ATPL

Splitting circuits for simulation and optimization

Johan Palm, Enrico Zanoni

18/12/2025

UNIVERSITY OF COPENHAGEN

Table of Contents

1. Introduction

The problem

- **NISQ:**
  - NISQ computers require short gate depth
  - CNOT gates are noisy
  - Can only do 1- and 2-qubit gates from a universal gateset

The problem

- **NISQ:**
  - NISQ computers require short gate depth
  - CNOT gates are noisy
  - Can only do 1- and 2-qubit gates from a universal gateset
- **Fault-Tolerant Quantum Computing:**
  - Not a readily available resource
  - Want to **minimize** the use of a quantum computer

Key Insights

- **Do we need to run entire circuits?**
  - **Clifford** gates are efficiently classically simulatable
  - **Rotational** or **T** gates are not simulatable efficiently

Key Insights

- **Do we need to run entire circuits?**
  - **Clifford** gates are efficiently classically simulatable
  - **Rotational** or **T** gates are not simulatable efficiently
- **Can we take advantage this?**
  - We could *move* gates to the end of the circuit and then classically simulate them or measure in a different basis
  - **Can't** simulate state-preparation gates

Project

- **Goal:**
  - Introduce a framework for handling the moving and optimization of gates in the Haskell framework
    - QuCLEAR that holds larger framework for this and smaller algorithms [1]
    - We can use certain algorithms from QuCLEAR
- **Specifically:**
  - Implement the Clifford extraction
  - **In interest of time** add qubit connectivity constraints

2. Theory

Theoretical Foundation: Weak Commutation

- Core Identity: Clifford circuits stabilize the Pauli group, meaning $U_{CL}^{\dagger} P_1 U_{CL} = P_2$, where P_2 is another Pauli string.
- A Pauli rotation of angle θ can be written of the form $e^{-i\frac{\theta}{2}P} = \cos(\frac{\theta}{2})I - i\sin(\frac{\theta}{2})P$.
- The Commutation Rule: $e^{iP_1\theta} U_{CL} = U_{CL} e^{iP_2\theta}$, where $P_2 = U_{CL}^{\dagger} P_1 U_{CL}$
- Sign Tracking: The rotation angle sign may flip if P_2 includes a negative sign $e^{i(-P)\theta} = e^{iP(-\theta)}$.

Single-Qubit Clifford Commutations

- **Basis Change**: Single-qubit Clifford gates (H, S) transform Pauli operators to different bases, which is essential for rotating X and Y terms into the Z basis for parity encoding.

Gate	**Original** P	**Transformed** P'
H	X	Z
H	Y	-Y
H	Z	X

Table: Single-qubit Pauli transformations under Clifford action.

Single-Qubit Clifford Commutations

- **Basis Change**: Single-qubit Clifford gates (H, S) transform Pauli operators to different bases, which is essential for rotating X and Y terms into the Z basis for parity encoding.

Gate	**Original** P	**Transformed** P'
S	X	Y
S	Y	-X
S	Z	Z

Table: Single-qubit Pauli transformations under Clifford action.

Control-X - Qubit Clifford Commutations

- **Basis Change**: Two-qubit Clifford gate, namely CNOT transform states depending on .

Gate	**Original** P	**Transformed** P'
CX	$X_C \otimes I_T$	$X_C \otimes X_T$
CX	$Z_C \otimes I_T$	$Z_C \otimes I_T$
CX	$Y_C \otimes I_T$	$Y_C \otimes X_T$

Table: Control-X - qubit Pauli transformations under Clifford action.

Control-X - Qubit Clifford Commutations

- **Basis Change**: Two-qubit Clifford gate, namely CNOT transform states depending on .

Gate	**Original** P	**Transformed** P'
CX	$I_C \otimes X_T$	$I_C \otimes X_T$
CX	$I_C \otimes Z_T$	$Z_C \otimes Z_T$
CX	$I_C \otimes Y_T$	$Z_C \otimes Y_T$

Table: Control-X - qubit Pauli transformations under Clifford action.

