PCQ: Parallel Compact Quantum Circuit Simulation

Shuang Liang, Yuncheng Lu, Ce Guo, Wayne Luk, Paul H. J. Kelly
Department of Computing, Imperial College London, London, UK
Email: {shuang.liang, yuncheng.lu19, c.guo, w.luk, p.kelly}@imperial.ac.uk

***Abstract*—Since quantum computers are not readily available, much quantum computing research such as quantum algorithm verification has to be conducted on classical computer platforms. While many quantum circuit simulators have been developed on CPUs and GPUs, the potential of FPGAs as a platform with parallel computing capabilities and high energy efficiency has not been fully explored. This paper describes a novel approach with two modes of data movement optimization for an FPGA-based parallel pipelined dataflow architecture targeting a compact computation format. A data decoupling method is adapted to partition computing tasks and data into non-interacting subsets, significantly reducing external data interaction overhead. The proposed approach shows significant promise in improving performance and energy efficiency compared with existing state vector based CPU, GPU, and FPGA implementations.**

I. Introduction

Quantum computing is a promising research topic due to its potential to deliver exponential speedups for specific problems compared with classical computers, such as integer factorization [1], database search [2], and optimization problems [3]. Back in the 1980s, physicists envisioned incorporating the principles of quantum mechanics into computation [4], [5]. Decades later, during the 2020s, numerous research institutions have successfully developed quantum device prototypes, such as IBM [6], [7], Google [8], and D-Wave [9], [10]. Some even claim to have achieved "quantum supremacy", indicating that quantum computers can solve problems intractable on classical computers [11], [12].

However, current quantum computers are neither large-scale nor reliable. At the same time, society is showing intense enthusiasm for identifying quantum advantages and discovering and verifying quantum algorithms [13]–[15]. Developing simulators on classical computing technology becomes a means of addressing quantum computer availability.

In recent times, simulating quantum circuits on classical hardware platforms has become a flourishing research topic. Many open-source libraries based on CPU/GPU [16]–[24] have been developed to simulate quantum circuits. Among these simulators, the state vector method is among the most commonly used [25], [26]. It works by capturing evolving quantum states in the quantum circuit. We implement our hardware based on this method because simulation time increases linearly rather than exponentially with the number of quantum gates.

With the advancement of technology, Field-Programmable Gate Arrays (FPGAs) have emerged as a promising alternative due to their high parallelism, flexibility, and high energy efficiency. Some efforts focus on deploying quantum circuits with fixed structures, such as the Quantum Fourier Transform (QFT) [27], [28] and the Shor's algorithm [29] to achieve speedup or high energy efficiency. There is also recent work on reconfigurable FPGA-based simulators that can compute various quantum circuits [30]–[33].

FPGA-based simulation, however, has not been fully explored. Three issues are as follows. (1) There is a significant gap in performance between FPGA-based simulators and GPU-based simulators. Moreover, there is scope to improve the efficiency of FPGA-based simulators. (2) Current simulation studies do not address memory wall issues that hinder the extension of simulation to cover more qubits. (3) The effect of fixed-point quantization on quantum circuits with an increasing number of qubits needs investigation to support a comprehensive evaluation.

This paper makes the following contributions to address the above three issues.

1) **General-Purpose FPGA Simulation and Balanced Workload Scheduling:** To improve its performance and efficiency, we present an FPGA-based simulator that features a set of parallel computing cores with a six-stage pipeline based on a compact computing format. Moreover, it supports data-aware automatic load distribution to keep computing cores balanced and highly utilized (Section III).
2) **Novel Dataflow Decoupling Approach:** To alleviate memory wall effects on our FPGA-based simulator, a data decoupling approach is introduced to split tasks and data into separate parts. A software tool has been developed to automate task partitioning and performance improvement (Section IV).
3) **Quantization Bitwidth Experiments and Comprehensive Evaluation:** Our experiments explore fixed-point quantization fidelity across different numbers of qubits on quantum circuit benchmarks, which can provide the basis for future low-bitwidth simulations. Through our normalization efforts, we make a comprehensive evaluation. Compared with state-of-the-art FPGA implementations, our design can achieve 1.3 to 497.2 times speedup and 3.4 to 81.7 times efficiency improvement (Section V).

An overview of the rest of the paper is as follows. Section II reviews the theory of quantum circuit simulation based on the state vector approach. Section III describes a workload balancing strategy and our hardware architecture. Section IV introduces the data decoupling approach and shows how it

can alleviate the memory wall effects on FPGA designs. Section V presents our experimental methodology, followed by comparisons with current FPGA/CPU/GPU designs and analysis of the effects of the data decoupling approach.

