port-mapped I/O, only the limited subset that interacts with I/O devices can be used, often providing only simple load-and-store operations between CPU registers and I/O ports.

RISC ISAs have a small set of simple and general instructions where the memory is accessed through loads and stores. RISC-V is an open ISA that has been designed to support computer architecture research and education [31] but has been also adopted by industry [11]. The philosophy behind RISC-V promotes open collaboration, base set simplicity, and special-purpose extensibility. We use this design alternative as a baseline to emphasize the limitations of the common approaches and motivate the need of specialized quantum-oriented extensions.

Figure 3 (a) shows an example of the assembly code for base 32-bit RISC-V (RV32i) ISA applying a single-qubit gate operation using the MMIO communication method. It includes loading of a QPU address to the register (`QPU_ADDR`), and consequently loading the time stamp `TIME_STAMP`, gate type `GATE_TYPE` and qubit ID `QUBIT_ID` following by a store instruction. The Quantum Status Register (QSR) is used as a control interface.

4.1.2. eQASM eQASM [12] is a 32-bit instruction set that targets a seven-qubit superconducting quantum processor with a two-dimensional square lattice connectivity [29]. eQASM ISA follows most of the requirements induced by hardware implementation described in Section 3.1. It consists of four

```
la      x1, QPU_ADDR;
addi    x2, 0, TIME_STAMP;
sw      x2, 0x0(x1);
addi    x2, 0, GATE_TYPE;
sw      x2, 0x5(x1);
addi    x2, 0, QUBIT_ID;
sw      x2, 0x9(x1);
addi    x2, 0, QSR;
sw      x2, 0x20(x1);
```

(a) Base RV32I

```
qwait   CYCLES;
smis    s0, {QUBIT_LIST};
x90     s0 | NOP;
```

(b) eQASM [6]

```
tsi     TIME_STAMP;
x90i    QUBIT_ID;
```

(c) QUASAR

```
tsi     TIME_STAMP;
ld      gpr1, QUBIT_ADDR;
vld     vreg1, gpr1;
vqqi    vreg1, GATE_TYPE;
```

(d) Double-indexed Vector

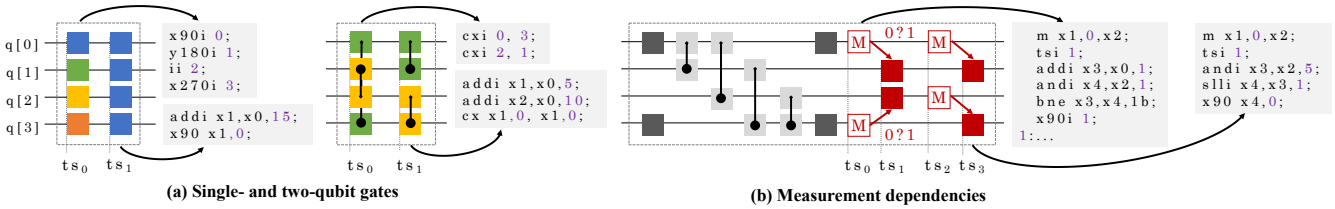
Figure 3: Circuit programming examples using four ISAs: not-extended 32-bit RISC-V (RV32) via memory-mapped IO communication, eQASM [6], the proposed QUASAR extension to RV32 and the double-indexed vector extension.

major groups of instructions: control flow, data flow, arithmetical and logical operations, and quantum-related instructions. Quantum-related instructions includes quantum `WAIT` of immediate and register formats to control timings, instructions that specify target qubit for gate application (`SMIS` for single-qubit ID list and `SMIT` for two-qubit pair list) and gate application instruction that can fit two types of gate.

Per each gate in a single time stamp, the assembly code requires one first specify the qubit ID list, and second, apply a quantum operation on the specified qubits. Figure 3 (b) shows an example of the assembly code composed of three 32-bit instructions. An additional gate will require one more target specification instruction (`SMIS`). `NOP` will be replaced with the second gate operation with no cost in instruction count. Two-qubit gate operations encoding is similar and requires the same number of instructions. The only difference is related to the 16-bit connectivity mask in the `SMIT` instruction while the `SMIS` instruction uses the 7-bit mask.

4.1.3. QUASAR QUASAR is an extension to the RISC-V ISA designed to support missing quantum instructions of four categories: (i) single-qubit gates, (ii) two-qubit gates, (iii) measurement and (iv) timing control. The proposed encoding focus on maximizing the number of qubits that can be addressed by a single instruction. The extension consists of 31 instructions using a fixed 32-bit length that requires a new instruction encoding space. QUASAR is a *greenfield* extension that uses the operation code (opcode) prefix `'10'` thereby conflicting with the standard compressed instruction extension `'C'`.