Simplified example

- **The circuit**: $R(P_1)\theta \circ CNOT$, where $P_1 = XI$
- Moving the CNOT will lead to: $CNOT \circ R(P_2)\theta$ where $P_2 = XX$

Less simplified example

- **The circuit**:
  - $H \otimes H \otimes H \otimes H$
  - $I \otimes I \otimes CNOT$
  - $CNOT \otimes I \otimes I$
  - $R(P_1)$, where $P_1 = ZZII$

Less simplified example

- **The circuit**:
  - $H \otimes H \otimes H \otimes H$
  - $I \otimes I \otimes CNOT$
  - $CNOT \otimes I \otimes I$
  - $R(P_1)$, where $P_1 = ZZII$
- **Can be moved to**:
  - $H \otimes H \otimes H \otimes H$
  - $I \otimes I \otimes CNOT$
  - $R(P_2)$, where $P_2 = IZII$
  - $CNOT \otimes I \otimes I$

Less simplified example

- **The circuit**:
  - $H \otimes H \otimes H \otimes H$
  - $I \otimes I \otimes CNOT$
  - $CNOT \otimes I \otimes I$
  - $R(P_1)$, where $P_1 = ZZII$
- **Can be moved to**:
  - $H \otimes H \otimes H \otimes H$
  - $R(P_3)$, where $P_3 = IZII$
  - $I \otimes I \otimes CNOT$
  - $CNOT \otimes I \otimes I$

Less simplified example

- **The circuit**:
  - $H \otimes H \otimes H \otimes H$
  - $I \otimes I \otimes CNOT$
  - $CNOT \otimes I \otimes I$
  - $R(P_1)$, where $P_1 = ZZII$
- **Can be moved to**:
  - $R(P_4)$, where $P_4 = IXII$
  - $H \otimes H \otimes H \otimes H$
  - $I \otimes I \otimes CNOT$
  - $CNOT \otimes I \otimes I$
- Where we have **decreased** the amount of rotations needed on our Pauli string

Implementation: The Symplectic Form

- **Mathematical Representation**: Any n-qubit Pauli string is uniquely identified by two bit-vectors $(x, z \in \{0, 1\}^n)$ and a sign bit $s \in \{0, 1\}$.

Implementation: The Symplectic Form

- We can represent the Pauli operators as tuples for X and Z:
- $I = (0, 0)$, $X = (1, 0)$, $Z = (0, 1)$, $Y = (1, 1)$
- Any Pauli string can then be represented as a $2n$ vector.
- **Example:** $XIYZ = [0011|1010]$, where the first n and last n bits represents Z and X respectively.

Conjugating Symplectic form

- **Conjugating by a CNOT**
- Given an Pauli encoding p of P, encoding p' of $CNOT_{i,j}$, where i is control and j is target, conjugated through is:
  - $\forall k \notin \{i, n+j\}, p'[k] = p[k]$
  - $p'[i] = p[j] \oplus p[i]$
  - $p'[n+j] = p[n+i] \oplus p[n+j]$
  - $(p[i] \wedge p[n+i]) \wedge (p[j] \wedge p[n+j]) \Rightarrow s' \to s \oplus 1$

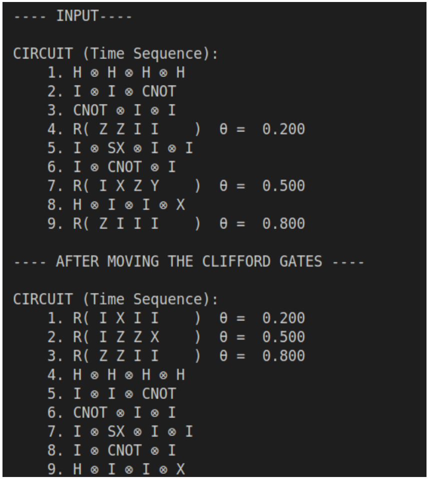
Conjugating Symplectic form

- **Conjugating by a H**
- Given an Pauli encoding p of P, encoding p' of H_i, where i is the used qubit, conjugated through is:
  - $\forall k \notin \{i, n+i\}, p'[k] = p[k]$
  - $p'[i] = p[n+i]$
  - $p'[n+i] = p[i]$
  - $p[i] \land p[n+i] \Rightarrow s' \rightarrow s \oplus 1$
- Basically swaps the X and Z components.