II. Background: Quantum Computing Theory

A. Qubit and Quantum Gate

The quantum bit (Qubit) is the smallest unit in quantum computing [34]. Our work targets *pure* state simulation only, as its generalization in supporting mixed states is straightforward by representing a mixed state as a linear combination of pure states. An n-qubit pure state, in its most general form as a state vector, is represented by a length-2^n complex-valued vector with norm 1. Each element in the state vector is called an *amplitude*. By convention, we adopt the bra-ket notation, wherein a state is denoted by a ket vector $|\psi\rangle \in \{\psi \in \mathbb{C}^{2^n} | \|\psi\| = 1\}$. Its corresponding bra vector $\langle\psi|$ is obtained by taking the Hermitian transpose of $|\psi\rangle$. We reserve the notation $|k\rangle$ for $k = 0, 1, \cdots$ for the computational basis vectors. An n-qubit $|k\rangle$ state is the 2^n-vector whose k-th amplitude is 1, and other amplitudes are 0.

An n-qubit quantum gate is a $2^n \times 2^n$ unitary matrix. A quantum circuit is an ordered collection of quantum gates. Our design choice also respects the fact that main quantum circuits are basically composed of single-qubit gates and two-qubit gates [34].

B. Quantum Circuit Simulation Using the State Vector Approach

Under the state vector description, applying a gate U on a state $|\psi\rangle$ is effectively performing a matrix-vector multiplication $U|\psi\rangle$. For example, applying a single-qubit gate $\begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix}$ on state $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ gives the state:

$$\begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} u_{00}\alpha + u_{01}\beta \\ u_{10}\alpha + u_{11}\beta \end{bmatrix} \quad (1)$$

C. Compact Computation Format

We extend (1) to multiple-qubit cases, and the matrix-vector representation can be seen on the left of Fig. 1 (applying a single-qubit gate on a 3-qubit state vector with various target qubits). The target qubit in a quantum gate reflects the entanglement between elements inside the state vector. Considering that most of the quantum gates in common circuits are single-qubit gates, that is, there is no entanglement between most elements, and the corresponding matrix representation has a large number of zero elements. In our hardware, we only store and compute nonzero elements to improve efficiency, which we call the compact computing format. Some recent high-performance works, such as QuEST [25], have adopted this computing format.

III. Hardware Architecture

In the compact computing format, a large matrix-vector multiplication can be decomposed into multiple smaller matrix-vector multiplications with non-zero elements. In our architecture, parallel PEs are implemented, each of which can perform compact matrix-vector multiplications with low latency. To provide sufficient internal bandwidth, we also implement multiple buffers with the same number of PEs using on-chip BRAMs to store the state vector in a distributed manner. The third column of Fig. 1 shows how elements are stored in multiple buffers. For instance, for P buffers, Buffer #1 stores the state $P * k$, Buffer #2 stores the state $P * k + 1$, $\cdots$, Buffer #P stores the state $P * k + P - 1$ $(k = 0, 1, \cdots)$.

A. Hardware Design Challenges and Our Solutions

For the parallel architecture mentioned above, to reduce the computing latency, the two main challenges and our solutions are as follows:

- How to maintain workload balance in the processing element (PE) array: Fig. 1 presents how to complete a balanced workload distribution when applying a single-qubit gate with various target qubits. Distinct target qubits signify the entanglement between different elements in the state vector, which results in the required memory access behavior variations. Since elements are distributed in multiple buffers, this may occur in two situations (Data Loading Mode in Fig. 1). Mode 1: Elements required by each PE are in the same buffer. Mode 2: Elements required by each PE are stored in two different buffers (Section III-B).
- How to reduce the computing latency inside each PE: To reduce the latency, our general-purpose PE features a 6-stage fine-grained pipeline. It allows our architecture to have a higher speedup than other FPGA simulators (Section III-C).

B. Workload Balancing

In our simulator, load balancing is automatically implemented by hardware modules based on the characteristics of the input data. The core of workload balancing involves delivering required data to PEs to complete computations. We have two modules that help with automatic data delivery. The first, Address Generator, generates addresses for buffers based on the target qubit (fourth column of Fig. 1). The second, Data Routing Module, is enabled in Mode 2 to help PEs get required data from appropriate output ports of buffers (third column of Fig. 1). The behavior of the memory access and data routing depends on the following factors: the number of PEs P, the number of qubits N, and the target qubit t.

