QASM-to-HLS: A Framework for Accelerating Quantum Circuit Emulation on High-Performance Reconfigurable Computers

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY®

Anshul Maurya, Naveed Mahmud

Department of Electrical Engineering and Computer Science, Florida Institute of Technology

amaurya2022@my.fit.edu, nmahmud@fit.edu

FLORIDA TECH

Introduction

- **Motivation**
  - Existing FPGA-based quantum circuit emulation methods [1][2] exhibit limited flexibility in emulating a variety of quantum algorithms
    - They have fixed, algorithm-specific hardware architectures.
  - Additionally, the computational load of generating transformation matrices generally fall on the CPU, thereby restricting the extent to which FPGAs can be leveraged for accelerating quantum algorithms.
  - Mapping a quantum algorithm to its corresponding FPGA architecture for emulation is challenging, particularly for algorithm developers with limited FPGA design experience.
- **Objectives and Approach**
  - Provide an interface between front-end quantum algorithm design and backend FPGA-based emulation.
  - Develop a methodology for converting quantum circuits represented in Quantum Assembly Language (QASM) into their corresponding architectures for emulation on FPGA backends.
  - Automatically derive quantum emulation architectures for High-Level Synthesis (HLS) on FPGAs. Investigate variations of hardware architectures with trade-offs between area and speed. Derived architectures support 64-bit FP precision and complex number arithmetic.

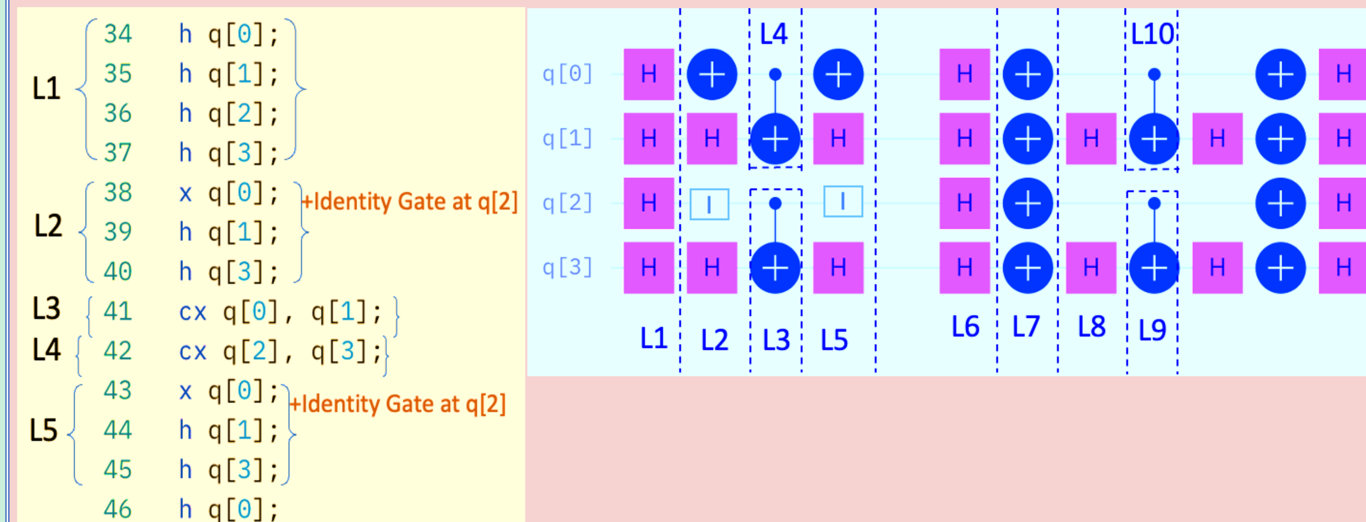
Background

- **QASM**
  - Quantum assembly languages [3] are machine-independent languages that traditionally describe quantum computation in the circuit model.
  - Open quantum assembly language (Open QASM 2) was proposed as an imperative programming language for quantum circuits based on earlier QASM dialects.
  - In principle, any quantum computation could be described using Open QASM 2.
- **High-level Synthesis**
  - High-level synthesis (HLS) [4] automates hardware design by converting high-level programming languages into optimized hardware code.
  - Let's hardware designers efficiently build and verify hardware, by allowing them to describe the design at a higher level of abstraction.

QASM Processing

In our methodology, we start with the QASM description of the quantum circuit. An efficient QASM parsing method (QASM-to-HLS) is developed that generates high-level hardware codes for HLS.

