Master's Thesis:

GNN for Time-Sliced Quantum Circuit Partitioning

Author: Ruizhe Yu Xia

Supervisor: Sergi Abadal Cavallé - AC - UPC

Internal Examiner: Cecilio Angulo Bahón - ESAII - UPC

January 25th, 2022

Master in Artificial Intelligence

FACULTAT D'INFORMATICA DE BARCELONA (FIB)

UNIVERSITAT POLITECNICA DE CATALUNYA (UPC) – BarcelonaTech

FACULTAT DE MATEMATIQUES (UB)

UNIVERSITAT DE BARCELONA (UB)

ESCOLA TECNICA SUPERIOR D'ENGINYERIA (URV)

UNIVERSITAT ROVIRA i VIRGILI (URV)

GNN for Time-Sliced Quantum Circuit Partitioning

Abstract:

To scale into architectures that are big enough to achieve supremacy over classical computers, quantum computer designs will likely adopt a multi-core approach. These processors will have high connectivity within clusters and higher latency communications between cores. Previous work by Baker *et al.* developed a heuristic partitioning algorithm to schedule qubits into cores for quantum programs, for which a static control flow is known beforehand. This heuristic algorithm aims to reduce the number of inter-core movements, which are the main source of error and latency in these architectures. In the present thesis, we develop a pipeline using a Graph Neural Networks (GNN) to imitate this heuristic algorithm. We do so in hopes that GPU or GNN-specific hardware acceleration provide better scalability and parallelization in the near future, as quantum processors and algorithms move from hundreds to thousands or even millions of qubits. This work also paves the way for future work using GNN's for optimization of quantum circuit partitioning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

By leveraging properties of quantum mechanics such as entanglement and superposition, quantum computers are aiming to run a broad set of algorithms exponentially faster than classical computers, for problems that are currently intractable. Thus, if successful, a quantum computer will revolutionize fields that require extremely large amounts of computation: cryptography, optimization, simulation or artificial intelligence to name a few.

We are currently immersed in the Noisy Intermediate-Scale Quantum (NISQ) era [28]. At present, quantum computers have around 100 qubits (quantum analog of classical bits) in the upper end and can only run small versions of certain quantum algorithms, often not exceeding a few hundreds of quantum gates due to the noisy nature of the qubits [4, 1]. This is not enough to bring about the end of RSA encryption, which is one of the most prominent promises of quantum computing [14]. Despite remarkable advances towards reducing it, quantum decoherence is still the main hindrance of scaling quantum computers and algorithms to sizes that are able to show the *quantum supremacy* that [27] speaks of. Decoherence is a term that refers to the loss of quantum information due to unwanted interactions with the environment. This is why quantum processors are isolated, kept at cryogenic temperatures and controlled externally. Even with these preventive measures, certain errors scaling with the number of qubits or increased difficulty of integrating required control circuits will limit the size a single-chip architecture can achieve to a few thousands [25].

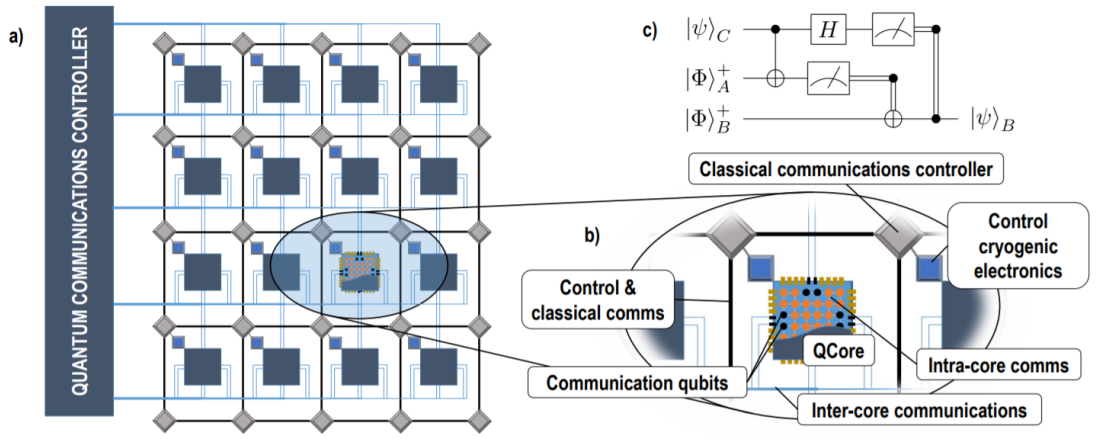

Figure 1.1: Adapted from [29] a) 2D diagram of a multi-core architecture. b) Enumeration of the components. c) Circuit for quantum teleportation.

Due to these challenges, quantum hardware is expected to scale via increasing the number of processors and not only the size of the processors, in a similar fashion to how classical computers have scaled. These multi-core quantum processors, would have high connectivity within each core and higher latency connections between cores, see Figure 1.1. For a multi-core processor of C cores with N qubits per core to retain the equivalent computational capability of a single processor of $N \times C$ qubits, the cores need to be connected via quantum-coherent links. For example, [24] proposes a way to connect ion traps (cores) via an optical switch, i.e. using photon-mediated teleportation.

Operations between qubits located at different cores are more costly than local operations, since the remote ones require multiple steps as shown in Figure 1.1. In particular, transfer of the quantum state of a qubit could entail (i) the distribution of entangled particles or pairs between transmitting and receiving cores, (ii) the execution of multiple quantum gates between one qubit of the entangled pair and the qubit to be transmitted, (iii) measurements of the entangled qubit at transmission, (iv) classical transmission of the measurement outcomes from transmitter to receiver, and (v) the application of single-qubit gates to the entangled bit at the receiving end. As a result, in the near term, these non-local communication operations between cores are expected have 5 to 100 times the latency of in-cluster communication [24]. Also, quantum teleportation has error rates 10 to 100 times larger than those of local two-qubit gates [6]. Thus, mapping qubits to cores while minimizing the number of non-local communication operations is key to a successful multi-core quantum

processor. Quantum program schedules can be represented as time-sliced interaction graphs where nodes represent qubits and each edge represents an interaction, i.e. qubits that need to be in the same core. Therefore, the problem of mapping quantum programs to a multi-core processor can be posed as a graph partitioning problem with a goal of minimizing edge crossings between partitions and, hence, minimizing movements between cores.

On the other hand, we have Graph Neural Networks (GNN). GNN's are able to operate on and learn from data that has graph structure. They have been gaining popularity due to their successes in many applications that involve graphs, such as molecular structure [15], social networks [33] or knowledge graphs [17] to name a few. As such, they pose a promising candidate for the task of mapping qubits to a multi-core processor.

In this work, we demonstrate the ability of GNN's to reduce iterations of a recent partitioning algorithm for time-sliced multi-core mapping [6]. The main contribution of this work is to serve as a stepping stone towards better optimization and scalability of qubit mapping methods using GNN. In particular, we postulate that as the number of qubits increases, the algorithm from [6] will not scale well due to its irregular and search-driven iterative nature and its operation over a large but probably very sparse qubit interaction graph. On the other hand, GNN acceleration is being researched intensely [2, 12] and promises hefty speedups using both software and hardware techniques, hence paving the way to more scalable qubit partitioning schemes for large-scale quantum computers. In this direction, the contributions of this thesis are:

- We implement a GNN that learns a baseline graph partitioning algorithm for quantum multi-core architectures [6].
- We validate the approach using well established quantum programs as benchmarks, obtaining solutions of similar optimality than the baseline.
- We evaluate the complexity of the algorithms and show the computational speedup that using our pipeline entails for the baseline approach.

The rest of the thesis is organized as follows. In Chapter 2, we introduce the basics of quantum circuits, GNN's and graph partitioning. In Chapter 3, we state

the goals we set out to accomplish with this work and outline our contributions. In Chapter 4, we introduce the benchmarks we test on, the circuits we use to train our model and present our proposed solution. In Chapter 5, we present our results and provide a brief discussion. In Chapter 6, we conclude with some final remarks and lines for future work.

