Tensor Networks for Simulating Quantum Circuits on FPGAs

Maksim Levental

May 29, 2021

Abstract

Most research in quantum computing today is performed against simulations of quantum computers rather than true quantum computers. Simulating a quantum computer entails implementing all of the unitary operators corresponding to the quantum gates as tensors. For high numbers of qubits, performing tensor multiplications for these simulations becomes quite expensive, since N-qubit gates correspond to 2^N-dimensional tensors. One way to accelerate such a simulation is to use field programmable gate array (FPGA) hardware to efficiently compute the matrix multiplications. Though FPGAs can efficiently perform tensor multiplications, they are memory bound, having relatively small block random access memory. One way to potentially reduce the memory footprint of a quantum computing system is to represent it as a tensor network; tensor networks are a formalism for representing compositions of tensors wherein economical tensor contractions are readily identified. Thus we explore tensor networks as a means to reducing the memory footprint of quantum computing systems and broadly accelerating simulations of such systems.

Contents

1 Introduction

Quantum computing (QC) refers to the manipulation and exploitation of properties of quantum mechanical (QM) systems to perform computation. QM systems exhibit properties such as superposition and entanglement and clever *quantum algorithms* operate on these systems to perform general computation. Unsurprisingly, the technique was intially conceived of as a way to simulate physical systems themselves:

> "... [N]ature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy."

This closing remark from the keynote at the 1st Physics of Computation Conference in 1981, delivered by the late Richard Feynman [10], succinctly, but accurately, expresses that initial goal of quantum computing. Although modeling and simulating physical systems on quantum computers remains a thriving area of research we narrow our focus here to QC as it pertains to solving general computational problems. Such problems include unstructured search [16], integer factorization [31], combinatorial optimization [9], and many others. It is conjectured that some quantum algorithms enable quantum computers to exceed the computational power of classical computers [39].

QC systems are composed of so-called quantum bits, or *qubits*, that encode initial and intermediate states of computations. Transformations between states are effected by time-reversible transforms, called *unitary operators*. A formalism for representing quantum computation is the *quantum circuit* formalism, where semenatically related collections of N qubits are represented as *registers* and transformations are represented as *gates*, connected to those registers by *wires*, and applied in sequence. As already mentioned, in hardware, quantum circuits correspond to physical systems that readily exhibit quantum mechnical properties; examples of physical qubits include transmons, ion traps and topological quantum computers [26]. Current state of the art QC systems are termed Noisy Intermediate-Scale Quantum (NISQ) systems. Such systems are characterized by moderate quantities of physical qubits (50-100) but relatively few logical qubits (i.e. qubits robust to inteference and noise). Due to these limitations (and, more broadly, the relative scarcity of functioning QC systems), most research on quantum algorithms is performed with the assistance of simulators of QC systems. Such simulators perform simulations by representing N-qubit circuits as 2^N-dimensional complex vectors and transformations on those vectors as 2^N-dimensional complex matrix-vector multiplication. Naturally, due to this exponential growth, naively executing such simulations quickly become infeasible for $N > 50$ qubits [27], both due to memory constraints and compute time.

It's the case that matrices are a subclass of a more general mathematical object called a *tensor* and composition of matrices can be expressed as *tensor contraction*. *Tensor networks* (TNs) are decompositions (i.e. factorizations) of very high-dimensional tensors into networks (i.e. graphs) of low-dimensional tensors. TNs have been successfullly employed in reducing the memory requirements of simulations of QC systems [27]. The critical feature of tensor networks that make them useful for QC is the potential to perform tensor contractions on the low-dimensional tensors in an order such that, ultimately, the memory and compute time requirements are lower than for the traditional representation. Existing applications of TNs to quantum circuits focus primarily on memory constraints on general purpose computers [11] and distributed environments [22].

FPGAs are known to be performant for matrix multiplication uses cases [25]. Though FPGAs typically run at lower clock speeds (100-300MHz) than either conventional processors or even graphics processors they, nonetheless, excel at latency constrained computations owing to their fully "synchronous" nature (all modules in the same *clock domain* execute simultaneously). At first glance FPGAs seem like a suitable platform for performant simulation of quantum systems when runtime is of the essence. Unfortunately, RAM is one of the more limited resources on an FPGA and therefore it becomes necessary to explore memory reduction strategies for simulations (as well as runtime reduction strategies). Hence, we explore tensor networks as a means of reducing the memory footprint of quantum circuits with particular attention to dimensions of the designs space as they pertain to deployment to FPGAs.

The remainder of this report is organized as follows: section 2 covers the necessary background wherein subsection 2.1 very briefly reviews quantum computation and quantum circuits (with particular focus on aspects that will be relevant for tensor networks and FPGAs), section 2.2 defines tensors and tensor networks fairly rigorously and discusses algorithms for identifying optimal contraction orders, section 2.3 discusses the constraints imposed by virtue of deploying to FPGA, section 3 describes our implementation of TNs on FPGAs, section 4 reports our results on various random circuits, and section 5 concludes with future research directions.

2 Background

2.1 Quantum Computing

We very (very) quickly review quantum computing and quantum circuits as they pertain to our project. For a much more pedagogically sound introduction consult [17]. As already alluded to, quantum computing exploits

properties of quantum mechanical systems in order to perform arbitrary computation. The fundamental unit of quantum computation is a qubit, defined as two-dimensional quantum system with state vector ψ an element of a Hilbert space[1] H:

$$\psi := \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} \equiv \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. This exhibits the superposition property of the qubit[2] in that the squares of the coefficients are the probabilities of measuring the system in the corresponding basis state. Collections of qubits have state vectors that represented by the *tensor product* of the individual states of each qubit; for example, two qubits ψ, ϕ have state vector

$$\psi \otimes \phi := \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \otimes \begin{pmatrix} \alpha' \\ \beta' \end{pmatrix} \equiv \begin{pmatrix} \alpha\alpha' \\ \alpha\beta' \\ \beta\alpha' \\ \beta\beta' \end{pmatrix}$$

where the second $\otimes$ is the Kronecker product and $\alpha\alpha'$ indicates conventional complex multiplication. Note that the basis relative to which $\psi \otimes \phi$ is represented is the standard basis for $\mathbb{C}^4$ and thus we observe exponential growth in the size of the representation of an N-qubit system. An alternative notation for state vectors is Dirac notation; for example, for a single qubit

$$|\psi\rangle \equiv \alpha\,|0\rangle + \beta\,|1\rangle$$

and a 2-qubit system

$$\begin{aligned} |\psi\rangle \otimes |\phi\rangle &\equiv (\alpha\,|0\rangle + \beta\,|1\rangle) \otimes (\alpha'\,|0\rangle + \beta'\,|1\rangle) \\ &\equiv \alpha\alpha'\,|0\rangle \otimes |0\rangle + \alpha\beta'\,|0\rangle \otimes |1\rangle + \beta\alpha'\,|1\rangle \otimes |0\rangle + \beta\beta'\,|1\rangle \otimes |1\rangle \\ &\equiv \alpha\alpha'\,|0\rangle\,|0\rangle + \alpha\beta'\,|0\rangle\,|1\rangle + \beta\alpha'\,|1\rangle\,|0\rangle + \beta\beta'\,|1\rangle\,|1\rangle \\ &\equiv \alpha\alpha'\,|00\rangle + \alpha\beta'\,|01\rangle + \beta\alpha'\,|10\rangle + \beta\beta'\,|11\rangle \\ &\equiv \alpha\alpha'\,|0\rangle + \alpha\beta'\,|1\rangle + \beta\alpha'\,|2\rangle + \beta\beta'\,|3\rangle \end{aligned}$$

where in the last line we've used the decimal representation for the bit strings identifying the basis states. Of particular import for QC are the *entangled* or *bell states*; they correspond to multi-qubit states, such as

$$|\xi\rangle = \frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle$$

that cannot be "factored" into component states[3]. Then, naturally, changes in qubit states are represented as unitary[4] matrices U; for example

$$\psi' = U\psi = \begin{pmatrix} U_{00} & U_{01} \\ U_{10} & U_{11} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} U_{00}\alpha + U_{01}\beta \\ U_{10}\alpha + U_{11}\beta \end{pmatrix}$$