Each instruction from these four groups can be encoded using immediate or register-based format. Both addressing types supports up to 512 qubits with a 9-bit immediate format or a 32-bit mask and 4-bit immediate offset. *Measurement* instruction uses the same encoding strategy, but also specifies the destination register `rd` address to place the measured values for further use. QUASAR provides the *time stamp* instructions `ts` to progress over time with the support of the special purpose register that contains the current value of the quantum circuit time stamp. A detailed description of the QUASAR



(a) Single- and two-qubit gates

(b) Measurement dependencies

Figure 4: Typical quantum circuits representation and the associated QUASAR assembly code. (a) Circuits are composed of single- or two-qubits gates. Different colors show different types of the gates. Black arrows indicate source-target direction in two-qubits gates. (b) Circuit consists of an algorithm subroutine shown in gray and a measurement-dependent subroutine shown in red. Red arrows indicate data dependencies.

ISA extension is publicly available for the community [6].

Figure 3 (c) shows an example of the single-qubit circuit code. Since both parameters, i.e. qubit ID and gate operation type are encoded within a single instruction, the fragment requires only two instructions.

4.1.4. Double Index Quantum Vector (qV) Quantum circuits express both SIMD (same gate, multiple qubits) and MIMD (multiple gates, multiple qubits) parallelism. Whereas scalar ISAs can easily execute such circuits, in this paper, we introduce several novel innovations that leverage and extend vector instruction sets to address both forms of parallelism. Although the 'vector length' is hard-coded in modern SIMD ISAs (SSE, AVX, VSX, etc...), traditional vector ISAs encode both a *maximum vector length* (MVL) and a operational vector length (VL) in two special control registers [21, 7]. The former is a read-only register that informs programs of the implementation-specific maximum vector length (programs read this register to determine how they stride through memory). The latter is a read-write register that allows programs to restrict execution to a subset of a vector. Historically, one views vector register elements as the data in question. In such a world, vector element i might represent the state of array element $i+j$ where j is some offset in memory. Unfortunately, such mappings are not appropriate for the quantum world as one cannot store a qubit in a digital vector. Rather, we use each vector element to encode the index to an arbitrary qubit. Thus, each vector register is essentially a list of qubit indices.

In addition to the traditional vector operations, e.g. load, store, add, etc., we add two new instruction classes for executing quantum gates. The first (`vqqi`) is a two-register format with an `op` field to specify a common gate. Such an instruction takes the two list of qubits (two vectors of indices) and applies the same operator (gate) to all of them. In essence, this expresses the traditional SIMD-style approach to parallelism with the caveat that it is only applied to a subset of qubits in the system. The second form (`vqqg`) is a three-register format. As before, we have two lists of qubits (vector registers encode indices of qubits) but augment this with a third register that encodes the gate to be applied (`gate[vreg3[]] qubits[vreg1[i]], qubits[vreg2[i]]`). This novel form that combines two qubit index lists with a quantum gate list allowing us to easily express MIMD parallelism (any combination of qubits and gates). Masking can be effected with either a `nop` field in the third register or enumerating qubits from 1 (qubit_1 is the first qubit) and treating any index of zero in the first register as a `nop`.

Vector architectures have a rich design space. In addition to the maximum vector length (number of gates that can be executed by a single instruction), there is the initiation rate (number of gates that can be executed per cycle), the number of vector registers (limited by the RISC-V instruction length), and the size of each vector element. One should tailor the MVL to match the typical SIMD or MIMD parallelism available in a quantum circuit (maximum number of gates in a window that address different qubits). Finally, one can select the vector element size (VES) used for qubit indices to support computers of varying numbers of qubits (8b, 16b, and 32b elements enable quantum computers of 255, 64K, and 4 billion qubits respectively).

An astute reader will note that execution of a quantum circuit will nominally iterate on three (`vld, vld, vqqi`) or four (`vld, vld, vld, vqqg`) instructions depending on whether one is attempting to exploit SIMD or MIMD parallelism (one must load the vectors of indices of the qubits, the vector of gates, and execute the gates).

4.2. Design Space Evaluation

Given the specifics of different ISA designs, we show how circuit characteristics affect the encoding on an example of the proposed QUASAR extension.

In Figure 4(a), two fragments of a quantum circuit are composed of different single- and two-qubit gates. In the case of a single-qubit gate circuit, the difference can be in the rotation axis or the rotation angle, e.g. x90 vs. y180. At time ts_0, all of the applied gates are different that can be expressed using the immediate instruction format resulting in four instructions. At time ts_1, all of the applied gates are the same and using the mask format the circuit fragment can be expressed in two instructions instead of four. In the case of the two-qubit gate circuit, the difference can be in the distance between two qubits or in target-control qubit direction. Due to the requirement to load the mask to the register prior to the quantum instruction execution, use of the immediate format

results in two instructions, while mask requires one additional load.

In Figure 4(b), we show a fragment of the surface error correction code circuit that contains measurement dependencies. At time ts_0, measurement gates are applied on qubits $q[0]$ and $q[3]$. After the measurement is completed, the values are expected in the register $x2$. Depending on the values in $x2$, we apply gates on qubits $q[1]$ and $q[2]$ at time stamp ts_1. To check the condition we need at least three instructions prior to the quantum gate instruction, i.e. `addi` to load the expected value, `andi` to pick the bits that correspond to the target qubits and `beq` for conditional branching. This set of instructions varies depending on the position of the target qubits, i.e. lower 12 bits, upper 20 bits or both as well as the expected values, i.e. `0` or `1`. However, the circuit may contain measurement dependencies with similar patterns as it is shown at time ts_2 and ts_3. This pattern contains two similarities: the distance between the measured and dependent qubits, and the type of the conditionally applied gates. If such a pattern exists in the circuit, the mask format can be used instead of the conditional branching resulting in a significant reduction of the instructions per circuit fragment.

