GATE-LEVEL SIMULATION OF QUANTUM CIRCUITS

GEORGE F. VIAMONTES, MANOJ RAJAGOPALAN,
IGOR L. MARKOV, AND JOHN P. HAYES
The Univ. of Michigan, Advanced Computer Architecture Lab., Ann Arbor, MI 48109-2122
E-mail: {gviamont,rmanoj,imarkov,jhayes}@eecs.umich.edu

Simulating quantum computation on a classical computer is a difficult problem. The matrices (operators) representing quantum gates, and vectors modeling qubit states grow exponentially with an increase in the number of qubits. However, by using a new data structure called the Quantum Information Decision Diagram (QuIDD) that exploits the structure of quantum operators, many of these matrices and vectors can be represented in a form that grows polynomially. Using QuIDDs, we implemented a general-purpose quantum computing simulator in C++ called QuIDDPro and tested it on Grover's algorithm. We observed that our QuIDD technique asymptotically outperforms other known simulation techniques.

1 Introduction

Richard Feynman observed in the early 1980s that simulating quantum processes on classical hardware seems to require super-polynomial (in the number of qubits) memory and time. Traditional array-based representations are often insensitive to the actual values stored, and even sparse matrix storage offers little improvement for quantum operators with no zero matrix elements (e.g. Hadamard operators). Gottesman[6] identified a number of special-case quantum circuits for which tailor-made simulation techniques require only polynomial memory and runtime. However, he noted that these "restricted types of quantum circuits fall short of the full power of quantum computation." Thus, in cases of major interest, such as Shor's and Grover's algorithms, quantum simulation is still performed with straightforward linear-algebraic tools and requires astronomic resources.

A number of "programming environments" for quantum computing were proposed recently that are mostly front-ends to quantum circuit simulators. Their back-ends typically use linear algebra methods to multiply matrices and require super-polynomial computational resources in the number of qubits. The potential benefits of efficient linear-algebraic operations on compressed arguments are immense.

Our approach uses graph-based techniques to improve asymptotic time and memory complexity of quantum simulations by exploiting the structure of quantum operators. Although abstract worst-case complexity is still exponential, our approach achieves very substantial performance gains in many important cases.

Other advanced simulation techniques, e.g., MATLAB's "packed" representation, include data compression, but often must decompress the operands of matrix-vector multiplication. A notable exception is Greve's simulation[5] of Shor's algorithm that uses *Binary Decision Diagrams* (BDDs)[2]. Probability amplitudes of individual qubits are modeled by single decision nodes. This only captures superpositions where every participating qubit is rotated by ± 45 degrees from $|0\rangle$ towards $|1\rangle$. Though Greve's BDD representation cannot simulate arbitrary quantum circuits, the idea of modeling quantum states with a BDD-*like* structure is appealing and motivates our approach.

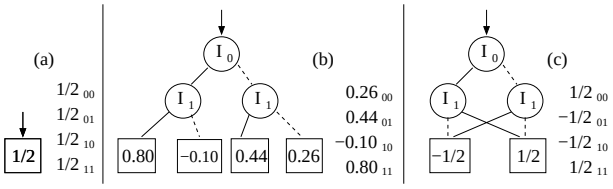

Figure 1. QuIDD examples of (a) best, (b) worst, and (c) mid-range complexity.

2 QuIDD Theory

The *Quantum Information Decision Diagram* (QuIDD) was born out of the observation that vectors and matrices which arise in quantum computing exhibit a lot of structure. More complex operators obtained from the tensor product of simpler matrices continue to exhibit these common substructures which BDDs can capture. BDDs and operations for manipulating them were originally developed by Lee[1] and extended to Reduced Ordered BDDs (ROBDDs) by Bryant[2] to handle large Boolean functions efficiently. An ROBDD is a *directed acyclic graph* (DAG) with up to two outgoing edges per node, labeled *then* and *else*. Algorithms that perform operations on ROBDDs are typically recursive traversals. While not improving worst-case asymptotics, in practice ROBDDs achieve exponential space compression and runtime improvements by exploiting various types of structure in applications.