Matrix representations of transformations on multi-qubit states are constructed using the Kronecker product on the individual matrix representations; for example

$$U \otimes V := \begin{pmatrix} U_{00}V & U_{01}V \\ U_{10}V & U_{11}V \end{pmatrix} := \begin{pmatrix} U_{00}V_{00} & U_{00}V_{01} & U_{01}V_{00} & U_{01}V_{01} \\ U_{00}V_{10} & U_{00}V_{11} & U_{01}V_{10} & U_{01}V_{11} \\ U_{10}V_{00} & U_{10}V_{01} & U_{11}V_{00} & U_{11}V_{01} \\ U_{10}V_{10} & U_{10}V_{11} & U_{11}V_{10} & U_{11}V_{11} \end{pmatrix}$$

Here we see again an exponential growth in representation size as a function of number of qubits.

[1] A Hilbert space H is a vector space augmented with an inner product such that, with respect to the metric induced by that inner product, all Cauchy sequences converge.

[2] We say that the qubit is in a superposition of the basis vectors/states.

[3] ξ cannot be factored because there is no solution to the set of equations (for $\alpha, \alpha', \beta, \beta'$)

$$\alpha\alpha' = \frac{1}{\sqrt{2}}, \quad \alpha\beta' = 0, \quad \beta\alpha' = 0, \quad \beta\beta' = \frac{1}{\sqrt{2}}$$

[4] A matrix U is unitary iff $UU^\dagger = U^\dagger U = I$, i.e. it is its own Hermitian conjugate or more simply if it is "self-inverse".

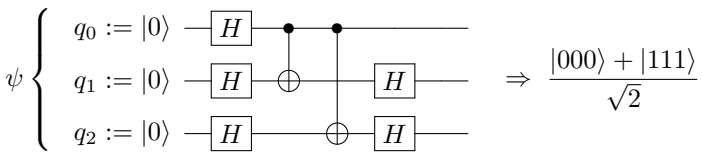


Figure 2.1: Quantum Circuit representing 3-qubit q_0, q_1, q_2 entanglement effected by application of successive Hadamard gates.

As already alluded to, quantum circuits are a formalism for representing quantum computation in general and algorithms designed for quantum computers in particular. In the quantum circuit formalism qubit states are represented by wires and unitary transformations are represented by gates (see figure 2.1), much like classical combinational logic circuits might be, though, whereas combinational logic is "memoryless"[5], sequences of quantum gates specified by a quantum circuit do in fact connote the evolution (in time) of the qubits. In addition quantum gates, as opposed to classical gates, are necessarily reversible and hence there are no quantum analogs to some classical gates such as NOT and OR.

2.2 Tensors and Tensor Networks

We quickly define tensors and tensor networks and then move on to tensor network methods for simulating quantum circuits.

2.2.1 Tensors

One definition of a tensor[6] T is as an element of a tensor product space[7]:

$$T \in \underbrace{V \otimes \cdots \otimes V}_{p \text{ copies}} \otimes \underbrace{V^* \otimes \cdots \otimes V^*}_{q \text{ copies}}$$

where V^* is dual[8] to V. Then T, in effect, acts a multilinear map

$$T : \underbrace{V^* \times \cdots \times V^*}_{p \text{ copies}} \times \underbrace{V \times \cdots \times V}_{q \text{ copies}} \to \mathbb{R}$$

by "applying" p elements from V to p elements of V^* and q elements from V^* to q elements of V. Note the swapping of the orders of V, V^* in both the definitions and the description. T's coordinate basis representation

$$T \equiv T^{i_1 \ldots i_p}_{j_1 \ldots j_q} \, \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_p} \otimes \mathbf{e}^{j_1} \otimes \cdots \otimes \mathbf{e}^{j_q} \tag{2.1}$$

is determined by its evaluation on each set of bases

$$T^{i_1 \ldots i_p}_{j_1 \ldots j_q} := T\left(\mathbf{e}^{i_1}, \ldots, \mathbf{e}^{i_p}, \mathbf{e}_{j_1}, \ldots, \mathbf{e}_{j_q}\right)$$

The pair (p, q) is called the *type* or *valence* of T while $(p + q)$ is the *order* of the tensor. **Note that we do not use rank to mean either of these things**[9]. Furthermore, eqn. (2.1) in fact represents a linear sum of basis elements, as it employs Einstein summation convention[10]. Note we make liberal use of summation

[5] The output of a combinational logic circuit at any time is only a function of the elements of the circuit and its inputs.

[6] There are several more at varying levels of mathematical sophistication. Chapter 14 of [28] is the standard reference. Ironically, it is this author's opinion that one should shy away from physics oriented expositions on tensors.

[7] The collection of tensor products of elements of the component spaces quotiented by an equivalence relation.

[8] The dual space to a vector space V is the vector V^* consisting of linear maps $f : V \to \mathbb{R}$. The dual basis of the dual space consists of f_i such that $f_i(\mathbf{e}_i) = \delta_{ij}$. It is convention to write f_i as $\mathbf{e}^i$ (note the superscript index).

[9] The *rank* of a tensor is the minimum number of distinct basis tensors necessary to define it; the tensor in eqn. (2.1) is in fact a rank 1 tensor. The definition is a generalization of the rank of a matrix (which, recalling, is the dimension of its column space, i.e. number of basis elements). Despite this obvious, reasonable definition for rank, one should be aware that almost all literature in this area of research uses rank to mean order.

[10] Repeated indices in juxtapose position indicate summation $a_i b^i := \sum_i a[i]b[i]$.

convention in the following but occasionally use explicit sums when it improves presentation (i.e. when we would like to emphasize a particular contraction).

There are two important operations on tensors we need to define. Firstly, we can form the tensor product Z of two tensors T, W, of types $(p, q), (r, s)$ respectively, to obtain a tensor of type $(p + r, q + s)$:

$$\begin{aligned}
Z &:= T \otimes W \\
&= \left(T^{i_1 \dots i_p}_{j_1 \dots j_q}\ \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_p} \otimes \mathbf{e}^{j_1} \otimes \cdots \otimes \mathbf{e}^{j_q}\right) \otimes \left(W^{k_1 \dots k_r}_{l_1 \dots l_s}\ \mathbf{e}_{k_1} \otimes \cdots \otimes \mathbf{e}_{k_r} \otimes \mathbf{e}^{l_1} \otimes \cdots \otimes \mathbf{e}^{l_s}\right) \\
&= \left(T^{i_1 \dots i_p}_{j_1 \dots j_q} W^{k_1 \dots k_r}_{l_1 \dots l_s}\ \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_p} \otimes \mathbf{e}_{k_1} \otimes \cdots \otimes \mathbf{e}_{k_r} \otimes \mathbf{e}^{j_1} \otimes \cdots \otimes \mathbf{e}^{j_q} \otimes \mathbf{e}^{l_1} \otimes \cdots \otimes \mathbf{e}^{l_s}\right) \\
&:= Z^{i_1 \dots i_{p+r}}_{j_1 \dots j_{q+s}}\ \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_{p+r}} \otimes \mathbf{e}^{j_1} \otimes \cdots \otimes \mathbf{e}^{j_{q+s}}
\end{aligned}$$