Chapter 2

Background

In this chapter, we provide background on the different fields that the present thesis touches upon. In particular, Section 2.1 outlines the theory behind GNN's and their characteristics. Then, in Section 2.2, we describe how quantum programs are expressed and the role of qubit movement in the execution of such programs. Then, Section 2.3 mentions a few approaches of graph partitioning as a way to facilitate the mapping of qubits in multi-core quantum architectures, to then describe the solution from [6] used as baseline in this thesis in Section 2.4.

2.1 Graph Neural Networks

In this section, we provide a brief introduction of Graph Neural Networks (GNN) adjusted to the one used in this thesis. For a more in depth description of GNN's refer to specialized texts, e.g. [18, 2].

Formally, a graph is a pair $G = (V, E)$ where V is the set of vertices or nodes and E is a set of pairs of nodes. Nodes represent objects and edges are relations between them. Graphs can be directed or undirected, depending on whether the relations represented by edges are unidirectional or bidirectional. One example of the former might be the CORA dataset [31] where nodes represent published papers and the edges represent the first node (paper) citing the second node (paper). An example of the latter would be a social network, where nodes represent people and edges represent the friendship relation.

GNN is used to refer to a model that leverages neural networks to ingest data with graph structure and make inferences. Traditional neural networks like Multi Layer Perceptrons (MLP) or Convolutional Neural Networks (CNN) usually deal with data in a real euclidean space $\mathbb{R}^n$ or isomorphic to one such space (images, vectors, ...). Others are suited for temporal sequences of varying length such as Recurrent Neural Networks (RNN). However, it is not obvious how to extend these models to deal with graph data.

Some first attempts to deal with graph data were to extract features from the graph live in a representation space that is of the type of traditional neural networks, i.e. a fixed size real vector. This approach, however, does not explicitly make use of the graph structure and may suffer from generalization issues. Indeed, graphs that come the way of the model may have different number of nodes or node degree distribution than those used to extract representations. Therefore, the features that are extracted must be graph topology-invariant in order to fit in to the same size feature vector, so the graph structure needs to be lost in the process.

To deal with these issues, GNN's were born. GNN's is a very active area of research and thus, a rapidly evolving field. In [34], the authors survey the field, finding applications in supervised, unsupervised and semi-supervised learning setting. They also find a wide range of extensions to GNN's from "traditional" deep learning, such as attention, recurrent GNN, Graph autoencoders among others. GNN can be designed to deal with many different types of graphs: multiplex graphs, hypergraphs, dynamic graphs and more. See figure 2.1 for an overview of GNN's by type and scale.

In this work we will focus our attention to Message Passing Neural Networks [15] (MPNN). See figure 2.2 for a diagram of information flow for these networks.

To achieve this information flow, a hidden state $\mathbf{h}_v^0 \in \mathbb{R}^n$ is assigned to each node. These states are initialized with node features. For example, in the case of molecular data, these features can be physical properties of the atom.

Then we enter the message passing stage, which is repeated T times. For each step $t+1$, each node u passes messages to its neighbors and each node v aggregates its received messages into $\mathbf{m}_v^{t+1}$

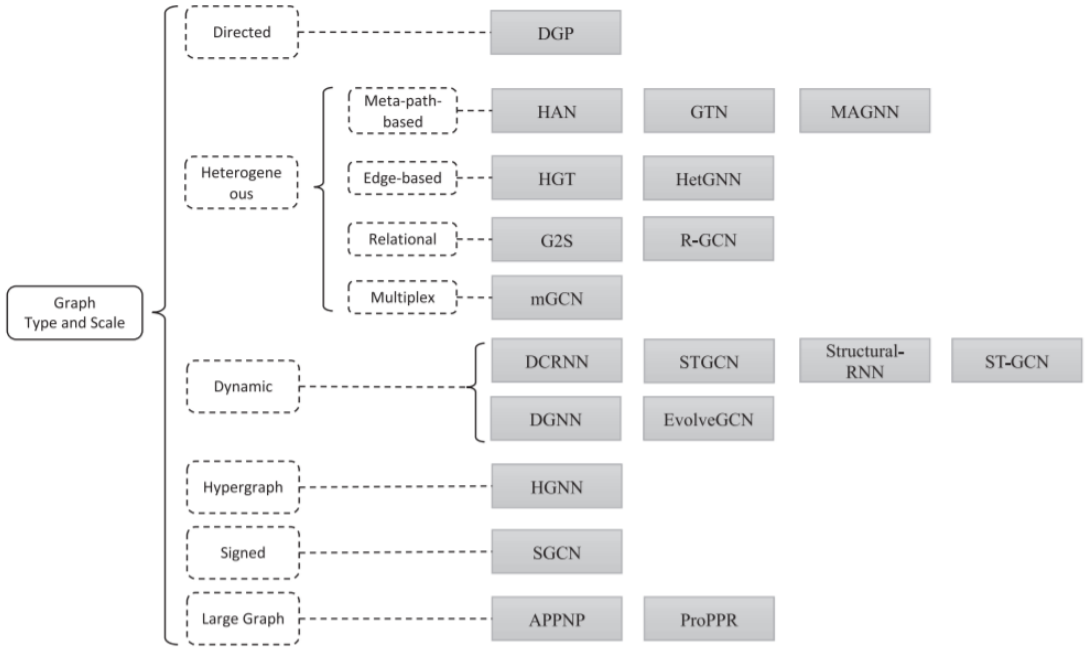

Figure 2.1: Adapted from [34]. An overview of GNN variants considering graph type and scale.

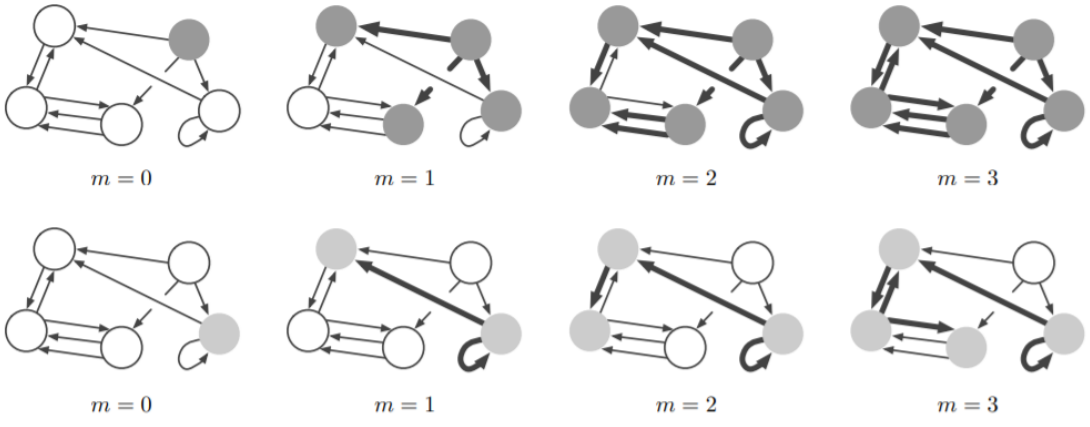

Figure 2.2: Adapted from [7] Example of message passing. Each row highlights the information that diffuses through the graph starting from a particular node. Shaded nodes indicate how far information from the original node has traveled; bolded edges indicate edges through which that information can travel. This propagation of information happens simultaneously for all nodes and edges.

$$\mathbf{m}_v^{t+1} = \frac{1}{|N_v|} \sum_{u \in N_v} M_t(\mathbf{h}_v^t, \mathbf{h}_u^t, \mathbf{e}_{u,v}) \tag{2.1}$$

where N_v is the set of neighbors and $\mathbf{e}_{uv}$ is a feature vector for the edge, for example atomic distance for molecular data. M denotes a learnable differentiable function that is referred to as message function. Each node uses this aggregated message to update its state

$$\mathbf{h}_v^{t+1} = U_t(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}) \tag{2.2}$$

where U_t is also a learnable differentiable function called update function.