- **Layering:**
  - To emulate quantum operation, each layer of the quantum circuit is considered, and the corresponding matrix operation is generated.
  - To conserve resources, layers with CNOT gates only are segregated.

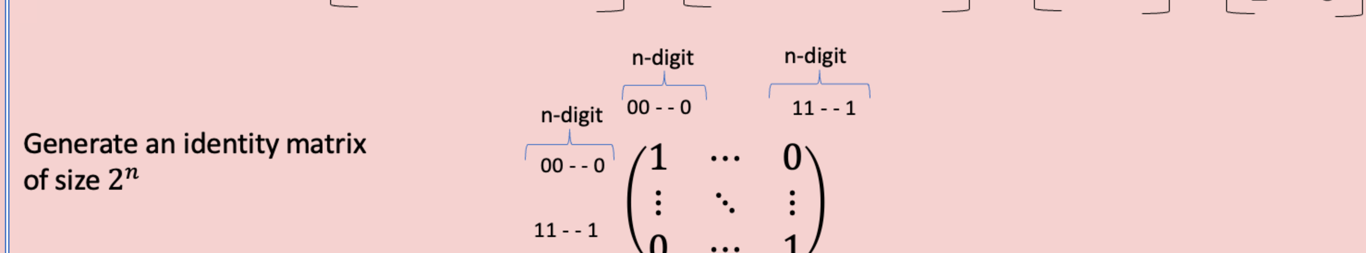


  - Multiplying the matrices of two adjacent layers can yield a consolidated single-layer matrix, leveraging this insight can effectively mitigate data transfer latency and accelerate the computation of results across multiple layers.
  - Controlled gates [5] will be processed differently without forming the pairs.
- **Matrix Generation :**
  - Each layer's matrix is computed using the developed python based QASM-to-HLS package.
  - The package contains functions for layer identification and classification from the QASM code.
  - Layer matrices are generated by tensor operations between individual gate matrices.

Ex:

$$L1: [h, h, I, x] = \begin{bmatrix} 0.707 & 0.707 \\ 0.707 & -0.707 \end{bmatrix} \otimes \begin{bmatrix} 0.707 & 0.707 \\ 0.707 & -0.707 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Generate an identity matrix of size 2^n

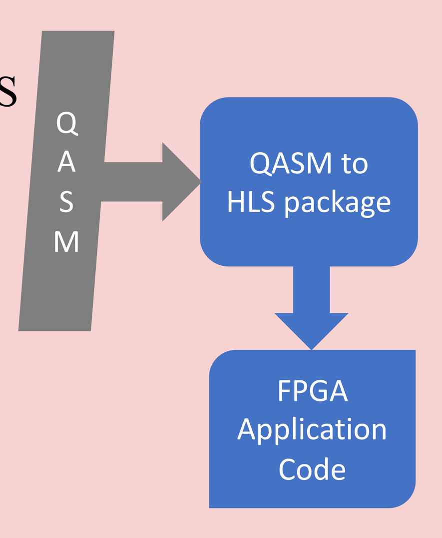


- Based on control bit, flip the target bit and at corresponding place of new binary string, update with 1

Ex.:

0 0 0 1 0 → 1 0 0 1 0
T C T C

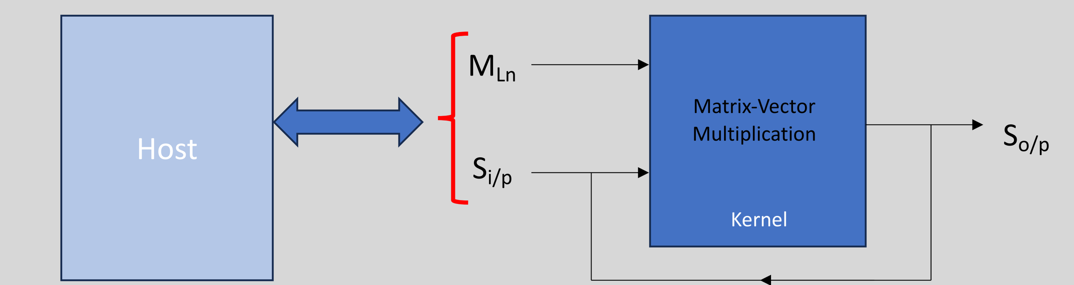
- QASM-to-HLS processes the QASM and produces the HLS application code, consisting of host (PC) code and kernel (FPGA) code
- For fixed number of qubits, only the host code changes for different algorithms and gate parameters. Kernel architecture remains same.