Despite it being obvious, its important to note that the tensor product Z produces a tensor of order $(p + r + q + s)$, i.e. higher than either of the operands. On the contrary, *tensor contraction* reduces the order of a tensor. We define the contraction Y of type (a, b) of a tensor T to be the "pairing" of the ath and bth bases:

$$\begin{aligned}
Y &:= T^{i_1 \dots i_p}_{j_1 \dots j_q}\ \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_{a-1}} \otimes \mathbf{e}_{i_{a+1}} \otimes \cdots \otimes \mathbf{e}_{i_p} \otimes \left(\mathbf{e}_{i_a} \cdot \mathbf{e}^{j_b}\right) \otimes \mathbf{e}^{j_1} \otimes \cdots \otimes \mathbf{e}^{j_{b-1}} \otimes \mathbf{e}^{j_{b+1}} \otimes \cdots \otimes \mathbf{e}^{j_q} \\
&= T^{i_1 \dots i_p}_{j_1 \dots j_q} \delta^{j_b}_{i_a}\ \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_{a-1}} \otimes \mathbf{e}_{i_{a+1}} \otimes \cdots \otimes \mathbf{e}_{i_p} \otimes \mathbf{e}^{j_1} \otimes \cdots \otimes \mathbf{e}^{j_{b-1}} \otimes \mathbf{e}^{j_{b+1}} \otimes \cdots \otimes \mathbf{e}^{j_q} \qquad \left(\text{since } \mathbf{e}_i \cdot \mathbf{e}^j = \delta^j_i\right) \\
&= \sum_{j_b} T^{i_1 \dots j_b \dots i_p}_{j_1 \dots j_b \dots j_q}\ \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_{a-1}} \otimes \mathbf{e}_{i_{a+1}} \otimes \cdots \otimes \mathbf{e}_{i_p} \otimes \mathbf{e}^{j_1} \otimes \cdots \otimes \mathbf{e}^{j_{b-1}} \otimes \mathbf{e}^{j_{b+1}} \otimes \cdots \otimes \mathbf{e}^{j_q} \\
&:= Y^{i_1 \dots i_{a-1} i_{a+1} \dots i_p}_{j_1 \dots i_{b-1} i_{b+1} \dots j_q}\ \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_{a-1}} \otimes \mathbf{e}_{i_{a+1}} \otimes \cdots \otimes \mathbf{e}_{i_p} \otimes \mathbf{e}^{j_1} \otimes \cdots \otimes \mathbf{e}^{j_{b-1}} \otimes \mathbf{e}^{j_{b+1}} \otimes \cdots \otimes \mathbf{e}^{j_q}
\end{aligned}$$

where $(\cdot)$ means inner product. Notice that the order of Y is $(p - 1, q - 1)$. Finally notice that we can omit writing out bases and just manipulate coordinates. We shall do as such when it simplifies presentation.

As mentioned in the introduction, matrices can be represented as tensors; for example, the two dimensional $N \times N$ matrix M is taken to be a tensor of type $(1, 1)$ with basis representation

$$M \equiv M^i_j\ \mathbf{e}_i \otimes \mathbf{e}^j$$

where upper indices correspond to the row index and lower indices correspond to the column index of the conventional matrix representation and both range from 1 to N. The attentive reader will notice that the coordinate representation of the tensor product for type $(1, 1)$ tensors is exactly the Kronecker product for matrices. Similarly, tensor contraction for type $(1, 1)$ tensors is the familiar matrix trace:

$$M^i_j \left(\mathbf{e}_i \cdot \mathbf{e}^j\right) = M^i_j \delta^j_i = \sum_{i=1}^{N} M^i_i$$

More usefully, we can express matrix-vector multiplication in terms of tensor contraction; let

$$\mathbf{x} := \begin{pmatrix} x^1 \\ x^2 \end{pmatrix} \equiv x^1 \mathbf{e}_1 + x^2 \mathbf{e}_2 \equiv x^i \mathbf{e}_i$$

where we switch to valence index notation in the column vector for closer affinity with tensor notation. Then it must be the case that

$$\mathbf{y} = M\mathbf{x} = \left(M^i_j \mathbf{e}_i \otimes \mathbf{e}^j\right)\left(x^k \mathbf{e}_k\right) = \left(M^i_j x^k \mathbf{e}_i\right)\left(\mathbf{e}^j \cdot \mathbf{e}_k\right) = M^i_j x^k \delta^j_k \mathbf{e}_i = M^i_j x^j \mathbf{e}_i$$

Letting $y^i := M^i_j x^j$ we recognize conventional matrix-vector multiplication. Employing tensor contraction in this way extends to matrix-matrix multiplication (and tensor composition more broadly); for two type $(1, 1)$ tensors M, L we can form the type $(1, 1)$ tensor Z corresponding to matrix product $M \cdot L$ of $N \times N$ by first taking the tensor product

$$Z^{ik}_{lj} := M^i_l L^k_j$$

The attentive reader will notice that the coordinate representation of two tensors is exactly the Kronecker product of two matrices. Then contracting along the off diagonal

$$Z^i_j := Z^{ik}_{kj} = M^i_k L^k_j \equiv \sum_{k=1}^{N} M^i_k L^k_j \tag{2.2}$$

One can confirm that this is indeed conventional matrix multipliation of two $N \times N$ matrices. In general, stated simply, when contracting indices of a tensor product, contraction can be understood to be a sum over shared indices.

2.2.2 Tensor Networks

Tensor networks (TNs) are a way to factor tensors with large orders into networks of tensors with lower orders; since the number of parameters a tensor consists of is exponential in the order of the tensor, smaller order tensors are much preferrable computationally. They were first used to study ground states of one dimensional quantum many-body systems [36] but have since been applied in other areas (such as machine learning [13]). TNs lend themselves to a diagrammatic representation which can be used to reason about such factorizations (figure 2.2a). We will primarily be interested in TNs as a means to factoring the state-vector of an N-qubit system (see figure 2.2b)

$$|\psi\rangle := \sum_{i_1 i_2 \ldots i_N} C^{i_1 i_2 \ldots i_N} |i_1\rangle |i_2\rangle \cdots |i_N\rangle \tag{2.3}$$

for which its common to propose an ansatz factorizations:

- **Matrix Product States (MPS)** [19], which yields factorization

  $$C^{i_1 i_2 \ldots i_N} \equiv A^{i_1}_{j_1} A^{i_1 j_1}_{j_2} \cdots A^{i_{N-1} j_{N-2}}_{j_{N-1}} A^{i_N j_{N-1}}$$

  where j are called *bond indices.* If each index i has dimension d (i.e. takes on values 1 to d) then C is specified by d^N parameters and can always be represented by an MPS factorization Ndm^2 parameters, where $m := d^{N/2}$ is the bond dimension. While for this naive representation $d^N < Ndm^2$, in practice m is fixed to some moderate size such that $d^N > Ndm^2$ and the MPS factorization functions as an approximation.

- **Projected Entangled Pair States (PEPS)** [32], which is a generalization of MPS to higher spatial dimensions, i.e. TNs that correspond to lattices of contractions of tensors, which themselves represent pairwise entangled quantum systems. Naturally, such a series of contractions doesn't lend itself to being expressed in traditional notation and thus we observe the power of tensor network diagrams (see PEPS in figure 2.2c).

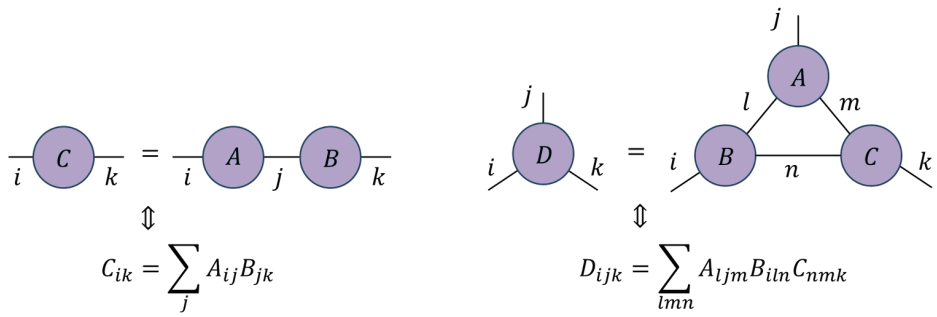
- **Tree Tensor Networks (TTN)** [30], a further generalization where tensors are entangled (and therefore contracted) hierarchically. In fact TTNs bear the closest resemblance to quantum circuits.