After the T message passing steps, it is the turn of the readout phase. In this phase we extract the inference target using the readout function, again learnable and differentiable. Its form will change depending on the task at hand, sometimes we might want a prediction per node, so it may take each final hidden state $\mathbf{h}_v^{t+1}$ and produce a prediction. We may also want to have a prediction for the whole graph, in that case it will take all final hidden states as input.

As one can imagine from the above description, there is a fair amount of freedom to design a MPNN. The above description is not even fixed. For example, one may want to use 2-hop neighbors for eligible message senders to each node. In Section 4 we will introduce our own modification to deal with different types of edges.

2.2 Quantum Programs and Interaction Graphs

As mentioned in the introduction, qubits are the quantum analog to classical bits. Thus, they constitute the basic unit of quantum information. Their state is described by a superposition of the basis states $|0\rangle$ and $|1\rangle$. Therefore, their possible states live in a continuous space, unlike classical bits, which live in a boolean space. To make operations with qubits, one uses quantum gates, which are reversible operations that may act on single qubits or may be used to interact pairs of qubits (two-qubit gates). However, there is one irreversible operation, the measurement operation, which collapses the state into either $|0\rangle$ or $|1\rangle$.

Each quantum processor is constructed with a certain topology that describes the connectivity or one qubit to another, i.e. whether a two-qubit gate can be applied

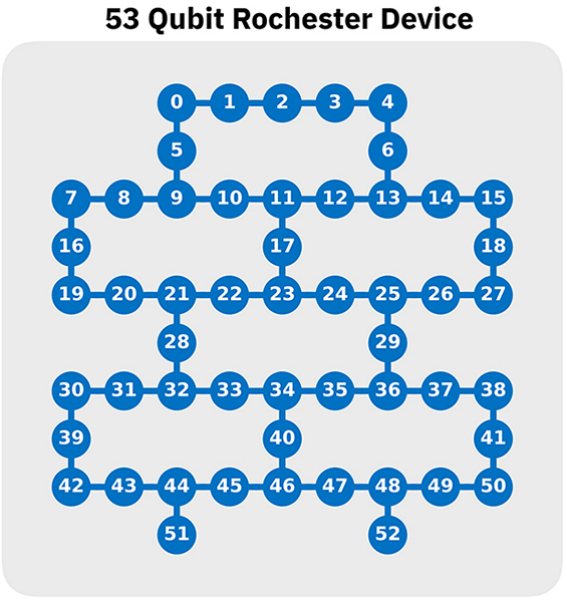

Figure 2.3: Topology of the IBM Rochester Device. Links between nodes indicate applying two-qubit gates to them is allowed.

to a certain pair of qubits. See Figure 2.3 for an example topology.

Therefore, to interact qubits that are not neighbors in the topology, one has to move them to suitable slots. These movements are also regarded as gates that imply either swapping qubits, teleportation, physical movement, or using a resonant cavity to transfer the quantum state from one and to another. As mentioned above, qubits suffer from decoherence, so their expected lifetime before becoming unusable is limited. Long range operations introduce significant latency so they need to be used sparingly if possible. Two-qubit gates are expected, in the current generation of quantum computers, to cause errors once in in every 100-1000 of them and long range communication errors are 10-100 times more frequent.

The most common way of representing quantum programs is by using circuit diagrams, hence quantum programs are also referred to as quantum circuits. These circuit diagrams are similar to those used in classical computing. In these diagrams programs are represented as a sequence of operations where each horizontal line represents the temporal line of a qubit and vertical lines represent two qubit gates.

Parallelism is allowed in these circuits : non-dependent operations may be performed at the same time. This parallelism is represented by placing operation at the same horizontal coordinate. Each set of parallel operations is what is defined as

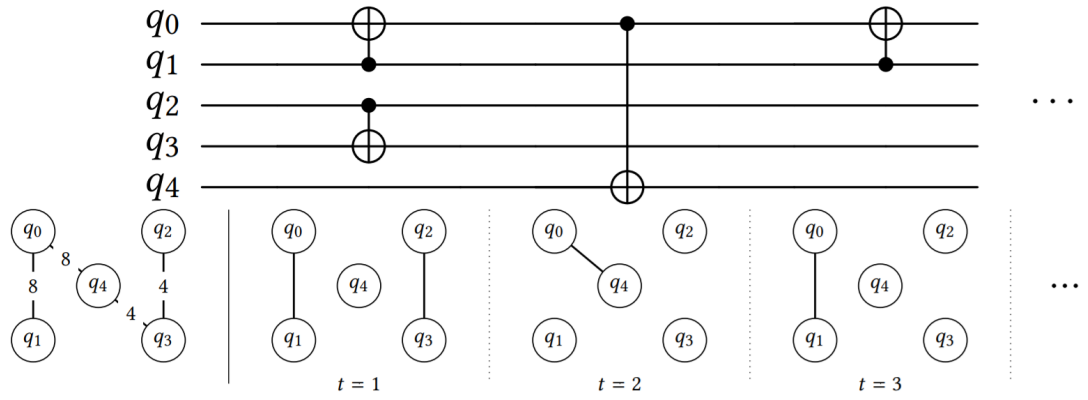

Figure 2.4: Adapted from [6]. (Top) Graph diagram of a quantum program. (Bottom) Total interaction graph and interaction graphs for each time slice of the above circuit.

a time slice of a quantum program. Time slices will depend not only on the specific quantum program but also on the architecture of the processor. Each time slice may be in turn be be summarized with an interaction graph, where nodes represent qubits and edges are gates to be applied to the involved nodes (qubits). Another graph of interest would be the total interaction graph, which is computed by summing all interaction graphs for a program, assigning weight 1 to each edge. See Figure 2.4 for an example of a circuit diagram and interaction graphs.

We are interested in processors with modular architectures in this thesis, for example [23], a trapped-ion based module blueprint. These computers have their qubits grouped in cores (modules, clusters, processors) where there is high connectivity between qubits in the same core and expensive, high latency non-local communication between cores. Thus reducing these operations is highly beneficial to avoid errors and allow for longer programs.

2.3 Graph Partitioning

For multi-core architectures, in each time slice, qubits will need to be assigned to certain cores so that the required gates can be applied. This can be posed as a problem of graph partitioning, where the objective is to minimize edges that go from nodes in one partition to nodes in other partitions. In particular, modular quantum computer architectures are fixed, i.e. a fixed number of partitions (cores)

with a fixed size for each partition.

This constraint discards the use of well known heuristic graph partitioning algorithms such as [20] or [11] as they do not offer guarantees on the size of the partitions. An alternative to this are constraint-based optimizers such as [10] which solves Satisfiability Modulo Theories for optimal solutions and easily takes on the previous constraint. The main limitation of this type is optimization is its poor scalability, which saturates at about 40 qubits.

Another consideration to take into account is that non-local communication is to be minimized due to its high latency. Thus, in each time slice, the interaction must be satisfied with the minimum number of (partition) swaps from the previous time slice.

2.4 Lookahead Graphs and rOEE

Baker et al. [6] propose the use of a lookahead graph at each time step t, which assigns infinite value to interactions (edges) that must be satisfied for the present slice. Then for each pair of qubits they add

$$w_t\left(q_i, q_j\right) = \sum_{t<m\leq T} I\left(m, q_i, q_j\right) \cdot D(m-t) \tag{2.3}$$

where T is the total number of slices, I is the indicator function for whether an interaction exists in time slice m for qubits q_i and q_j and D is a monotonically decreasing, non-negative function. This way, interactions that are closer in time are weighted higher.