1) Mode 1 (Unicast): When the target qubit $t \geq \log P$, the data transfer between buffers and PEs is in the unicast mode. Actually, t determines the stride of reading elements in the state vector, and P represents the stride between adjacent elements in the same buffer. Therefore, $\log P$ can be seen as the boundary for whether data reading involves two buffers. In this mode, PE #k only receives data from Buffer #k, and

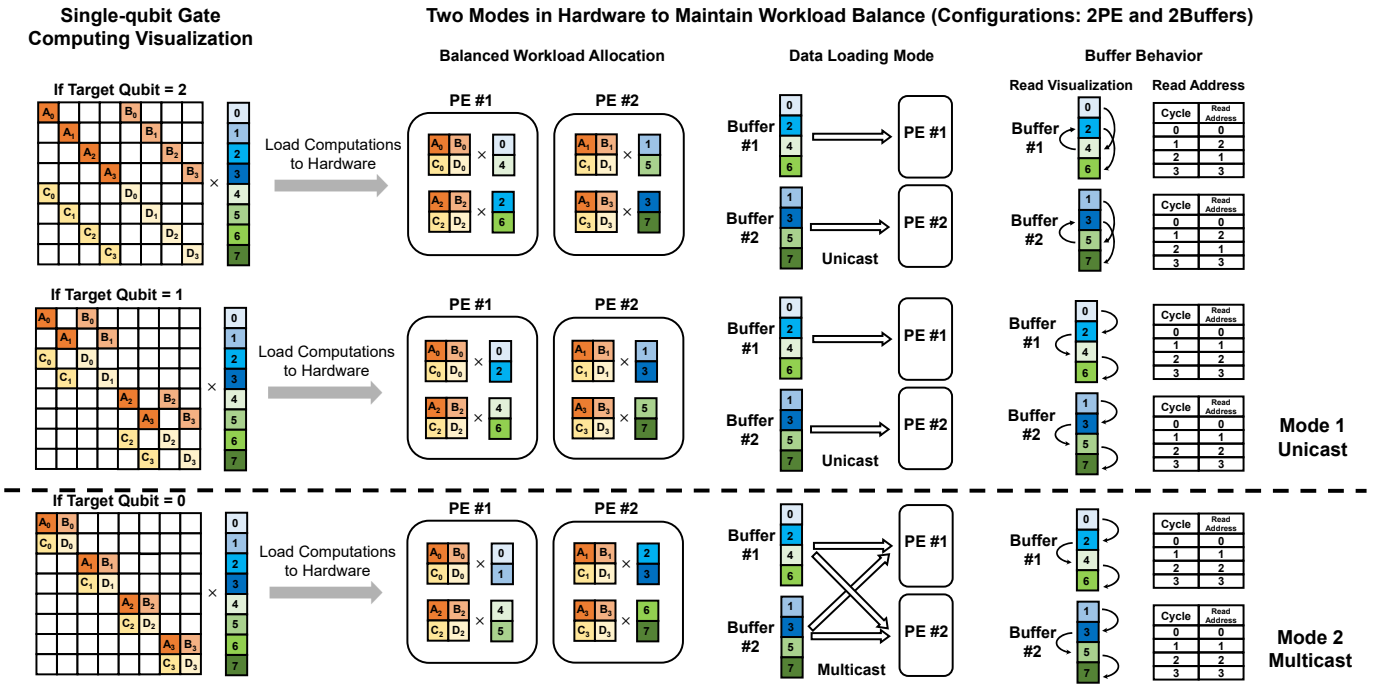

Fig. 1. Two modes (Unicast and multicast data loading modes) to maintain workload balance.

addresses for reading and writing are in jump order according to t. A total of $2^{N-1}/P$ compact matrix operations are performed on PE #k. Meanwhile, the read address of Buffer #k changes as $[0, 2^{t-\log P}, 1, 1+2^{t-\log P}, \cdots]$ until all data have been accessed.

2) Mode 2 (Multicast): When the target qubit $t < \log P$, the data transfer between buffers and PEs is in the multicast mode. In this mode, PE #k and PE #$(k+2^t)$ receive data from Buffer #k and Buffer #$(k+2^t)$. At the input port of each PE, there is a MUX to help it fetch data from different buffers at different cycles. In this mode, the Address Generator generates successive read addresses $[0, 1, 2, \cdots]$ for all buffers.

C. Fully Pipelined Processing Element

The processing element (PE), the basic computing unit of our hardware architecture, completes matrix operations on 2×1 vectors. The hardware architecture of the PE to complete compact matrix-vector operations is shown in Fig. 2. If the function of the 2×2 matrix is to exchange elements' positions (such as *PauliX* and *PauliY*), PEs will use a set of MUX bypasses to directly exchange input data instead of using the arithmetic circuits in Fig. 2.

This computing core has a six-stage pipeline without pipeline bubbles. Multiplication of 2×2 matrix and 2×1 vector is calculated in the following order to reuse hardware: For instance, to complete $[w_{00}, w_{01}; w_{10}, w_{11}] \times [d_0; d_1]$, the computing core will compute $w_{00} \times d_0$ and $w_{10} \times d_0$ first, then $w_{01} \times d_1$ and $w_{11} \times d_1$.