- **Multi-scale Entanglement Renormalization Ansatz (MERA)** [33], a specific type of TTN where the tensors are alternatingly unitaries and isometries[11].

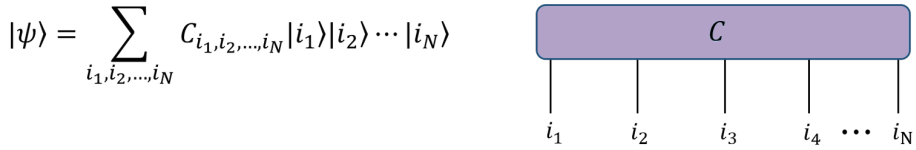
2.2.3 TNs for Simulating Quantum Circuits

Factoring eqn. (2.3) is only the first step to successfully simulating a quantum circuit. By representing some final state as a tensor as well, and contracting across all indices (called *contracting the network*), we can calculate the amplitude for that particular state. Since tensor contraction is associative[12], the order in which tensors are actually contracted is a "hyperparameter" of TN methods; finding the optimal contraction order, with respect to accuracy (assuming some approximation has been made in constructing the factorization), compute time, and memory requirements is critical.

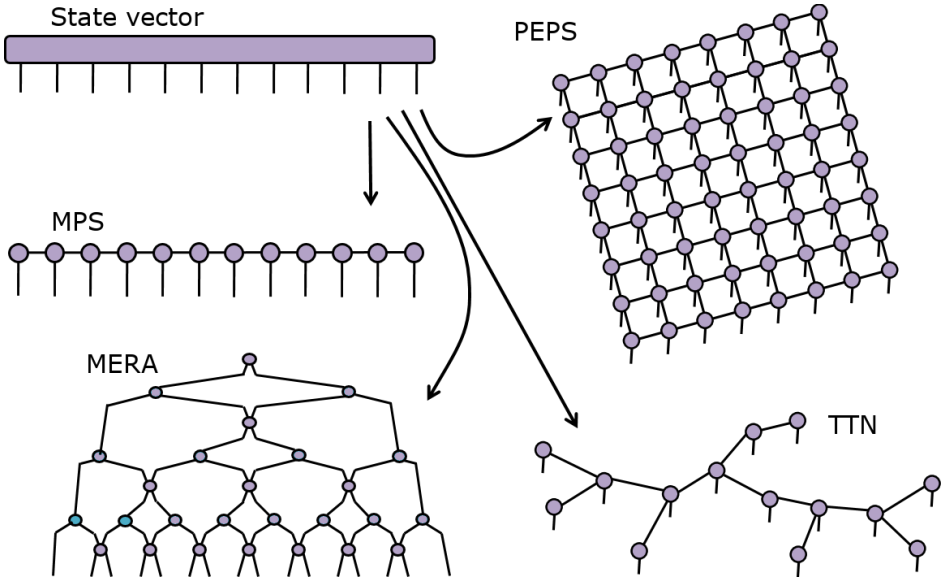
[11] A tensor, seen as a multlinear map, that preserves distances under the ambient distance metric.
[12] This can be observed by noting that summing is an associative operation (or by analog with matrix-matrix multiplication).



(a) Contraction



(b) State vector representation



(c) State vector factorization

Figure 2.2: Tensor network diagrammatic contraction. [8]

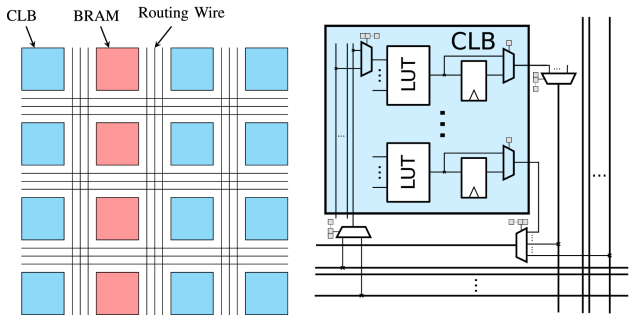


Figure 2.3: FPGA floorplan diagram [24].

In particular we focus on contraction orders for TTNs as they most closely resemble quantum circuits. For a TTN consisting of N tensors (corresponding to N gates) with maximum order p, worst case, we can see that contraction time takes $O\left(N \exp\left(O\left(p\right)\right)\right)$ since, in general, contracting across all indices of a pair of tensors is exponential in their orders[13]. Markov et al. [21] showed that there in fact exists a contraction ordering which results in a contraction time of $O\left(N^{O(1)} \exp\left(O\left(\text{tw}\left(G^L\right)\right)\right)\right)$ where G^L is the line graph[14] of the tensor network and $\text{tw}\left(G^L\right)$ is the tree-width[15] of G^L. For quantum circuits consisting of many few qubit gates this technique produces a much more (runtime) efficient evaluation of the circuit; indeed Markov et al. further show that any TTN corresponding to a quantum circuit with N gates, where the number of gates that act on any pair of qubits is bounded by r, has contraction time $O\left(N^{O(1)} \exp\left(O\left(r\right)\right)\right)$.

Markov et al.'s results are not tight; their construction finds some tree-decomposition with the correct corresponding tensor contraction order that suits their aim (overall runtime complexity of the translation from quantum circuit to TTN and the ultimate contraction). In reality there are often contraction orders that are much more space and runtime efficient. Though, in general problem is NP-hard [3], for particular TTNs (corresponding to circuits) there are heuristics, such as non-adjacent contractions [27], that produce more efficient orders. Alternatively, randomized search and Bayesian optimization can be used to identify efficient contraction orders [15, 11].

2.3 FPGAs

A field-programmable gate array (FPGA) is a device designed to be configured by a user into various arrangements of (classical) gates and memory. FPGAs consist of arrays (hence the name) of configurable logic blocks (CLBs), static ram (SRAM), and programmable busses that connect CLBs and SRAM into various topologies (see figure 2.3). The CLBs typically contain arithmetic units (such as adders, multipliers, and accumulators) and lookup tables (LUTs), that can be programmed to represent truth tables for many boolean functions. Using hardware description languages (such as VHDL or Verilog) designers specify modules and compose them into circuits (also known as a *dataflows*) that perform arbitrary computation. These circuits then go through a *place and route* procedure before ultimately being instantiated on the FPGA as *processing*

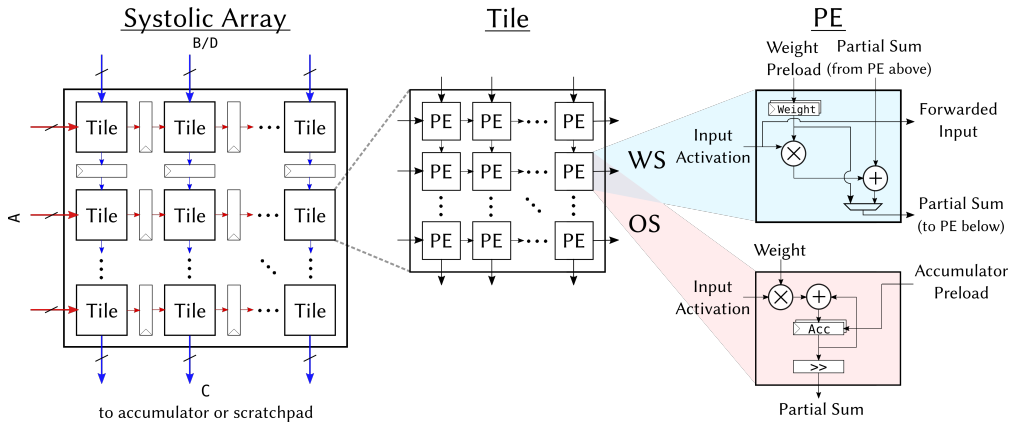
[13] Consider contracting two $(1, 1)$ tensors (as in eqn. (2.2)), i.e. two order 2 tensors, which effectively is matrix multplication followed by trace. The complexity of this contraction is then $O\left(N^{2+1} + N\right) \equiv O\left(\exp\left(2 \log N\right)\left(1 + N\right)\right)$ (where N here is the characteristic dimension of the matrix). Assuming the ranges of all tensor indices is the same (i.e. N is constant across all tensors), for example $N = 2$ as in the case of matrices derived of unitary transformations operating on single qubits, we recover the stated complexity.

