Tools For Quantum and Reversible Circuit Compilation

- MARTIN ROETTELER
- PRESENTED BY HARSH KHETAWAT
- 11/19/2018
Introduction/Motivation

Multistage compilation of QAlgos:

- High level description of program → Net lists of circuits → Pulse sequences → Physical Quantum Computer

Key: Implement classical subroutines (oracles):

- Why?
- Underlying problem often involves classical data:
  - factoring (Shor’s),
  - HHL – for solving linear equations,
  - quantum walks
  - quantum simulation, etc.
- How best to implement on quantum computer?
Reversible Computing

How best to implement classical subroutines (oracles) on a quantum computer

Deals with:
- Minimize gate count for a given universal gate set
- Minimize resources such as:
  - Circuit depth
  - Number of qubits required, etc.

Compiling irreversible programs to QC:
- Hide classical subroutines in libraries – optimized collection of functions
- Tools to convert classical code → network of Toffoli gates (Quipper)

LIQUID provides REVS – tool to automatically convert Classical code → reversible networks
Idea behind REVS

Bennet’s method (1973)
- Reverse each time step
- Perform forward computation using step-wise reversible processes
- Copy out the result
- Undo all steps in the forward computation in reverse order

Solves reversible embedding problem
- Cost – large memory footprint as each intermediate results has to be stored
- Solution - Bennet’s new and improved method!! (1989)
- Pebble games
- Space vs Time tradeoff
Pebble game: case of 1D graph

Rules of the game: [Bennett, SIAM J. Comp., 1989]
- n boxes, labeled i = 1, ..., n
- in each move, either add or remove a pebble
- a pebble can be added or removed in i=1 at any time
- a pebble can be added or removed in i>1 if and only if there is a pebble in i-1
- 1D nature arises from decomposing a computation into “stages”

Example:

```
   #    i
1  1
```

```
   1  2  3  4
```

1 blue dot
Pebble game: case of 1D graph

Rules of the game: [Bennett, SIAM J. Comp., 1989]
- n boxes, labeled i = 1, ..., n
- in each move, either add or remove a pebble
- a pebble can be added or removed in i=1 at any time
- a pebble can be added or removed in i>1 if and only if there is a pebble in i-1
- 1D nature arises from decomposing a computation into “stages”

Example:

```
# i
1 1
2 2
```

```
1 2 3 4
```

1 2

1 2
Pebble game: case of 1D graph

Rules of the game: [Bennett, SIAM J. Comp., 1989]
• n boxes, labeled i = 1, ..., n
• in each move, either add or remove a pebble
• a pebble can be added or removed in i=1 at any time
• a pebble can be added of removed in i>1 if and only if there is a pebble in i-1
• 1D nature arises from decomposing a computation into “stages”

Example:

```
  #  i
  1  1
  2  2
  3  3
```

```
1  2  3  4
```

1  2  3  4
Pebble game: case of 1D graph

Rules of the game: [Bennett, SIAM J. Comp., 1989]
• n boxes, labeled i = 1, ..., n
• in each move, either add or remove a pebble
• a pebble can be added or removed in i=1 at any time
• a pebble can be added of removed in i>1 if and only if there is a pebble in i-1
• 1D nature arises from decomposing a computation into “stages”

Example:

```
<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
```

1 2 3 4
Pebble game: case of 1D graph

Rules of the game: [Bennett, SIAM J. Comp., 1989]
- n boxes, labeled $i = 1, \ldots, n$
- in each move, either add or remove a pebble
- a pebble can be added or removed in $i=1$ at any time
- a pebble can be added or removed in $i>1$ if and only if there is a pebble in $i-1$
- 1D nature arises from decomposing a computation into “stages”

Example:

<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

1 2 3 4
Pebble game: case of 1D graph

Rules of the game: [Bennett, SIAM J. Comp., 1989]

- n boxes, labeled i = 1, ..., n
- in each move, either add or remove a pebble
- a pebble can be added or removed in i=1 at any time
- a pebble can be added of removed in i>1 if and only if there is a pebble in i-1
- 1D nature arises from decomposing a computation into "stages"

Example:

```
#  i
1  1
2  2
3  3
4  4
5  3
6  2
```

```
1  2  3  4

1

2

3

4
```
Pebble game: case of 1D graph