Conjugating Symplectic form

- **Conjugating by a S**
- Given an Pauli encoding p of P, encoding p' of S_i, where i is the used qubit, conjugated through is:
  - $\forall k \notin \{i\}, p'[k] = p[k]$
  - $p'[i] = p[n+i] \oplus p[i]$
  - $p[i] \land p[n+i] \Rightarrow s' \rightarrow s \oplus 1$

Output on a larger circuit



```
---- INPUT----

CIRCUIT (Time Sequence):
    1. H ⊗ H ⊗ H ⊗ H
    2. I ⊗ I ⊗ CNOT
    3. CNOT ⊗ I ⊗ I
    4. R( Z Z I I     )  θ =  0.200
    5. I ⊗ SX ⊗ I ⊗ I
    6. I ⊗ CNOT ⊗ I
    7. R( I X Z Y     )  θ =  0.500
    8. H ⊗ I ⊗ I ⊗ X
    9. R( Z I I I     )  θ =  0.800

---- AFTER MOVING THE CLIFFORD GATES ----

CIRCUIT (Time Sequence):
    1. R( I X I I     )  θ =  0.200
    2. R( I Z Z X     )  θ =  0.500
    3. R( Z Z I I     )  θ =  0.800
    4. H ⊗ H ⊗ H ⊗ H
    5. I ⊗ I ⊗ CNOT
    6. CNOT ⊗ I ⊗ I
    7. I ⊗ SX ⊗ I ⊗ I
    8. I ⊗ CNOT ⊗ I
    9. H ⊗ I ⊗ I ⊗ X
```

Decomposing Pauli Rotations

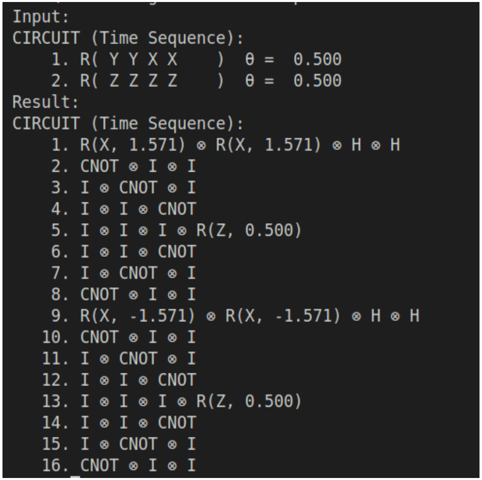
- **Core Concept:** Any multi-qubit Pauli rotation $R_P(\theta) = e^{-i\frac{\theta}{2}P}$ can be implemented using a **single-qubit rotation** flanked by Clifford gates.
- This structure is often called a **Pauli Gadget**.
- **The 3-Step "Sandwich" Protocol:**
    1. **Basis Change:** Map all non-Z terms (X, Y) to the Z-basis using single-qubit Cliffords (H, R_X).
    2. **Parity Computation:** Use a ladder of **CNOT** gates to compute the parity onto a target qubit.
    3. **Rotation & Uncompute:** Apply $R_Z(\theta)$ to the target, then apply the inverse of steps 1 and 2 to restore the basis.
- Mathematically:

$$e^{-i\frac{\theta}{2}P} = C^{\dagger}\left(I \otimes \cdots \otimes R_Z(\theta)\right) C$$

Optimizing Parity Logic (1/2)

- **The Challenge:** Constructing a full CNOT ladder (parity tree) for every single rotation is inefficient. In large circuits, this leads to excessive depth and gate count.
- **The Inefficiency:** A naive decomposition performs "Uncomputation" immediately after every rotation: Basis Change $\rightarrow$ CNOTs $\rightarrow$ $R_Z(\theta)$ $\rightarrow$ **Inverse CNOTs** $\rightarrow$ **Inverse Basis**
- **The Strategy:** Instead of uncomputing immediately, we **delay** the inverse Clifford gates. We push them forward through the circuit to merge with the next operations.

