# A Decision Diagram Package for Reversible and Quantum Circuit Simulation

D. Michael Miller, *Member, IEEE*, Mitchell A. Thornton, *Member, IEEE* and David Goodman

***Abstract*— This paper presents a new decision diagram structure and considers its application to the simulation of reversible and quantum circuits. The structure is designed to efficiently represent the matrices describing reversible and quantum gates and incorporates key properties of those matrices. Algorithms are given for efficiently building the decision diagram representation directly from a gate without constructing the actual matrix and for performing matrix multiplication which is the fundamental operation to determine the function realized by a cascade of gates. Experimental results show the methods presented are applicable to circuits at the state of the art in reversible and quantum synthesis and design.**

## I. Introduction

Recently, there has been considerable renewed interest in reversible circuits [28]. This is due to their potential for low power consumption and because quantum circuits are reversible. Much of the work in the literature considers circuit synthesis and optimization [2], [6], [7], [8], [10], [11], [12], [13], [14], [15], [16], [17], [18], [20], [21], [23], [24], [26], [25], [29], [31].

Simulation is a key step in circuit design and synthesis. In this paper, we present a decision diagram structure specifically oriented to reversible and quantum circuits and consider its application to circuit simulation and verification.

The functional behaviour of a primitive reversible (including quantum) gate can be represented as a transformation matrix and the functional behaviour of a circuit(cascade) composed of such gates is given by the product of those matrices. The matrices are complex-valued for quantum gates. For a binary circuit with $n$ lines, each matrix has dimension $2^n \times 2^n$ so in order to handle all but circuits with a few lines it is necessary to develop efficient methods for the storage and manipulation of the matrices.

Decision diagram techniques have been used for storing and manipulating matrices [5], [9]. Recently, Viamontes, Markov and Hayes [34], [35], [36] presented the QuIDDPro package for handling the matrices encountered in specifying and simulating quantum circuits. QuIDDPro uses CUDD, a robust and highly optimized decision diagram package developed by Somenzi [32]. QuIDDPro uses algebraic decision diagrams [3] as implemented in CUDD, with some extensions. A major advantage of this approach is it makes use of the extensive optimization in CUDD.

D. Michael Miller is with the Department of Computer Science, University of Victoria, PO Box 3055, Victoria, BC, Canada V8W 3P6, mmiller@cs.uvic.ca

Mitchell A. Thornton and David Goodman are with the Department of Computer Science and Engineering,Southern Methodist University, P.O. Box 750122 Dallas, TX 75275-0122 USA, mitch,dgoodman@engr.smu.edu

Here, we consider an alternative approach which, since it is applicable to multiple-valued as well as binary circuits, we call quantum multiple-valued decision diagrams (QMDD). In this paper, we emphasize the binary ($r = 2$) case. Application of the approach to the multiple-valued case is discussed in [27]. This paper presents many key implementation concepts which greatly improve upon the approaches described in [27].

The goal in developing the QMDD representation and package is to effectively represent and manipulate the complex-valued matrices encountered for quantum circuits. The matrices needed for non-quantum reversible circuits are 0-1 permutation matrices and hence a subset of the more general complex case. We use well known concepts from the myriad of decision diagram structures developed to date, edge attributes in particular. We also use some key implementation ideas from QuIDDPro. Multiple outgoing edges from each nonterminal vertex are employed in QMDD and is a key to the compactness and consequent efficiency of the representation. We also consider how different representations of complex numbers can improve efficiency and accuracy for certain types of problems.

We assume the reader is familiar with binary decision diagram terminology and techniques (see [30]). Section II provides the background on reversible and quantum gates and circuits necessary for this paper. Matrix representation of such gates and circuits is reviewed in Section III. The QMDD approach is described in detail in Section IV. Key implementation issues for QMDD are discussed in Section V. Section VI presents experimental results and the paper concludes with suggestions for further research in Section VII.

## II. Reversible and Quantum Gates and Circuits

*Definition 1:* An $n$-input, $n$-output (denoted $n \times n$) totally-specified function is reversible if it maps each input assignment to a unique output assignment.

*Definition 2:* An $m$-input, $m$-output (denoted $m \times m$) gate is reversible if it realizes a reversible function.

A reversible function defines a permutation of the input patterns and there are thus $2^n!$ $n \times n$ reversible functions. A reversible function is realized by a cascade of reversible gates.

### A. Toffoli Gates

A variety of binary reversible gates have been considered. The family of Toffoli gates [33] is defined as follows:

*Definition 3:* An $m \times m$ Toffoli gate passes the first $m-1$ lines (controls) through unchanged, and inverts the $m^{th}$ line (target) if the control lines all equal 1.

The $2 \times 2$ Toffoli gate and the $3 \times 3$ Toffoli gate have been named the *controlled-NOT* and *controlled-controlled-NOT* gates, respectively. A Toffoli gate with no controls is a *NOT* gate.