Then for each lookahead graph they apply the relaxed Overall Extreme Exchange (rOEE). This is a modification of the standard Overall Extreme Exchange (OEE), presented in [26]. OEE takes an initial partition and efficiently calculates a sequence of swaps that improves the benefit, calculated from lookahead graphs. Their relaxed version stops the swapping once the partition is "valid", meaning all interactions in the immediate time slice are able to be performed (each pair of qubits is in the same partition). They find that this relaxed version is necessary because standard OEE overcorrects and makes more movements than needed.

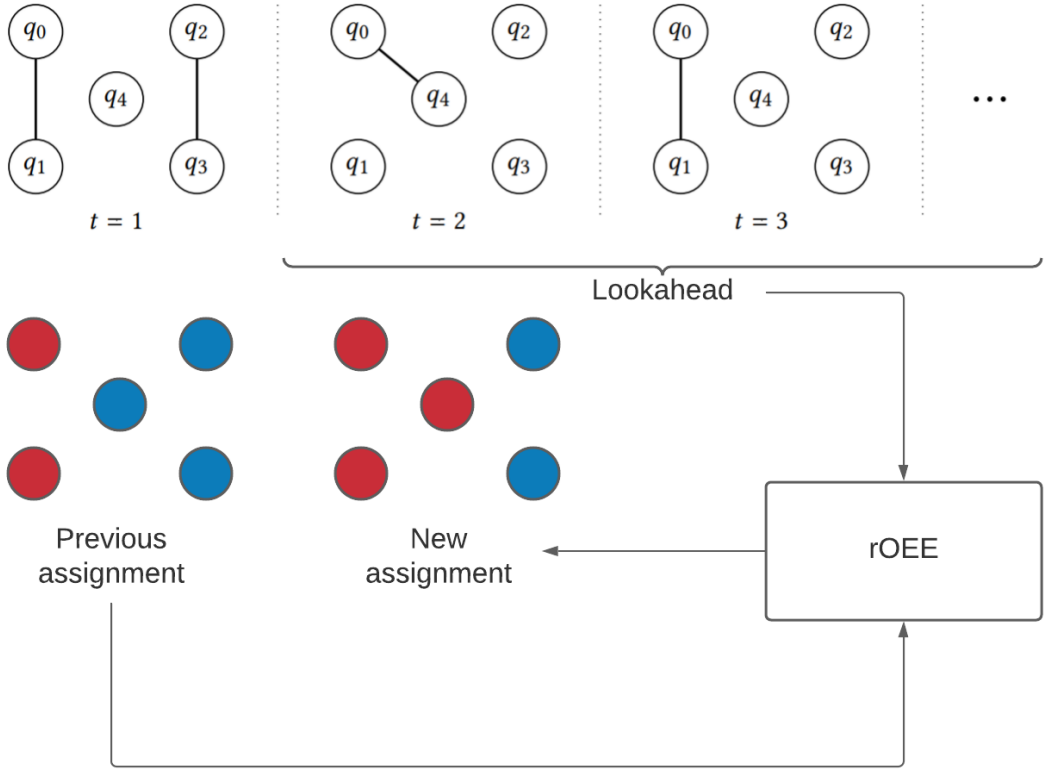

Figure 2.5: A diagram of Baker's algorithm applied to a single time slice.

With this method they aim to minimize the non-local communication. Which is the number of operations needed to go from the assignment of one time slice to the next. To calculate and optimize it, a second graph is created. This time, nodes represent cores and the edges represent qubits that need to go from one core to another. Then k-cycles are removed in increasing order of k. This because given a cycle, we can optimize the number of swaps needed since for a k-cycle we need $k-1$ to make all required changes. Thus, if we denote c_k the number of k-cycles removed and r the number of remaining edges, the non-local communication is measured as

$$C = r + \sum_{k=2}(k-1) \cdot c_k \tag{2.4}$$

The process is repeated for all time slices and summed to obtain the non-local communication for that quantum program. This also provides a path to go from the old assignment to the new one. See figure 2.5 for a diagram of Baker's algorithm applied to one time slice.

Chapter 3

Goals and Contribution

The OEE has a computational complexity of $O((kn)^2 \log(kn))$ [26] where k is the number of partitions and n is the number of nodes for each pass, this complexity is the same for rOEE passes. However, the relaxed version will always make less passes than the standard version.

On the other hand, GNN's in their basic form [15] have a computational complexity of $O(n+m)$ where m is the number of edges. Note that m strictly less than n^2, and the sparser the graph, the lower the complexity[1].

In addition, GNN's are able to benefit from GPU's which can further increase their computation speed. There is also research into developing specific hardware to accelerate GNN's, for example [5, 2, 12].

The goals for GNN in the field of quantum are two fold. First, develop GNN's that are able to replicate traditional partitioning algorithms. Succesful GNN's will be able to scale better into the bigger quantum architectures needed for meaningful computations that are currently intractable with classical computers.

The second goal, is to not only replicate traditional algorithms, but also improve their results. To do so in a supervised manner, a tractable way of finding near-optimal solutions would be needed, which as motivated in the previous chapter is not a trivial task. However, future work may consider the use of graph reinforcement

[1] We do not make any claims on the dependence on k, while it would most likely imply a sub-linear increase with n, it also requires training a new GNN.

learning. For example, [13] uses GNN's for two-way graph partition problems.

For this thesis we set out to contribute to the first goal, leaving the second goal for future work. We develop a GNN pipeline that is able to replicate the behavior of Baker's algorithm [6] for time-sliced quantum circuit partitioning. The pipeline includes a way of enforcing partition size constraints and rolls back to the original algorithms if it fails. We propose two different approaches that tackle this goal: one well suited for Baker's algorithm and another for more general use. Our GNN's, trained with randomly generated circuits are shown to generalize well to unseen real circuits and reduce rOEE computations by 20 to 30%.

Chapter 4

Methodology

We adopt the assumptions made in [6]. All-to-all inter-core connectivity is assumed, meaning that all inter-core quantum state transfers take one hop each and hence are equally expensive. We also adopt their architecture of 10 cores and 10 qubits per core. However, our methods presented below can be adapted with ease to accommodate different assumptions.

In section 4.1 we introduce the data we will be using for training our model as well as some real circuits that will be used to benchmark its performance on real unseen data. Then in section 4.2 we introduce our proposed solution which includes the specific GNN's we used and post-processing steps: maximum likelihood and rollback.

4.1 Data and Benchmarks

We decided to use randomly generated circuits to train our model. The reason is twofold: to train our model we need a sizable dataset of quantum programs and we want to see how well our model generalizes to unseen circuits with structurally different interaction graphs. Random circuits are analogous to synthetic traffic patterns. They are used frequently in early quantum computing works as it allows to stress the architectures in a controlled manner.

We generate 1000 random circuits like described in [30] with an average of 4000

two-qubit gates. These are scheduled using an as late a possible (ALAP) scheduling heuristic to obtain the sequence of interaction graphs. These come out with an average of 300 time-slices each. Interaction graphs do not include single qubit gates and empty time slices (or with only single qubit gates) are removed. Then, we apply the algorithm of [6], for which we made an implementation in Python, to obtain the target partitions for each qubit in each time slice. A standard 0.6, 0.2, 0.2 train/validation/test split of the dataset is adopted, where the train dataset is used for training, the validation set is used to tune parameters and the test set is only used for final testing.

Besides using the test set of random circuits as a benchmark, we are interested in the generalization power to more structured circuits, like real algorithms. To represent this kind of algorithms we chose the Quantum Fourier Transform (QFT) and Grover's search algorithm as benchmarks [8, 16]. These circuits have a predefined structure that depends only on the number of qubits, so we use these algorithms for qubit counts ranging from 50 to 100. Note that architectures of 100 qubits are able to handle programs with a lower quantity of qubits, the remaining qubits can be used as ancilla or control qubits.

We use implementations of the mentioned circuits in OpenQL [21] and schedule them using said platform for which an adaptation of our Python code to C++ for Baker's algorithm is underway.

4.2 GNN Pipeline

In this section, we present our proposed pipeline for the partitioning problem. First, we make predictions using an MPNN. Then, we enforce problem constraints such as the partition size constraint and the valid partition constraint, to be elaborated on below. Figure 4.1 shows a diagram of our proposed pipeline.