The specific dataflow is as follows: First, load d_0, w_{00}, and w_{10} to registers in Multiplier Arrays. It then proceeds to compute $w_{00}d_0$ and $w_{10}d_0$ and store them in the Accumulation Register Group. Subsequently, accumulation results are transferred to registers in the Intermediate Register Group via DEMUX. Likewise, the next set of data d_1, w_{01}, and w_{11} are loaded to compute $w_{01}d_1$ and $w_{11}d_1$. Finally, the results of compact matrix-vector multiplication can be obtained by $w_{00}d_0 + w_{01}d_1$ and $w_{10}d_0 + w_{11}d_1$.

IV. Data Decoupling and Distributed Simulation

In our architecture, external memory accesses are only performed at the beginning and end of the simulation, leading to high efficiency. However, this approach requires all state vectors to stay on-chip for optimized efficiency; otherwise external memory access can cause a bottleneck. To circumvent this problem, most studies focus on a small number of qubits (less than 21 qubits [30]–[32]) or resolve this issue with high-bandwidth memory [33]. Inspired by a GPU-based simulator [35], we present a data decoupling approach for FPGA to simulate circuits with a large number of qubits efficiently without using specialized memory. Furthermore, we develop software to automate the decoupling according to the circuit structure and memory limitation. This software tool can also evaluate the improvement in external memory access efficiency based on the data decoupling method.

Fig. 3 uses the directed graph to present how to decouple a massive computing task (directed graph) to small tasks (connected components) running on independent devices with limited memory space. This directed graph has two types of nodes: circles representing data and squares representing

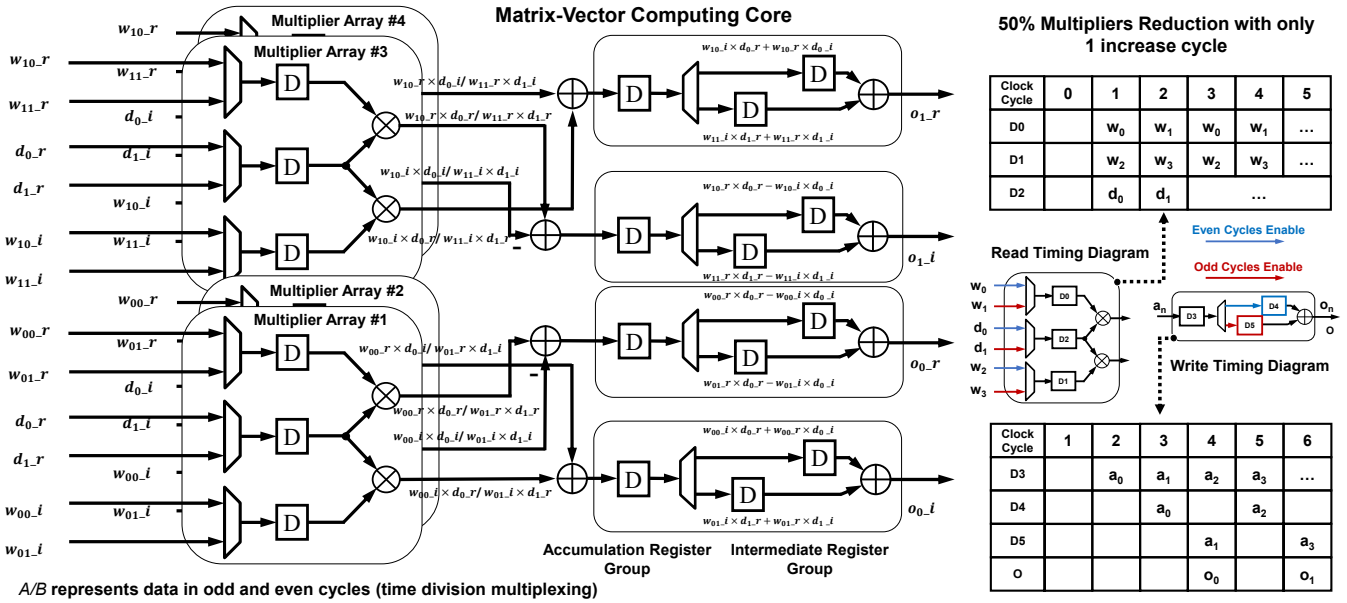

Fig. 2. Hardware architecture with time division multiplexing and full pipeline (Left: Hardware details. Right: Timing diagram of time division multiplexing). Only the hardware responsible for dense matrix calculations is shown, the MUX bypass for data exchange is omitted for clarity.

operations. The data involved in the same operation are placed on the same device because they will be calculated together. In Fig. 3, we assign data on the same connected component to the same device. Specifically, we will compute OP1, OP3, and OP5 on device 1 and OP2, OP4, and OP6 on device 2. After initialization, all devices only use local data for computation, which eliminates the external communication overhead.