[14] A *line graph* captures edge adjacency; given a graph G, G^L is defined such that each edge of G corresponds to a vertex of G^L and two vertices are are connected in G^L if the edges in G that they correspond to are adjacent on the same vertex (in G).

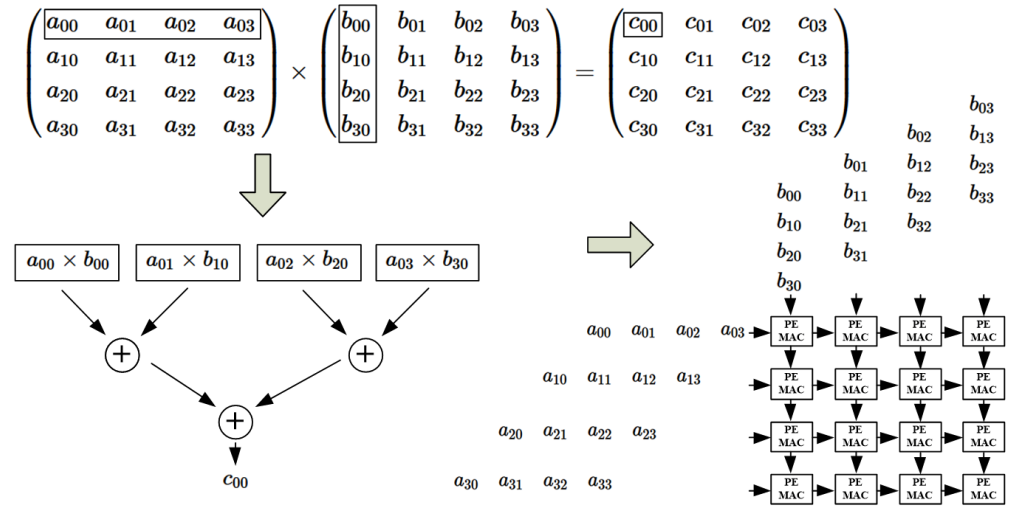
[15] A *tree decomposition* of a graph G is a tree T and a mapping from the vertices of G into "bags" that satisfy the following properties

1. Each vertex must appear in at least one bag.
2. For each edge in G, at least one bag must contain both of the vertices it is adjacent on.
3. All bags containing a given vertex in G must be connected in T.

The width w of a tree decomposition is the cardinality of the largest bag (minus one). Finally the *tree-width* of G is the minimum width over all possible tree decompositions. Intuitively, a graph has low tree-width if it can be constructed by joining small graphs together into a tree.



(a) Gemmini systolic array architecture. The processing elements (PEs) are either of type Weight Stationary (WS) or Output Stationary (OS). [12].



(b) Systolic array architecture implementing matrix multiplication. Input matrices A and B stream by to produce output matrix C via successive mutiply-accumulate (MAC) operations. Note that C remains in the processing elements (i.e. this is a diagram of an OS architecture). [38].

Figure 2.4: Systolic arrays

elements (PEs) and connections between PEs.

While modules consisting purely of combinational logic compute their outputs at the stated clock speed of the FPGA, inevitably I/O (i.e. fetching data from memory) interleaved with such modules (otherwise arranged into a pipeline architecture) creates pipeline stalls. Thus, it's essential that FPGA designs are as compute bound as possible (rather than I/O bound). In particular, we explore I/O minimal generalized matrix multiplication (GEMM) [7] and other *systolic array* architectures [34, 12]. A systolic architecture [20] is a gridded, pipelined, array of PEs that processes data as the data flows[16] through the array. Crucially, a systolic architecture propagates partial results as well as input data through the pipeline (see figure 2.4a). Systolic arrays are particularly suited for I/O efficient matrix multiplication owing to the pipelining of inputs (see figure 2.4b).

One remaining hurdle to simulating quantum compuations (i.e. carrying out tensor contractions) on FPGAs is SRAM. The standard remediation is to perform arithmetic with reduced precision[17]. There is

[16]The relationship to cardiovascular "systolic" is in association with the flow of data into the array, akin to how blood flows through the veins into the human heart.

[17]Germaine to this issue is the fact that arithmetic on FPGAs is typically performed in fixed precision (via an integer representation), owing to higher compute cost incurred for floating point arithmetic.

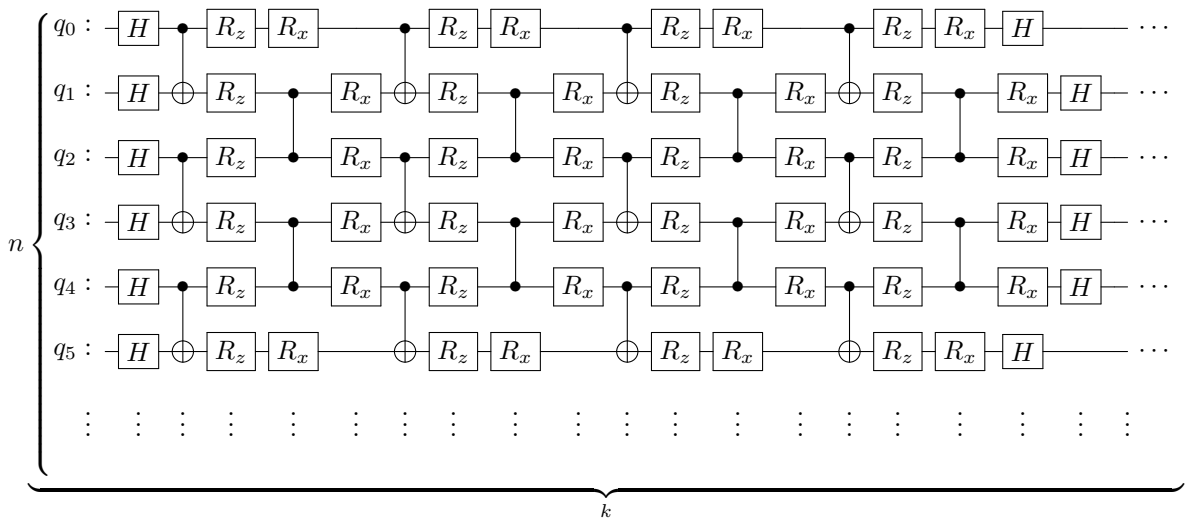


Figure 3.1: Test quantum circuit for n qubits and rounds k.

evidence that suggests that simulations of quantum circuits, of varying depths [6], are robust to reduced precision computation as long as that loss of precision is uncorrelated [37] i.e. insofar as it can be treated as uncorrelated noise.