4.2.1 Message Passing Neural Network

For our GNN, we will use an MPNN similar to the one proposed in [15] and the following characteristics.

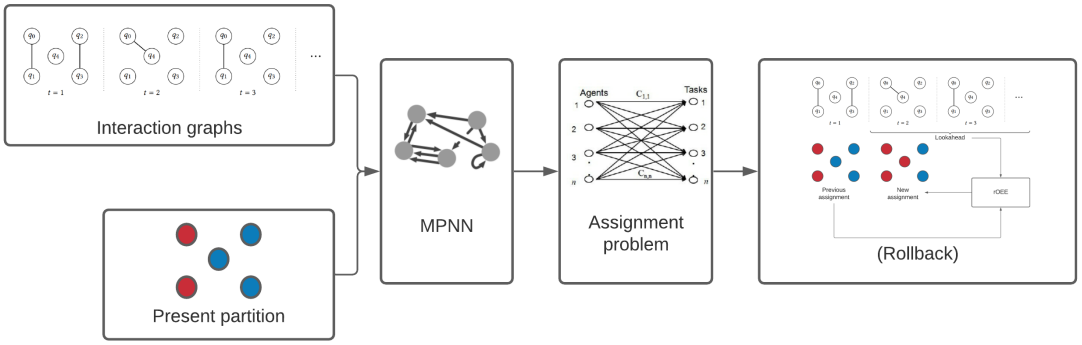

Figure 4.1: Diagram of our proposed pipeline. Present and future information is encoded to the edges either through lookaheads or using windows of interaction graphs. Along with this, present partitions are encoded as node features of the MPNN. The MPNN infers the probabilities of each node being in each partition for the next slice. Then, we solve the assignment problem to maximize the likelihood. If the partition is not valid we rollback to Baker.

Edge Features

For edge feature selection we tried two different approaches. The first one is using the lookahead weights from [6] as the edge feature. We will call this approach Lookahead approach.

The second approach is to use windows of slices of certain width with no edge features, from the present slice to future slices. See Figure 4.2 for an example. This approach will be named Window approach. There are two types of edges: interaction edges and temporal edges. Interaction edges are self explanatory, they are the edges from the interaction graphs. On the other hand, temporal edges require a more detailed description.

Temporal edges are edges that connect nodes to themselves on a different time slice. We tried different connection schemes, as shown in Figure 4.3:

- Single hop: Each node is connected to itself one time step into the future and into the past.
- Cycle: Same as above, but the present node is connected to itself in the last time step.
- Center: The present node is the hub that connects to all other time steps.

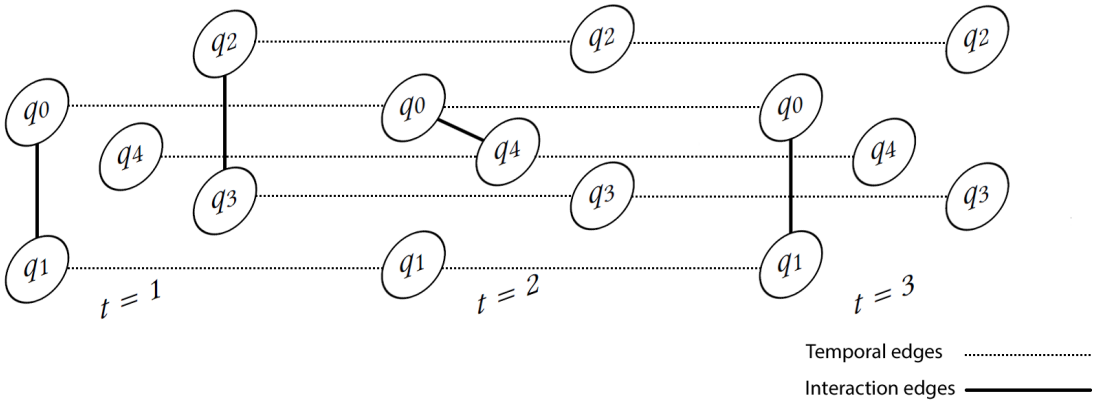

Figure 4.2: Example of window approach (single hop). There are two types of edges: interaction edges and temporal edges.

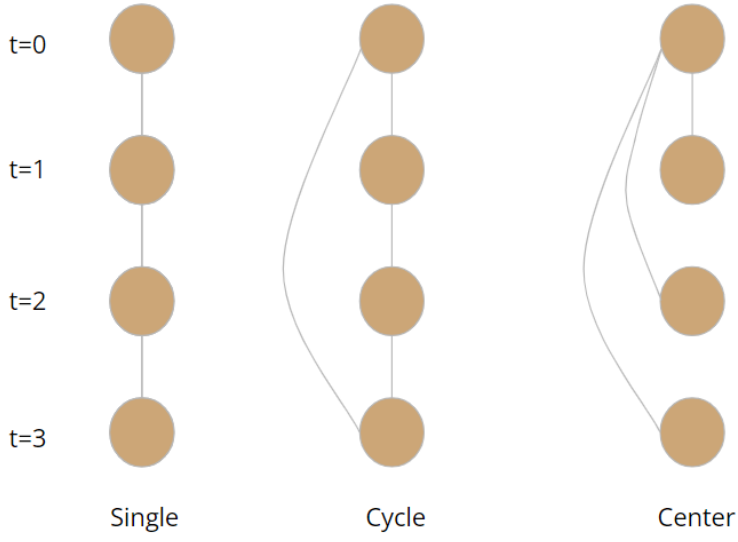

Figure 4.3: Different approaches to temporal connections.

Single hop is the most simple way of connecting a temporal sequence. However, with the intuition to improve the flow of information, we experimented with the cycle and center schemes.

We alternate interaction message passing and temporal message passing: messages are first sent through interaction edges. Then, after the node states are updated, we repeat with the temporal edges. One can also consider to use a different rate of updating each type of edge, e.g. message pass through interaction edges 3 times before doing so for temporal edges once.

Node Features

For the node feature we use the partition for the previous time slice, randomly initialized if we are dealing with the first time slice. This information will be encoded in to the node hidden state dimension by means of an embedding layer.

Other Hyperparameters

Some other hyperparameters to consider for this particular MPNN are:

- The number of message passing steps T.
- The aggregation function: fixed to a Gated Recurrent Unit (GRU) cell.
- The message function: fixed to an MLP, we can tune the number of layers, dropout and hidden activations. The last layer's width is fixed to the hidden state dimension.
- The readout function: same tunable parameters as above. The last layer's width is fixed to the number of partitions with softmax activation.
- The loss function: fixed to cross entropy.

Development and training

To implement our MPNN, we used Python as general language and Tensorflow 2.6.0 [3] as deep learning framework. The machine used for training included two Nvidia

GEFORCE GTX 1080 Ti GPU's and an Intel(R) Xeon(R) CPU E5-1630 v3 @ 3.70GHz CPU.

We performed hand-tuning of the results. This is because each instance of training would take around 12h-24h with our setup (depending on hyperparameters and epochs) so performing grid search (or even reasonably wide random search) with cross validation was unfeasible. Below we summarize some of our findings.

We start with some general findings:

- We used the Adam optimizer [22] with an exponential decay schedule. While the benefits of using decay with Adam are inconclusive in general, we found that for our case it would result in more stable training while allowing a bigger initial learning rate.
- Dropout was not necessary and even hindered training. The model was barely overfitting the data. Only on later stages of training (around epoch 50), would the accuracy on the validation accuracy curve go below the training accuracy, without decreasing.
- We found slight improvements in accuracy by using GELU [19] hidden activations instead of the RELU family.
- Sufficiently big MLP's for the message and readout functions would not differ much in performance, so they were kept conservatively small to reduce computation.

