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:

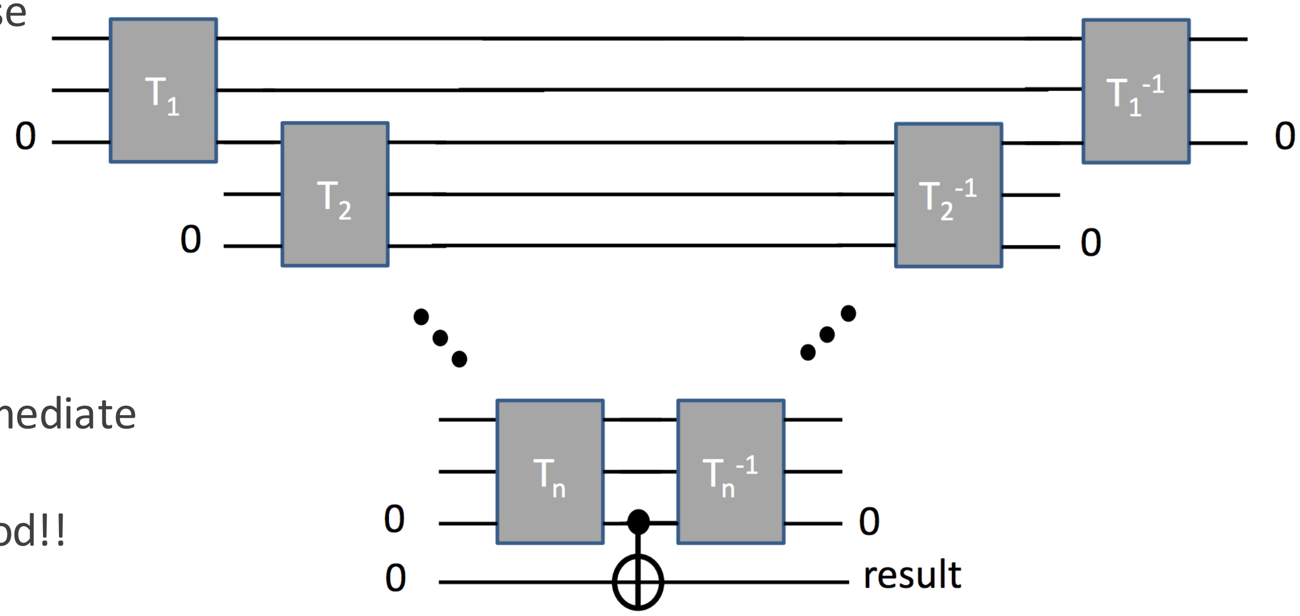
- Hide classical subroutines in libraries – optimized collection of functions
- Tools to convert classical code → network of Toffoli gates (Quipper)

LIQU|> 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

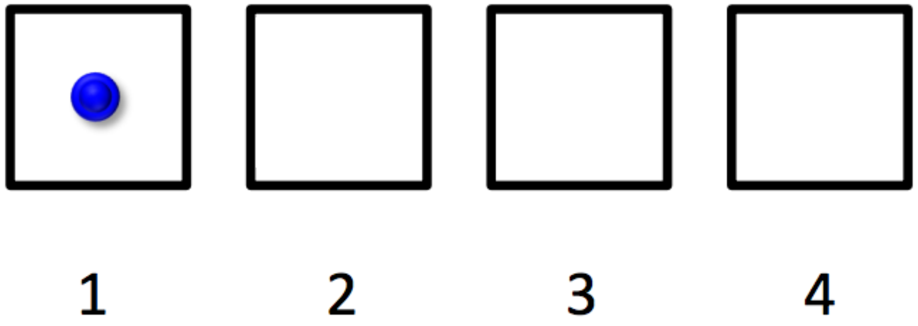
Pebble game: case of 1D graph

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 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”

#	i
1	1

Example:



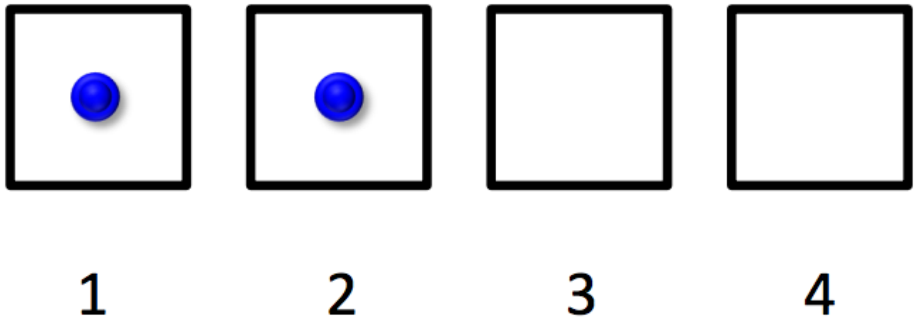
Pebble game: case of 1D graph

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



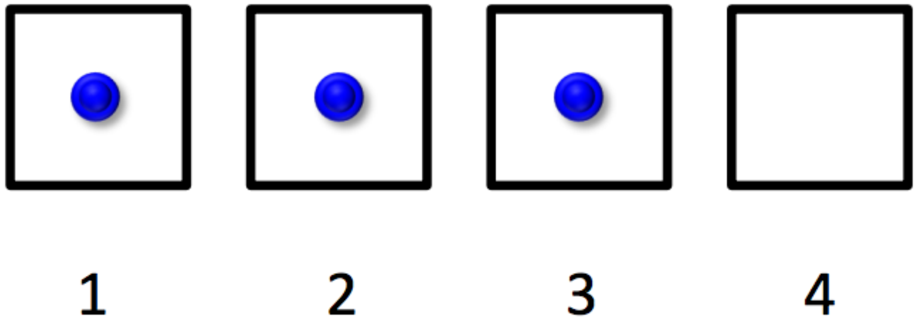
Pebble game: case of 1D graph

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”

#	i
1	1
2	2
3	3

Example:



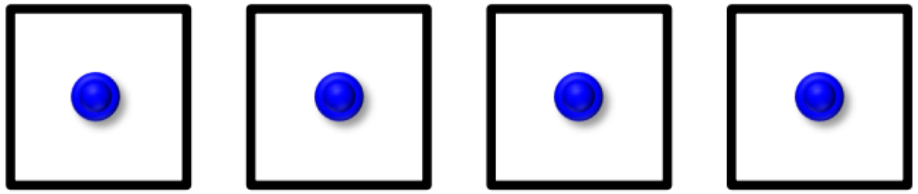
Pebble game: case of 1D graph

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”

#	i
1	1
2	2
3	3
4	4

Example:



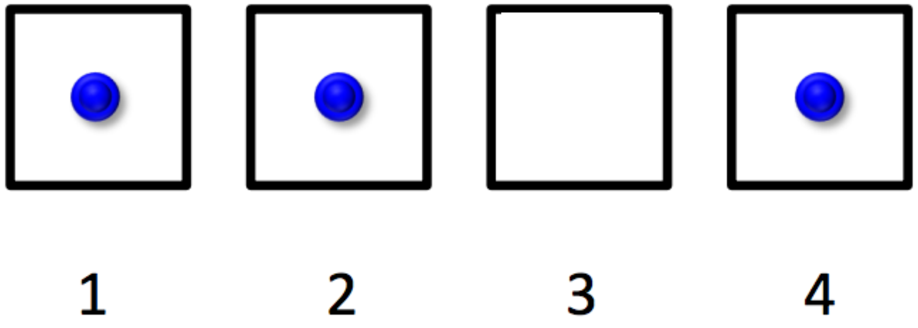
1 2 3 4

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:



