Recent Approaches for Efficient Compiling of Quantum Circuits

Michele Amoretti$^{1,2}$
1. Department of Engineering and Architecture - University of Parma, Italy
2. Quantum Information Science @ University of Parma, Italy
www.qis.unipr.it
Contact: michele.amoretti@unipr.it

Quantum Computing and HPC
CINECA, 19 December 2019
Quantum Software @ UniPr

Team

- Michele Amoretti (Associate Professor)
- Davide Ferrari (PhD student)

Topics

- high performance computing (classical and quantum)
- quantum compiling
- quantum networking (protocols for distributed monitoring, anonymity, leader election, etc.)

Source code

https://github.com/qis-unipr
Molecular Magnetism Group @ UniPr

Team

- Stefano Carretta (Full Professor)
- Paolo Santini (Full Professor)
- Elena Garlatti (Research Associate)
- Alessandro Chiesa (Post-doc)
- Roberto De Renzi (Full Professor)
- Francesco Petiziol (Post-Doc)
- Emilio Macaluso (PhD student)
- Simone Chicco (PhD student)
- Luca Crippa (PhD student / IBM)

Topics

Quantum hardware simulating four-dimensional inelastic neutron scattering

Multilevel structure exploited to encode a qubit with embedded Quantum Error Correction.

QC&HPC @ CINECA, 19/12/2019
Outline

- Quantum Compilation Problem
- State of the Art
- Deterministic Pattern-oriented Quantum Compiler (DPQC)
  - Performance Evaluation
Quantum Compilation Problem

Fast, device-aware implementation of quantum algorithms

A good quantum compiler must translate an input program into the most efficient equivalent of itself, getting the most out of the available hardware.

The quantum compilation problem in general is **NP-Hard**
Quantum Compilation Problem

On noisy intermediate-scale quantum (NISQ) devices:

- **Gate synthesis** = decomposition of an arbitrary unitary operation into a sequence of gates from a discrete set
- Compliance with the **coupling map**
- Noise awareness

Quality indicators:

- Circuit **depth**
- **Gate count**
- **Fidelity** of quantum states

Actions:

- Decide **initial mapping**
- Exchange qubit states using **SWAP gates**
Example: GHZ State

The ideal quantum circuit that generates the GHZ state requires only H and CNOT gates

Example: GHZ State

Ideal vs. compiled 8-qubit GHZ circuit
State of the Art

Zulehner et al., An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures, IEEE TCAD vol.38 no.7, 2019

- Partition circuit into layers
- For each layer find a CNOT compliant mapping
  - \( m!/(m-n)! \) possible mappings with \( m \) physical qubits and \( n \) logical qubits
  - **A* search algorithm** to find less expensive swap sequence
  - minimize additional operations to switch between subsequent mappings
State of the Art

Jandura, 2018 (LookaheadSwap in Qiskit)

Mapping = logical qubits ↔ physical qubits

Given a mapping and a list of upcoming CNOTs, repeat 4 times:

1. Find 4 most promising swaps and calculate the new mapping for each of them
2. For each mapping count how many CNOTs can be executed and remove them from the list

From the $4^4 = 256$ final mappings, choose the one that allowed for the most CNOTs to be executed and execute the first swap on the path to this mapping

Then start the whole algorithm again, until the circuit is completed
State of the Art

Li et al., Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices, ASPLOS ‘19, 2019

SWAP-based BidiREctional heuristic search (SABRE)

- Preprocessing
  - Compute distance matrix $D$ over the coupling map
  - Generate DAG from circuit to represent two-qubit gates dependencies
  - Initialize the front layer $F$ as the set of two-qubit gates without unexecuted predecessors
  - Generate random initial mapping

- Iterate over front layer $F$, stop when $F$ is empty
  - Remove executable gates from $F$ and add their successor to $F$
  - For those gates in $F$ that cannot be executed, select best SWAP sequence using an heuristic cost function based on distance matrix $D$
State of the Art

`qiskit.transpiler`

```
qc_comp = transpile(qc, backend=backend, coupling_map=coupling_map, pass_manager=pm)
```

The PassManager schedules the passes:

- Layout selection
- Unrolling
- Swap Mapping
  - BasicSwap
  - StochasticSwap
  - LookaheadSwap
- Gate optimizations
- etc.
Deterministic Pattern-oriented Quantum Compiler (DPQC)

The DPQC software is the implementation of **efficient and deterministic strategies** for the compilation of quantum circuits characterized by peculiar sequences (patterns) of two-qubit operators.