Below some considerations for the case of window input:

- Different message passing steps for interaction edges and temporal edges did not make a noticeable difference in accuracy.
- When increasing the width of the window, the accuracy would improve, saturating at around a width of 16. A further increase of the width would improve accuracy only marginally, while requiring more resources.
- When using the single hop scheme, the number of message passing steps would have to scale with the width of the window to achieve the improvements mentioned above. With the cycle scheme, it would need to scale at a lower rate. With the center scheme, no increase was needed for the message passing steps.

Table 4.1: Parameters for the final MPNN models. For the window approach we use the center time scheme with a window width of 16.

Message passing steps	4
Message function hidden layers	[64,64,64,64]
Readout function hidden layers	[64,64,64,64]
Hidden layers activation function	GELU
Loss	Cross entropy
Node state dimension	128
Learning rate	0.0017
Decay steps	1000
Decay rate	0.99
Batch size	64
Epochs	100
Shuffle buffer	4096

We perform batch training by creating an adjacency matrix that contains all adjacency matrices as a block in the diagonal and set the batch size to 64. We trained our models for 100 epochs with local shuffling with a buffer of 4096, due the the data set not fitting in memory. Table 4.1 summarizes our final training parameters.

4.2.2 Maximum Likelihood and Assignment Problem

Taking the highest probability prediction of the MPNN for each node does not guarantee that each partition has the designated number of assigned qubits. Thus, we run a post-processing step to obtain partitions that comply with that constraint.

We can obtain the maximum likelihood partition that complies with the partition size constraint by solving an assignment problem. The assignment problem is a well known optimization problem that is defined by a cost matrix C. The cost matrix summarizes the cost of assigning an element of a set of "workers" to an element of a set of "jobs". The task is then to find the assignment that minimizes its sum of costs. In our case, our "workers" are the qubits q_i and our "jobs" are the partitions,

$$\min_p \sum_i C(q_i, p(q_i)), \tag{4.1}$$

where p is the partition function, that assigns qubits to their partitions.

To obtain the final maximum likelihood partition that complies with the partition size constraint, we proceed as follows:

1. We take the negative logarithm of the predictions for each node: since we are modelling probabilities with our MPNN, the logarithm is needed to have a linear cost function; we take the negative to pose it as a minimization problem as follows:
$$\max_p \prod_i y_{i,p(i)} = \min_p \sum_i -\log(y_{i,p(i)}). \tag{4.2}$$

2. We repeat the negative log predictions for each partition as many times as slots that partition has: we need a square matrix for the assignment problem, i.e. as many slots as qubits. This is where the partition size constraint is enforced.

3. We solve the assignment problem with the Jonker-Volgenant algorithm [9].

4.2.3 Valid partitions

We still have one constraint that is not guaranteed to be fulfilled. That is, that all interactions for the present interaction graph must have its interaction qubits in the same partition to be a valid partition. To deal with this constraint we decide to rollback and use Baker's algorithm for invalid predictions of the GNN pipeline.

Note that Baker's algorithm actually does not guarantee valid partitions. However, due to certain properties of quantum programs, the output of rOEE is almost always a valid partition. Quantum gates are mostly two-qubit gates, and when higher n-qubit gates are present, they can be decomposed to two-qubit gates. Thus, when the partition sizes are sufficiently large, the Baker's algorithm guarantees valid partitions.

Chapter 5

Results and Discussion

In this chapter, we present the results of our GNN pipeline. First, some metrics for the MPNNs will be shown for the random circuits. Then, we will show the results for the non-local communication and computational aspects for the pipeline on real circuits.

5.1 Random circuits

In Table 5.1 we find some metrics concerning the performance of the Lookahead and Window approaches. It is important to mention that accuracy must only be viewed as a proxy for performance, since the partition size constraint is not yet applied.

Table 5.1: Results for Lookahead and Window approaches. Non-local communication is shown for the pipeline using each GNN, the results for Baker's algorithm are in parenthesis.

		Training	Validation	Test
Lookahead	Accuracy	0.9274	0.9245	0.9249
	Loss	0.2364	0.2481	0.2476
	Valid partitions (%)	21.11	21.05	21.06
	Non-local communication	2061 (2059)	2062 (2057)	2063 (2056)
Window	Accuracy	0.8140	0.8082	0.8087
	Loss	0.6521	0.6712	0.6754
	Valid partitions (%)	5.87	5.53	5.57
	Non-local communication	2070 (2059)	2077 (2057)	2075 (2056)

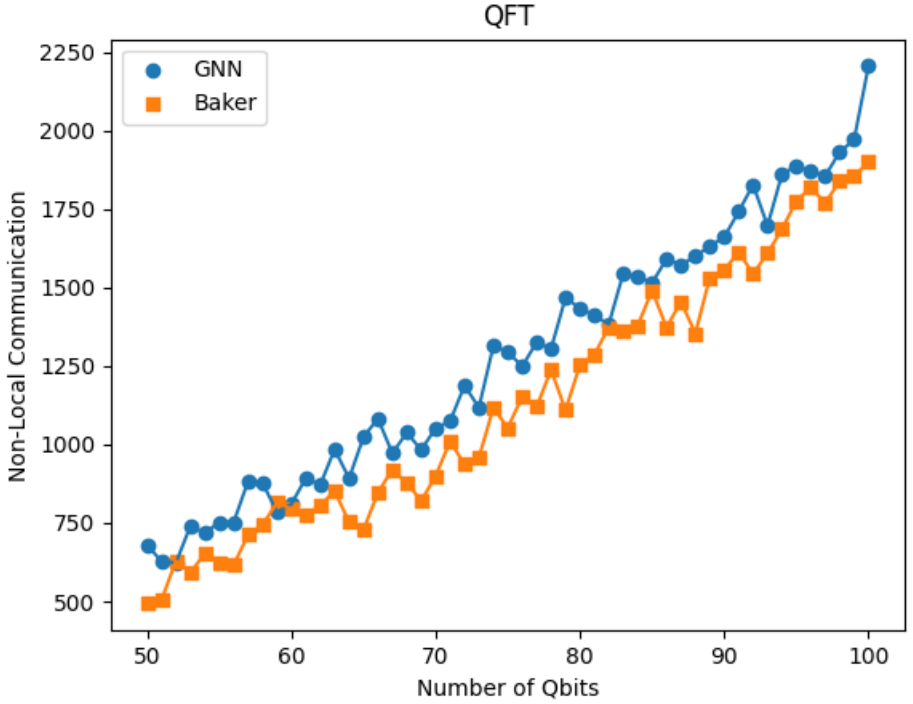

Figure 5.1: The non-local communication operations for the QFT circuits.

We observe that the Lookahead approach obtains better results than the Window approach. This is to be expected, since we are trying to replicate an algorithm, it makes sense that using the same information yields better results.

However, we see that the Window approach is also able to generate valid assignments, albeit in a lower percentage. We believe that further refinement of this approach may lead to a model that is more general in the sense of being able to not only imitate Baker's algorithm and other algorithms that may be developed, but also for optimization of the solution, say by using Reinforcement Learning for example. All said, the Window approach is not up to par in performance in its current state, so for the real circuits we only compare the Lookahead approach to Baker's algorithm.

