Streamlined OpenQASM Circuit Generation

Cedric Gaberle[1], Thomas Lippert[1,2], Manpreet S. Jattana[1,2]
[1]Modular Supercomputing and Quantum Computing, Goethe University Frankfurt am Main, Kettenhofweg 139, 60325 Frankfurt am Main, Germany
[2]Institute for Advanced Simulation, Jülich Supercomputing Centre, Forschungszentrum Jülich, D-52425 Jülich, Germany

Introduction and Motivation

Generating quantum computer instructions, even when using an intermediate representation such as OpenQASM, becomes increasingly tedious and time-consuming for larger ansätze.

We propose an open-source library implementing:
- An algorithm taking ansätze in Pauli string representation as input, commonly used for ansätze and Hamiltonians
- A classical parallel approach to further speed up the process

Pauli string input representation
- Any ansatz can be expressed using a suitable transformation, since Pauli and identity operations form a complete basis set
- Pauli string representation is much shorter, single character per operation, than corresponding OpenQASM representation
- Parameterized rotations and CNOT gates form a universal gate set, which is used to implement the ansatz

Why parallelization?
- Circuit generation is a possible bottleneck
  → Problem Hamiltonians and ansätze expected to grow with quantum computing capabilities
- Making use of the available computational resources
  → Quantum computers are expected to be part of larger supercomputing environments
- Even on personal computers, a lot of computing resources are idle during quantum computer's instruction generation

Ease of use
- Library implemented in highly efficient C++
- Made available as Python binding [1], simplifying integration into existing workflows

Parallelizable Generation Approach

n-qubit operators in an ansatz are of the form $e^{-ic\theta\boldsymbol{\sigma}^{\otimes n}} : \boldsymbol{\sigma} \in \{\sigma_x, \sigma_y, \sigma_z, I\}$, where c is the coefficient and θ the parameter

→ Sandwich parameterized rotation in the z-basis between CNOT gates [2]
→ For Pauli operations other than σ_z: Appropriate rotation of $\pi/2$ are used before and after CNOT

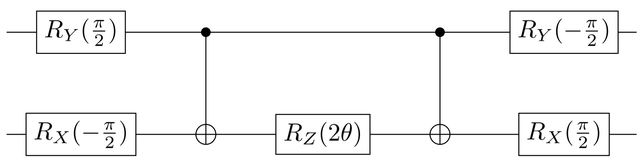


Fig. 1: Circuit implementation for $e^{-i\theta\sigma_x\otimes\sigma_y}$

- Input (see Fig. 2) encodes all the information
  → Pauli operator string
  → Coefficient
  → Grouping index

```
IIIXY  0.5  1
IIXIY  1.2  2
IIXYZ  0.8  3
```

Fig. 2: Example input file

Grouping index is used to indicate order and commuting operators
→ Commuting operators can share the same parameter
→ We consider each term separately and ansätze already in factorized form [3]

Encode each operator, i.e. each line of the input file, in parallel:

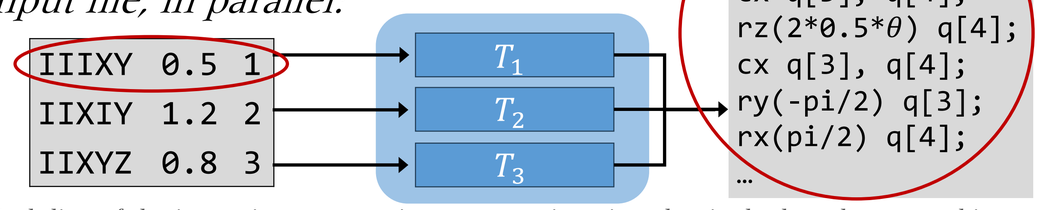


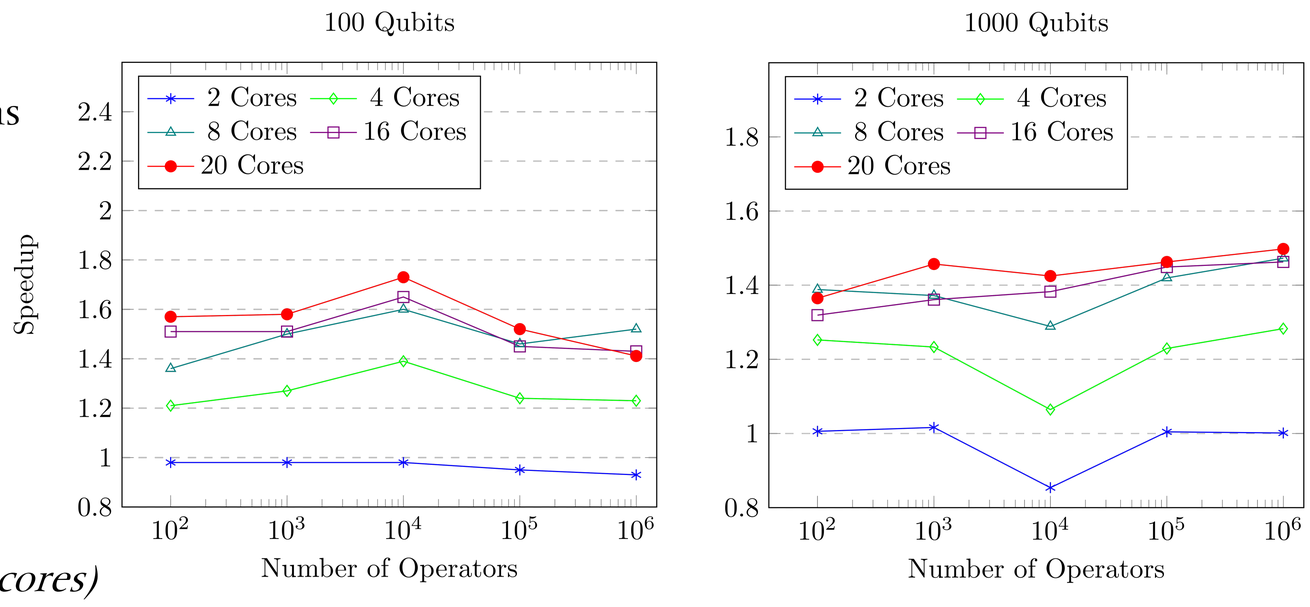
Fig. 3: Each line of the input, i.e. operator in an ansatz, is assigned a single thread, executed in parallel, transforming the Pauli string representation to OpenQASM per operator. Grouping indices ensure the correct order in the resulting OpenQASM file. Red circles highlight the conversion of one operator.