By compilation we mean the transformation of abstract quantum circuits into circuits suited to the characteristics of the available hardware device, preserving their correctness and trying to maximize their efficiency.

Among the possible characteristics of a quantum computer, the DPQC software considers the coupling map and tries to **minimize circuit depth and gate count**.
CNOT Sequences Without Gaps

- Algorithm `chain_layout()` analyzes the coupling map and orders the qubits in a *chain*
- Every logical qubit $q_i$ is associated with the *i*-th physical qubit in the chain
- Initial mapping satisfies all CNOTs
- In general, the problem of finding a path that visits every node only once is NP-complete
- `chain_layout()` exploits the qubit indexing and the graph structure to find a path with a deterministic and efficient approach
- Computational complexity $O(m)$, where $m$ is the number of physical qubits
CNOT Sequences With Gaps

- Initial mapping is not enough
- Algorithm \textit{path()} moves the qubits $q_1$ and $q_2$ involved in a remote CNOT close to each other:
  - iteratively computes the SWAP path between $q_1$ and $q_2$ neighbors
  - evaluates the distance between $q_x$ and $q_y$ as $|x - y|$ with $x$ and $y$ being their physical qubit indices in the coupling map
  - uses a cost function to choose which qubit to add to the path

\[
\begin{align*}
f(x, y) &= |x - y| \times SWAP\_DEPTH
\end{align*}
\]

- updates $q_1$ and $q_2$ mapping after every iteration
- Computational complexity $O(m)$
CNOT Cascades

Repeated sequences of inverted CNOT cascades

- CNOT cascades are equivalent to CNOT sequences
- Algorithm `analyze_circuit()` detects CNOT cascades and turns them into CNOT sequences
- Computational complexity $O(gn)$ with $g$ being the number of gates in the circuit
Inverted CNOT Cascades

- CNOT gates can be inverted by applying $H$ gates before and after involved qubits
- `analyze_circuit()` also detects inverted CNOT cascades and turns them into CNOT cascades
- Algorithm `gate_cancellation()` deletes double CNOT and double $H$ gates
- Computational complexity $O(dn^2)$
GHZ Results

Backend: QX20 Tokyo

QC&HPC @ CINECA, 19/12/2019
Quantum Chemistry Circuits

Used to compute the ground state wave function of simple molecular systems such as:

- Hydrogen $\text{H}_2$ with 4 qubits
- Lithium Hydride $\text{LiH}$ with 12 qubits
- Water $\text{H}_2\text{O}$ with 14 qubits
# UCCSD Results

## Circuit Depth

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Input Circuit</th>
<th>DPQC</th>
<th>Basic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2_UCCSD</td>
<td>82</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>LiH_UCCSD</td>
<td>8845</td>
<td>6717</td>
<td>12201</td>
<td>8026</td>
</tr>
<tr>
<td>H2O_UCCSD</td>
<td>15388</td>
<td>13413</td>
<td>12797</td>
<td>14789</td>
</tr>
<tr>
<td>Random20_UCCSD</td>
<td>125638</td>
<td>132865</td>
<td>204927</td>
<td>173672</td>
</tr>
</tbody>
</table>

## Compilation Time (s)

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>DPQC</th>
<th>Basic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2_UCCSD</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LiH_UCCSD</td>
<td>221</td>
<td>247</td>
<td>314</td>
</tr>
<tr>
<td>H2O_UCCSD</td>
<td>575</td>
<td>393</td>
<td>756</td>
</tr>
<tr>
<td>Random20_UCCSD</td>
<td>18448</td>
<td>11230</td>
<td>20127</td>
</tr>
</tbody>
</table>

Backend: QX20 Tokyo
## RyRz Results

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Input Circuit</th>
<th>DPQC</th>
<th>Basic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2_RyRz</td>
<td>73</td>
<td>40</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>LiH_RyRz</td>
<td>233</td>
<td>160</td>
<td>898</td>
<td>615</td>
</tr>
<tr>
<td>H2O_RyRz</td>
<td>273</td>
<td>190</td>
<td>1258</td>
<td>841</td>
</tr>
<tr>
<td>Random20_RyRz</td>
<td>393</td>
<td>279</td>
<td>3552</td>
<td>1629</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>DPQC</th>
<th>Basic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2_RyRz</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LiH_RyRz</td>
<td>4</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>H2O_RyRz</td>
<td>5</td>
<td>36</td>
<td>28</td>
</tr>
<tr>
<td>Random20_RyRz</td>
<td>8</td>
<td>93</td>
<td>65</td>
</tr>
</tbody>
</table>

