Dynamic Programming Algorithms as Reversible Circuits: Symmetric Function Realization

Dmitri Maslov
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3 Canada
cetrau@mail.ru

ABSTRACT

The work starts with a general idea of how to realize a dynamic programming algorithm as a reversible circuit. This realization is not tied to a specific reversible design model and technology or a class of dynamic algorithms, it shows an approach for such synthesis. As an illustration of this approach, a class of all symmetric functions is realized in a dynamic programming algorithm manner as a reversible circuit of Toffoli elements. The garbage, quantum and reversible costs of the presented implementation are calculated and compared to the costs of previously described reversible synthesis methods. The summary of results of this comparison is as follows. The quantum cost of the proposed method is better than the quantum cost of any other known systematic approach. The garbage is usually lower (except for comparison with the synthesis methods, primary goal of which is synthesis with theoretically minimal garbage). For the large functions reversible cost is better or has the same asymptotic that of other methods. Although the reversible cost comparison may not look beneficial for the small functions, the possible technological implementation (quantum) shows that it is beneficial to use the presented approach even for the small functions.

1. INTRODUCTION

General recursion and dynamic programming algorithms [4] form a very important class of algorithms. Many of the real-world problems can be solved by dynamic programming algorithms: evaluating symmetric function, algorithms on strings, most of the approximation methods. The complete list is very large.

Reversible logic implementations are known to dissipate zero heat due to the information loss [10, 3], therefore helping to solve at least the following two problems: overheating, power saving (longer life for batteries). The reversible logic solution may be especially important in low-voltage designs of mobile systems, where both power saving and overheating are very important due to the light weight and independent battery. The reversible logic design differs significantly from the conventional logic design. In addition to the reversibility of gates, no fan-outs and no feed-backs [15] restrictions are applied. This leaves us with the cascade as the only possible structure.

2. GENERAL DYNAMIC PROGRAMMING SOLUTION

By the definition, a **reversible gate (function)** is such a gate that the input-to-output mapping performed by the gate is bijective. That is, the inverse always exists. Reversible synthesis is a synthesis procedure which uses the reversible gates only (with no fan-outs and no feed-backs restrictions that are commonly used). A good characteristic of how good the reversible synthesis method is, is the cost of the circuit synthesized. Reversible synthesis costs differ from the costs of the non-reversible implementations. Reversible synthesis cost is not simply a number of gates used, it is essentially different. The cost of a reversible implementation consists of the two values: the number of gates used, and the number of non-useful outputs, called **garbage**. The need in garbage as a special parameter can be illustrated on the following example. If one wants to compute a Boolean product, xy, a gate (circuit) with two inputs and one output cannot be used — the output only is not sufficient to build the inverse of the function and uncover information on what the inputs were. It also happens that 2-input 2-output circuits cannot compute this function. Only a 3-input 3-output circuit can [19]. But, such a circuit produces two outputs that are not needed, thus, having the garbage value of 2. The garbage itself is an important consideration, since for some technologies a high price is paid for the large number of inputs and outputs. For instance, quantum computers of maximal size of 7 only were reported [1].

The main idea of a dynamic programming algorithm is decomposing the problem f into a set of other problems, where the answer for f can be found in terms of a "simple" operation from the answers of subproblems. When the dynamic algorithm is completely specified, a set of operations needed to make a step of calculation, and a set of subfunctions needed for the computation is defined. Further, to build a reversible circuit we will realize each of the algorithm operations as a reversible cascade. But first, we need to find the garbage cost of the implementation.

Given a problem (function) f with Boolean inputs $(x_1, x_2, ..., x_n)$ and a dynamic algorithm, that terminates after S steps of

calculation first calculate the two numbers:

- $M = \max_{s=1..S}(m_s)$, where m_s is the number of Boolean values (intermediate storage) needed in addition to the input values in order to complete the calculation starting at step s. That is, M represents maximum number of subfunction values which are needed in order to complete the algorithm calculation starting from step s. For simplicity assume that on each step of the algorithm only one subfunction value is updated.
- Given a reversible design model and the set of all operations the dynamic algorithm performs at one step, find the number $G = \max_{s=1..S}(g_s)$, where g_s is the minimal amount of garbage needed for the reversible implementation of the s-th step of the algorithm in terms of the considered model.

When the two numbers are determined, the circuit is constructed as follows:

- Create the set of input constants size $M+G$ and assign values zero to all of them. First M will be used to store intermediate results of the calculation and the answer itself, and G are needed for the calculation.
- Use the reversible design model to create a circuit for each of the S steps of the specified dynamic programming algorithm. The circuit changes one subfunction value at a time.