Thus, the number of instructions per a circuit fragment can significantly vary, especially in the large scale realistic scenario. It is important to quantify these variations and determine a set of rules for efficient code generation. Moreover, the number of instructions per time stamp directly affects whether or not the control processor will meet timing in order to deliver the gates for the next time stamp cycle.

5. Results

5.1. Experimental Setup

5.1.1. Evaluation metrics Our study targets two evaluation metrics: *encoding efficiency* and *execution time*. The encoding shows how many `Bytes` is required to represent a circuit or a circuit fragment when using a specific ISA. It includes both characteristics related to the ISA design, the instruction count and the data to be moved from the memory to the registers. Moreover, the encoding efficiency allows us to evaluate the program size and predict the memory requirements for efficient code execution.

Execution time is a typical performance metric in classical computing research. In our study, we use this metric to evaluate the timing constrains satisfaction. To satisfy the constrain, per every time stamp, the execution time of the related subcircuit is required to be less then the threshold value. We chose the threshold value of 20ns that corresponds to the duration of a typical single-qubit gate in superconducting technology. We estimate the execution time by calculating the Instruction Per Cycle (IPC) rate and applying the knowledge of the processor architecture implementation described in Section 5.1.3 running at different speeds.

5.1.2. Benchmarks We evaluate the capabilities and limitations of the quantum control processors using the proposed circuit characterization model. Our experiments include two types of circuits: *synthetic circuits* and circuits based on the *real algorithms*. The synthetic circuits are built based on the three circuit characteristics, i.e. gate density, gate diversity and distribution balance. They are composed of single- and two-qubit gates representing the variety of possible combinations in a 3-dimensional space. Synthetic circuits allow us asses ISA characteristics in the most critical conditions.

We chose two circuits from existing algorithms that are the major building blocks of the most real quantum algorithms: *Quantum Fourier Transform (QFT)* [24] and *Grover's operator* [24].

QFT is the quantum analogue of the inverse discrete Fourier transform. It is a key building block for many existing quantum algorithms, such as Shor's algorithm and quantum phase estimation [24]. The QFT circuit is composed of two quantum logic gates, i.e. the Hadamard gate (`H`) and the Controlled Phase gate (`CF`). These two types of gates are decomposed into a subset of base rotations supported by both superconducting quantum device and instruction set.

Grover's algorithm [14] is a quantum search algorithm that allows solving the problem in the order of $O(\sqrt{N})$ versus $O(N)$ operations on a classical computer. The algorithm is composed of repeated application of a quantum subroutine called *Grover operator* [24]. The Grover operator is built out of Hadamard gates surrounding an operation that performs the conditional phase shift. Similar to the QFT, the Hadamard and phase shift gates are decomposed into simple rotations and controlled-not gates.

5.1.3. Processor characteristics Our timing constrain satisfaction experiments require the basic knowledge on the processor implementation characteristics. Thus we propose a 32-bit 5-stage in-order processor architecture called *ICE* core. *ICE* executes the RISC-V base integer set and supports the non-standard QUASAR extension.

The processor has a typical architecture with data and instruction memories, general purpose register file, Arithmetical Logic Unit (ALU), etc., but it also includes three components designed to support and improve quantum extension execution, i.e. special purpose *time stamp register*, *measurement status register file* and *the quantum backend interface*. To resolve potential hazards, the processor supports *pipeline bubbles* and fully bypassed *operand forwarding*.

A detailed description of the ICE processor implementation as well as its HDL source code are publicly available to the community [6].

5.2. Encoding Efficiency

Figure 5 summarizes the encoding capabilities and limitations of the considered ISA designs. It compares the following parameters: instruction count per a single-qubit and two-qubit gate (**1q** and **2q**), the required number of registers (**register**

	RV32		eQASM		QUASAR		Vector	
	1q	2q	1q	2q	1q	2q	1q	2q
inst count	8 (9)	10 (12)	3	3	2 (3)	2 (4)	4	6
register use	2		1		0 (1)	0 (2)	2	3
	indx	mask						
max QN	2^{12} (2^{32})	12 (32)	7		512		$2^{VES}-1$	
max QON	2^{12}		2^9		2^5		$2^{VES}-1$	

Figure 5: Quantum ISA encoding summary: instruction count (inst count) per 1-qubit and 2-qubit gates (1q and 2q), register use, maximum qubit number (max QN) via index or mask encoding (indx or mask) and maximum gate operation number (max GON). VES is the size of each vector element (e.g. 8b) in the vector ISA

.

use), maximum number of qubits to be covered by the encoding (**max QN**) and the maximum number of the gate type operations (**max GON**).

