Specializing the hash table of a quantum decision diagram simulator for graph states

Author Antoni Spaanderman
a.v.spaanderman@student.tudelft.nl
Responsible Professor Tim Coopmans
Supervisor Juul Sanders

1. Background

Naive quantum algorithm simulation is slow and memory expensive. A Quantum Multiple-Valued Decision Diagram (QMDD) compresses the state vector by merging nodes in a tree made from the quantum state vector:

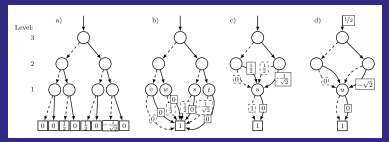
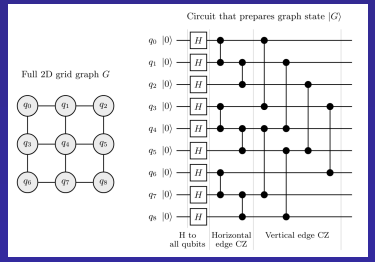


Diagram credit: [1]

When a new node is about to be added to a level, it needs to be compared to all existing nodes on that level, which is the job for a hash table. This research is about this hash table. The hash table will be used in a specific way by the QMDD simulator when running specific quantum algorithms. In this work, the quantum algorithms used are graph states, primarily 2D grid graph states. The final implementation will also be tested using the Deutsch-Jozsa algorithm to check if understanding and optimizations from graph states applies there too.

Graph states are an active research topic because of their use in measurement-based quantum computing and quantum error correction. This is a full 2D 3×3 grid graph, along with a quantum circuit to prepare the corresponding graph state, which is done using H and CZ gates:



2. Research question

How can a QMDD based quantum algorithm simulator be improved by specializing the hash table used for node merging, when preparing graph states?
Sub questions:

1. How many collisions does the hash table have to handle when preparing a graph state?
2. How is the hash table used, what patterns are there in the hash table calls?
3. How can patterns in the hash table calls be predicted?
4. Based on the prediction, how can the hash table be improved?
5. How well does the new hash table work on a different quantum circuit family?

3. Methodology

General hash table design:

- Nodes are never compared across levels: give every level its own hash table.
- Insertions always follow a failed lookup of the same node: combine the lookup and insert function to lookup-or-insert.
- No floating point complex number handling needed in most cases, and in all cases for graph states: turn off floating point complex number handling until it is needed.
- Use SHA256 hash for research purposes: almost perfect uniformly distributed output, but as a cryptographic hash it is slow (takes less than a microsecond still) compared to practical hash functions.

Specialized hash table design for graph states:

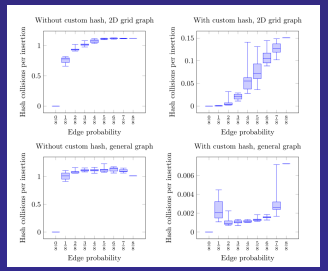
- Low edge label is constant 1, high edge label is -1 or 1. Nodes are almost uniformly distributed over high edge node IDs: use high label sign bit and high edge node ID in the hash function.
- Assign node IDs on each level using a slab allocator to reuse IDs of deleted nodes. This lowers the maximum hash value and thus the needed memory.

Measuring performance:

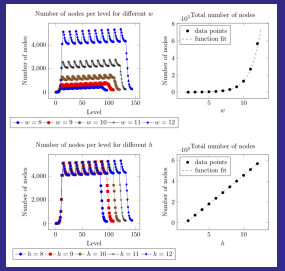
- Instrument hash tables with collision trackers.
- Visualize nodes on a level as pixels in images to assess effectiveness of the hash function.
- Testing using the Deutsch-Jozsa algorithm.

4. Results

Comparing with or without specialized hash table when simulating 2D grid graph states, collision count drops to 7 times less, on general graph states it drops to more than 100 times less.



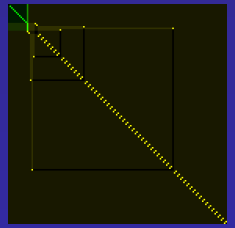
Node counts are plotted for each level to identify how they relate to the dimensions of the grid graph. A sawtooth pattern appears and node counts react differently to increasing the grid width (top 2 graphs) compared to increasing the grid height (bottom 2 graphs).



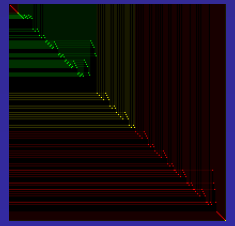
Visualizations were made to assess specialized hash function effectiveness:

- One image corresponds to one QMDD level.
- A bright pixel at (x,y) means a node with low edge node ID x and high node ID y exists.
- The real part of the high edge label determines the color: green for positive, red for zero and negative. Yellow is the result of blending green and red at the same pixel.
- Dim pixels are created to the left and above bright pixels to clearly see when multiple nodes share the same x or y.

Visualization of level 8 (stores 8-qubit states) after preparing a graph state.
The x and y values of the nodes are very correlated, most are seen very close to the y=x diagonal.
The pixels are also spread out very evenly across the x and y axis, as can be seen from the almost constant color in the dim pixels in the top row and left column.



Visualization of level 8 after running the Deutsch-Jozsa algorithm.
Nodes are spread out more evenly across the x axis than the y axis, but the other way around also happens in other levels. They are also more scattered than comparing with the visualization on the left, but still grouped around the y=x diagonal.



5. Discussion

- Most nodes inserted in the hash table do not need collision handling.
- The sawtooth pattern is not caused by properties of the graph state, instead it is caused by the preparation process.
- Node counts are very predictable and follow an exponential relation when increasing grid width and a linear relation when increasing grid height.
- As can be seen in the bottom left image of the results, only five rows have more than one pixel of the same color, meaning only five nodes in that level produce colliding hash values. This is in line with the first graph.

6. Conclusion and Future work

A QMDD can be substantially improved for graph states specifically, by adding a hash table that handles most QMDD nodes without needing collision handling thanks to a specialized hash function. Along with that, it speeds up other algorithms as well, when those algorithms are implemented using a specific gate set, if possible. In practice this was tested using the Deutsch-Jozsa algorithm.

Future work:

- Model and/or proof for QMDD level sizes in terms of grid width and height.
- Multithreading on the QMDD.

References
[1] Lieuwe Vinkhuijzen, Tim Coopmans, David Elkouss, Vedran Dunjko, and Alfons Laarman. Limdd: A decision diagram for simulation of quantum computing including stabilizer states. *Quantum*, 7:1108, 2023.

TU Delft