3 Implementation

We use `quimb` [14] to specify quantum circuits and generate TNs therefrom. In particular we simulate circuits for various n qubits and *rounds* k where each consists of alternating qubit couplings according of the form in figure 3.1. We also use Bayesian parameter optimization (BPO) [15] to find tensor contraction orders and compare against naive greedy search. Note that for both strategy we set a timeout of 600 seconds. We then deploy the contraction strategy that produces the fewest number of tensor contractions balanced against the orders of intermediate tensors[18]. In order to expedite the process of deploying we precompute some first few tensor contraction such that all tensors deployed to the FPGA are square and congruent (i.e. all of the same dimensions). For tensors of order greater than $(1, 1)$ (i.e. tensors that are not matrices) we transform them into $(1, 1)$ tensors by taking the Kronecker product of all component $(1, 1)$ tensors; to be precise we perform the following operation on the (p, q) tensor

```
mats = [t[idx] for idx in np.ndindex(t.shape[:-2])]
block_mat = block_diag(*qubit_mats)
```

where `block_diag` builds a block diagonal matrix of its arguments. All of our code has been made available on GitHub[19].

For deploying circuits to FPGAs we use Chisel [4] as a HDL, by way of an adaptation of the Gemmini systolic array generator [12]. Notably we experiment with using Gemmini as an accelerator (i.e. fully parameterizing the weights/entries of the matrices) and "hardcoding" certain gates/tensors. One possible advantage of the latter approach over the former is a reduction in loads from memory for the weights. The success of the chosen approach depends heavily on whether certain sequences of fixed gates can actually be pipelined or alternatively deployed in toto to the FPGA. We hypothesize that this might depend on the

[18] This choice was purely due to platform constraints in that large intermediate tensors could not be effectively simulated.
[19] https://github.com/makslevental/fpga_stuff/ on the `complexmatmul` branch.

	Systolic	Naive
LUTs	512	110,074
Registers	896	2,048
Pins	642	3,074
DSPs	32	232

Table 1: Synthesis and place and route for an Arria II GX for both systolic arrays and naive matmul.

depth and gate count of the circuits/TNs. See figure 3.2 for the netlists corresponding to our systolic array and matmul implementations. Note that (complex) arithmetic was done in 32 bit fixed precision for both implementations, with 28 bits allocated behind the binary point.

One challenge we faced was in deploying to real hardware[20]; unfortunately time and administrative challenges[21] prevented us from actually deploying to real FPGAs. As a substitute we used the well-known and trusted Verilog simulator[22] Verilator[23], which transpiles Verilog (which Chisel generates) to a cycle-accurate model in C++. We then executed this model to collect proxy measurements. Note that for certain configurations (generally those with high qubit and round count) we could successfully simulate due to memory constraints on the workstation running the Verilator produced model.

4 Evaluation

We perform two sets of evaluations. Even though it was not the central goal of our exploration we first compare the time required to compute a tensor contraction strategy across n qubits and rounds k for the greedy search strategy and the BPO search strategy. We then address our central goal in comparing the actual runtime for performing the discovered tensor contraction on both the systolic array implementation (see fig. 3.2b) and the "hardcoded" naive matmul (see fig. 3.2c).

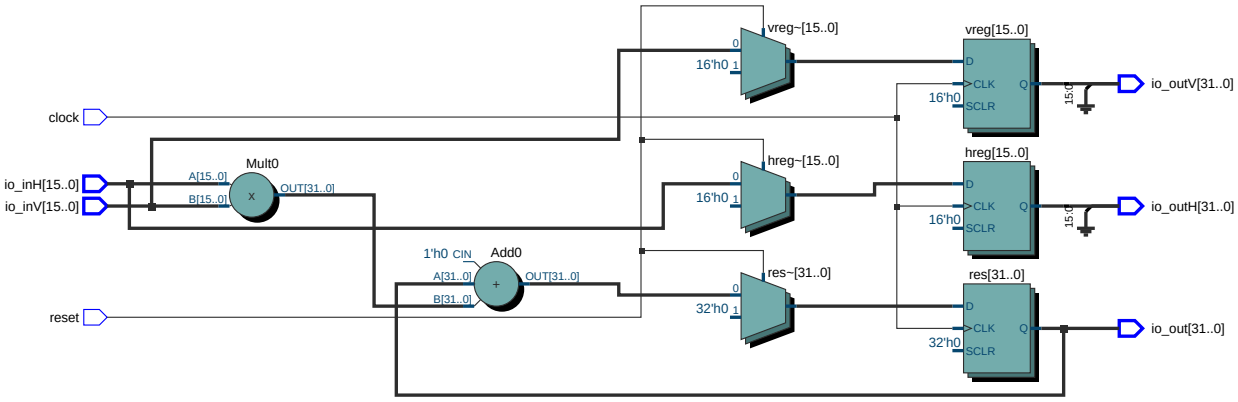
Some interesting things to note searching for contractions: computing (not evaluating) the optimal contraction strategy (i.e. using BPO) is generally more performant that greedy search (see figures 4.1a, 4.1b). The likely reason for this is that BPO converges more quickly and more efficiently searches the space of possible contraction orders than greedy search (which greedily optimizes some surrogate objective). Also note that, in fact, for certain configurations greedy didn't converge within the timeout.

Regarding the differences in the evaluation times of the contraction orders (figures 4.2a, 4.2b) it's clear that the systolic array implementation is more performant in terms of both memory requirements and runtime. This is paradoxically both obvious and suprising. As already mentioned, one expects systolic arrays to have improved performance relative to naive matrix multiplication for streaming data (and indeed, in general, they do) but for this use case (where all matrix elements are known at deploy time) one also expects that latency incurred by pipelining would offset that performance improvement. One hypothesis for this is that the difference is an artifact of simulating the FPGA implementations insofar as simulating a more densely connected FPGA implementation (see the differences between 3.2b and 3.2c) is more compute intensive, especially with respect to heap allocations (since systolic arrays incur more loads from memory). To corroborate this hypothesis we used Intel's Quartus EDA[24] tool to synthesize and place and route (for an Arria II GX). Indeed the naive implementation an order of magnitude (sometimes several) of each type of resource (see table 1).

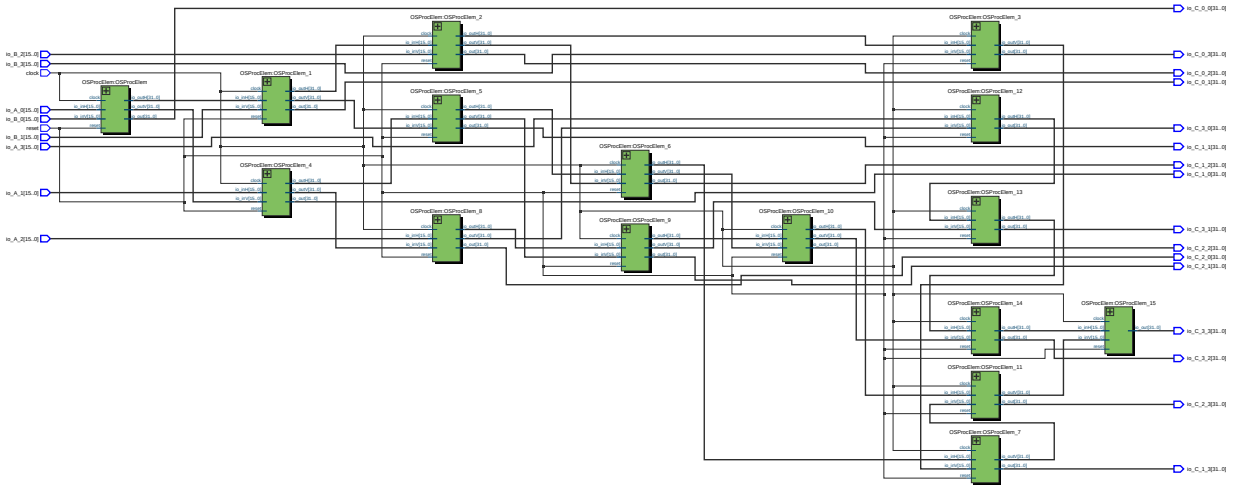
5 Conclusion

We explored tensor networks deployed to FPGAs as a mean of accelerating simulations of quantum circuits. In order to accomplish this goal we expressed tensor contraction as sequences of matrix multiplication and