Backend: QX20 Tokyo

QC&HPC @ CINECA, 19/12/2019
Other Benchmarks

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Input Circuit</th>
<th>DPQC</th>
<th>Basic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>9symml_195</td>
<td>19235</td>
<td>35406</td>
<td>38489</td>
<td>40002</td>
</tr>
<tr>
<td>clip_206</td>
<td>17879</td>
<td>34715</td>
<td>33877</td>
<td>38726</td>
</tr>
<tr>
<td>co14_215</td>
<td>8570</td>
<td>18294</td>
<td>18551</td>
<td>19458</td>
</tr>
<tr>
<td>dist_223</td>
<td>19694</td>
<td>36144</td>
<td>36488</td>
<td>42289</td>
</tr>
<tr>
<td>life_238</td>
<td>12511</td>
<td>23319</td>
<td>24974</td>
<td>25990</td>
</tr>
<tr>
<td>max46_240</td>
<td>14257</td>
<td>27466</td>
<td>27550</td>
<td>29650</td>
</tr>
<tr>
<td>sao2_257</td>
<td>19563</td>
<td>39268</td>
<td>40202</td>
<td>42988</td>
</tr>
<tr>
<td>sqn_258</td>
<td>5458</td>
<td>9999</td>
<td>10599</td>
<td>11271</td>
</tr>
<tr>
<td>sym10_262</td>
<td>35575</td>
<td>67362</td>
<td>64885</td>
<td>75168</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Compilation Time (s)</th>
<th>DPQC</th>
<th>Basic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>9symml_195</td>
<td>612</td>
<td>626</td>
<td>802</td>
<td></td>
</tr>
<tr>
<td>clip_206</td>
<td>759</td>
<td>570</td>
<td>817</td>
<td></td>
</tr>
<tr>
<td>co14_215</td>
<td>377</td>
<td>316</td>
<td>450</td>
<td></td>
</tr>
<tr>
<td>dist_223</td>
<td>783</td>
<td>626</td>
<td>951</td>
<td></td>
</tr>
<tr>
<td>life_238</td>
<td>358</td>
<td>397</td>
<td>538</td>
<td></td>
</tr>
<tr>
<td>max46_240</td>
<td>431</td>
<td>429</td>
<td>660</td>
<td></td>
</tr>
<tr>
<td>sao2_257</td>
<td>871</td>
<td>691</td>
<td>1036</td>
<td></td>
</tr>
<tr>
<td>sqn_258</td>
<td>148</td>
<td>170</td>
<td>239</td>
<td></td>
</tr>
<tr>
<td>sym10_262</td>
<td>1543</td>
<td>1119</td>
<td>1987</td>
<td></td>
</tr>
</tbody>
</table>

Backend: QX20 Tokyo
Future Work

- Testing the proposed approach with the **new coupling maps**
- Using extra qubits
- Studying **other circuit patterns**
Future Work

While effective, circuit depth and gate count are only pseudo-objectives. In reality, what matters is the **fidelity** of a computation when run on actual quantum hardware.

NISQ compilers excel when more **information** is made available to them **from the device**.

IBM Q **device properties** are shared openly and can be benchmarked, resulting in a bunch of compiler innovations.


Thank You!

Questions?
Sequence Offset

- $0 \leq \text{offset} < (m-n)$

- Used to map the qubits in the circuit to the qubits in the chain in the interval $\{0,..,n + \text{offset}\}$ instead of $\{0,..,n\}$

8 qubits chain with offset=12
Impact of Sequence Offset

UCCSD LiH

<table>
<thead>
<tr>
<th>offset</th>
<th>depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14652</td>
</tr>
<tr>
<td>1</td>
<td>10518</td>
</tr>
<tr>
<td>2</td>
<td>9984</td>
</tr>
<tr>
<td>3</td>
<td>8065</td>
</tr>
<tr>
<td>4</td>
<td>12720</td>
</tr>
<tr>
<td>5</td>
<td>11061</td>
</tr>
<tr>
<td>6</td>
<td>11875</td>
</tr>
<tr>
<td>7</td>
<td>9756</td>
</tr>
<tr>
<td>8</td>
<td>9575</td>
</tr>
</tbody>
</table>