Rules of the game: [Bennett, SIAM J. Comp., 1989]

- n boxes, labeled i = 1, ..., n
- in each move, either add or remove a pebble
- a pebble can be added or removed in i=1 at any time
- a pebble can be added or removed in i>1 if and only if there is a pebble in i-1
- 1D nature arises from decomposing a computation into “stages”

Example:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\text{1} & \text{2} & \text{3} & \text{4} \\
\end{array}
\]

\[
\begin{array}{cc}
\# & i \\
1 & 1 \\
2 & 2 \\
3 & 3 \\
4 & 4 \\
5 & 3 \\
6 & 2 \\
7 & 1 \\
\end{array}
\]
Pebble game: 1D plus space constraints

Imposing resource constraints:
• only a total of $S$ pebbles are allowed
• corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: ($n=3$, $S=3$)
Pebble game: 1D plus space constraints

Imposing resource constraints:
- only a total of $S$ pebbles are allowed
- corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: $(n=3, S=3)$

<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

1 2 3 4
Pebble game: 1D plus space constraints

Imposing resource constraints:
• only a total of $S$ pebbles are allowed
• corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: ($n=3, S=3$)

```
#  i
1  1
2  2
3  3
```

```
+---+---+---+---+
|   |   |   |   |
+---+---+---+---+
     |   |   |
+---+---+---+---+
     |   |   |
+---+---+---+---+
     |   |   |
+---+---+---+---+

1  2  3  4
```
Pebble game: 1D plus space constraints

Imposing resource constraints:
• only a total of $S$ pebbles are allowed
• corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: ($n=3$, $S=3$)

<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Pebble game: 1D plus space constraints

Imposing resource constraints:
• only a total of $S$ pebbles are allowed
• corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: $(n=3, S=3)$

<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Pebble game: 1D plus space constraints

Imposing resource constraints:
- only a total of $S$ pebbles are allowed
- corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: ($n=3$, $S=3$)

<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
Pebble game: 1D plus space constraints

Imposing resource constraints:
• only a total of $S$ pebbles are allowed
• corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: $(n=3, S=3)$

<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>
Pebble game: 1D plus space constraints

Imposing resource constraints:
- only a total of $S$ pebbles are allowed
- corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: ($n=3$, $S=3$)

```
    #  i
    1  1
    2  2
    3  3
    4  1
    5  4
    6  3
    7  1
    8  2
```
Pebble game: 1D plus space constraints

Imposing resource constraints:
• only a total of $S$ pebbles are allowed
• corresponds to reversible algorithm with at most $S$ ancilla qubits

Example: ($n=3$, $S=3$)

<table>
<thead>
<tr>
<th>#</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>
Use dynamic programming to determine best strategy for given n (steps) and S (pebbles)

Works for 1-D chains
More complex for general graphs

[Lange-McKenzie-Tapp 2000]

[Bennett ‘73]
REVS

Determining best strategy is program dependent and non-trivial

REVS:
- Boolean functions synthesized using heuristics and optimizations (ESOP)
- Circuits made reversible using:
  - Bennet’s method(s)
  - Uncompute data that is no longer needed (from data dependencies)

For example – SHA256
- No branching, uses simple boolean functions such as XOR, AND and bit rotations
- However, it has internal state between rounds
REVS

Modeled using Mutable Data Dependency (MDD) graphs
- Tracks data flow during classing computation
- Identify which parts can be overwritten / uncompumed (clean-up)

Clean-up on QC ≈ Garbage collection on classic computers

Outputs Toffoli network
- Imported in LIQU|>
- Used as part of quantum communication
- Supports compilation for different target architectures / abstract QC machine models
SHA-256