5.2.1. Single- and two-qubit gates As shown in Figure 3 and 5, a single-qubit gate operations requires at least 8 RV32i instructions that is 32 `Bytes`. Every additional type of a single-qubit gate per time stamp requires at least 4 instructions (16 `Bytes`). Moreover, the `addi` instruction operates on a 12-bit immediate value. In order to address a 32-bit value, the `lui` instruction is used, which operates on the upper 20-bit range. Thus, to address a larger range of qubits an additional instruction is required. The maximum number of qubits depends on the chosen addressing method: index that can cover up to 2^{32} qubits or mask that is limited by the general purpose register width to support up to 32 qubits. A two-qubit gate operation requires an additional block of 2 instructions to specify the source qubit in a pair.

Figure 6 shows the encoding scalability for single- and two-qubit gates for the circuits with up to 32 qubits and all-to-all connectivity. The figure is divided into three sub-figures: (i) circuits with the maximum gate diversity, (ii) circuits with the same type of gates and (iii) circuits with two types of gates. The light green region shows the circuit size that corresponds to the eQASM design goal, i.e. 7 qubits. Within this region, the encoding shows good scalability. However, as the circuit size grow the number of instructions significantly increases.

Figure 6 (iii) shows the encoding for circuits with 2 types of the gates. In this example we show the variation of eQASM in case gates are applied on qubits located in multiple different sections, i.e. 1s, 2s, 2s, 4s and 5s and with different locality distribution, i.e. best case (BC) or worst case (WC). Therefore, eQASM ISA requires to analyze more than 9 potential encoding scenarios. With that, we aim to emphasize that such a variety of potential encoding result in severe complexity for code generation that only worsen with the gate diversity increase.

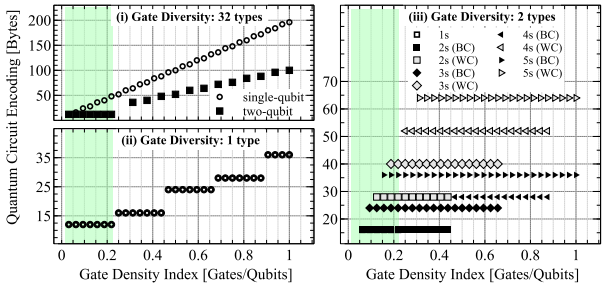


Figure 6: Quantum Circuit Encoding with eQASM ISA.

For two-qubit gates, eQASM instruction contains one mask of 16 bits where each bit represents a unique qubit coupling. According to the author's description [12], target quantum circuit of 7 qubits implements the topology with 8 couplings. Each coupling provides a possibility for two-qubit gate of two different directions. Thus, such a setup requires $8*2=16$ bits. Such an approach allows one to encode maximum number of two-qubit gates in one instruction even with the highest gate diversity.

Such an approach of the topology-aware instruction set that dramatically limits its flexibility. First, it requires continuous changes to instruction format, processor micro-architecture, and the entire software stack every time the topology or size of the quantum chip changes, even if the change is minimal. Second, if we generalize this approach encoding instead all possible couplings, but not only these physically present, we end up dealing with the complete graph. The number of edges of a complete graph with n vertices is calculated as $n*(n-1)/2$. Taking into account that each edge has two directions, this number is multiplied but 2. For a reference 7-qubit chip, the number of bits required to cover any possible topology becomes 42 vs. 16 and already does not fit into the 32-bit instruction format. For a 32-qubit chip, the number goes up to 992 bits.

Since this number does not fit into the instruction format, we make our experiments based on the assumption that the range can be covered with multiple instructions issued in sequence. Thus, with 16-bit immediate mask per instruction eQASM needs 62 instructions. Also, we ignore the fact that eQASM lacks additional bits to represent the target range, which is 6 bits as $2^6=64>62$.

Figures 7 a) and b) show single- and two-qubit gates encoding per single time stamp. We provide encoding results using QUASAR immediate, register mask, and a combination of both formats. To determine the encoding benefits and limitations, we vary three circuit characteristics described in Section 3.2 as follows: *gate density index* varies from $1/32$ to $32/32$ Gates/Qubits, *gate diversity* assumes one scenario from the set of $[1,2,4,8,16]$ of different gate types with the *balanced distribution* or *unbalanced distribution* of gate number per each type in three scenarios, i.e. $[4,8,16]$ types.

The red-cross line (32 types) represents the worst case scenario in terms of *gate diversity* encoded with the immediate

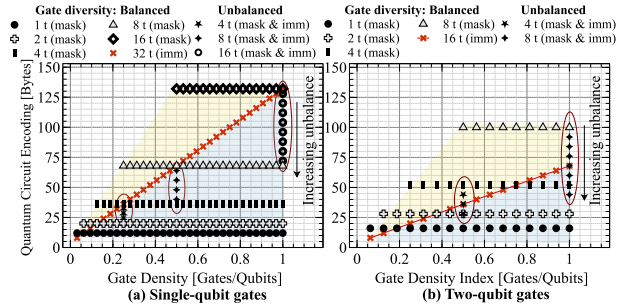


Figure 7: Quantum Circuit Encoding with QUASAR ISA.