### B. Quantum Gates

Many quantum gates have been defined and studied in the literature [28]. We here restrict our attention to quantum circuits composed of:

- NOT gates;
- Controlled-NOT gates (CNOT);
- The 2-line controlled-$V$ gate that changes the target line using the transformation given by the matrix $\mathbf{V} = \frac{1+i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$ if the control line has the value 1;
- The 2-line controlled-$V^+$ that changes the target line using the transformation $\mathbf{V}^+ = \mathbf{V}^{-1} = \frac{1-i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$ if the control line has the value 1.

Gates $V$ and $V^+$ are often referred to as *controlled-square-root-of*-NOT gates since $V^2 = (V^+)^2 = NOT$.

It is important to note that while we only use a small set of gates for illustration in this paper, QMDD are directly applicable to other common quantum gates such as the Hadamard gate, Pauli gates and rotation gates [28] amongst others. The only requirement is that the behaviour of the gate be describable as a matrix in the manner outlined in the next Section.

## III. Matrix Representation of Gates and Circuits

The common feature of all the gates mentioned above (and other reversible and quantum gates) is that the transformation performed by a gate, and the transformation performed by a cascade of gates, can be represented as a matrix of dimension $2^n \times 2^n$ for an $n$-line circuit. For example for $n = 3$, the Toffoli gate $T(x_0, x_2; x_1)$ ($x_0, x_2$ controls, $x_1$ target) has the transformation matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{1}$$

Note that we treat $x_0$ as the least significant variable in all examples.

The controlled-$V$ gate $V(x2; x1)$ with $n = 3$ has the transformation matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} \\ 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} \end{bmatrix} \tag{2}$$

This matrix has complex-valued entries and is not a permutation matrix.

## IV. Quantum Multiple-valued Decision Diagrams

The representation of a matrix by a decision diagram is discussed in [5], [9]. In general, a matrix $M$ with $p$ rows and $q$ columns can be represented by a decision diagram with binary decision variables with $\lceil log_2 p \rceil$ *row selection* variables and $\lceil log_2 q \rceil$ *column selection* variables. Redundant and duplicate vertices are eliminated in the usual manner. The diagram in general has multiple terminal vertices corresponding to the matrix element values. If the number of rows or columns is not a power of two, the matrix is considered to be appropriately padded with zeroes. The need for such padding does not arise in this work.

### A. The QMDD Structure

The motivation for the QMDD structure comes from the regular structure of the matrices encountered for reversible and quantum gates and circuits. In general, a $2^n \times 2^n$ matrix $\mathbf{M}$ can be partitioned as

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{pmatrix} \tag{3}$$

where each of the $\mathbf{M}_i$ has dimension $2^{n-1} \times 2^{n-1}$. For example the matrix in (2) is partitioned as

$$\left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} \\ 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} \end{array}\right] \tag{4}$$

The submatrices can be further partitioned into four smaller blocks. For many blocks, such as the one in the bottom right of (4), repeated blocks are identified. Only one occurrence of a repeated block needs to be represented and this leads to considerable compaction in the representation of the matrices encountered for reversible and quantum gates and circuits.

The idea of a QMDD is to recursively apply the quadrant partitioning outlined above identifying each partitioning with a vertex in a decision diagram. Each vertex has four outgoing edges one for each quadrant numbered as in (3). Duplicate blocks are handled by having the appropriate edges point to

the same sub-DD. As the above example shows, zero and identity matrices frequently arise. These are very effectively handled by the QMDD structure as will be described below.

For generality, we shall define QMDD for $r$ values. The binary case is $r = 2$.

*Definition 4:* A **QMDD** is a directed acyclic graph with the following properties:

1) There is a single **terminal vertex** with associated value 1. The terminal vertex has no outgoing edges.
2) There are some number of **non-terminal vertices** each labeled by an $r^2$-valued selection variable. Each nonterminal vertex has $r^2$ outgoing edges designated $e_0, e_1, ..., e_{r^2-1}$.
3) One vertex is the **start vertex** and has a single incoming edge that itself has no source vertex.
4) Every edge in the QMDD (including the one leading to the start vertex) has an associated complex-valued **weight**. An edge with weight 0 always points to the terminal vertex.
5) The selection variables are **ordered**, assume with no loss of generality $x_0 \prec x_1 \prec ... \prec x_{n-1}$, and the QMDD satisfies the following two rules:
   - Each selection variable appears at most once on each path from the start vertex to the terminal vertex.
   - An edge from a nonterminal vertex labeled $x_i$ points to a nonterminal vertex labeled $x_j, j < i$ or to the terminal vertex. Hence $x_0$ is closest to the terminal vertex and $x_k$ for the highest $k$ present labels the start vertex.
6) No nonterminal vertex is **redundant**, *i.e.* no nonterminal vertex has its $r^2$ outgoing edges all with the same weights and pointing to a common vertex.
7) Each nonterminal vertex is **normalized** such that $e_j, 0 \leq j \leq r^2 - 1$ has weight 1 and $\forall e_i, i < j$ have weight 0. Note, such an $e_j$ must exist or the vertex would be redundant (all weights 0).
8) Nonterminal vertices are **unique**, *i.e.* no two nonterminal vertices labeled by the same $x_i$ can have the same set of outgoing edges (destinations and weights).