Beyond the domain of digital logic design, ROBDD variants have been adopted in many contexts. Multi-Terminal Binary Decision Diagrams (MTBDDs)[3] and Algebraic Decision Diagrams (ADDs)[4] with their integrated linear-algebraic operations are particularly relevant to the task of simulating quantum systems. We have developed a further refinement, the QuIDD, to compress complex-valued matrices and vectors and operate on them in compressed form. Our C++ implementation of QuIDDs, called QuIDDPro, uses ADDs, but can, in principle, use MTBDDs as well. Space and time complexities of our simulations of n-qubit systems range from $O(1)$ to $O(2^n)$, but the worst case is not typical.

Vectors and Matrices. Figure 1 shows the QuIDD structure for three 2-qubit states. We consider the indices of the four vector elements in binary, and define their bits as decision variables of QuIDDs. A similar definition is used in ADDs.[4] For example, traversing the *then* edge (solid line) of node I_0 in Figure 1c is equivalent to assigning the value 1 to the first binary digit of the vector index. Traversing the *else* edge (dotted line) of node I_1 in the same figure is equivalent to assigning the value 0 to the second binary digit of the index. These traversals bring us to the terminal node $-\frac{1}{2}$, which is precisely the value at index 10 in the vector representation.

QuIDD representations of matrices extend those of vectors by adding a second type of variable node and enjoy the same reduction rules and compression benefits. Consider the 2-qubit Hadamard matrix annotated with binary row and column indices shown in Figure 2a. In this case there are two sets of indices: The first (vertical) set corresponds to the rows, while the second (horizontal) set corresponds to the columns. We assign the variable name R_i and C_i to the row and column index variables respectively. This distinction between the two sets of variables was

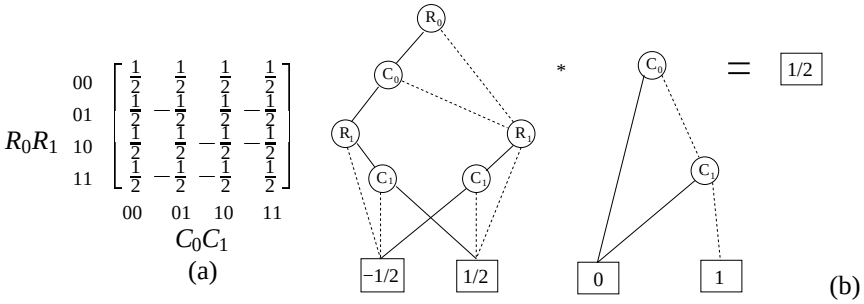

Figure 2. (a) 2-qubit Hadamard, and (b) its QuIDD representation multiplied by $|00\rangle = (1,0,0,0)$.

originally noted in several works including that of Bahar et al.[4] Figure 2b shows the QuIDD form of this sample matrix where it is used to modify the state vector $|00\rangle = (1,0,0,0)$ via matrix-vector multiplication.

QuIDD Operations. Most operations defined for ADDs also work on QuIDDs with only slight modification. A key example is matrix multiplication, which is an extension of the dot-product operation and implemented as a recursive procedure adapted from the well-known *BDD-Apply* function.[2] For more details, see our technical report.[7] Tensor products can be implemented as follows. By definition, $A \otimes B$ multiplies each element of A by the entire matrix (or vector) B producing a new matrix (or vector). Multiplication of the terminal values is done by first shifting the variable numbers in B after those in A followed by a call to the recursive *Apply* function with an argument that directs *Apply* to multiply when it reaches the terminals of both operands.[7] Since QuIDD operations are variants of *Apply*, they have complexity $O(|A| \cdot |B|)$, which is polynomial in the number of qubits if both $|A|$ and $|B|$ are.

3 Simulating Grover's Algorithm with QuIDDs

We simulated quantum circuits for Grover's algorithm[8] in the state-vector representation by performing matrix-vector multiplications and tensor products using QuIDDs. Ideal numbers of iterations for Grover's algorithm are computed following Boyer et al.[9] Oracle circuits are implemented using k-CNOT gates. To evaluate QuIDD representations of all relevant quantum gates, we measured their size at n-qubits, where $n = 20..100$ and observed that memory usage grew linearly.[7] Thus, using QuIDDs, simulation of Grover's algorithm is not memory limited even at 100 qubits. Results in Table 1 deal with an oracle searching for one element out of 2^n. As shown, QuIDDPro achieves asymptotic memory savings compared to known interpreted and compiled numerical analysis packages. The overall runtimes are still exponential in n because Grover's algorithm requires an exponential number of iterations, even on an actual quantum computer. We also studied a "mod-1024" oracle[7] searching for elements whose ten least significant bits are 1. We observed that memory usage for the mod-1024 oracle for up to $n = 25$ qubits grew as $(7.592 + 0.041n)$. For additional details on numerical precision, round-off errors, etc. see our technical report.[7]