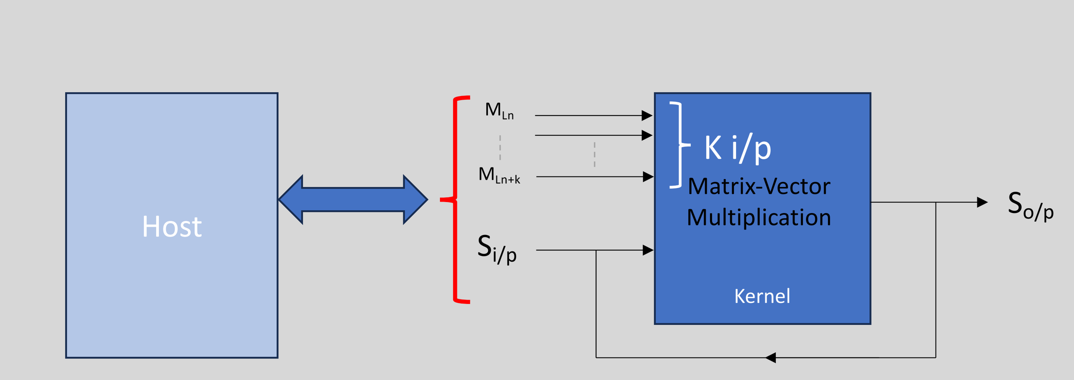
FPGA Architectures

- **QASM-to-HLS can generate 3 design variants of kernel architecture for quantum emulation**

1. **Type-1 Design:** This design performs a sequence of matrix-vector multiplications to determine a final output quantum state. This is performed by a single kernel that takes a layer matrix and state vector as inputs and performs complex matrix-vector computation.



2. **Type-2 Design:** A parallelized version of Type-1 design which uses concurrent input streams and reduces the data transfer time. This design is optimal for emulation of small circuits and utilizes the FPGA resources more efficiently.