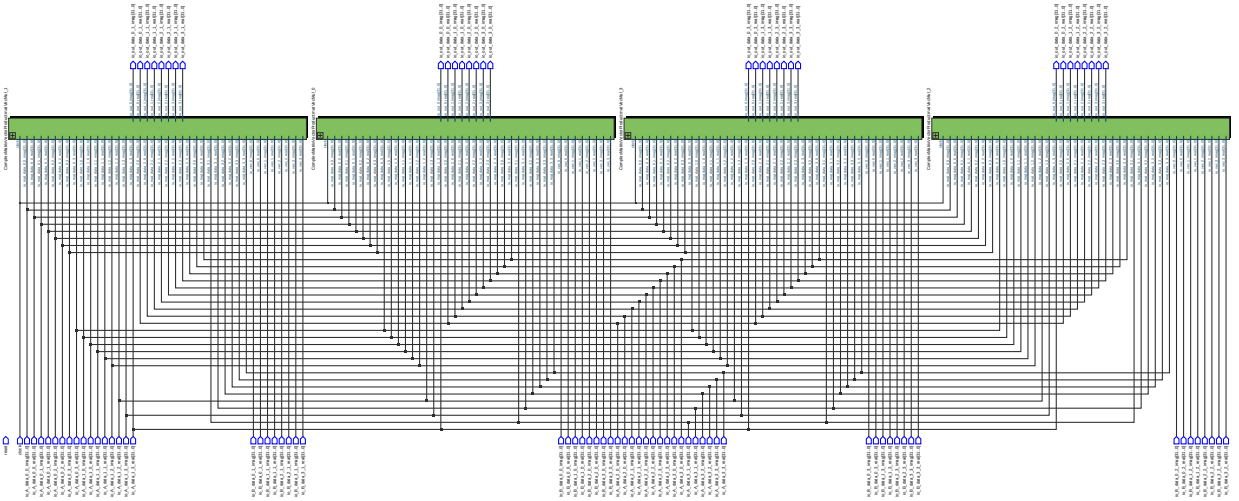
[20] A challenge not unfamiliar to the seasoned QC researcher.
[21] We were not able to get allocations on CHI@TACC in a timely fashion (the issue is ongoing...).
[22] It is simulations all the way down.
[23] https://www.veripool.org/verilator/
[24] Electronic design automation.



(a) Output stationary systolic array processing element.

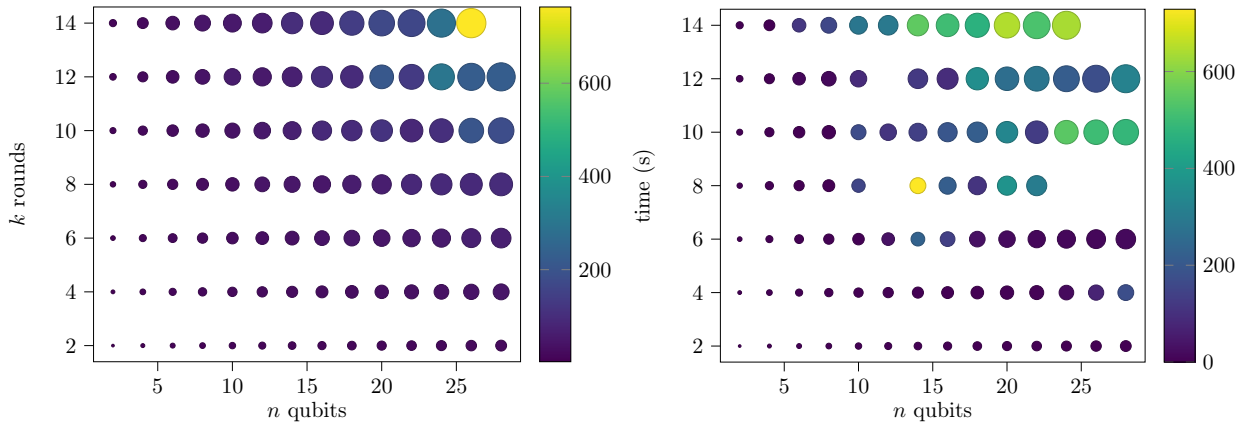


(b) 4×4 matrix multiplication systolic array implementation.



(c) Naive 4×4 matrix multiplication implementation.

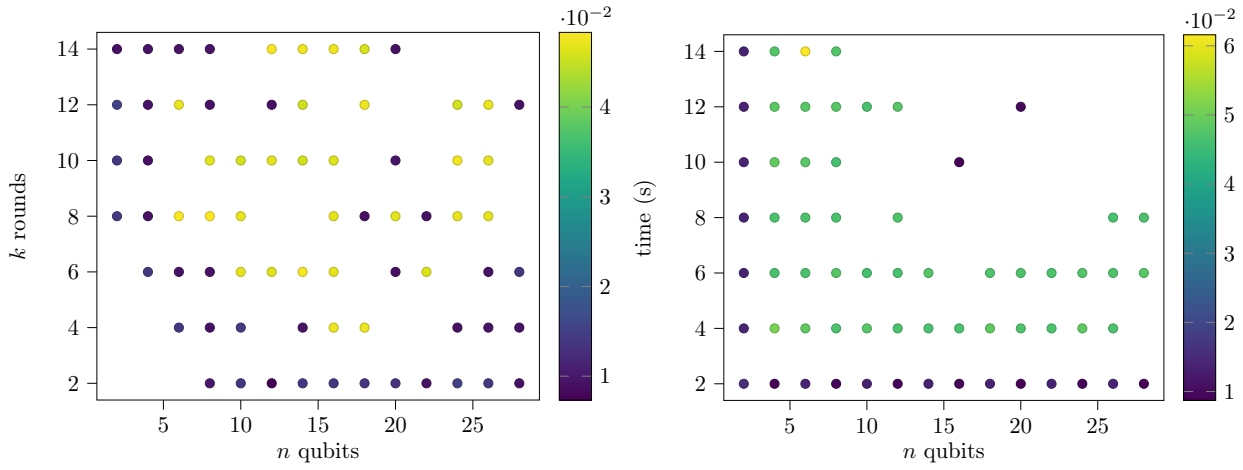
Figure 3.2: FPGA implementations



(a) Runtimes for computing BPO contraction strategy.

(b) Runtimes for computing greedy contraction strategy.

Figure 4.1: Runtimes for computing contraction strategies for circuits for various n qubits and rounds k. Color scale represents time and marker size represents, qualitatively, the number of tensors in the tensor network corresponding to the quantum circuit. Note that absent markers correspond to searches that didn't converge.



(a) Runtimes for contracting with systolic array.

(b) Runtimes for contracting with matmul.

Figure 4.2: Runtimes for test circuits for various n qubits and rounds k. Note that absent markers correspond to contractions that could not be simulated due to memory constraints imposed by using the Verilator model rather than an actual FPGA.

implemented two different matrix multiplication FPGA designs: systolic arrays, which operate on streaming matrix elements and naive matrix multiplication, which wholesale instantiates all the necessary MAC operations. In order to choose the contraction orders we used an "off the shelf" library which searches for a suitable contraction by either performing greedy search or Bayesian optimization. We compared the performance of both the contraction search strategy and each contraction evaluation implementation. Unfortunately we were unable to obtain acecss to FPGA devices and thus we made due with cycle-accurate simulations. Results for both comparison were generally in agreement with intuition: BPO converged to a contraction order more effectively (more quickly and more robustly) than greedy search and systolic arrays evaluated the contraction more efficiently than naive matrix multiplication.