Naive decomposition example



```
Input:
CIRCUIT (Time Sequence):
    1. R( Y Y X X    )  θ =  0.500
    2. R( Z Z Z Z    )  θ =  0.500
Result:
CIRCUIT (Time Sequence):
    1. R(X, 1.571) ⊗ R(X, 1.571) ⊗ H ⊗ H
    2. CNOT ⊗ I ⊗ I
    3. I ⊗ CNOT ⊗ I
    4. I ⊗ I ⊗ CNOT
    5. I ⊗ I ⊗ I ⊗ R(Z, 0.500)
    6. I ⊗ I ⊗ CNOT
    7. I ⊗ CNOT ⊗ I
    8. CNOT ⊗ I ⊗ I
    9. R(X, -1.571) ⊗ R(X, -1.571) ⊗ H ⊗ H
   10. CNOT ⊗ I ⊗ I
   11. I ⊗ CNOT ⊗ I
   12. I ⊗ I ⊗ CNOT
   13. I ⊗ I ⊗ I ⊗ R(Z, 0.500)
   14. I ⊗ I ⊗ CNOT
   15. I ⊗ CNOT ⊗ I
   16. CNOT ⊗ I ⊗ I
```

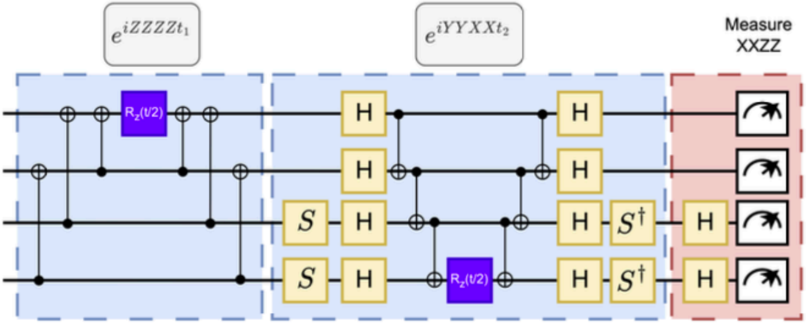
Optimizing Parity Logic (2/2)

- **Mechanism (Gate Commutation):** The "Uncomputation" Cliffords ($C^\dagger$) are pushed past the next rotation (R_{Next}).
- **The Effect:** This removes the physical gates between operations and might transforms the Pauli basis of R_{Next} to a better (with lower Hamming weight) rotation.

Mathematical Transformation:

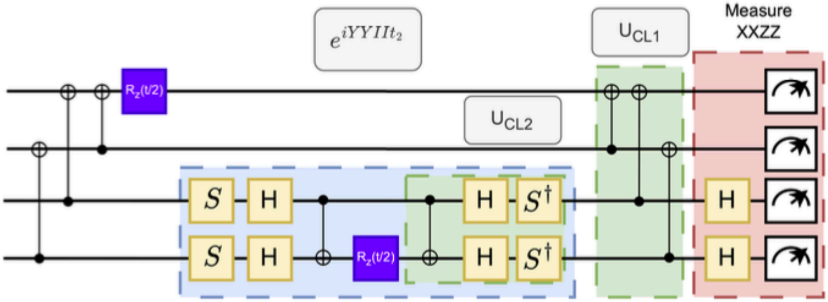
$$\underbrace{R_Z(\theta)\cdot \mathbf{C}^\dagger}_{\text{Op 1}} \cdot \mathbf{R_{Next}} \xrightarrow{\text{Push } C^\dagger} R_Z(\theta)\cdot \underbrace{(\mathbf{C}^\dagger \cdot R_{Next} \cdot \mathbf{C})}_{\text{Transformed Op 2}} \cdot \mathbf{C}^\dagger$$

Example of smarter CNOT tree (1/3)



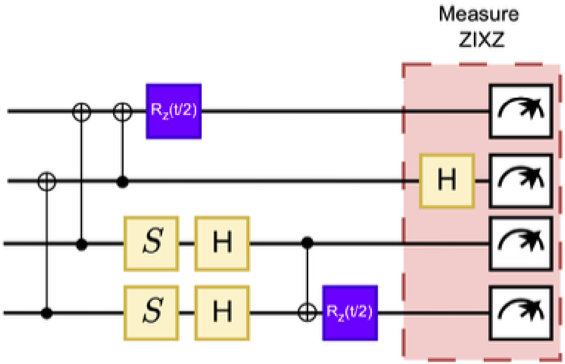
(a)

Example of smarter CNOT tree (2/3)



(b)

Example of smarter CNOT tree (3/3)



(c)

CNOT Tree Synthesis: The ”Look-Ahead” Strategy

Goal

Construct a CNOT parity tree for the current Pauli string (P_{curr}) while optimizing gate cancellation for future strings.