We can partition the circuit into several segments for deeper circuits, where qubits become increasingly entangled. Segmenting circuits can extend this approach to cover simulations with more quantum gates. In addition to its use in distributed computing like similar approaches designed for GPU-based simulations, this method can be used to execute divided subtasks in sequence on a single device. Moreover, this method is universal and orthogonal to other partitioning methods. It can be applied to arbitrary quantum circuit simulations in combination with other distributed methods, such as circuit division [36].

V. Evaluation

A. Experimental Methodology

Hardware Design Configuration and Testing Platform: The proposed simulator (Configuration: 18-bit fixed-point quantization/32 PEs/32 Buffers) is designed and implemented in Verilog HDL through Vivado 2022.1. The performance of this design is evaluated on AMD Xilinx XCVU9P.

CPU/GPU Comparison Baseline: Our CPU simulations are performed with an Intel Core i9-11900K with 32GB RAM. GPU simulations are performed with a Nvidia GeForce RTX 3080 Ti with 12GB memory. We compare the proposed simulator with three state-of-the-art CPU/GPU libraries with default settings (the compilation time is deducted): QuEST [25] (version 3.7.0), Qiskit [37] (version 0.45.1), and CUDA Quantum [38] (version 0.5.0). Double-precision floating-points are used among all CPU and GPU simulations. We measure power consumption using a socket power meter. The power for each design is offset by its idle power.

B. Quantization Experiment

The fidelity of quantum devices is a critical factor that affects the performance of quantum algorithms. The low fidelity of devices causes errors to accumulate in circuits, which can result in wrong results in the quantum algorithm. Similarly, the errors introduced by quantization should also be considered in the quantum circuit simulation.

We evaluate the fidelity of fixed-point simulations on QFT, known as one of the most commonly used quantum circuits. We take various numbers of qubits n and quantization bitwidths q_{width}. To ensure our design does not bias towards any state vector configuration, in each run, the input state vector is (Haar-)randomly initialized. We take Discrete Fourier Transform (DFT) with double-precision floating-point as the ground truth and compute the fidelity of fixed-point FPGA simulations.

To help researchers explore the tradeoff between fidelity and quantization bitwidth, we record the quantization bitwidth needed to achieve a fidelity of, respectively, 0.95, 0.99, and 0.999 in Fig. 4. In this work, we choose the bitwidth that can achieve a fidelity of 0.99, which exceeds the commonly defined high fidelity [39]–[41].

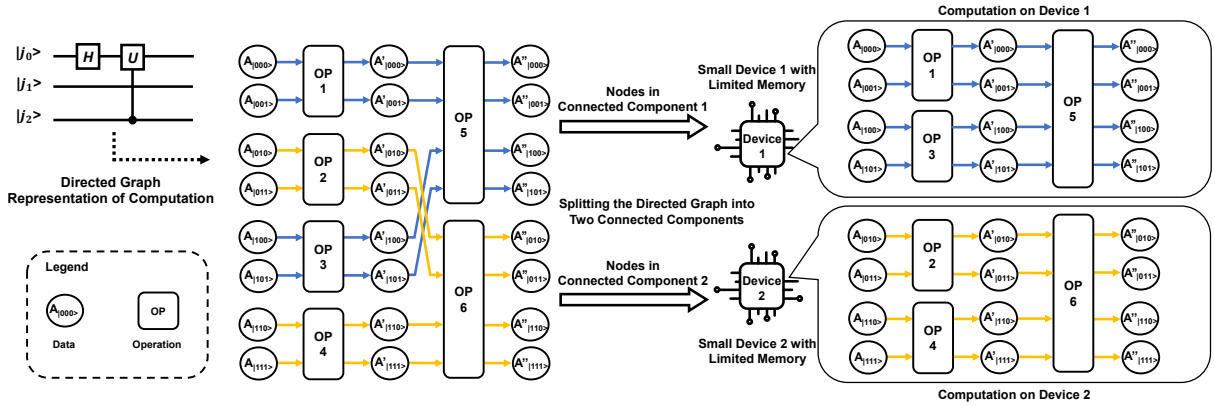

Fig. 3. Split the directed graph of computation to connected components, where each component represents a sub-task that operates independently.

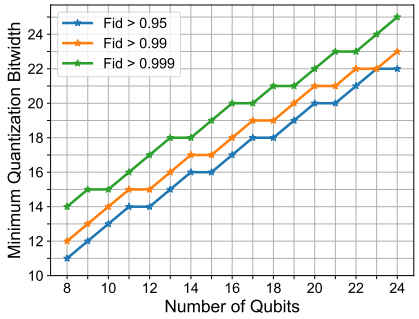

Fig. 4. Minimum quantization bitwidth needed to achieve target fidelity.

C. Comparison with FPGA Designs

Since there are many differences in the performance reports of current FPGA-based simulators [32], we choose a benchmark, QFT, which is shared among most works. It should be noted that our comparison does not include FPGA-based simulators customized for the specific quantum circuits.