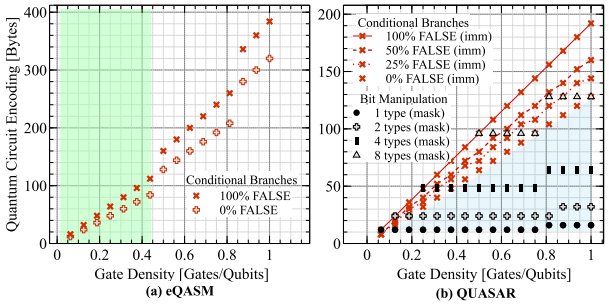


Figure 8: Measurement-dependent feedback.

instruction format only. The encoding shows linear dependency on gate density such as 1 instruction or 4 Bytes per each gate plus 1 instruction or 4 Bytes to increment the time stamp. All of the *gate diversity* scenarios with the *balanced distribution* are encoded using mask register format. Because all of the qubits are located within the 32-bit range, gate density does not impact the encoding efficiency. Each gate type requires 2 instructions or 16 Bytes plus 1 instruction or 4 Bytes to increment the time stamp.

The yellow zone indicates the cases in which mask encoding is less efficient than immediate and vice versa blue zone shows immediate encoding inefficiency. In case of an unbalanced distribution a combination of two instruction formats results in significant reduction in the requisite number of instructions. The reduction increases with the increasing unbalance. For example, in case of 8 gate types per 16 gates that is 0.5 gate density index, maximum reduction is achieved when the circuit fragment is composed of 7 gates of 7 different types encoded with immediate format and 9 gates of the same type encoded with the mask format.

Two qubits encoding shows similar to single-qubit circuit trends (Figure 7 (b)). Since two-qubit gates require twice more qubits than single-qubit gates, the gate density is calculated as the number of two-qubit gates per half of qubits, i.e. 16. The yellow zone of mask inefficiency is significantly larger. Circuits with gate diversity of half of the possible maximum, i.e. 8 types and balanced distribution do not benefit from mask even at the largest gate density of 16 gates. Also, only half of the unbalanced circuits can be efficiently encoded with the combination of two formats. The reason lies in the previously discussed need to load two mask values instead of one. The demonstrated model assumes that a 32-bit mask can be loaded with a single `la` instruction. This load requires 4 Bytes of encoding space, but causes longer execution due to the memory latency. An alternative load is composed of two immediate instructions `lui` and `addi` that do not require memory access, but take 8 Bytes to encode the sequence. Alternative encoding results in even stronger mask inefficiency.

As shown in Figure 5, qV requires 4 instructions for single-qubit and 6 instructions for two-qubit operations. Vector format requires a general purpose register to hold the memory address per each vector register used. qV enables SIMD and MIMD (different gates on different qubits) parallelism in a single instruction. Moreover, it also allows to encode single- and two-qubit gates in a single instruction. The encoding efficiency is independent of circuit characteristics.

5.2.2. Measurement-dependent feedback Figure 8 shows the encoding of quantum circuit fragments composed of two steps: qubit data measurement and measurement-dependent feedback.

Figure 8(a) shows eQASM encoding. There three factors that make eQASM feedback control encoding with branches less efficient for 32-qubit circuits. First, the processor does not have direct access to the quantum data. First, measured data first arrive to the special-purpose register, then an additional move-like instruction has to be executed to copy data to the general purpose register. After that, the data can be analyzed for conditional branch execution. Second, the eQASM operates on a 7-qubit/bit granularity. Thus, at each point the number of qubits exceeds this number, an additional move instruction has to be applied. Moreover, because eQASM is a 32-bit instruction set, all classical instructions, such as logical and arithmetical operations are of 32-bit granularity. Moving and operating a 7-bit value within the 32-bit register is unproductive and challenging. Third, eQASM lacks several crucial logical operations, such as left and right shifts to enable proper bit manipulation.

In Figure 8(b), we evaluate two encoding approaches. The first approach uses QUASAR RV32 conditional branches instructions and immediate single-qubit gate format. The results are illustrated with red-cross lines. The application of the measurement gates is similar to the single-qubit gate encoding. Thus, for measurement-dependent feedback model, we use the circuit fragment that contains only these instructions related to the conditional branches and feedback gates. Different red-cross lines indicate the percentage of conditions to be `TRUE` or `FALSE`. If the condition is `FALSE` the feedback gate is applied resulting in additional instructions execution. Otherwise, the only instruction to be executed is the conditional branch. While the encoding stays the same for all of these cases, with this we illustrate the difference in the number of instructions to be executed.

Second approach uses mask register format and bit manipulation techniques on circuits with the gate diversity of 1 type,

Table 1: Decomposed QFT sub-circuit for 32 qubits

time ⟶

1 y90	1 x180	1 z	1 cnot	1 z	1 cnot	1 z	...
8	8	11	10	11	10	11	...
3	3	3	4	3	4	3	...
4	4	6	6	6	6	6	...
1	1	3	2	3	2	3	...
2	2	3	2	3	2	3	...