Key Innovation: Recursive Look-Ahead

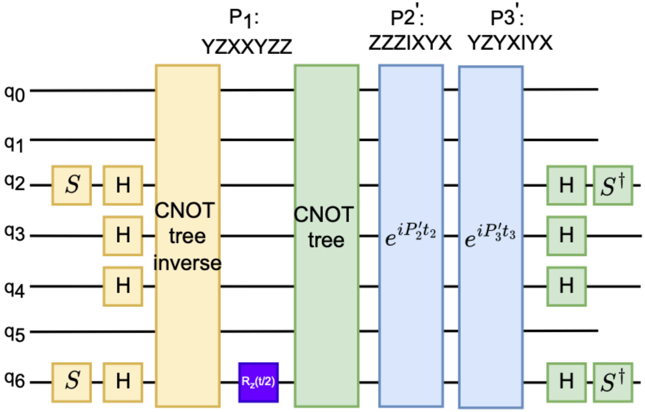
- Standard approaches optimize locally, but QuCLEAR looks ahead to the next Pauli string (P_{next}).
- **Grouping Principle:** Qubits that share identical operators in P_{next} are grouped together into subtrees.
- This ensures that the structure built now will be trivial (or easier) to implement for the subsequent operation.

CNOT Tree Synthesis: Algorithmic Steps

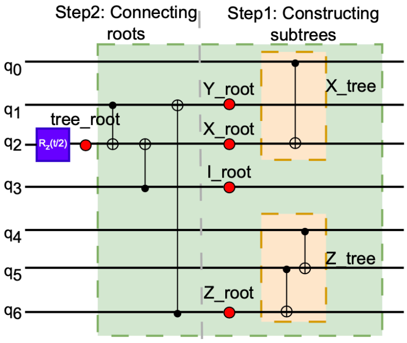
The synthesis follows these steps:

1. **Update Next Pauli:** The algorithm computes the state of the *next* Pauli string by applying the Clifford gates already extracted (*extr_clf*).
2. **Partitioning (Subtree Generation):** Qubits are partitioned into four groups based on their operator in the updated P_{next}: *I_tree*, *X_tree*, *Y_tree*, and *Z_tree*.
3. **Recursive Synthesis:** If a subtree contains more than one qubit, the algorithm calls itself recursively, looking further ahead to P_{next+1} to determine the internal connection order.

Algorithm


P_1:
YZXXYZZ
P2':
ZZZIXYX
P3':
YZYXIYX
q0
q1
q2
q3
q4
q5
q6
S
H
CNOT tree inverse
$R_z(t/2)$
CNOT tree
$e^{iP_2' t_2}$
$e^{iP_3' t_3}$
$S^\dagger$

Algorithm



CNOT subtrees only based on P2' ZZZIXYX:

I_tree: 3	I_root: 3
X_tree: 0, 2	X_root: 2
Y_tree: 1	Y_root: 1
Z_tree: 4, 5, 6	Z_root: 6

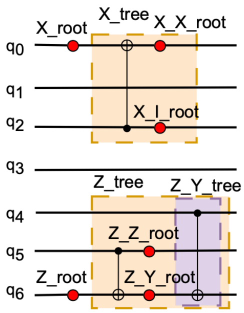
(b)

CNOT Tree Synthesis: Connecting Roots

After synthesizing the subtrees, their "roots" (the final qubit of each group) must be connected to form the full parity tree.

- **Connection Logic:** Roots are connected based on a priority list derived from commutation rules.
- **Optimization:** For example, connecting a *Z_root* with a *Y_root* is prioritized because it transforms the resulting Pauli operators into identities or simpler forms.
- The final remaining qubit becomes the global tree root where the R_z rotation is applied.