4 Conclusions and Future Work

In this work, we proposed and tested a new technique for simulating quantum circuits using a data structure called a QuIDD. We have shown that QuIDDs enable practical,

Table 1. Simulating Grover's algorithm with n qubits using Octave (Oct), MATLAB (MAT), Blitz++ (B++) and our simulator QuIDDPro (QP). Results were produced on a 1.2GHz AMD Athlon with 1GB RAM running Linux. Memory usage for MAT and Oct is lower bounded by the size of the state vector and conditional phase shift operator; B++ and QP memory usage is measured as the size of the entire program. Runtimes for MAT and Oct are not shown past 15 qubits as simulation time was limited to 20 hours.

Runtime (s)				
n	Oct	MAT	B++	QP
10	89.4	14.0	0.22	0.20
11	2.94e2	45.9	0.72	0.39
12	9.26e2	1.53e2	2.22	0.88
13	3.09e3	5.80e2	6.92	1.94
14	1.36e4	5.90e3	23.09	4.79
15	7.10e4	5.92e4	70.4	9.32
16	TIME-OUT	TIME-OUT	2.13e2	22.2
17	TIME-OUT	TIME-OUT	6.34e2	50.7
18	TIME-OUT	TIME-OUT	1.92e3	1.13e2
19	TIME-OUT	TIME-OUT	5.74e3	2.00e2
20	TIME-OUT	TIME-OUT	1.74e4	3.25e2

Memory Usage (MB)				
n	Oct	MAT	B++	QP
10	3.60e-2	2.00e-2	1.95e-2	0.211
11	6.80e-2	4.40e-2	7.03e-2	0.207
12	0.132	9.20e-2	7.42e-2	0.281
13	0.260	0.188	0.129	0.426
14	0.268	0.264	0.250	0.444
15	0.524	0.520	0.500	0.605
16	1.04	1.03	1.00	0.840
17	2.06	2.06	2.00	0.965
18	4.11	4.10	4.00	1.59
19	8.20	8.20	8.00	1.77
20	16.4	16.4	16.0	2.04

generic and reasonably efficient simulation of quantum computation. Their key advantages are faster execution and lower memory usage. In our experiments, QuIDD-Pro achieves exponential memory savings compared to other known techniques. Our ongoing work focuses on the growth of required precision and on simulating other quantum algorithms, such as Shor's, where we plan to simulate the effects of errors and decoherence. We are also attempting to describe quantum gates that have polynomial QuIDD representations and thus facilitate fast classical simulation.

Acknowledgments This work was partially supported by the DARPA QuIST program. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements of the Defense Advanced Research Projects Agency, or the U.S. Government.

References

1. C.Y. Lee, "Representation of Switching Circuits by Binary Decision Diagrams", *Bell System Technical Jour.*, 38:985-999, 1959.
2. R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers*, C35:677-691, Aug 1986.
3. E. Clarke et al., Multi-Terminal Binary Decision Diagrams and Hybrid Decision Diagrams, In T. Sasao and M. Fujita, eds, *Representations of Discrete Functions*, pp. 93-108, Kluwer, 1996.
4. R. I. Bahar et al., "Algebraic Decision Diagrams and their Applications", *In Proc. IEEE/ACM ICCAD 1993*, 188-191, 1993.
5. D. Greve, "QDD: A Quantum Computer Emulation Library", 1999 `http://home.plutonium.net/~dagreve/qdd.html`
6. D. Gottesman, "The Heisenberg Representation of Quantum Computers", `quant-ph/9807006`.
7. G. Viamontes, M. Rajagopalan, I. Markov, J. Hayes, "Gate-Level Simulation of Quantum Circuits", `quant-ph/0208003`
8. L. Grover, "Quantum Mechanics Helps In Searching For a Needle in a Haystack", *Phys. Rev. Lett.* 79:325-328, 1997.
9. M. Boyer, G. Brassard, P. Hoyer and A. Tapp, "Tight Bounds on Quantum Searching", *4th Workshop on Physics and Computation*, Nov. 1996.