Note that the start vertex is a nonterminal vertex except in the case where the QMDD has no nonterminal vertices in which case the QMDD consists of just the terminal vertex which is also the start vertex. This extreme case corresponds to a constant valued matrix.

Figure 1 illustrates the QMDD representation for a quantum $V$ gate. The 0,1,2,3 edges from each vertex are from left to right. The weights are shown next to each line.

*Definition 5:* To **evaluate a QMDD** for a particular selection variable assignment, find the path from the start vertex to the terminal vertex (including the edge leading to the start vertex) determined by the assignment. In particular, from a vertex labeled $x_i$ follow edge $e_j$ where $j$ is the value assigned to $x_i$. Note that some variables may not appear on the designated path. The value associated with the selection variable assignment is the product of the weights on the edges in the corresponding path.

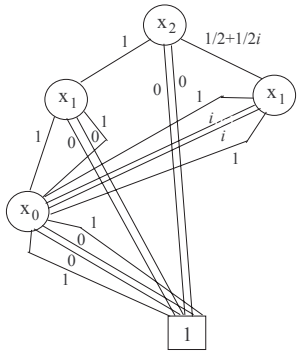


Fig. 1. QMDD for $V(x_2; x_1)$ with $n = 3$ (see eqn. 2).

*Theorem 1:* An $r^n \times r^n$ complex valued matrix $M$ has a unique (up to variable reordering or relabeling) QMDD representation.

**Proof:** The proof is given in [27].

### *B. Matrix Operations*

We here outline methods for performing basic matrix operations directly on QMDD. Given an edge $e$, we use: $w(e)$ to denote the weight on $e$; $v(e)$ to denote the vertex $e$ points to; $x(e)$ to denote the variable that labels the vertex $e$ points to (not defined for the terminal vertex); $E_i(e)$ to denote the $i^{th}$ edge out of the vertex that $e$ points to; and $T(e)$ to denote a Boolean test that is true if $e$ points to the terminal vertex. The variables adhere to the same order in all QMDD and we shall use $\prec$ to denote the fact one variable precedes another and hence appears closer to the terminal vertex in the QMDD. Since we restrict our attention to the binary case, every nonterminal vertex has 4 outgoing edges.

**Matrix Addition** Matrix addition of two $2^n \times 2^n$ matrices is easily implemented using QMDD since adding two matrices of the same dimension can be seen to be a matter of adding the four quadrants separately. This can be applied recursively. The only complication that arises is when a QMDD is missing a decision variable due to repeated blocks. This is accounted for in the procedure outlined below.

Let $e_0$ and $e_1$ be two edges pointing to two QMDD (matrices) to be added. The procedure is recursive and involves the following steps:

1) If $T(e_1)$, swap $e_0$ and $e_1$.
2) If $T(e_0)$,
   a) if $w(e_0) = 0$, the result is $e_1$;
   b) if $T(e_1)$, the result is an edge pointing to the terminal vertex with weight $w(e_0) + w(e_1)$.
3) If $T(e_0)$ or $x(e_0) \prec x(e_1)$, swap $e_0$ and $e_1$.
4) For $i = 0, 1, 2, 3$
   a) Create an edge $p$ pointing to $v(E_i(e_0))$ with weight $w(e_0) \times w(E_i(e_0))$.
   b) If $x(e_0) = x(e_1)$, create an edge $q$ pointing to $v(E_i(e_1))$ with weight $w(e_1) \times w(E_i(e_1))$, else set $q = y$.

c) Recursively invoke this procedure to add $p$ and $q$ giving $z_i$.

5) The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, 2, 3$. This vertex and the edge pointing to it are normalized as described above.

**Matrix Multiplication** Matrix multiplication of two $2^n \times 2^n$ matrices can be decomposed as follows:

$$\begin{pmatrix} \mathrm{M}_0 & \mathrm{M}_1 \\ \mathrm{M}_2 & \mathrm{M}_3 \end{pmatrix} \begin{pmatrix} \mathrm{N}_0 & \mathrm{N}_1 \\ \mathrm{N}_2 & \mathrm{N}_3 \end{pmatrix} = \begin{pmatrix} \mathrm{M}_0\mathrm{N}_0 + \mathrm{M}_1\mathrm{N}_2 & \mathrm{M}_0\mathrm{N}_1 + \mathrm{M}_1\mathrm{N}_3 \\ \mathrm{M}_2\mathrm{N}_0 + \mathrm{M}_3\mathrm{N}_2 & \mathrm{M}_2\mathrm{N}_1 + \mathrm{M}_3\mathrm{N}_3 \end{pmatrix} \quad (5)$$

which requires multiplication and addition of smaller matrices. Addition is implemented as described above. The multiplication is implemented by recursively applying (5) down to the multiplication of individual elements. Once again the major complication is missing decision variables in one of the QMDD which is accounted for in the algorithm outlined below.