When the circuit is built, the outputs which contain the answer are specified by the dynamic algorithm.

The amount of the resulting reversible design garbage, as well as the final cost of the network will depend on the design of the dynamic programming algorithm and strength of the reversible design model.

Note, that a benefit of using this method comes from the following consideration. If a multiple output function to be realized has outputs for which the dynamic programming algorithm is known and those for which the dynamic programming algorithm is not known, the dynamic programming realizable outputs can be built first by the suggested method. The resulting circuit passes the input values through unchanged. Therefore, the remaining outputs can be built by any other procedure without any special preprocessing.

3. APPLICATION: MULTIPLE OUTPUT SYMMETRIC FUNCTIONS

To run the application we need: reversible logic gates (design model), synthesis model, and the dynamic programming algorithm. Define the model by taking the most popular in reversible logic theory gates. $TOF(x_1) = \bar{x_1}$ is a NOT gate. $TOF(x_1; x_2) = (x_1, x_1 \oplus x_2)$ has been termed Feynman [6] or controlled-NOT gate (CNOT). $TOF(x_1, x_2; x_3) = (x_1, x_2, x_3 \oplus x_1 x_2)$ is usually referred as a Toffoli gate [19]. These gates are depicted in Figure 1. The set of these gates was shown to be complete. This is a minimal complete set of gates in the sense that excluding either one from this set will result in incompleteness if no garbage addition is allowed. We do not choose any specific design procedure as it will be showed that the implementation of a single step of the dynamic programming algorithm is very simple.

To design a dynamic programming algorithm we need several definitions.

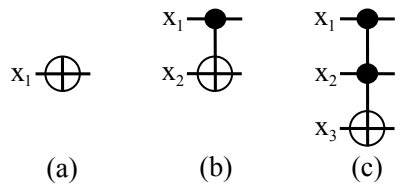

Figure 1: (a) $TOF(x_1)$, (b) $TOF(x_1; x_2)$ and (c) $TOF(x_1, x_2; x_3)$ Toffoli Gates.

DEFINITION 1. **Multiple output symmetric Boolean function** $\overrightarrow{F}(x_1, x_2, ..., x_n) = (y_1, y_2, ..., y_m)$ *is such a function that* $\overrightarrow{F}(x_1, x_2, ..., x_n) = \overrightarrow{F}(\pi(x_1, x_2, ..., x_n))$, *where* π *is a permutation of* n *elements.*

DEFINITION 2. *The* **σ-function** $\sigma_n^k(x_1, x_2, ..., x_n)$ *is defined as* $\bigoplus_{\{i_1<i_2<...<i_k\}} x_{i_1} x_{i_2} ... x_{i_k}$ *for* $k = 0, 1, ..., n$.

LEMMA 1. *Every symmetric function can be written as a linear combination (with respect to operation EXOR) of not more than* $(n + 1)$ σ*-functions.*

PROOF. There are $(n + 1)$ different σ-functions, all are linearly independent. In fact, their kernels (set of inputs for each the function is not equal zero) do not intersect, thus different linear combinations produce different functions. All σ-functions are symmetric, so are their linear combinations. The number of symmetric functions of n variables is 2^{n+1}, therefore the different linear combinations of σ-functions form the set of all symmetric functions and each symmetric function has unique linear form made of σ-functions. □

LEMMA 2. $\sigma_n^k(x_1, x_2, ..., x_n) = x_n \sigma_{n-1}^{k-1}(x_1, x_2, ..., x_{n-1}) \oplus \sigma_{n-1}^k(x_1, x_2, ..., x_{n-1})$ *for* $k \geq 2$.

PROOF. Observe that the first part of the right hand side has all the terms of degree k which include variable x_n as a multiple when the second part has all the terms of degree k which do not include the variable x_n. Thus, right hand side has all the terms of degree k, which, by definition, forms the left hand side. □

Note, that the statement of the Lemma 2 holds for $k = 1$. The result for $k = 1$ will be used to calculate $\sigma_n^1(x_1, x_2, ..., x_n)$. Use Lemma 2 to form the dynamic programming algorithm for calculating a symmetric function by saying that in order to build a multiple output symmetric function, first build the set of σ-functions recursively and then construct the output vector as a linear combination of the outputs of dynamic programming algorithm. Formally the algorithm works as follows:

```
1. create Boolean array sigma[1..n]=0;
2. for i=1 to n
3.    for k=i down to 1
4.       if k>1 sigma[k]
               = (sigma[k] + x[i]*sigma[k-1]) mod 2;
5.       if k=1 sigma[k] = (sigma[k] + x[i]) mod 2;
6.    end for;
7. end for.
```