Ideal candidate:
- Stores state between rounds
- Simple binary functions

4x improvement in number of qubits required

Can also be applied to other hash functions
- SHA-3 and MD5

REVS allows exploration of trade-off space

<table>
<thead>
<tr>
<th>Rnd</th>
<th>Bennett Bits</th>
<th>Bennett Gates</th>
<th>Bennett Time</th>
<th>Eager Bits</th>
<th>Eager Gates</th>
<th>Eager Time</th>
<th>Reference Bits</th>
<th>Reference Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>704</td>
<td>1124</td>
<td>0.254</td>
<td>353</td>
<td>690</td>
<td>0.329</td>
<td>353</td>
<td>683</td>
</tr>
<tr>
<td>2</td>
<td>832</td>
<td>2248</td>
<td>0.263</td>
<td>353</td>
<td>1380</td>
<td>0.336</td>
<td>353</td>
<td>1366</td>
</tr>
<tr>
<td>3</td>
<td>960</td>
<td>3372</td>
<td>0.282</td>
<td>353</td>
<td>2070</td>
<td>0.342</td>
<td>353</td>
<td>2049</td>
</tr>
<tr>
<td>4</td>
<td>1088</td>
<td>4496</td>
<td>0.282</td>
<td>353</td>
<td>2760</td>
<td>0.354</td>
<td>353</td>
<td>2732</td>
</tr>
<tr>
<td>5</td>
<td>1216</td>
<td>5620</td>
<td>0.290</td>
<td>353</td>
<td>3450</td>
<td>0.366</td>
<td>353</td>
<td>3415</td>
</tr>
<tr>
<td>6</td>
<td>1344</td>
<td>6744</td>
<td>0.304</td>
<td>353</td>
<td>4140</td>
<td>0.378</td>
<td>353</td>
<td>4098</td>
</tr>
<tr>
<td>7</td>
<td>1472</td>
<td>7868</td>
<td>0.312</td>
<td>353</td>
<td>4830</td>
<td>0.391</td>
<td>353</td>
<td>4781</td>
</tr>
<tr>
<td>8</td>
<td>1600</td>
<td>8992</td>
<td>0.328</td>
<td>353</td>
<td>5520</td>
<td>0.402</td>
<td>353</td>
<td>5464</td>
</tr>
<tr>
<td>9</td>
<td>1728</td>
<td>10116</td>
<td>0.334</td>
<td>353</td>
<td>6210</td>
<td>0.413</td>
<td>353</td>
<td>6147</td>
</tr>
<tr>
<td>10</td>
<td>1856</td>
<td>11240</td>
<td>0.344</td>
<td>353</td>
<td>6900</td>
<td>0.430</td>
<td>353</td>
<td>6830</td>
</tr>
</tbody>
</table>
Using Dirty Ancillas

What are dirty ancillas?
- Qubits in unknown state
- Might be entangled in unknown way
- Available as scratch space

How can dirty ancillas be useful? Two scenarios currently known:
- Multiply controlled NOT operation
- Constant incrementer \( |x> \rightarrow |x + c> \)

Increment \( |x> \) by 1 example using unknown \( |g> \):
- \( g' \) is 2’s complement of \( g \Rightarrow g' – 1 = \text{not}(g) \)
- \( g + g' = 0 \)
- \( |x>|g> \rightarrow |x – g>|g> \rightarrow |x – g>|g' – 1> \rightarrow |x – g – g' + 1>|g' – 1> \rightarrow |x + 1>|g> \)
Repeat-Until-Success Circuits

Key idea: Use non-deterministic circuits (RUS circuits) for decomposition (Paetznick & Svore, 2014)

- Substantial reduction in T gates
- Shorter expected circuit length compared to purely unitary design
- Approximating to desired precision $\mathcal{E}$

Has been shown to efficiently synthesize any 1-qubit unitary

Number of repetitions are provably finite

Fig. 3. Repeat-Until-Success (RUS) protocol to implement a unitary $V$. 
Conclusion

REVS:
- Translate classical, irreversible programs $\rightarrow$ reversible circuits
- Not required to think in circuit centric manner
- Capture data dependencies/mutations using MDDs
- Heuristics to find optimal pebbling strategies

Reuse of qubits even if state is unknown/entangled
- Reduce circuit sizes

Implement unitaries probabilistically using protocols such as RUS
- Constant factor improvement in circuit size
Discussion

Reuse of dirty ancillas only possible for very specific situations

RUS protocol very interesting:
- Can we implement multi-qubit unitaries using RUS?

The paper doesn’t discuss heuristics used for finding optimal pebbling strategy
- What heuristics are used?
- Can we improve on it?