Let $e_0$ and $e_1$ be two edges pointing to two QMDD (matrices) to be multiplied. The procedure is recursive and involves the following steps:

1) If $T(e_1)$, swap $e_0$ and $e_1$.
2) If $T(e_0)$ then
   a) if $w(e_0) = 0$, the result is $e_0$;
   b) if $w(e_0) = 1$, the result is $e_1$;
   c) otherwise, the result is an edge pointing to $v(e_1)$ with weight $w(e_0) \times w(e_1)$.
3) If $x(e_0) \prec x(e_1)$, swap $e_0$ and $e_1$.
4) For $i = 0, 2$
   For $j = 0, 1$
   Set $z_{i+j}$ to be an edge with weight 0 pointing to the terminal vertex.
   For $k = 0, 1$
   (i) Create an edge $p$ pointing to $v(E_{i+k}(e_0))$ with weight $w(e_0) \times w(E_{i+k}(e_0))$.
   (ii) If $x(e_0) = x(e_1)$, create an edge $q$ pointing to $v(E_{j+2\times k}(e_1))$ with weight $w(e_1) \times w(E_{j+2\times k}(e_1))$, else set $q = y$.
   (iii) Recursively invoke this procedure to multiply the QMDD pointed to by $p$ and $q$ and then use the procedure above to add the result to the QMDD pointed to by $z_{i+j}$. The result of the addition replaces $z_{i+j}$.
5) The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, 2, 3$. This vertex and the edge pointing to it are normalized as described above.

**Kronecker Product** Implementing the Kronecker product of QMDD uses partitioning and recursion techniques analogous to those for addition and multiplication. Let $e_0$ and $e_1$ be two edges pointing to two QMDD (matrices) for which we want to compute the Kronecker product $A \bigotimes B$ (note that this operation is not commutative). For the application considered here, the selection variables for $B$ precede the selection variables for $A$. This greatly reduces the complexity of the algorithm for computing the Kronecker product of two QMDD. The procedure is recursive and involves the following steps:

1) If $T(e_0)$ then
   a) if $w(e_0) = 0$, the result is $e_0$;
   b) if $w(e_0) = 1$, the result is $e_1$;
   c) otherwise, the result is an edge pointing to $v(e_1)$ with weight $w(e_0) \times w(e_1)$.
2) For $i = 0, 1, 2, 3$
   Recursively invoke this procedure to find the Kronecker product of $E_i(e_0)$ and $e_1$ setting $z_i$ to the result.
3) The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, 2, 3$. This vertex and the edge pointing to it are normalized as described above.

### *C. Gate and Circuit QMDD Construction*

Given the basic operations above, an important procedure we have developed builds the QMDD representation of the matrix defining a gate transformation. Assume, as above, the variable order $x_0, x_1, ..., x_{n-1}$ from the terminal vertex towards the start vertex. A gate $G$ is specified by the $2 \times 2$ base transition matrix $M$, the target line $x_t$ and a possibly empty set of control lines $C$. We here provide an outline of the procedure which is sufficient to appreciate the key facets of the method. A detailed understanding of building QMDD for individual gates and the matrix operations outlined above is best gained from the C code implementation available from the authors.

The procedure builds the QMDD from the terminal vertex towards the start vertex. It has three phases:

1) **Variables below the target**: For these variables, 4 separate QMDD are constructed. The $(i \times 2 + j)^{th}$ QMDD has a path following the active values (the values required to activate the controls) for the control variables to the terminal value $M_{i,j}$ with all other paths leading to 0. Non-control variables (unconnected lines) are taken into account through certain identity matrices combined with the active part of the QMDD using Kronecker products.
2) **The target variable**: A single QMDD is formed with the source vertex controlled by the target variable and the edges from that vertex leading to the QMDDs constructed in (1).
3) **Variables above the target**: The upper part of the QMDD has a path following the active values for the control variables with all other paths leading to 0. Again, non-control variables are taken into account through appropriate identity matrices and Kronecker products.

The most important aspect of this procedure is that it builds the QMDD corresponding to a gate from terminal to source vertex in a single pass by iterating through $x_0$ to $x_{n-1}$ with no backtracking or recursion. It is thus quite efficient.

The above outlines how the QMDD giving the matrix transformation defining a single gate is constructed. For a

cascade of gates (circuit) $G_0G_1...G_{t-1}$ where transformation for gate $G_i$ is defined by matrix $M_i$, the transformation for the complete circuit is $M_{t-1} \times M_{t-2} \times ... \times M_0$. This is computed using QMDD based matrix multiplication.