Define the two numbers needed for calculation and build the circuit:

- In order to calculate $\sigma_b^a(x_1, x_2, ..., x_b)$ we need $\sigma_{b-1}^{a-1}(x_1, x_2, ..., x_{b-1})$ and $\sigma_{b-1}^{a}(x_1, x_2, ..., x_n)$, and in order to continue and complete the calculations we will need $\sigma_{b-1}^{a-1}(x_1, x_2, ..., x_{b-1})$, $\sigma_{b-1}^{a-2}(x_1, x_2, ..., x_{b-1}), ...,$ $\sigma_{b-1}^{1}(x_1, x_2, ..., x_{b-1})$. If $a = 1$ use the formula $\sigma_b^1(x_1, x_2, ..., x_b) = \sigma_{b-1}^1(x_1, x_2, ..., x_{b-1}) \oplus x_b$. For $a = 0$ do not create anything, since addition of a unit can be done in-place when the outputs are created by a single NOT gate. Therefore, $M = \max_{b=1,2,...,n}(b) = n$.

- For the intermediate calculations no garbage is needed, G=0. To calculate function $\sigma_b^1(x_1, x_2, ..., x_b)$ use the gate $TOF(x_b; \sigma_{b-1}^1(x_1, x_2, ..., x_{b-1}))$, string 4. of the pseudo code. Function $\sigma_{b-1}^1(x_1, x_2, ..., x_{b-1})$ will not be used in further calculations, therefore we can overwrite it. To calculate $\sigma_b^a(x_1, x_2, ..., x_b)$ use the gate $TOF(x_b, \sigma_{b-1}^a(x_1, x_2, ..., x_n); \quad \sigma_{b-1}^{a-1}(x_1, x_2, ..., x_{b-1}))$, string 5. of the pseudo code. Again, function $\sigma_{b-1}^{a-1}(x_1, x_2, ..., x_{b-1})$ will not be used in further calculations, so we can overwrite it.

When the dynamic programming part creates the set of outputs $\sigma_n^k(x_1, x_2, ..., x_n)$ for $k = 1, 2, .., n$, construct the output by including the needed σ-functions. Depending on what the function is, we may or may not need to create the additional garbage outputs. A greedy method may create m additional zero constants for the outputs. In practice, a better solution is usually possible. The described synthesis algorithm allows to formulate the following result.

THEOREM 1. *Every symmetric multiple output function* $\overrightarrow{F}(x_1, x_2, ..., x_n) = (y_1, y_2, ..., y_m)$ *can be realized with the cost of:*

- *at most* m *NOT gates, at most* $n + mn$ *CNOT gates, and* $\frac{n(n-1)}{2}$ *Toffoli gates;*
- *garbage of at most* $2n$ *bits.*

Less garbage can be achieved if we notice that there is no need to create a special constant line for $\sigma_b^1(x_1, x_2, ..., x_b)$, which can be stored on the input line x_b. This allows to decrease both garbage and reversible implementation costs by 1. Other garbage and cost saving comes from the observation that if all the outputs can be composed of first $k+1$ ($2 \leq k \leq n$) σ-functions $\sigma_n^0(x_1, x_2, ..., x_n), \sigma_n^1(x_1, x_2, ..., x_n), ...,$ $\sigma_n^k(x_1, x_2, ..., x_n)$, we do not need to run the dynamic programming algorithm to create the remaining $(n - k)$ σ-functions. These two observations allow to formulate the following result:

THEOREM 2. *Every symmetric multiple output function* $\overrightarrow{F}(x_1, x_2, ..., x_n) = (y_1, y_2, ..., y_m)$*, where linear* σ*-function decomposition requires a function of maximal degree* k *(*$2 \leq k \leq n$*) can be realized with the cost of:*

- *at most* m *NOT gates, at most* $n + mn - 1$ *CNOT gates, and* $\frac{(2n-k)(k-1)}{2}$ *Toffoli gates;*
- *garbage of at most* $n + k - 1$ *bits.*

Another benefit of using this method comes from the following consideration. If a multiple output function to be realized has both symmetric and non symmetric outputs, the symmetric outputs can be realized first by the suggested method. Then, the set of gates $TOF(x_{n-1}; x_n)$ $TOF(x_{n-2}; x_{n-1})$ ... $TOF(x_2; x_1)$ is applied to the end of the cascade, which creates the whole set of unchanged inputs $\{x_1, x_2, ..., x_n\}$ on the first n output lines of the cascade. Therefore, the remaining outputs can be built by any other procedure. Such an operation adds $(n - 1)$ gates to the cascade. If for the target technology the cost of a single garbage bit is less than the cost of $(n-2)$ CNOT gates, the very first approach when a special line is created for $\sigma_1(x_1, x_2, ..., x_n)$ can be used.

