Data structures for quantum circuit verification and how to compare them

Vinkhuijzen, L.T.

Summary

Quantum computers are a proposed fundamentally new type of computer. They aim to perform some computations much faster than previously possible by exploiting phenomena at the quantum scale, called superposition and entanglement. If and when large ones are be built, then they promise computational speedups for several important problems, notably in chemistry and physics, potentially contributing to important applications such as better medicines, batteries and solar panels, among others, thus contributing to and accelerating the transition to green energy and a sustainable economy. In order to realize these promises, we will need tools which analyze, simulate, compile, and verify these new computers and the algorithms that run on them. Meanwhile, of course, we still require tools to compile, analyze and verify classical software. The goal of this thesis is to contribute to a tool set for these two use cases.

Before we describe our contributions, however, let us describe the problem we are trying to solve and what makes it so difficult.

- When analyzing a quantum algorithm, we have a description of it in terms of a set of qubits and a series of quantum gates and measurements that are to be applied to the qubits. We wish to answer questions such as, "what is the most likely output?" or, "how likely is the computer to end up in a particular state?"

  The biggest difficulty is that, if a quantum computer consists of n qubits, then its state at any point in time is described by a list of 2^n complex numbers. This exponential increase makes it infeasible to store the list without using compression techniques when n becomes too large.

- In classical model checking, the goal is to decide whether an algorithm satisfies a specification. For example, we may wish to check that a program never enters

a state where two threads receive access to a resource that is supposed to be mutually exclusive. The difficulty is that, if a system consists of n bits, then the set of possible states grows as 2^n – exponentially, again, in the number n of bits.

In both cases, a tool which analyzes the system will be successful if it manages to tackle the exponentially-sized state space. Because the challenges in the quantum and classical cases are so similar, this allows us to focus on tackling their shared challenge: to find enough structure in the state space to compress it into a manageable amount of space. A powerful tool may then deliver value for both the above use cases.

Our tool of choice in this thesis is the *decision diagram* (DD). Decision diagrams are a versatile data structure with a rich theory and with applications in many parts of computer science, including analyzing quantum computers and doing (classical) model checking. The role of decision diagrams in these scenarios is to provide a complete snapshot of the state of the computation at each moment.

This summary is too short for a thorough description of DDs. Instead, we briefly list the properties that makes them attractive for our purposes. A DD is a lossless compression of a list of numbers (that is, the diagram represents the same list but uses only a modest amount of memory. This is analogous to saying that a Boolean formula is a compressed representation of its truth table). Importantly – and this distinguishes DDs from other compression methods – one can perform operations on the data without decompressing the DD first. For example, suppose that we are considering a certain quantum state $|\varphi\rangle$, which may represent the intermediate state during a quantum computation, and suppose that we know which quantum logic gate this algorithm will next apply. We wish to obtain a description of the state $|\psi\rangle$ after the quantum logic gate has been applied. This is relatively straightforward to do if we operate on the state vector; unfortunately, this always takes $\mathcal{O}(2^n)$ time, which prohibits us from applying the state vector method for problems of interesting size (because then n is too large). Instead, given a DD representing the current quantum state $|\varphi\rangle$, there is an algorithm which constructs a new DD representing the state $|\psi\rangle$ of the system after the quantum logic gate has been applied. In many cases, this process is efficient, because we do not need to decompress the DD to an (exponentially long) state vector in order to obtain a description of the state after the gate is applied. Designing such algorithms and the DDs on which they operate is the primary technical objective of this thesis.

The method described above allows us to analyze any quantum circuit in principle;

in practice, however, the DD may become too big to fit in computer memory, or operations on the DD take too much time; when that happens, our analysis has failed. The size of the DD and time taken by operations depends on how complex the quantum state is and on the type of DD used. Again, understanding and addressing these bottlenecks by analyzing existing and designing new DDs and algorithms to operate on them, is the primary technical goal of this thesis, and is arguably the primary goal of research on decision diagrams more broadly.

Indeed, existing decision diagrams could not adequately analyze many quantum algorithms; not even, for example, so-called stabilizer circuits, even though stabilizer circuits can in fact be efficiently simulated and analyzed using other methods. Therefore, in this thesis, we present a novel type of decision diagram, which we call the Local Invertible Map Decision Diagram (LIMDD) which can do everything that previous decision diagrams could do, plus stabilizer circuits, and, as we shall see, much more. By analyzing our design on paper and implementing it in software, we are able to show that our new DD outperforms existing DDs both in theory and in practice, in Chapter 3 and Chapter 4.

There are many different architectures for decision diagrams, both because new architectures improve upon previous ones and because different designs cater to different use cases. The strengths and weaknesses of these DDs are typically assessed using three criteria: *succinctness* (how large is the decision diagram?), *tractability* (which operations on a DD run in polynomial-time?) and *rapidity* (is one DD faster than another?). However, we found that (1) existing methods for quantum simulation had not previously been mapped out along these criteria; and (2) there was no good method to tell which of two data stuctures is more rapid. Therefore, (1) we provide such a map and (2) we introduce just such a method, by which one can tell when one data structure is more rapid than another, in Chapter 5. In the process, we discover a surprising connection between two existing methods; namely, QMDDs are a special type of matrix product state. Moreover, matrix product states are strictly more rapid than QMDDs. Previously, these two data structures had not been compared; our work therefore brings together two directions of research.

In Chapter 6, we examine an old decision diagram called the *Disjoint Support Decomposition Decision Diagram* (DSDBDD). This DD is not a quantum tool; it can be used for classical model checking. Although the DSDBDD had a software implementation, little was known analytically about its expressive power. We fill this gap by showing

that it is, in fact, exponentially more succinct than several other decision diagrams.

Lastly, in Chapter 7, we use the *Sentential Decision Diagram* for classical model checking. Here we are given a computer program with a finite number of variables and are asked what the reachable states are. The goal is to build an SDD which represents the set of reachable states of the input program. Our principal challenge in this case is to choose the SDD's *variable tree* ("vtree") well, because the vtree determines the size of the SDD. To this end, we have the opportunity to analyze the program before we start computing the reachable states using decision diagrams. This allows us to tabulate, for each variable, with which other variables it comes into "contact;" e.g., in an assignment statement, several variables are used to compute the value of another. We empirically compare several different heuristics for choosing the vtree based on information about the interaction of the program's variables.