## V. QMDD Implementation

We have implemented a QMDD package in C. Because of its unique approach, this is a stand-alone package which is not built on top of an existing package such as CUDD. Here we describe the key features of the implementation. The package is available at www.cs.uvic.ca/~mmiller/QMDD.

### A. Vertices and Edges

Memory space for vertices is dynamically allocated with freed vertex space put onto an available space chain for reuse. A reference count based garbage collection scheme is used. The reference count is the number of edges pointing to a vertex. If the total vertex space allocated exceeds a user-defined limit, garbage collection takes place with the space for vertices with reference counts of 0 added to the available space chain. There is a single terminal vertex with associated value 1. Nonterminal vertices are normalized as described above and the standard DD unique table approach is used to ensure each nonterminal vertex is created and stored only once (see [27] for details).

Two differences here are that we use a separate unique table for each QMDD variable and we have modified the hashing function used for the unique table. Both lead to shorter search chains from the unique table buckets. As before, the hashing function combines the pointer and weight values on the edges emanating from a vertex. However, it now shifts the values of the pointers so that the order of the edges affects the hash key. Before, without this shifting, the order was irrelevant which led to an uneven distribution of vertices across the unique table.

As described above, every edge has an associated complex-valued weight. Rather than storing this weight with each edge, we associate an integer which is an index to a table of complex values. This has two advantages: each complex number is stored only once; and the storage required per edge (an integer) is less than is required for a complex number. We use indices 0 and 1 for the two edge weights 0 and 1. This offers a speedup since those two key values can be checked by the index without having to access the table of complex values.

### B. Operation Tables

Since operations on complex numbers are relatively expensive, and the same computations arise frequently, we employ addition, multiplication and division lookup tables. Subtraction is not required in this work. Initially each table is empty.

Each time a complex number operation must be evaluated, the appropriate table entry is checked. If it is not empty, the computation has been previously computed and the index to the result in the complex value table is found in the appropriate operation table. If the operation table entry is empty, the required computation is performed and the index to the result is placed in the table for future reference. We take into account commutativity for addition and multiplication so that only the entries above and on the diagonal are employed in those computation tables.

At present, we use simple arrays for these tables as the number of complex values encountered for our largest examples to date is on the order of 55. Should arrays prove impractical in the future, more sophisticated data structures could be used. We have found this very simple idea yields considerable execution speedup at relatively low memory cost.

### C. Computed Table

As is common for DD packages, a computed table [4] is used to reduce duplicate computations. When a matrix addition, matrix multiplication or Kronecker product is to be performed on two QMDD, the computed table is checked to see if the result of the computation is available there before performing the full computation. Since the computations are themselves recursive, this check is performed at each recursive step so that results known for subcomputations can also be employed.

The detailed approach is as described in [27] except the hash function used to look up entries in the computed table is based on the edges pointing to the two QMDD (addresses and weight) and the type of operation. The implementation described in [27] did not use the edge weights in the hash function. Experiments on large problems have shown the revised hash function to be more effective. The medium to large examples considered below can not be completed in reasonable time without the use of a computed table.

### D. Zero and Identity Matrices

The nature of the matrices encountered in the problem at hand is that they have many occurrences of zero and identity submatrices. The QMDD structure described above handles the zero submatrix case directly since any such submatrix is simply a single edge pointing to the terminal vertex with an edge weight of 0. The situation for zero submatrices is then easily taken into account as shown in the outlines for the matrix operations given above.

To reduce computation for matrix multiplication and the Kronecker product, a flag is associated with each vertex to identify whether the QMDD it and its descendants comprise represents an identity matrix. For multiplication, the result is simply the other matrix if one of the argument matrices is the identity matrix or that matrix multiplied by $w$ if the other is the identity multiplied by a scalar $w$. Such a scaler only affects the edge pointing to the start vertex of the QMDD.

The Kronecker product $A \bigotimes B$ can be formed without recursion if $A$ is the identity matrix or the identity multiplied by a scalar $w$. The result is a block matrix with $wB$ along the diagonal and 0 matrices in all off-diagonal entries.

The terminal vertex, which has value 1, represents the single element identity matrix. Each nonterminal vertex represents a $2 \times 2$ block matrix and is an identity matrix

only if the blocks on the diagonal are identity matrices and the two off diagonal entries are 0 matrices. These conditions are readily checked when a vertex is formed and entered into the unique table. Note that the conditions need only be checked for the immediate descendants and not recursively so the overhead is negligible. The vertex labeled $x_0$ and the left vertex labeled $x_1$ in Fig. 1 are both start vertices for sub-QMDD representing identity matrices.

### *E. Representing Complex Numbers*

The most straightforward method for representing complex numbers is to store the real and the imaginary parts as floating-point values. This has two disadvantages: computations using floating-point are time-consuming; and significant round-off problems can arise particularly given the amount of computation required for larger circuits.