Table I summarizes the hardware performance of our design and state-of-the-art general-purpose FPGA-based simulators. Compared to these FPGA-based simulators, our design can achieve 1.3 to 497.2 times speedup and 3.4 to 81.7 times better efficiency. We instantiated XDMA (8GT/s and 8 lanes) in our architecture for external memory access. To make a fair comparison, when the memory access time or hardware resource usage for a design is not available, we estimate the performance using the same settings as our design.

Compared to [30], our architecture can achieve higher performance due to skipping zero elements, high hardware utilization, and a six-stage pipeline. Compared with [31], we choose a suitable quantization bitwidth according to our quantization experiments. It ensures that our simulations are sufficiently accurate without having redundant bitwidth. In addition, our design features more fine-grained optimization and hardware reuse compared to [33].

For simulations with more qubits, some published designs are limited by increased hardware resource consumption [27], [30]. Performance of some designs is affected by external communication overhead [31]. Although some designs, such as [33], adopt high-bandwidth memory to alleviate memory wall effects, memory access time still greatly affects the overall latency. Due to the versatility of our design, the computing resource usage will not increase with increasing number of qubits. By applying the data decoupling method in FPGA-based simulation, the limitations of on-chip memory and external communication overhead can be significantly reduced in this work.

D. Comparison with High-Performance CPU/GPU Libraries

Table II shows the performance comparisons of our work and three high-performance CPU/GPU simulators on the same benchmark. Compared to these CPU-based simulators, our design can achieve 5.3 to 181.5 times speedup. Compared with GPU-based simulators with powerful back-end optimization and a high degree of parallelism, our design still achieves 0.3 to 8.1 times acceleration and 14.7 to 338.5 times better energy efficiency. As the parallelism of the current design is 32, versions with higher bandwidth and increased parallelism can achieve higher speedup. Due to the characteristics of workload balancing in our design, the computational latency decreases linearly with the increase in parallelism.

E. Effect of Data Decoupling and Analysis

The above comparison demonstrates that an FPGA design, as a competitive simulation platform, provides excellent energy efficiency. Next, we will illustrate the effectiveness of the data decoupling approach with an example.

Still taking QFT as the benchmark, we assume a single computing node can store the state vector of less than 20 qubits locally (in line with the storage resources of mainstream FPGAs). Fig. 5 illustrates how much external memory access overhead will be reduced with the number of qubits. Compared with the conventional dataflow, this method can significantly reduce the external memory access overhead when simulating

TABLE I
HARDWARE PERFORMANCE COMPARISONS WITH STATE-OF-THE-ART FPGA-BASED SIMULATORS ON QFT-16

		Leidö, 2022[a][30]	Hong et al, 2022 [31]	Waidyasooriya et al, 2022[a][33]	Our Design
Platform		Xilinx PYNQ Z-2	Xilinx XCKU115	Intel Stratix 10 MX2100	Xilinx XCVU9P
Technology (nm)		28	20	14	16
Frequency (MHz)		300[b]	160	299	232
Quantization		32-bit fixed-point	16-bit fixed-point	32-bit floating-point	18-bit fixed-point
LUT		14K (40K)[c]	19K	307K	40K (66K)
FF		22K (50K)	4K	673K	13K (41K)
BRAM		1.5 (51.5)	NA	1513	64 (114)
DSP		128	128	1008	448
Latency (ms)		646.2 (646.3)	270.0	1.7	1.2 (1.3)
Throughput[d](GOPS)		0.72	1.23	137.07	204.85
Efficiency[d]	GOPS/DSP	0.0056	0.0096	0.1360	0.4573
	GOPS/MLUT	18.0	64.7	446.5	3103.8

[a.] For different benchmarks, the normalization is carried out according to the number of quantum gates and the data size. Moreover, we use Xilinx FPGA as a benchmark for uniform resource usage calculation across platforms.
[b.] [30] did not report the frequency, 300MHz is the ideal result given by authors.
[c.] A (B): A is the computing time/hardware resources, and B represents the total time/hardware resources (computing+data transfer).
[d.] For fair comparison, the performance of all designs is scaled linearly based on the 16nm technology.

TABLE II
PERFORMANCE COMPARISON OF OUR FPGA SIMULATOR AND MAINSTREAM CPU/GPU SIMULATION LIBRARIES ON QFT-16

	Qiskit		QuEST		CUDA Quantum		Our Design#Parallel 32	Our Design#Parallel 128[a]
Platform	Intel Core i9-11900K	Nvidia 3080Ti	Intel Core i9-11900K	Nvidia 3080Ti	Intel Core i9-11900K	Nvidia 3080Ti	Xilinx XCVU9P	scaling to 8nm
Technology (nm)	14	8	14	8	14	8	16	8
Power (W)	150.0	157.9	39.0	197.6	246.5	227.5	3.8	15.2
Latency (ms)	15.90	10.10	6.55	0.35	225.00	1.06	1.20 (1.24)	0.15 (0.19)
Executions per second	62.9	99.0	152.7	2857.1	4.4	943.4	806.5	5263.2
Efficiency (Exe/s/W)	0.419	0.627	3.915	14.459	0.018	4.147	212.237	346.263

[a.] Estimation: According to TSMC's technical report, an approximate linear frequency scaling is used here. Data transfer using PCIe Gen3 16lanes on XCVU9P.