5.2 Real circuits

As mentioned, QFT and Grover circuits have a predefined structure depending on the number of qubits. The non-local communication of both circuits ranging from

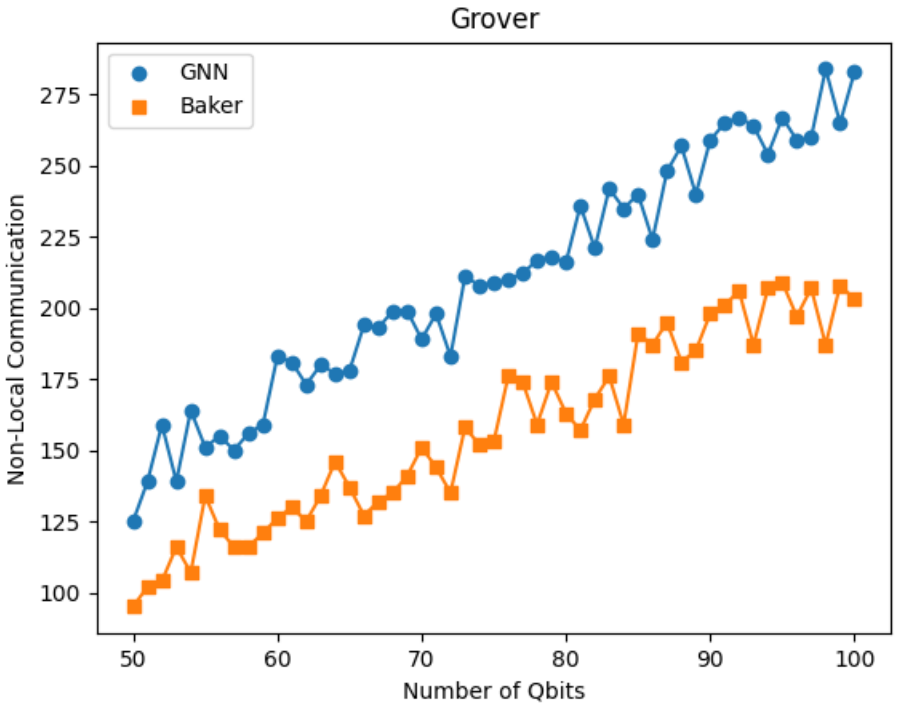

Figure 5.2: The non-local communication operations for the Grover Search Algorithm circuits.

50 to 100 qubits is shown in Figures 5.1 and 5.2.

We obtain results on-par (slightly worse) with Baker's algorithm for real unseen quantum circuits. The GNN is only trained on random circuits of 100 qubits, so this shows that the GNN is able to generalize well to unseen circuits with fundamentally different structures. To confirm this statement we also need to look at figures 5.3 and 5.4 where we plot the number of iterations of rOEE needed for each circuit. Note that the pipeline uses rOEE only if the predicted partiton is not valid. We see a decrease ranging from 20 to 30%. The decrease in iterations shows that the GNN is able to produce a portion of valid partitions for each circuit. As a side note, rOEE usually only needs one pass to obtain a valid partition.

In Figures 5.5 and 5.6 we plot the computation time rOEE. We leave out the computation time for the whole GNN pipeline because optimizing for speed was not a priority in this work, so results are not indicative of the computation speedup of GNN's. As we can see, the amount of computation time that is reduced ranges from 10 to 20%.

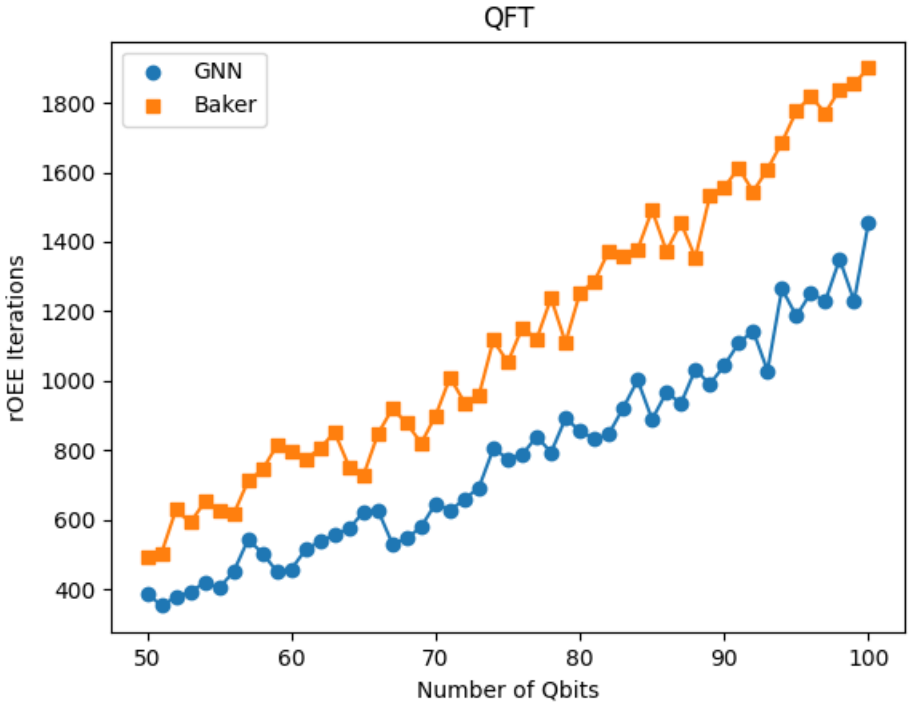

Figure 5.3: rOEE iterations for the QFT circuits.

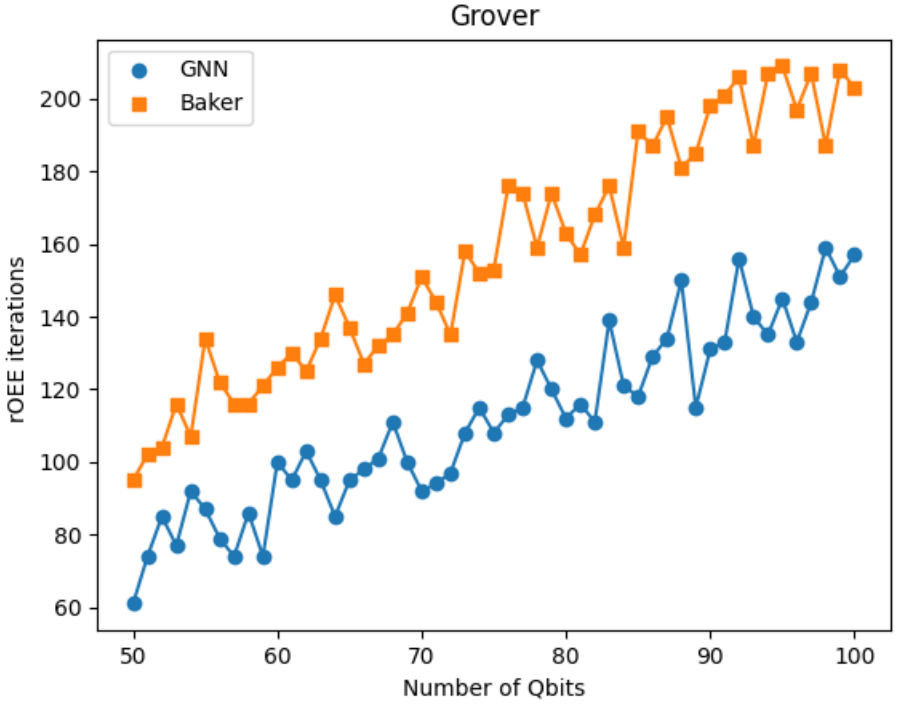

Figure 5.4: rOEE iterations for the Grover Search Algorithm circuits.