For Toffoli gate circuits, no complex values are needed and the matrices contain only values 0 and 1. For circuits composed of NOT, CNOT, controlled-$V$ and controlled-$V^+$ gates, it is readily shown that the required complex numbers all have real and imaginary parts which are rational numbers. A rational can be accurately stored as two integers and computation over the rationals can be performed with no round-off error. Other types of complex gate can be included with out moving beyond ratioanl real and imaginary parts for the complex numbers.

However, when rotation gates are used, rationals are not sufficient. The transformation matrices for rotation about $\theta$ involve $\sin\theta/2$ and $\cos\theta/2$. Rotation by 0 of $\pi$ is not a problem as the $\sin$ and $\cos$ values involved are 0 or 1. However, rotation by $\pi/2$, introduces $\sin\pi/2$ and $\cos\pi/2$ both of which equal $\sqrt{2}/2$ which is irrational. In this case, it transpires that the real and imaginary parts of the complex numbers involved and all be represented as quadratic irrational numbers of the form $\frac{a+b\sqrt{2}}{c}$ where $a$, $b$ and $c$ are integers. As for the rationals, these numbers can be stored and manipulated in the integer domain with no round-off. The computations are fairly involved but are faster, and of course more accurate, than using floating-point.

The above does not handle all cases. For example, rotation of $\pi/4$ involves the $sin$ and $cos$ of $\pi/8$ which has no convenient representation and in this case floats must be used.

Our current package uses the quadratic irrational notation noted above or floating-point but does not mix them. The choice is specified by the user. A better approach would employ complex number routines that adapt using the most accurate representation possible for each number. All examples presented below were run using the quadratic irrational representation.

## VI. Experimental Results

The experimental results presented here were run on a Toshiba PORTEGE with a 750 MHz Pentium III CPU and 256 MB memory running Windows XP. The Metrowerks Codewarrior C compiler Version 4 was used with level 4 gloabal optimization targeting faster execution speed.

All experiments were run with 8,192 buckets in the unique table associated with each QMDD variable and a compute table with 4,096 slots. The garbage collection limit was set at 50,000 vertices.

There are $8! = 40,320$ 3-variable binary reversible functions each of which is uniquely defined by a permutation of $0, 1, ..., 7$. Maslov and Miller [22] presented a search method that finds minimal gate count NCV circuits for those function using breadth first search. To verify the correctness of those circuits and as test of our QMDD procedures, we wrote a program that reads the function permutation and builds a QMDD representing the corresponding permutation. The program then builds a QMDD for the circuit for that function and verifies that the two QMDD are the same. Note that due to the uniqueness of the representation, verifying two QMDD are the same only requires confirmation that the edges pointing to the start vertices for the two QMDD are the same (both weight and pointer). Verification for all 40,320 functions took 8.67 CPU seconds.

Building a QMDD for a given permutation is a relatively straightforward recursive procedure. Basically, the permutation is partitioned into four pieces corresponding to the four quadrants of the permutation matrix. This procedure is applied recursively to each of the quadrants and the recursion continues till a block is a single element. This approach is limited to small $n$ since the permutation must be specified as $2^n$ integers and the work in the recursion is exponential in $n$.

For garbage collection, the total number of vertices was checked after each function was verified and garbage vertices were collected if the number of vertices in the unique tables exceeded the limit of 50,000. The garbage collection process was performed twice.

Table I gives results for a number of circuits taken from D. Maslov's benchmark web site [19] or provided directly by him (the NCV circuits in particular). For each we specify whether it has NOT, controlled-NOT and Toffoli gates (including multiple-control Toffoli gates) which is denoted *NCT*, or NOT, controlled-NOT, $V$ and $V^+$ gates which is denoted *NCV*. We give the numbers of *gates* and the *size* which is the number of vertices in the QMDD. *Total* is the number of vertices allocated when the program completed. A * following this value indicates the garbage collector was activated one or more times. Many of the large examples can not be completed in reasonable time, or not at all on the computer used, without garbage collection. *CPU* is the number of user CPU seconds as reported by the UNIX time command.

The examples in the top half of Table I are all *NCT* circuits. The results show that the sizes of the resulting QMDD are quite reasonable and that our yet to be fully optimized implementation can handle quite large circuits in very reasonable time.

There are three examples in the bottom half of Table I. For each, we have considered both an *NCT* and an *NCV* circuit. A QMDD is built for each of the circuits. In each case,