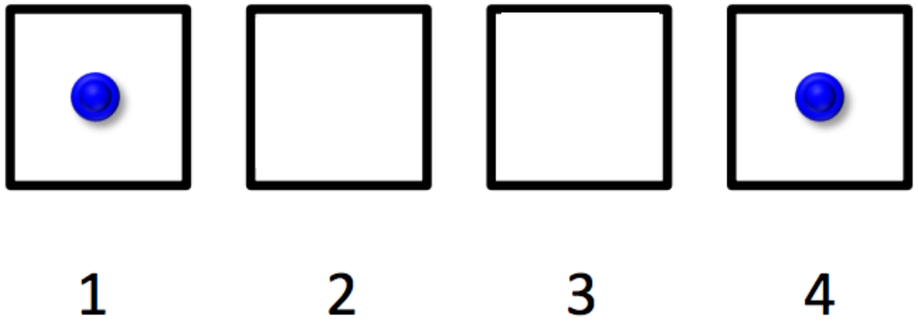
#	i
1	1
2	2
3	3
4	4
5	3

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:



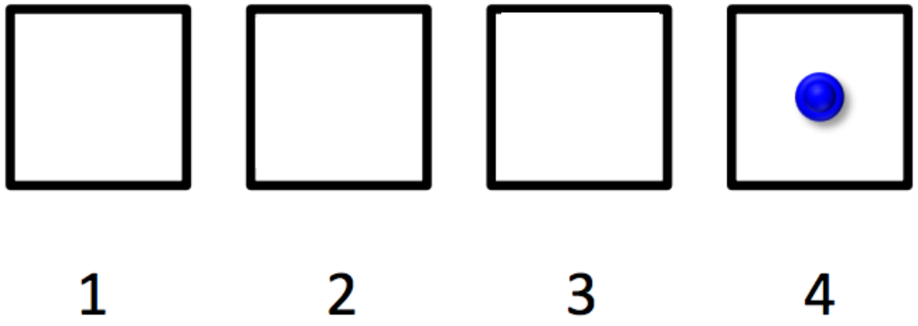
#	i
1	1
2	2
3	3
4	4
5	3
6	2

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
7	1

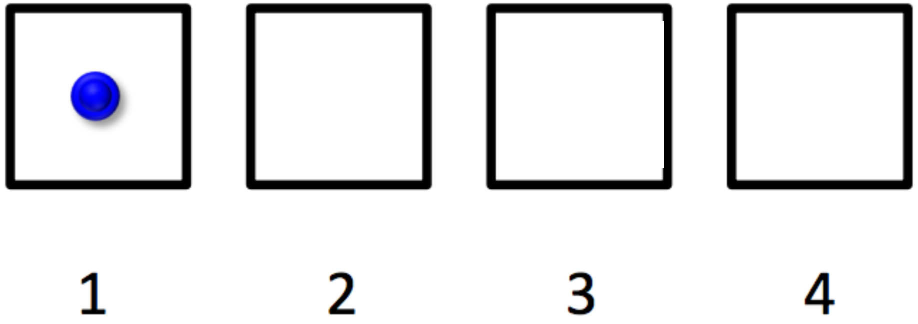
Pebble game: 1D plus space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

#	i
1	1

Example: (n=3, S=3)



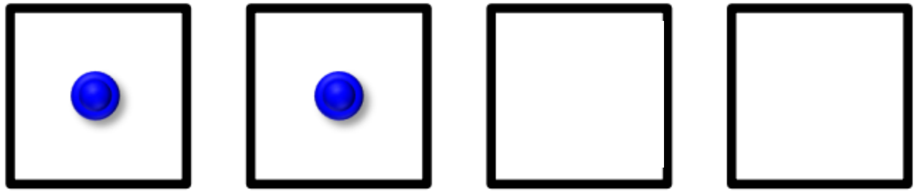
Pebble game: 1D plus space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

#	i
1	1
2	2

Example: (n=3, S=3)



1 2 3 4

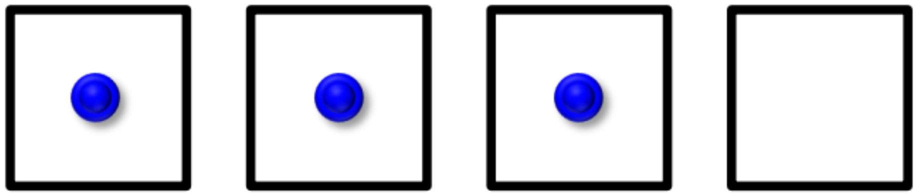
Pebble game: 1D plus space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