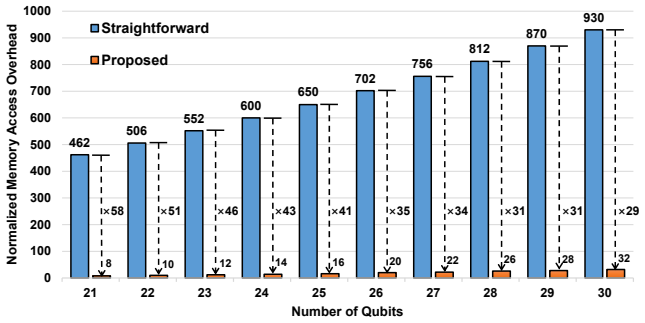

Fig. 5. Memory access efficiency improvement caused by proposed data decoupling (Normalization means that the entire state vector movement is treated as one transmission).

large quantum circuits. Specifically, it can increase memory access efficiency by 29 to 58 times.

VI. CONCLUSION

This paper develops PCQ, a general-purpose FPGA-based quantum circuit simulator with a compact computing format, fully pipelined computing cores, and balanced workload distribution. In comparison to Qiskit, QuEST, and CUDA Quantum, three high-performance GPU simulators, it can achieve 0.3 to 8.1 times speedup and 14.7 to 338.5 times better energy efficiency.

Our work includes analyzing the memory wall in FPGA-based simulations. A data decoupling method and corresponding automation software have been developed to alleviate its effects. On a device capable of storing less than 2^{20} complex numbers locally, when dealing with QFT with the number of qubits from 21 to 30, this approach increases the external memory access efficiency by 29 to 58 times.

Current and future work includes exploring further optimizations that can benefit the proposed approach, and developing tools that support the automation of design implementation and debugging.

ACKNOWLEDGMENT

The support of UK EPSRC (grant number EP/W032635/1, EP/V028251/1, EP/S030069/1 and EP/X036006/1), Intel and AMD is gratefully acknowledged.

References

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE, 1994, pp. 124–134.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, 1996, pp. 212–219.

[3] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[4] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, no. 6/7, 1982.

[5] D. Deutsch, "Quantum theory, the church–turing principle and the universal quantum computer," *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[6] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. Van Den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme *et al.*, "Evidence for the utility of quantum computing before fault tolerance," *Nature*, vol. 618, no. 7965, pp. 500–505, 2023.

[7] E. H. Chen, G.-Y. Zhu, R. Verresen, A. Seif, E. Baümer, D. Layden, N. Tantivasadakarn, G. Zhu, S. Sheldon, A. Vishwanath *et al.*, "Realizing the nishimori transition across the error threshold for constant-depth quantum circuits," *arXiv preprint arXiv:2309.02863*, 2023.

[8] "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, no. 7949, pp. 676–681, 2023.

[9] A. D. King, S. Suzuki, J. Raymond, A. Zucca, T. Lanting, F. Altomare, A. J. Berkley, S. Ejtemaee, E. Hoskinson, S. Huang *et al.*, "Coherent quantum annealing in a programmable 2,000 qubit ising chain," *Nature Physics*, vol. 18, no. 11, pp. 1324–1328, 2022.

[10] A. D. King, J. Raymond, T. Lanting, R. Harris, A. Zucca, F. Altomare, A. J. Berkley, K. Boothby, S. Ejtemaee, C. Enderud *et al.*, "Quantum critical dynamics in a 5,000-qubit programmable spin glass," *Nature*, pp. 1–6, 2023.

[11] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, pp. 595–600, 2018.

[12] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[13] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio *et al.*, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.

[14] E. Peters, J. Caldeira, A. Ho, S. Leichenauer, M. Mohseni, H. Neven, P. Spentzouris, D. Strain, and G. N. Perdue, "Machine learning of high dimensional data on a noisy quantum processor," *npj Quantum Information*, vol. 7, no. 1, p. 161, 2021.

[15] A. Assouel, A. Jacquier, and A. Kondratyev, "A quantum generative adversarial network for distributions," *Quantum Machine Intelligence*, vol. 4, no. 2, p. 28, 2022.

[16] Y. Gao, J. Xu, and H. Wang, "CuNH: Efficient GPU implementations of post-quantum kem newhope," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 551–568, 2021.