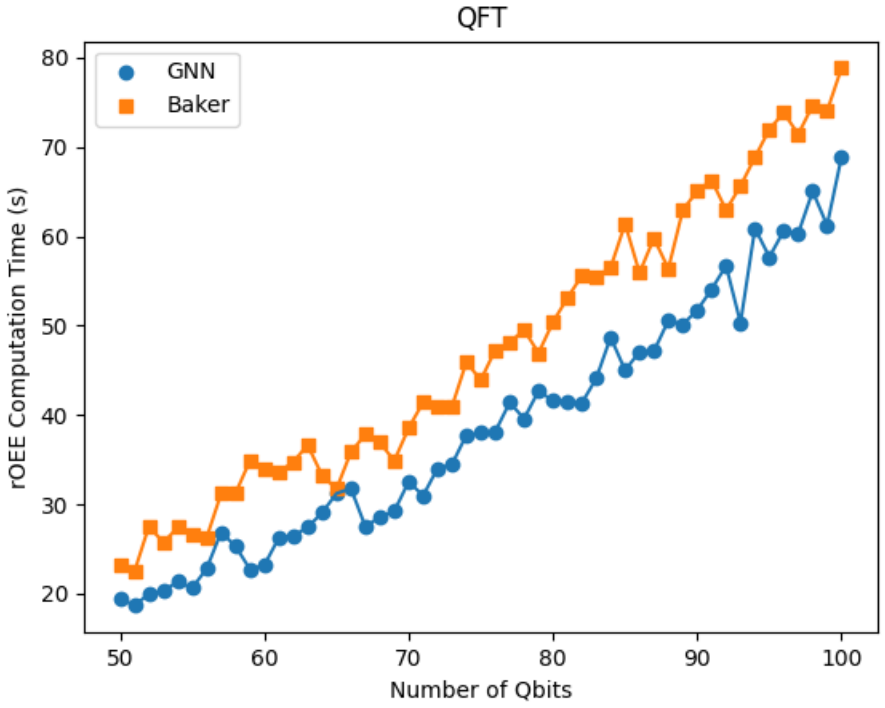

Figure 5.5: rOEE computation time for QFT circuits.

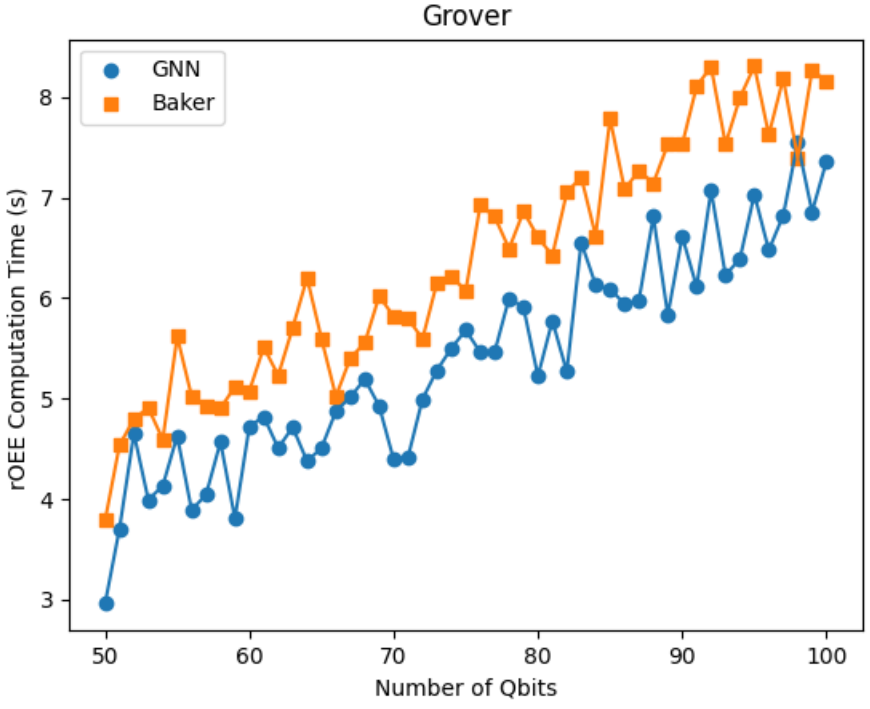

Figure 5.6: rOEE computation time for Grover Search Algorithm circuits.

Chapter 6

Conclusions and Future Work

In this thesis, we propose a GNN pipeline to imitate Baker's algorithm for time-sliced quantum circuit partitioning. We show that our pipeline using lookahead is able to replicate the algorithm and obtain valid qubit assignments in 20 to 30% of the cases on unseen real circuits, while being trained only with randomly generated circuits.

The results show the potential of GNN's for this task, specially as an accelerator for better scalability of quatum circuit partitioning. This is because GNN's can be optimized to run on GPU and only need to be trained once. We also show a way to feed the interaction graphs to a GNN without aggregation with our Window approach, while it yields worse results in the imitation task (as expected), it may be used for future models that try to optimize instead of imitate.

Future work towards a better GNN model can be explored in many different directions. One such direction is to generate a richer training dataset that includes both synthetically generated circuits as well as real circuits. Gathering such a dataset is not a trivial task itself since many implementations are made in different frameworks and quantum assembly languages are not standardized fully.

On the model side, improvements can be made by using more complex and/or efficient GNN's. For example, Graph Attention Layers [32] have found great success in different tasks. In addition to this, more information about the circuits can be incorporated into the inputs: gate information can be encoded into the interaction edges and one qubit gates can be represented as self loops in the interaction graph.

Another particularly interesting direction to explore is the use of Graph Reinforcement Learning. Using GNN's in recent Deep Reinforcement Learning frameworks is an approach that need not rely on solutions of other algorithms to find good solutions. In addition, certain constraints can be modeled into the reward function, for example the need for a valid partition can be enforced by only giving rewards if the partition is valid.

Chapter 7

Bibliography

[1] IBM unveils 127-qubit quantum processor. `https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor`. Accessed: 2022-01-08.

[2] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)*, 54(9):1–38, 2021.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, and et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, Oct 2019.

[5] A. Auten, M. Tomei, and R. Kumar. Hardware acceleration of graph neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[6] J. M. Baker, C. Duckering, A. Hoover, and F. T. Chong. Time-sliced quantum circuit partitioning for modular architectures. *Proceedings of the 17th ACM International Conference on Computing Frontiers*, May 2020.

[7] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[8] D. Coppersmith. An approximate fourier transform useful in quantum factoring, 2002.

[9] D. F. Crouse. On implementing 2D rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016.

[10] L. de Moura and N. Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[11] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181, 1982.

[12] R. Garg, E. Qin, F. Muñoz-Martínez, R. Guirado, A. Jain, S. Abadal, J. L. Abellán, M. E. Acacio, E. Alarcón, S. Rajamanickam, et al. Understanding the design space of sparse/dense multiphase dataflows for mapping graph neural networks on spatial accelerators. *arXiv preprint arXiv:2103.07977*, 2021.

[13] A. Gatti, Z. Hu, T. Smidt, E. G. Ng, and P. Ghysels. Graph partitioning and sparse matrix ordering using reinforcement learning and graph neural networks, 2021.

[14] C. Gidney and M. Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, Apr 2021.

[15] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[16] L. 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.

[17] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto. Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 1802–1808, 2017.

[18] W. L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.

[19] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2020.

[20] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[21] N. Khammassi, I. Ashraf, J. v. Someren, R. Nane, A. M. Krol, M. A. Rol, L. Lao, K. Bertels, and C. G. Almudever. Openql : A portable quantum programming framework for quantum accelerators, 2020.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[23] B. Lekitsch, S. Weidt, A. G. Fowler, K. Mølmer, S. J. Devitt, C. Wunderlich, and W. K. Hensinger. Blueprint for a microwave trapped ion quantum computer. *Science Advances*, 3(2):e1601540, 2017.

[24] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Physical Review A*, 89(2), Feb 2014.

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

[26] T. Park and C. Y. Lee. Algorithms for partitioning a graph. *Comput. Ind. Eng.*, 28(4):899–909, oct 1995.

[27] J. Preskill. Quantum computing and the entanglement frontier, 2012.

[28] J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, Aug 2018.

[29] S. Rodrigo, S. Abadal, E. Alarcón, M. Bandic, H. v. Someren, and C. G. Almudéver. On double full-stack communication-enabled architectures for multi-core quantum computers. *IEEE Micro*, 41(5):48–56, 2021.

[30] S. Rodrigo, M. Bandic, S. Abadal, H. van Someren, E. Alarcón, and C. G. Almudéver. Scaling of multi-core quantum architectures: A communications-aware structured gap analysis. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, page 144–151, New York, NY, USA, 2021. Association for Computing Machinery.

[31] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008.

[32] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018.

[33] Y. Wu, D. Lian, Y. Xu, L. Wu, and E. Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):1054–1061, Apr. 2020.

[34] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.