#	i
1	1
2	2
3	3

Example: (n=3, S=3)



1 2 3 4

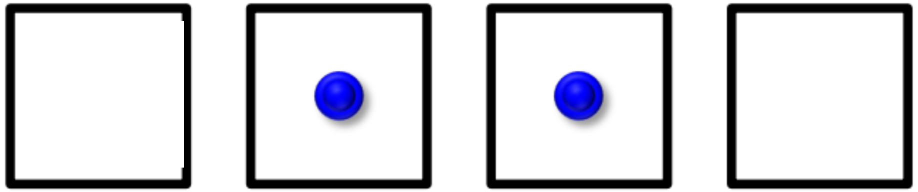
Pebble game: 1D plus space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

#	i
1	1
2	2
3	3
4	1

Example: (n=3, S=3)



1 2 3 4

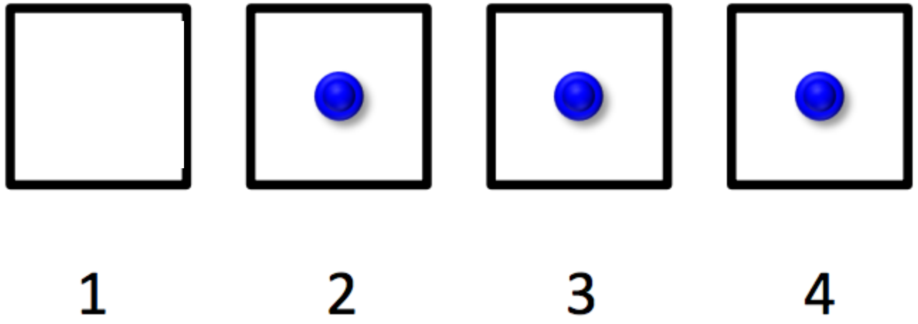
Pebble game: 1D plus space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

#	i
1	1
2	2
3	3
4	1
5	4

Example: (n=3, S=3)



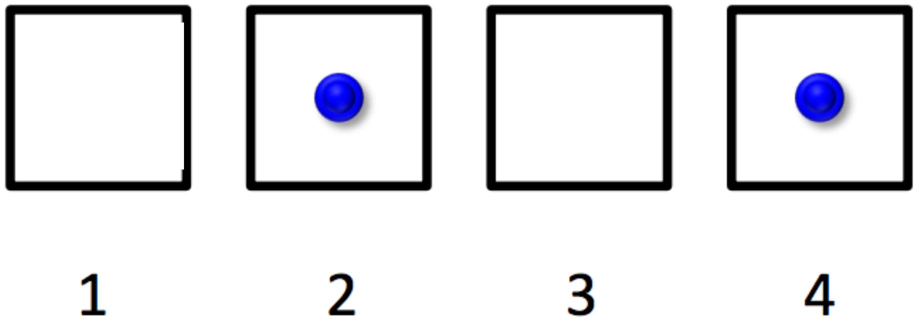
Pebble game: 1D plus space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

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

Example: (n=3, S=3)



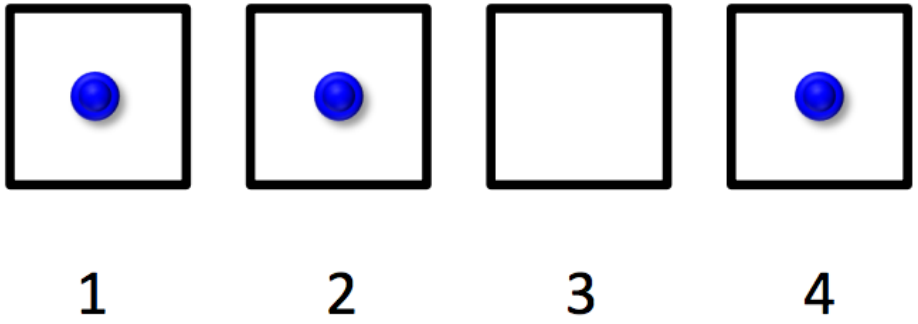
Pebble game: 1D plus space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