RV32i eQASM qV (i/d) QUASAR

Table 2: Decomposed Grover operator for 32 qubits

time ⟶

32 y90	32 x180	32 x180	1 y90	1 x180	1 cnot	...
9	9	9	9	9	12	...
9	9	9	3	3	4	...
4	4	4	4	4	6	...
1	1	1	1	1	2	...
3	3	3	3	3	2	...

RV32i eQASM qV (i/d) QUASAR

2 types, 4 types and 8 types. Here, conditional branch instructions are eliminated in the way we discuss it in section 4.2 and show in Figure 4(b). In case a pattern of 1, 2 or 4 types is present in the measurement-dependent feedback circuit, such a circuit can be encoded up to 12×, 6× and 3× times more efficient respectively. This reduction can be achieved by using only standard logical operations, such as `and`, `or`, `not`, left and right shifts, etc. Using explicit bit manipulation extension such as Bitmanip [33], measurement-dependent feedback can be encoded even more efficiently. At the end, the feedback quantum instruction is executed despite the measured data. However, the mask register indicates whether or not the gate has to be executed. If all zeros - non of the gates will be applied.

5.2.3. Program Size Using the information from Figure 5 and the encoding study results, we examine the program size estimates for two groups of quantum circuits. First group consists of synthetic circuits with different levels of single-qubit gate density, i.e. 3%, 10%, 50%, 75% and 100%. The circuit depth is calculated based on the assumptions that the qubit lifetime is 100us and the gate time is 20ns. Second group of circuits consists of real algorithm sub-routines, i.e. QFT and Grover's operator. In both cases, the circuit size is 32 qubits. Table 1 and Table 2 shows how many words (32 bit instruction or data) the QFT and Grover's operator circuits require per each time stamp using four different ISAs: RV32I, eQASM, QUASAR and qV. We use this information to estimate program size.

Figure 9 shows the program size for each circuit encoded for each ISAs. We highlight three categories in the logarithmic scale that correspond to the following memory requirements: low (below 2 kB), medium (16-32 kB corresponding to a L1 cache), and high (512-2048KB corresponding to a L2 cache). Note, qV data is split into two bars corresponding to instruction and data.

With 8 instructions per single-qubit operation, RV32I has the highest memory requirements among all ISAs. Even with a gate density as low as 3%, the program will rarely fit into a typical L1 cache. Moreover, with a gate density of 50% or higher, the program will not even fit into the typical L2 cache. The non-deterministic behaviour of caches and and high miss penalty increases the risk of timing failures.

By contrast, eQASM can encode low density circuits within L1 cache capacities. However, because of its restriction to 7-qubit quantum devices, the program size grows rapidly with increasing gate density.

qV encoding is independent of the circuit characteristics. The program instruction sizes are comparable to the typical L1 cache size, and data requirements are below 512MB. qV provides the best encoding for the high density circuits.

Ultimately, QUASAR provides the smallest program size for most of the circuits, but suffers compared to qV for high density circuits (starting from 50%).

Most real algorithm circuits demonstrate low gate density level. Here, program size is comparable to the typical L1 cache size. However, these circuits represent only a fragment of the algorithm, thus the program size will grow proportionally to the application length. Moreover, for real systems, we expect gate density to be higher due to need for additional gates required to mitigate errors and stabilize the circuit.

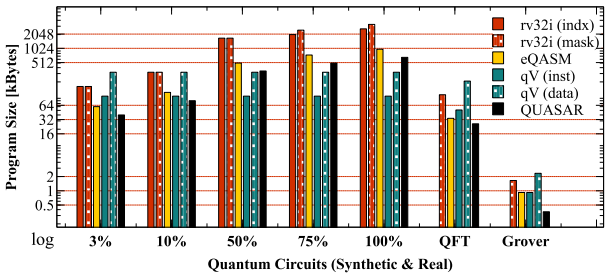


Figure 9: Program size of the quantum circuits of different density and high gate diversity.

5.3. Timing Constrain Satisfaction

Given the number of cycles per each circuit scenario, i.e. immediate instruction for the highest diversity circuits of 32 types and mask register format for 1-, 2-, 4-, 8- and 16-type diversity circuits, we report the time per single time stamp execution depending on the processor speed. We consider 200 MHz and 500 MHz that represent the execution on an FPGA board and 1 GHz, 1.5 GHz and 2 GHz for a typical ASIC implementation. Grey bars represent the execution time of the immediate instruction execution and marked lines represent sliding mask instruction execution. The red line at 20ns execution time corresponds to the duration of a typical single-qubit gate in superconducting technology. The area above the threshold

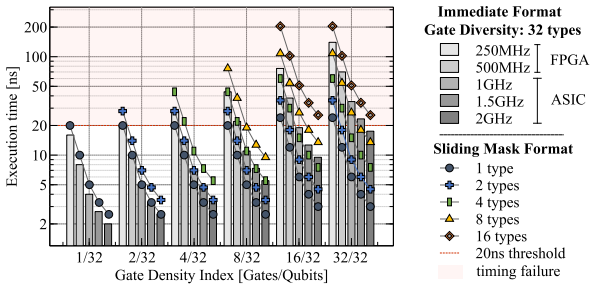


Figure 10: Timing control for a single quantum circuit cycle. Execution time is estimated based on five processor frequencies available for FPGA or ASIC implementation. A 20ns threshold corresponds to the duration of a single-qubit gate. Cases below the threshold line shows the processor failure to deliver control signals on time.

line indicates timing constrain satisfaction failure. In all the cases located in this area, the processor fails to deliver control gates on time resulting in circuit execution inaccuracy and erroneous computing results.