TABLE I

QMDD EXPERIMENTAL RESULTS

| name | type | n | gates | size | total | CPU(sec.) |
|---|---|---|---|---|---|---|
| ham3 | NCT | 3 | 5 | 10 | 39 | 0.000 |
| 3..17 | NCT | 3 | 6 | 10 | 42 | 0.000 |
| 5mod5 | NCT | 6 | 17 | 28 | 373 | 0.000 |
| hwb7 | NCT | 7 | 289 | 179 | 8,616 | 0.141 |
| hwb8 | NCT | 8 | 614 | 343 | 29,990 | 0.430 |
| hwb9 | NCT | 9 | 1,541 | 683 | 37,305* | 1.632 |
| hwb10 | NCT | 10 | 3,595 | 1,331 | 8,890* | 6.950 |
| hwb11 | NCT | 11 | 9,314 | 2,639 | 9,926* | 35.331 |
| hwb12 | NCT | 12 | 18,393 | 5,167 | 20,866* | 129.667 |
| ham15 | NCT | 15 | 132 | 4522 | 4521* | 0.631 |
| cycle17..3 | NCT | 20 | 48 | 236 | 2,959 | 0.00 |
| hwb4 | NCT | 4 | 11 | 22 | 122 | 0.000 |
| hwb4 | NCV | 4 | 11 | 22 | 247 | 0.000 |
| 0410184 | NCT | 14 | 46 | 39 | 1,091 | 0.010 |
| 0410184 | NCV | 14 | 74 | 39 | 1,917 | 0.030 |
| c2 | NCT | 35 | 116 | 150 | 12,359 | 0.200 |
| c2 | NCV | 35 | 305 | 150 | 40,617 | 0.631 |

we determined the two circuits yielded the same QMDD by checking the edges pointing to the start vertices.

Circuit c2 implements a reversible specification for the sum of two 10-bit numbers. Our initial implementation using *doubles* to represent the real and imaginary parts of a complex number took just over a minute and a half of CPU time for the NCV circuit. As the table above shows, our current implementation using quadratic irrationals which reduces the computation to integers and guarantees accuracy together with other improvements we have made reduces the time for the NCV circuit to less than 1 second. In addition to the speedup resulting from integer computation, the use of the quadratic irrationals avoids unnecessarily large intermediate QMDD arising from superfluous vertices resulting from round-off errors.

## VII. CONCLUSION

This paper has presented QMDD, a decision diagram structure specifically intended for reversible and quantum circuit simulation. We emphasize again that while the results presented considered only a small number of gate types and only NOT, controlled-NOT, $V$ and $V^+$ quantum gates in particular, the approach is applicable to any gate whose operation can be expressed as a transformation matrix. The experimental results given show that the approach has considerable promise. However, there are numerous issues that need to be further investigated.

We are investigating further alternatives for complex number representation and computation so that the package will not slow down significantly when applied to general rotation or other gates that don't fit the quadratic irrational representation. We are also investigating the effect of variable order on the size and construction cost for QMDD. Our initial work in this area indicates the effect can be quite pronounced for large diagrams.

We have performed some preliminary studies comparing our QMDD package and QuIDDPro [36] in terms of building DD for binary reversible circuits. Results for a small number of benchmark circuits indicate the QMDD package is from 2 to 10 times faster and requires less memory. A more detailed study is required to be conclusive and experiments on larger examples are required. QuIDDPro offers more functionality than our package and a far more sophisticated user interface so there is a higher overhead cost in terms of both time and memory. Also, QuIDDPro is based on CUDD and uses quite different memory management techniques than our package and in fact requires significant memory even for fairly small examples.

We believe the efficiency of our QMDD package can be significantly improved. We are studying the implementation techniques in the CUDD package to see which will yield improvements in our QMDD program.

Finally, and perhaps most interesting, while QMDD were designed initially for simulation and verification, we are now investigating using QMDD as a representation for reversible and quantum circuit synthesis. Abdollahi and Pedram [1] have presented a significantly different decision diagram structure for quantum circuits which they call QDD. Their structure is specifically directed to circuits composed of $R_x(\theta)$ rotation gates. Like QuIDDPro, QDD use two edges from each vertex and we do not believe the representation captures the underlying matrix properties as well as the QMDD representation does. We are studying Abdollahi and Pedram's synthesis approach and others in the literature to see if they can be adapted to QMDD.

## ACKNOWLEDGMENTS

This research was supported in part by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada.

## REFERENCES

[1] A. Abdollahi and M. Pedram. Analysis and synthesis of quantum circuits by using quantum decision diagrams. In *Proceedings of the IEEE Design Automation and Test in Europe*, to appear 2006.

[2] A. Agrawal and N. K. Jha. Reversible logic synthesis. In *Proceedings of the IEEE Design Automation and Test in Europe*, pages 1384–1385, Paris, France, February 2004.

[3] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric decision diagrams and their applications. *Form. Methods Syst. Des.*, 10(2-3):171–206, 1997.

[4] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *27th Design Automation Conference*, June 1990.

[5] E. Clarke, K. Mcmillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. *Formal Methods in System Design*, 10(2-3):137–148, 1997.

[6] E. Curtis and M. Perkowski. A transformation based algorithm for ternary reversible logic synthesis using universally controlled ternary gates. In *International Workshop on Logic Synthesis*, June 2004.

[7] G. W. Dueck and D. Maslov. Reversible function synthesis with minimum garbage outputs. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 154–161, March 2003.