#	i
1	1
2	2
3	3
4	1
5	4
6	3
7	1

Example: (n=3, S=3)



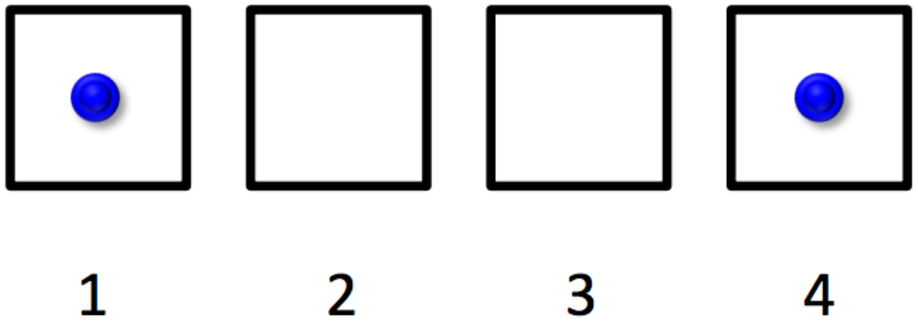
Pebble game: 1D plus space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

#	i
1	1
2	2
3	3
4	1
5	4
6	3
7	1
8	2

Example: (n=3, S=3)



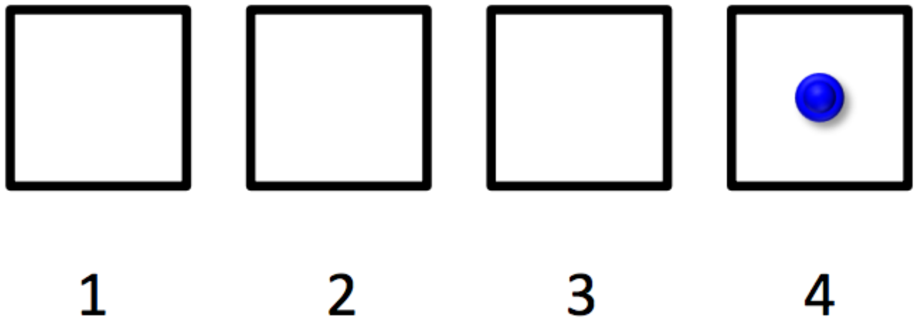
Pebble game: 1D plus space constraints

Imposing resource constraints:

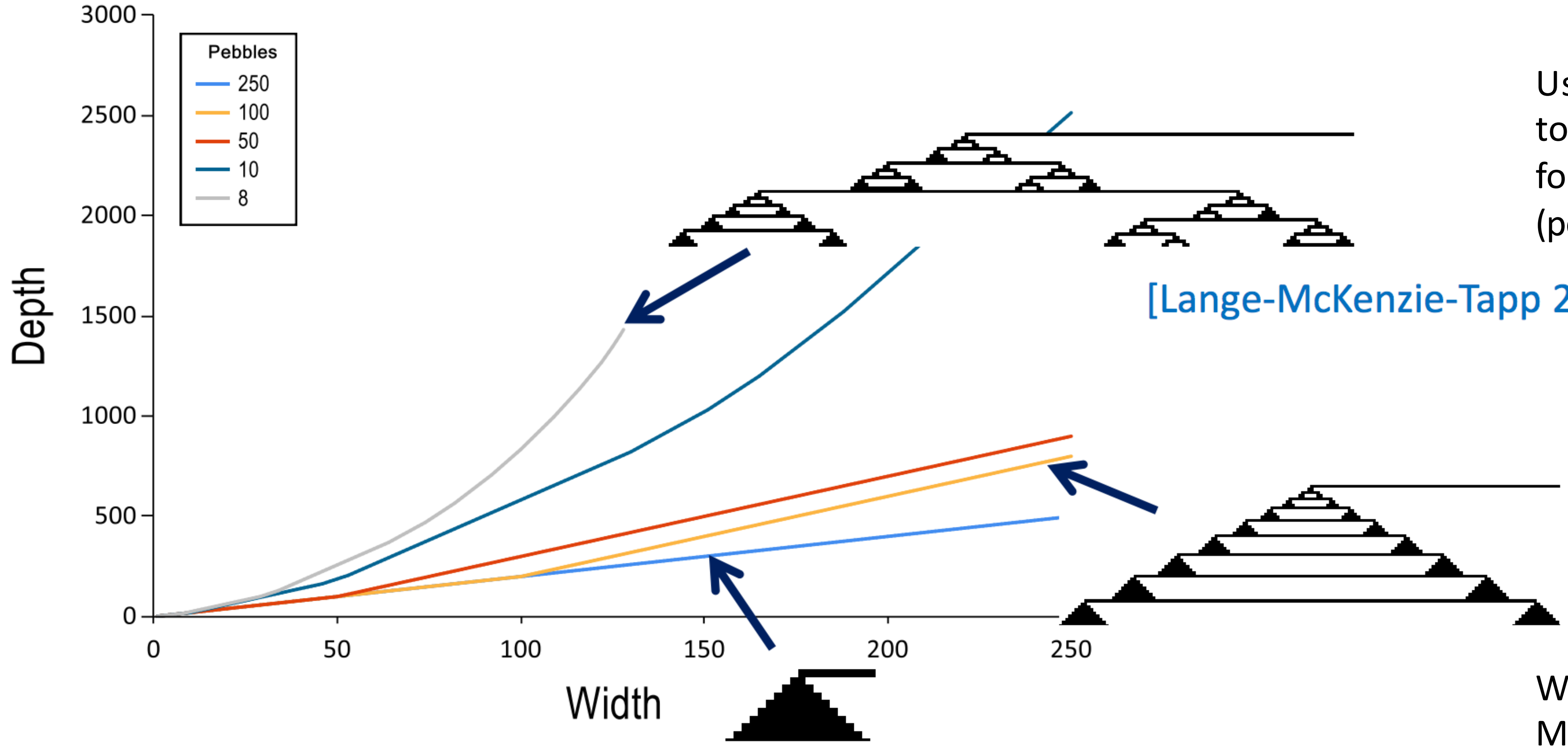
- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

#	i
1	1
2	2
3	3
4	1
5	4
6	3
7	1
8	2
9	1

Example: (n=3, S=3)



Pebble game: 1D plus space constraints



Use dynamic programming to determine best strategy for given n (steps) and S (pebbles)

[Lange-McKenzie-Tapp 2000]

[Bennett '73]

Works for 1-D chains
More complex for general graphs

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

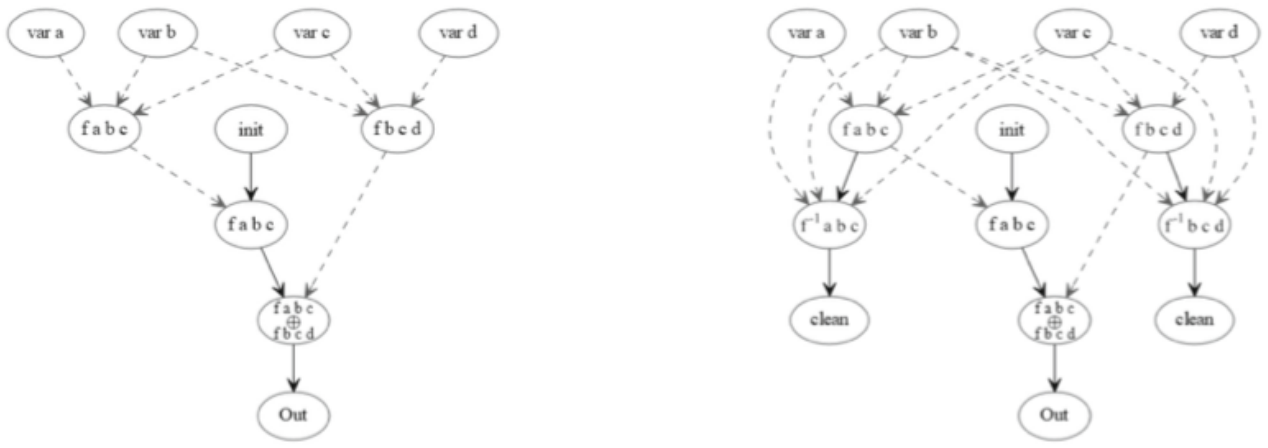
Modeled using Mutable Data Dependency (MDD) graphs