- Input representation is much more compact than OpenQASM
- Operator OpenQASM representation scales $\mathcal{O}(4k+1)$ where k is the number of non-identity operations
  → max. 2 rotations and 2CNOT gates per operator

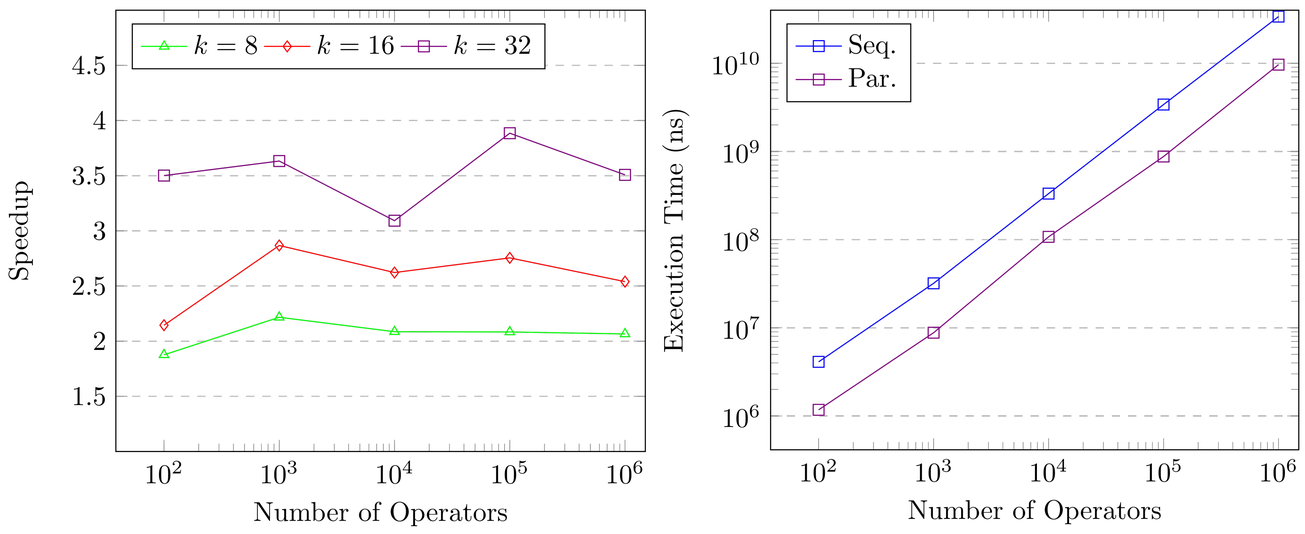
Parallel Implementation Speedup

Investigating the speedup for 100 & 1000 qubits per ansatz

Speedup for small k
- non-identity operations $k \leq 4$
- (100 qubits) ~1.6 overall speedup
- (1000 qubits) ~1.5 overall speedup
- An increase in resources leads to better performance



Speedup for larger k (20 cores)
- Workload increases
- Higher qubit utilization per operator benefits most from parallel execution
→ For k=32, execution time reduces from ~35 sec. to <10 sec.



Conclusion

- Algorithm reduces manual instruction generation overhead
- Parallel implementation further improves algorithm's runtime
- Considering today's hardware capabilities, it is not yet necessary to parallelize the algorithm
  However: We overcome a potential bottleneck for future cases
- Parallel algorithm performs best for large k

References

1. Cedric Gaberle, (2023) OpenQASM Parsing Library [Source Code]. https://github.com/msqc-goethe/QasmParserLibrary/tree/master
2. James D. Whitfield, et al.(2011) Simulation of electronic structure Hamiltonians using quantum computers, Molecular Physics, 109:5, 735-750
3. Manpreet Singh Jattana (2022) Applications of variational methods for quantum computers, Dissertation, RWTH Aachen University, pp. vii, 160

Contact E-Mails: gaberle@em.uni-frankfurt.de, th.lippert@fz-juelich.de, jattana@em.uni-frankfurt.de

msqc.uni-frankfurt.de