[8] G. W. Dueck, D. Maslov, and D. M. Miller. Transformation-based synthesis of networks of Toffoli/Fredkin gates. In *IEEE Canadian Conference on Electrical and Computer Engineering*, Montreal, Canada, May 2003.

[9] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient datastructure for matrix representation. *Form. Methods Syst. Des.*, 10(2-3):149–169, 1997.

[10] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Quantum logic synthesis by symbolic reachability analysis. In *Design Automation Conference*, pages 838–841, June 2004.

[11] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing CNOT-based quantum circuits. In *Design Automation Conference*, pages 419–424, New Orleans, Louisiana, USA, June 2002.

[12] P. Kerntopf. Synthesis of multipurpose reversible logic gates. In *EUROMICRO Symposium on Digital Systems Design*, pages 259–266, 2002.

[13] P. Kerntopf. A new heuristic algorithm for reversible logic synthesis. In *Design Automation Conference*, pages 834–837, June 2004.

[14] M. H. A. Khan and M. Perkowski. Logic synthesis with cascades of new reversible gate families. In *6th International Symposium on Representations and Methodology of Future Computing Technology (Reed-Muller)*, pages 43–55, March 2003.

[15] M. H. A. Khan and M. Perkowski. Multi-output ESOP synthesis with cascades of new reversible gate family. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 144–153, March 2003.

[16] A. Khlopotine, M. Perkowski, and P. Kerntopf. Reversible logic synthesis by iterative compositions. *International Workshop on Logic Synthesis*, pages 261–266, 2002.

[17] J.-S. Lee, Y. Chung, J. Kim, and S. Lee. A prectical method of constructing quantum combinatorial logic circuits. Technical report, Nov. 1999.

[18] M. Lukac, M. Pivtoraiko, A. Mishchenko, and M. Perkowski. Automated synthesis of generalized reversible cascades using genetic algorithms. In *5th International Workshop on Boolean Problems*, pages 33–45, Freiburg, Germany, September 2002.

[19] D. Maslov. Reversible logic synthesis benchmarks page. http://www.cs.uvic.ca/~dmaslov/, 2005.

[20] D. Maslov and G. W. Dueck. Asymptotically optimal regular synthesis of reversible networks. In *International Workshop on Logic Synthesis*, pages 226–231, Laguna Beach, CA, 2003.

[21] D. Maslov, G. W. Dueck, and D. M. Miller. Simplification of Toffoli networks via templates. In *Symposium on Integrated Circuits and System Design*, pages 53–58, September 2003.

[22] D. Maslov and D. M. Miller. Comparison of the cost metrics for reversible and quantum logic synthesis. *Quantum Information & Computation*, submitted Nov. 2005. available as quant-ph/0511008.

[23] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck. Quantum circuit simplification using templates. In *Design and Test Europe (DATE)*, pages 1208–1213, Munich, Germany, March 2005.

[24] D. M. Miller. Spectral and two-place decomposition techniques in reversible logic. In *Midwest Symposium on Circuits and Systems*, Aug. 2002.

[25] D. M. Miller, G. Dueck, and D. Maslov. A synthesis method for MVL reversible logic. In *International Symposium on Multiple-Valued Logic*, pages 74–80, Toronto, Canada, May 2004.

[26] D. M. Miller and G. W. Dueck. Spectral techniques for reversible logic synthesis. In *6th Int. Symp. on Representations and Methodology of Future Computing Technologies*, pages 56–62, March 2003.

[27] D. M. Miller and M. A. Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. In *2006 Int. Symp. on Multiple-Valued Logic*, accepted.

[28] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[29] M. Perkowski, L. Jozwiak, P. Kerntopf, A. Mishchenko, A. Al-Rabadi, A. Coppola, A. Buller, X. Song, M. M. H. A. Khan, S. Yanushkevich, V. Shmerko, and M. Chrzanowska-Jeske. A general decomposition for reversible logic. In *5th International Reed-Muller Workshop*, pages 119–138, 2001.

[30] V. P. S. S. N. Yanushkevich, D. M. Miller and R. S. Stankovic. *Decision Diagram Techniques for Micro- and Nanoelectronic Design Handbook*. CRC:Taylor & Francis, 2006.

[31] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. In *International Conference on Computer Aided Design*, pages 125–132, San Jose, California, USA, Nov 10-14 2002.

[32] F. Somenzi. CUDD: Cu decision diagram package - release 2.4.1. http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html, 2005.

[33] T. Toffoli. Reversible computing. *Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci*, 1980.

[34] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Graph-based simulation of quantum computation in the state-vector and density-matrix representation. In *Proceedings of SPIE*, volume 5436, April 2004.

[35] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Graph-based simulation of quantum computation in the density matrix representation. *Quantum Information & Computation*, 5(2):113–130, 2005.

[36] G. F. Viamontes, I. L. Markov, and J. P. Hayes. QuIDDPro: High-performance quantum circuit simulation. http://vlsicad.eecs.umich.edu/Quantum/qp/, 2005.