As shown, the ICE processor running on a FPGA can guarantee timely gate delivery for up to eight gates density with immediate format and up to 32 gates density with mask format if there are only 2 types of the gates. ASIC implementation running at around 2GHz allows executing any 32-qubit circuit using the immediate format or mask for the circuits with the gate diversity of 8 types and lower.

5.4. Summary

No single architecture is optimal for every quantum circuit. Rather, as gate diversity and density vary, so to does the optimal architecture. Similarly, as quantum system architecture evolves (qubits and topology), so to does the requirements placed on the architecture. In this section, we summarize the costs and benefits of each architecture.

RV32:

pros does not require any changes in software or hardware; supports various extensions that provide benefits for quantum computing, i.e. bit manipulation and floating-point;

cons inefficiency quantum circuit encoding requires a large number of instructions per gate that (1) demand high processor instruction throughput, and (2) result in program sizes larger than typical instruction cache capacities. (real time can only be guaranteed for simple circuits)

eQASM:

pros provides efficient encoding for a 7-qubit chip with a two-dimensional square lattice connectivity; flexible in terms of quantum gate operations allowing one to redefine the set of gates at compile-time;

cons encoding is narrowly focused and does not easily adapt to quantum systems with more qubits or different connectivity; masking of each bi-directional connection is expensive and does not scale; does not support bit manipulation; requires specialized software ecosystem.

QUASAR:

pros encoding supports both qubit addressing formats (immediate and register) which in turn allows QUASAR to support a wide range of quantum circuits; supports bit manipulation; easily integrates with the existing RISC-V ecosystem;

cons number of qubits is fixed; incompatible with the RISC-V compressed instructions extension;

qV:

pros enables SIMD and MIMD (different gates on different qubits) parallelism in a single instruction; encodes single- and two-qubit gates in a single instruction; encoding efficiency is independent of circuit characteristics; can easily be scaled to support an arbitrary number of qubits;

cons requires more data movement and more memory capacity per gate regardless of circuit; vector element size (VES) is log_2 of the maximum of the number of qubits and the number of unique gates; vector register-based ISA adds substantial complexity and hardware costs; requires a vector extension to the RISC-V software ecosystem.

6. Related Work

The need for quantum architecture exploration has been repeatedly mentioned by many authors over the last decades [10]. Despite the general consensus on its crucial role, however, very few works address this challenge in full.

In [2], authors proposed the instruction set together with the compilation and simulation frameworks for ion-trap based quantum architectures. Their software infrastructure includes a source compiler, an error correction compiler, a device scheduler and a simulator. However, the described ISA is only a "high-level" description of the execution model usually referred to as a "virtual ISA". Such an ISA does not represent any practical encoding and has no notion of the physical hardware resources.

Another virtual ISA has been recently proposed in [28]. The architecture consists of a virtual machine, an instruction language called Quil, and a quantum programming toolkit called Forest. The instruction set consists of the standard gate set used in theoretical quantum computation and superconducting qubits. Similar to the work described in [2], authors target the software frontend layers of the control stack, leaving the hardware-aware backend out of scope.

Unlike previous works, in [5] authors discuss the instruction set architecture from the hardware point of view comparing benefits and limitations of the well known RISC and CISC approaches. Yet, the paper does not propose any concrete implementation.

Finally, a team of industrial and university collaborators have recently introduced a full software-hardware stack for superconducting quantum accelerators [13]. Their work consists of the multiple levels of abstractions, including a specialized quantum ISA called eQASM [12], an assembly language (cQASM), and a compiler [19]. This is the first effort to detail the logical software and hardware architecture of the control

hardware pipeline for contemporary quantum processors. As this is a huge system building endeavor for a continuously moving target, their work focuses on functionality rather than performance evaluation. Our effort is complementary to theirs. We propose an Instruction Set Architecture for the QCP together with a simple but intuitive methodology for assessing its efficacy and scaling potential.

7. Conclusions

We analyzed the capabilities and limitations of QCPs depending on the size and specific characteristics of the quantum circuits. We consider four ISA scenarious: base RISC-V 32-bit ISA (RV32I), eQASM [12], and two RISC-V extensions that support quantum operations QUASAR and qV. We developed an evaluation methodology and circuit characterization model. It allowed us to analyze the encoding efficiency, program size and timing constrain satisfaction with synthetic circuits and two real algorithms (QFT and Grover's operator) We summarize our findings in each ISA benefirs and costs.

For future work, we plan to extend our analysis to cover a larger design space to include not only alternative architectural solutions, such as out-of-order execution, SIMD and MIMD parallelism, etc., but also emerging technologies such as superconducting digital circuits and cryo-CMOS.