Possible future work includes actually deploying to real FPGAs and then further comparing performance to the simulations performed here. Another particularly interesting research direction is the tangential problem of discovering optimal tensor contraction orders. Finding such tensor contraction orders is ultimately a combinatorial optimization problem. It occurs to us that possibly a deep learning approach could be effective. Recently there has been work on RL for combinatorial optimization[5] and MCTS for combinatorial optimization[2] that could, possibly, be adapted to this problem in a straightforward fashion.

References

[1] K. Abdelouahab, M. Pelcat, J. Sérot, C. Bourrasset, and F. Berry. Tactics to directly map cnn graphs on embedded fpgas. *IEEE Embedded Systems Letters*, 9(4):113–116, 2017.

[2] Kenshin Abe, Zijian Xu, Issei Sato, and Masashi Sugiyama. Solving np-hard problems on graphs with extended alphago zero, 2020. 14

[3] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM JOURNAL OF DISCRETE MATHEMATICS*, 8(2):277–284, 1987. 8

[4] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: Constructing hardware in a scala embedded language. In *DAC Design Automation Conference 2012*, pages 1212–1221, 2012. 10

[5] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3243–3250, Apr. 2020. 14

[6] Santiago I. Betelu. The limits of quantum circuit simulation with low precision arithmetic, 2020. 10

[7] Johannes de Fine Licht, Grzegorz Kwasniewski, and Torsten Hoefler. Flexible communication avoiding matrix multiplication on fpga with high-level synthesis. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, page 244–254, New York, NY, USA, 2020. Association for Computing Machinery. 9

[8] Glen Evenbly. Tensortrace: an application to contract tensor networks, 2019. 7

[9] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014. 2

[10] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6/7):467–488, 1982. 2

[11] E. Schuyler Fried, Nicolas P. D. Sawaya, Yudong Cao, Ian D. Kivlichan, Jhonathan Romero, and Alán Aspuru-Guzik. qtorch: The quantum tensor contraction handler. *PLOS ONE*, 13(12):e0208510, Dec 2018. 2, 8

[12] Hasan Genc, Ameer Haj-Ali, Vighnesh Iyer, Alon Amid, Howard Mao, John Wright, Colin Schmidt, Jerry Zhao, Albert Ou, Max Banister, Yakun Sophia Shao, Borivoje Nikolic, Ion Stoica, and Krste Asanovic. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures, 2019. 9, 10

[13] Ivan Glasser, Ryan Sweke, Nicola Pancotti, Jens Eisert, and J. Ignacio Cirac. Expressive power of tensor-network factorizations for probabilistic modeling, with applications from hidden markov models to quantum machine learning, 2019. 6

[14] Johnnie Gray. quimb: A python package for quantum information and many-body calculations. *Journal of Open Source Software*, 3(29):819, 2018. 10

[15] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, Mar 2021. 8, 10

[16] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. 2

[17] Abhijith J., Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, Andreas Bärtschi, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, David Gunter, Satish Karra, Nathan Lemons, Shizeng Lin, Alexander Malyzhenkov, David Mascarenas, Susan Mniszewski, Balu Nadiga, Daniel O'Malley, Diane Oyen, Scott Pakin, Lakshman Prasad, Randy Roberts, Phillip Romero, Nandakishore Santhi, Nikolai Sinitsyn, Pieter J. Swart, James G. Wendelberger, Boram Yoon, Richard Zamora, Wei Zhu, Stephan Eidenbenz, Patrick J. Coles, Marc Vuffray, and Andrey Y. Lokhov. Quantum algorithm implementations for beginners, 2020. 2

[18] Liancheng Jia, Liqiang Lu, Xuechao Wei, and Yun Liang. Generating systolic array accelerators with reusable blocks. *IEEE Micro*, 40(4):85–92, 2020.

[19] A Klümper, A Schadschneider, and J Zittartz. Matrix product ground states for one-dimensional spin-1 quantum antiferromagnets. *Europhysics Letters (EPL)*, 24(4):293–297, Nov 1993. 6

[20] Kung. Why systolic architectures? *Computer*, 15(1):37–46, 1982. 9

[21] Igor L. Markov and Yaoyun Shi. Simulating quantum computation by contracting tensor networks. *SIAM J. Comput.*, 38(3):963–981, June 2008. 8

[22] Alexander McCaskey, Eugene Dumitrescu, Mengsu Chen, Dmitry Lyakh, and Travis Humble. Validating quantum-classical programming models with tensor network simulations. *PLOS ONE*, 13(12):e0206704, Dec 2018. 2

[23] Thierry Moreau, Tianqi Chen, Luis Vega, Jared Roesch, Eddie Yan, Lianmin Zheng, Josh Fromm, Ziheng Jiang, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. A hardware–software blueprint for flexible deep learning specialization. *IEEE Micro*, 39(5):8–16, 2019.

[24] K. E. Murray, M. A. Elgammal, V. Betz, T. Ansell, K. Rothman, and A. Comodi. Symbiflow and vpr: An open-source design flow for commercial and novel fpgas. *IEEE Micro*, 40(04):49–57, jul 2020. 8

[25] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, and Guy Boudoukh. Can fpgas beat gpus in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, page 5–14, New York, NY, USA, 2017. Association for Computing Machinery. 2

[26] National Academies of Sciences Engineering and Medicine. *Quantum Computing: Progress and Prospects*. The National Academies Press, Washington, DC, 2019. 2

[27] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, Thomas Magerlein, Edgar Solomonik, Erik W. Draeger, Eric T. Holland, and Robert Wisnieff. Pareto-efficient quantum circuit simulation using tensor contraction deferral, 2020. 2, 8

[28] S. Roman. *Advanced Linear Algebra*. Graduate Texts in Mathematics. Springer New York, 2007. 4

[29] Justin Sanchez, Nasim Soltani, Pratik Kulkarni, Ramachandra Vikas Chamarthi, and Hamed Tabkhi. A reconfigurable streaming processor for real-time low-power execution of convolutional neural networks at the edge. In Shijun Liu, Bedir Tekinerdogan, Mikio Aoyama, and Liang-Jie Zhang, editors, *Edge Computing – EDGE 2018*, pages 49–64, Cham, 2018. Springer International Publishing.

[30] Y.-Y. Shi, L.-M. Duan, and G. Vidal. Classical simulation of quantum many-body systems with a tree tensor network. *Phys. Rev. A*, 74:022320, Aug 2006. 6

[31] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. 2

[32] F. Verstraete and J. I. Cirac. Renormalization algorithms for quantum-many body systems in two and higher dimensions. 7 2004. 6

[33] G. Vidal. Entanglement renormalization. *Phys. Rev. Lett.*, 99:220405, Nov 2007. 6

[34] Jie Wang, Licheng Guo, and Jason Cong. Autosa: A polyhedral compiler for high-performance systolic arrays on fpga. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, page 93–104, New York, NY, USA, 2021. Association for Computing Machinery. 9

[35] Pete Warden. Why gemm is at the heart of deep learning, Apr 2015.

[36] Steven R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69:2863–2866, Nov 1992. 6

[37] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic 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*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery. 10

[38] Zhijie Yang, Lei Wang, Dong Ding, Xiangyu Zhang, Yu Deng, Shiming Li, and Qiang Dou. Systolic array based accelerator and algorithm mapping for deep learning algorithms. In Feng Zhang, Jidong Zhai, Marc Snir, Hai Jin, Hironori Kasahara, and Mateo Valero, editors, *Network and Parallel Computing*, pages 153–158, Cham, 2018. Springer International Publishing. 9

[39] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020. 2