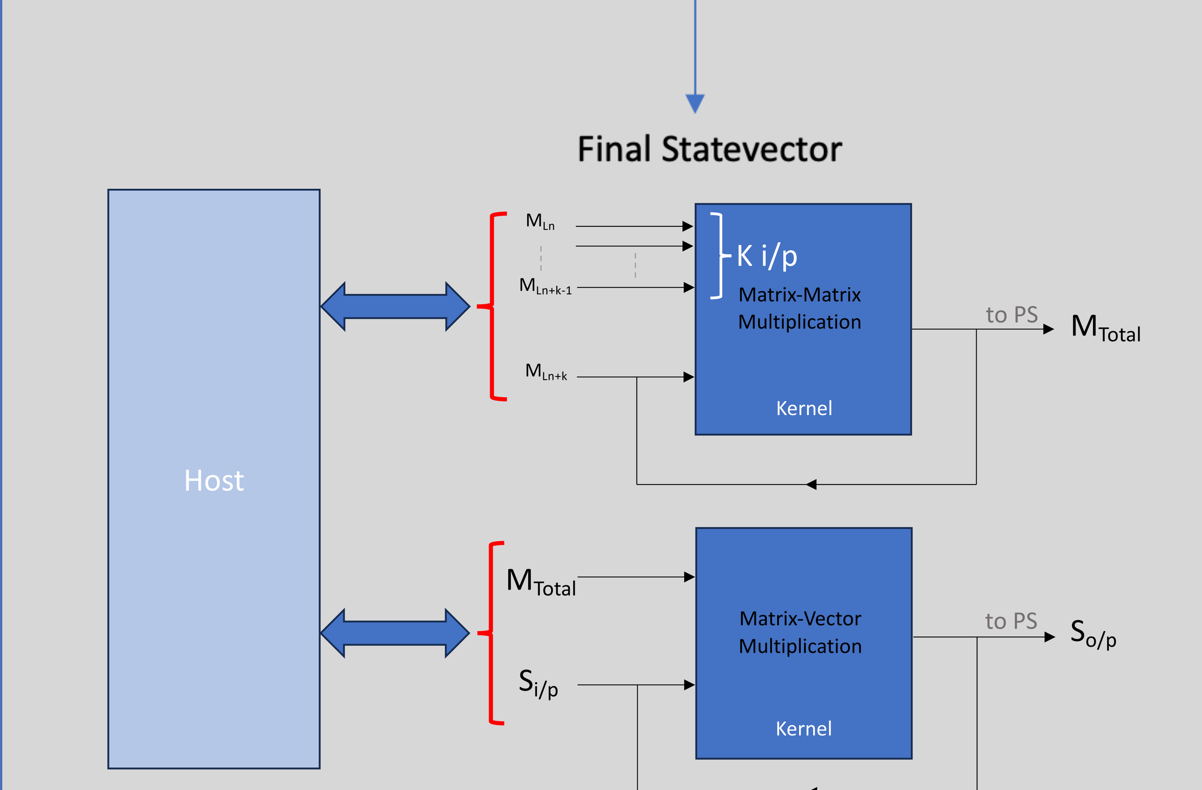


3. **Type-3 Design:** This design offers the highest throughput and acceleration; however, it comes at the cost of increased memory utilization. The approach involves passing k matrices into the buffer, where each pair of matrix computation results is stored in dedicated buffers. These intermediate results are then multiplied later to produce the resultant matrix comprising all the passed m matrices in a dataflow design. The resultant matrix is multiplied by the initial statevector.

$M_{Total} = M_{L1}\ M_{L2}\ M_{L3}\ M_{L4}\ M_{L5}\ M_{L6}\ M_{L7}\ M_{L8}$

M_{int1} M_{Int2}

M_{Total} ← [init Statevector]



Complexity Analysis

Type-1 Design:

Time Complexity: $(t_{avg} + t^c_{avg}) \times D$; D, depth of circuit

Space Complexity: $2^{n+5} + 2^{2n+4}$

$Average\ Computation\ time\ (t^c_{avg}) \rightarrow O(N^2); N, number\ of\ elements\ in\ matrix$

$t^c_{avg} =\ Avg\ time\ for\ data\ transfer$

Type-2 Design:

Time Complexity: $r.t_{avg} + D.t^c_{avg}$; $r = \frac{D}{K}$

Space Complexity: $2^{n+5} + K.2^{2n+4}$

$Average\ Computation\ time\ (t^c_{avg}) \rightarrow O(N^2); N, number\ of\ elements\ in\ matrix$

$t^c_{avg} =\ Avg\ time\ for\ data\ transfer$

Type-3 Design:

Time Complexity: $(t^c_{avg}.\log_2 K + t_{avg}) \times r + t^{c'}$; $r = \frac{D}{K}$; $K = 2, 4, 8, 16 - - - -$

Space Complexity for max parallelism: $K\left[\frac{K}{2} + 1\right].2^{2n+4}$

Matrix-Matrix Multiplication $time\ (t^c_{avg}) \rightarrow O(N^3)$

$Matrix - Vector\ Multiplication\ time\ (t^{c'}) \rightarrow O(N^2)$

$N, number\ of\ elements\ in\ matrix$

$t^c_{avg} =\ Avg\ time\ for\ data\ transfer$

Experimental Results

Number of Qubits	FPGA Kernel Execution Time (ms)	LUT%	Register%	BRAM%	DSP%	Software Simulation Time
3	0.012	0.52	0.46	1.06	0.16	7.16
5	0.131	0.51	0.47	1.25	0.16	12
7	1.923	0.52	0.46	3.89	0.16	37.1

Conclusions

- FPGAs can be used for efficient emulation of quantum algorithms, however mapping quantum circuits to FPGA emulation architectures is challenging
- The proposed automation framework facilitates the mapping of quantum circuits to FPGA emulation architectures.
- Experimental results include implementation on Xilinx Alveo U-200 FPGA [7]. Compared to state-of-the-art software simulator[7], speedup of almost up to ×100.
- Future work includes more implementations and optimizations.

References

1) Yee Hui Lee, Mohamed Khalil-Hani, Muhammad Nadzir Marsono, et al. An fpga-based quantum computing emulation framework based on serial-parallel architecture. International Journal of Reconfigurable Computing, 2016, 2016
2) Yunpyo Hong, Seokhun Jeon, Sihyeong Park, and Byung-Soo Kim. Quantum circuit simulator based on fpga. In 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), pages 1909–1911, 2022
3) Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. arXiv preprint arXiv:1707.03429, 2017.
4) Philippe Coussy and Adam Morawiec. High-level synthesis, volume 1. Springer, 2010
5) Marc Bataille. Quantum circuits of cnot gates. arXiv preprint arXiv:2009.13247, 2020.
6) https://www.xilinx.com/products/boards-and-kits/alveo/u200.htmloverview
7) IBM Simulators https://quantumcomputing.ibm.com/lab/docs/iql/manage/simulator.