Algorithm

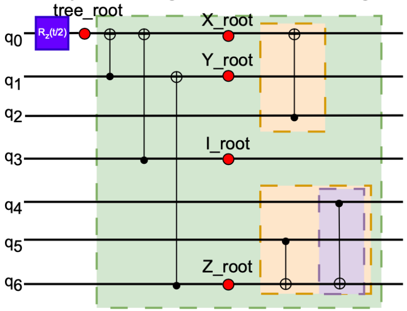


X_tree based on P3' YZYXIYX:		After connecting
X_I_tree: 2	X_I_root: 2	X_I_root and X_X_root,
X_X_tree: 0	X_X_root: 0	X_root: 0

Z_tree based on P3' YZYXIYX:		After connecting
Z_Y_tree: 4, 6	Z_Y_root: 6	Z_Z_root and Z_Y_root,
Z_Z_tree: 5	Z_Z_root: 5	Z_root: 6

(c)

Recursively constructing subtrees and connecting roots



(d)

Clifford Extraction: Workflow

The extraction process iterates through the circuit:

1. **Commuting Blocks:** Pauli strings are grouped into blocks where operations commute. The order of operations within a block can be changed freely.
2. **Greedy Selection ("Find Next Pauli"):** Within a block, the algorithm simulates extraction for every available Pauli. It selects the one that results in the lowest cost (fewest non-identity terms) for the *next* step.
3. **Accumulation:** The synthesized CNOT tree is added to the *extr_clf* register (conceptually at the end of the circuit) rather than remaining between rotations.

Clifford Extraction: Update and Complexity

As gates are moved to the end, the remaining uncompiled circuit must be updated.

- **Tableau Update:** Future Pauli strings are updated using the stabilizer tableau formalism: $P_{new} \leftarrow U_{CL}^{\dagger} P_{old} U_{CL}$.
- **Scalability:** The update operation is efficient, requiring only $O(n^2)$ classical operations.
- **Final Outcome:** The procedure results in a significantly reduced quantum circuit followed by a Clifford subcircuit that is absorbed into measurement observables or post-processing.

Current Status I: Efficient Clifford Moving

We have successfully implemented the core mechanism for ”moving” Clifford gates through the circuit.

Symplectic Representation

Instead of using dense matrices, we utilize the **symplectic form** (Stabilizer Tableau formalism).

Advantages:

- **Efficiency:** Allows representing Pauli strings and Clifford operators as binary vectors.
- **Speed:** The mathematical update of Pauli strings ($P_{new} \leftarrow U_{CL}^{\dagger} P_{old} U_{CL}$) is performed in $O(n^2)$ time.
- This ensures that the overhead of recalculating strings during extraction remains negligible.

Current Status II: Baseline Decomposition

To establish a baseline for our optimization benchmarks, we have implemented a standard synthesis method for Pauli rotations ($e^{iP\theta}$).

Trivial Decomposition

We currently synthesize rotations using the "Linear Chain" approach.

Details:

- The parity logic is computed using a cascade of CNOT gates strictly between **adjacent qubits**.
- **Role:** This provides a physically valid (though not optimized) starting point.
- It serves as the reference against which we will measure the improvements gained by the new recursive tree synthesis.

Future Work I: Framework Integration

Porting to Haskell

Our immediate objective is to integrate the optimization logic into our existing **Haskell** quantum compilation framework.

Implementation Goals:

- Translate the *Recursive CNOT Tree Synthesis* algorithm (Algorithm 1) into functional Haskell code.
- Implement the *Clifford Extraction* loop (Algorithm 2) to manage the commutation of gates and the accumulation of the final Clifford subcircuit.
- Ensure the new modules interface correctly with our existing circuit representation data structures.

Future Work II: Connectivity Constraints

The Connectivity Challenge

The current algorithm assumes **all-to-all** qubit connectivity. However, most hardware has sparse connectivity graphs.

Problem:

- A theoretically ”optimal” CNOT tree might require interacting qubits that are physically far apart.
- This forces the compiler to insert SWAP gates, which are expensive and can negate the benefits of the optimization.

Proposed Solution (Time Permitting):

- Develop a **topology-aware** version of the tree synthesis.
- Restrict the ”root connection” step so that CNOTs are only generated between qubits connected on the physical hardware graph.

Ji Liu, Alvin Gonzales, Benchen Huang, Zain Hamid Saleem, and Paul Hovland.
QuCLEAR: Clifford extraction and absorption for quantum circuit optimization.