8. Acknowledgments

The research leading to these results has received funding from the the U.S. Department of Energy, grant agreement n^{o} DE-AC02-05CH11231.

References

[1] Alán Aspuru-Guzik, Anthony D. Dutoi, Peter J. Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704–1707, 2005.

[2] S. Balensiefer, L. Kregor-Stickles, and M. Oskin. An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 186–196, June 2005.

[3] R. Barends, J. Kelly, A. Megrant, D. Sank, E. Jeffrey, Y. Chen, Y. Yin, B. Chiaro, J. Mutus, C. Neill, P. O'Malley, P. Roushan, J. Wenner, T. C. White, A. N. Cleland, and John M. Martinis. Coherent Josephson Qubit Suitable for Scalable Quantum Integrated Circuits. 111:080502, Aug 2013.

[4] Lev S. Bishop, Sergey Bravyi, Andrew Cross, Jay M. Gambetta, and John Smolin. Quantum volume. Technical Report 2017. https://dal.objectstorage.open.softlayer.com/v1/AUTH_039c3bf6e6e54d7, 2017.

[5] Keith A. Britt and Travis S. Humble. Instruction Set Architectures for Quantum Processing Units. *arXiv e-prints*, page arXiv:1707.06202, July 2017.

[6] Anastasii Butko. Quasar: Quantum instruction set architecture extension, 2019.

[7] Cray. Cray Assembly Language (CAL) for Cray X1 Systems Reference Manual, 2002.

[8] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536:63 EP –, 08 2016.

[9] D Deutsch. Quantum computational networks. 425:73–90, 09 1989.

[10] David P. Divincenzo. The Physical Implementation of Quantum Computation. *Fortschritte der Physik*, 48:771–783, Jan 2000.

[11] RISC-V Foundation. Risc-v cores and soc overview. https://riscv.org/risc-v-cores/, 2019.

[12] X. Fu, L. Riesebos, M. A. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, V. Newsum, K. K. L. Loh, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. eQASM: An Executable Quantum Instruction Set Architecture. *arXiv e-prints*, page arXiv:1808.02449, August 2018.

[13] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. A microarchitecture for a superconducting quantum processor. *IEEE Micro*, 38(3):40–47, May./Jun. 2018.

[14] Lov K. Grover. A fast quantum mechanical algorithm for database search. *arXiv e-prints*, pages quant–ph/9605043, May 1996.

[15] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*, 103(15):150502, October 2009.

[16] Jeremy Hsu. Ces 2018: Intel's 49-qubit chip shoots for quantum supremacy. https://spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy, 2018.

[17] D-Wave Systems Inc. D-wave. the quantum computing company. https://www.dwavesys.com.

[18] Julian Kelly. A preview of bristlecone, google's new quantum processor. https://research.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, 2018.

[19] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, and K. Bertels. cQASM v1.0: Towards a Common Quantum Assembly Language. *arXiv e-prints*, page arXiv:1805.09607, May 2018.

[20] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland. Randomized benchmarking of quantum gates. *Physical Review A, vol. 77, Issue 1*, 77(1):012307, January 2008.

[21] David Martin. Vector Extensions to the MIPS-IV Instruction Set Architecture. Technical report, University of California at Berkeley, 03 2000.

[22] Samuel K. Moore. Ibm edges closer to quantum supremacy with 50-qubit processor. https://spectrum.ieee.org/tech-talk/computing/hardware/ibm-edges-closer-to-quantum-supremacy-with-50qubit-processor, 2017.

[23] Michael A. Nielsen and Isaac L. Chuang. *Frontmatter*, pages i–viii. Cambridge University Press, 2010.

[24] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011.

[25] E. A. Sete, W. J. Zeng, and C. T. Rigetti. A functional architecture for scalable quantum computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–6, Oct 2016.

[26] J. M. Shalf and R. Leland. Computing beyond moore's law. *Computer*, 48(12):14–23, Dec 2015.

[27] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[28] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A Practical Quantum Instruction Set Architecture. *arXiv e-prints*, page arXiv:1608.03355, August 2016.

[29] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. DiCarlo. Scalable Quantum Circuit and Control for a Superconducting Surface Code. *Physical Review Applied*, 8(3):034021, Sep 2017.

[30] Joel J. Wallman and Joseph Emerson. Noise tailoring for scalable quantum computation via randomized compiling. *Phys. Rev. A*, 94:052325, Nov 2016.

[31] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. The risc-v instruction set manual, volume i: User-level isa, version 2.0. Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley, May 2014.

[32] Sheng Wei, Saro Meguerdichian, and Miodrag Potkonjak. Gate-level characterization: Foundations and hardware security applications. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 222–227, New York, NY, USA, 2010. ACM.

[33] Clifford Wolf. Risc-v bitmanip extension. https://github.com/riscv/riscv-bitmanip, 2019.

[34] J. Q. You and Franco Nori. Atomic physics and quantum optics using superconducting circuits. *Nature*, 474(7353):589–597, June 2011.