[17] A. Li, O. Subasi, X. Yang, and S. Krishnamoorthy, "Density matrix quantum circuit simulation via the bsp machine on modern GPU clusters," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[18] D. Lykov, A. Chen, H. Chen, K. Keipert, Z. Zhang, T. Gibbs, and Y. Alexeev, "Performance evaluation and acceleration of the qtensor quantum circuit simulator on GPUs," in *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*. IEEE, 2021, pp. 27–34.

[19] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. Garcia-Saez, J. I. Latorre, and S. Carrazza, "Qibo: A framework for quantum simulation with hardware acceleration," *Quantum Science and Technology*, vol. 7, no. 1, p. 015018, 2021.

[20] S. Efthymiou, M. Lazzarin, A. Pasquale, and S. Carrazza, "Quantum simulation with just-in-time compilation," *Quantum*, vol. 6, p. 814, 2022.

[21] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen *et al.*, "Qiskit: An open-source framework for quantum computing," *Accessed on: Mar*, vol. 16, 2019.

[22] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: An open source software framework for quantum computing," *Quantum*, vol. 2, p. 49, 2018.

[23] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, "A software methodology for compiling quantum programs," *Quantum Science and Technology*, vol. 3, no. 2, p. 020501, 2018.

[24] S. Mandrà, J. Marshall, E. G. Rieffel, and R. Biswas, "HybridQ: A hybrid simulator for quantum circuits," in *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*. IEEE, 2021, pp. 99–109.

[25] T. Jones, A. Brown, I. Bush, and S. C. Benjamin, "QuEST and high performance simulation of quantum computers," *Scientific reports*, vol. 9, no. 1, p. 10736, 2019.

[26] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Full-state quantum circuit simulation by using data compression," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–24.

[27] Y. Qian, M. Wang, J. Chen, L. Wang, and Z. Feng, "Efficient FPGA emulation of quantum fourier transform," in *2019 China Semiconductor Technology International Conference (CSTIC)*. IEEE, 2019, pp. 1–3.

[28] T. M. Aye and M. Iwahashi, "Implementation and analysis of quantum fourier transform gates over FPGA framework," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2019, pp. 1–5.

[29] X. Zhang, Y. Zhao, R. Li, X. Li, Z. Guo, X. Zhu, and G. Dong, "The quantum shor algorithm simulated on FPGA," in *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2019, pp. 542–546.

[30] E. Leidö, "Optimizing quantum computer simulation," Master's thesis, Chalmers University of Technology, 2022.

[31] Y. Hong, S. Jeon, S. Park, and B.-S. Kim, "Quantum circuit simulator based on FPGA," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2022, pp. 1909–1911.

[32] N. Mahmud and E. El-Araby, "Improving emulation of quantum algorithms using space-efficient hardware architectures," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 206–213.

[33] H. M. Waidyasooriya, H. Oshiyama, Y. Kurebayashi, M. Hariyama, and M. Ohzeki, "A scalable emulator for quantum fourier transform using multiple-FPGAs with high-bandwidth-memory," *IEEE Access*, vol. 10, pp. 65 103–65 117, 2022.

[34] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2010.

[35] P. Zhang, J. Yuan, and X. Lu, "Quantum computer simulation on multi-GPU incorporating data locality," in *Algorithms and Architectures for Parallel Processing: 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part I 15*. Springer, 2015, pp. 241–256.

[36] R. Li, B. Wu, M. Ying, X. Sun, and G. Yang, "Quantum supremacy circuit simulation on sunway taihulight," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 805–816, 2019.

[37] IBM, "Qiskit," https://qiskit.org/, last accessed 2023-12-23, Version 0.45.1.

[38] J.-S. Kim, A. McCaskey, B. Heim, M. Modani, S. Stanwyck, and T. Costa, "CUDA Quantum: The platform for integrated quantum-classical computing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–4.

[39] J. Gibbs, K. Gili, Z. Holmes, B. Commeau, A. Arrasmith, L. Cincio, P. J. Coles, and A. Sornborger, "Long-time simulations with high fidelity on quantum hardware," *arXiv preprint arXiv:2102.04313*, 2021.

[40] L. Stephenson, D. Nadlinger, B. Nichol, S. An, P. Drmota, T. Ballance, K. Thirumalai, J. Goodwin, D. Lucas, and C. Ballance, "High-rate, high-fidelity entanglement of qubits across an elementary quantum network," *Physical Review Letters*, vol. 124, no. 11, p. 110501, 2020.

[41] A. Laing, A. Peruzzo, A. Politi, M. R. Verde, M. Halder, T. C. Ralph, M. G. Thompson, and J. L. O'Brien, "High-fidelity operation of quantum photonic circuits," *Applied Physics Letters*, vol. 97, no. 21, 2010.