- Tracks data flow during classing computation
- Identify which parts can be overwritten / uncomputed (clean-up)

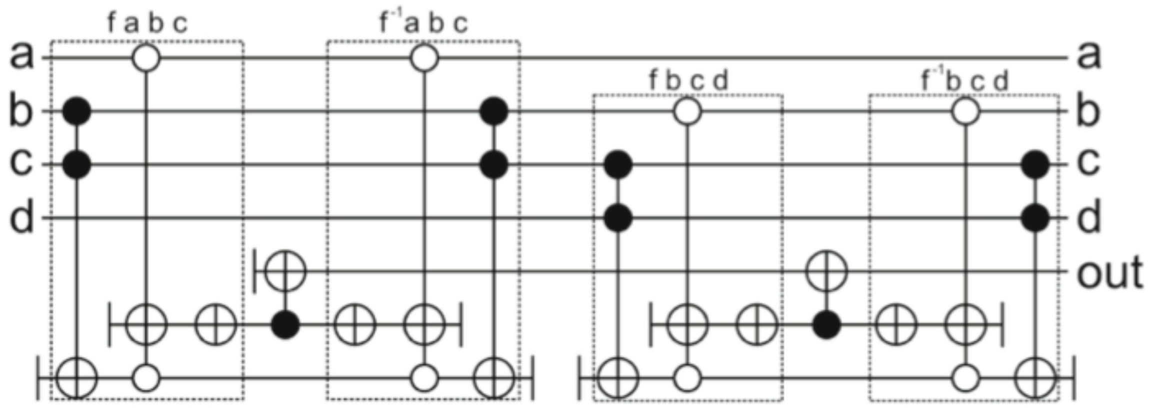
Clean-up on QC ≅ Garbage collection on classic computers

Outputs Toffoli network

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



(a) MDD for h before cleanup (b) MDD for h after eager cleanup



(c) Final resulting Toffoli network implementing the function h.

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 1. Comparison of different compilation strategies for the cryptographic hash function SHA-256.

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

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> → |x + c>

Increment |x> by 1 example using unknown |g>:

- g’ is 2’s complement of g => g’ – 1 = not(g)
- g + g’ = 0
- |x>|g> → |x – g>|g> → |x – g>|g’ – 1> → |x – g – g’ + 1>|g’ – 1> → |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 ε

Has been shown to efficiently synthesize any 1-qubit unitary

Number of repetitions are provably finite

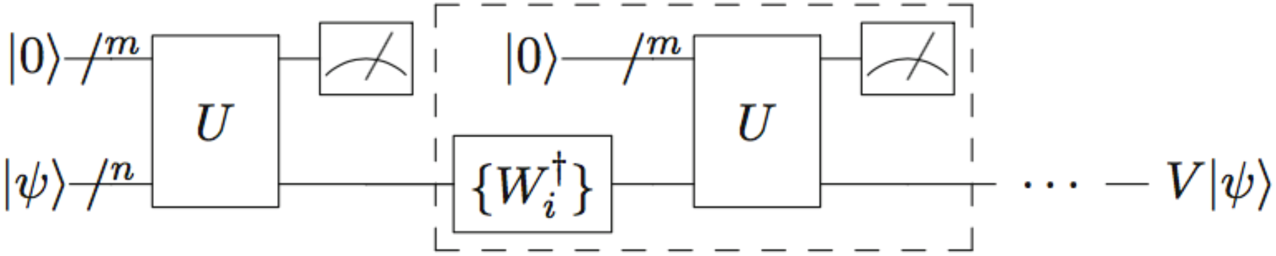


Fig. 3. Repeat-Until-Success (RUS) protocol to implement a unitary V.

Conclusion

REVS:

- Translate classical, irreversible programs → 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?