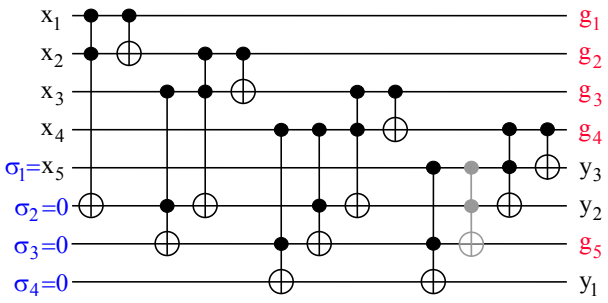

Figure 2: Circuit for $rd53$

EXAMPLE 1. *Take a multiple output function* $rd53$*, which is the 5-input 3-output symmetric function whose output is the binary representation of the number of ones in its input. First, find its* σ*-representation* $(\sigma^4, \sigma^2, \sigma^1)$*. Notice, that* σ^5*-function must not be built. Build the dynamic programming part. Observe, that the gates which affect a garbage bit whose changed value is not used by the design afterwards (the gate colored gray in Fig 2) can be deleted from the circuit without changing the output of the target function. The resulting circuit will contain 12 gates only.*

In all of our further designs, if a gate affects a garbage bit whose changed value is not used by the circuit to affect output bits afterwards, it is deleted from the design. This trivial procedure brings some simplification in almost every case.

The reversible Toffoli, CNOT and NOT gates can be implemented in quantum technology with the costs 5, 1 and 1 respectively. Note, that the quantum cost of the grey parts in the circuit is 4 [18], an implementation which was known to Peres [16, 7], and it is one smaller than the quantum cost of a single Toffoli gate (5). This gate was not considered as a separate in the literature, although it is clearly beneficial to do so.

This observations allow us to create the formula for quantum complexity of the method. It can be easily shown that the quantum complexity of a symmetric multiple output function $\overrightarrow{F}(x_1, x_2, ..., x_n) = (y_1, y_2, ..., y_m)$ requiring σ-functions of maximal order k is at most

$$m + n + mn - 1 + \frac{5(2n-k)(k-1)}{2} - 2(n-1) =$$

$$mn + m - n + 1 + \frac{5(2n-k)(k-1)}{2}.$$

4. COMPARISON OF THE RESULTS

There were several design methods proposed in the literature for the reversible design of multiple output Boolean functions. We would like to compare our results to the results of RPGA method by Perkowski *et al.* [17] (the method designed to synthesize the symmetric functions with reversible gates), reversible wave cascades [14], Khan gate family synthesis [9, 8], generalized Toffoli gates family [12, 5] and design of the Toffoli circuits using the templates [13]. The comparison consists essentially of the three parts: comparison of the garbage, number of gates in the reversible cascade and comparison of the quantum costs.

Unfortunately, [17] do not provide a table of results, which makes it hard to make the precise comparison. The asymptotic reversible cost (number of gates) of the both realizations are the same, namely $O(n^2)$. But, the RPGA method has excessive garbage, $\frac{n(n+1)}{2}$ (calculated in [12]), when the presented method has the garbage of maximum $(2n-1)$. A good quantum realization of the Kerntopf gates used in [17] was never found, therefore we claim that from the point of view of quantum cost our method will produce quantum circuits which will be constant (> 1) times cheaper. Comparison to the reversible wave cascades [14] (RWC columns), Khan gate family synthesis [9] (KGF columns) and generalized Toffoli gates family [12, 5] (GT columns) reversible synthesis results is summarized in Table 1. Actual circuits for the presented design can be found in [11].

The presented comparison is not quite fair. From one side, the mentioned methods are the general synthesis methods, which do not use special properties of functions such as ability to be calculated as a dynamic programming algorithm. From the other side, the cardinality of the set of gates of the mentioned methods is greater on the order than the number of gates used by the presented method.

From the table it can be seen that our method starts producing better results for larger functions both from the point of view of the reversible cost and garbage. The presented method can never beat the generalized Toffoli gates family synthesis method in terms of garbage, since the last has theoretically minimal garbage. But, the GT method scales badly, it can produce the circuits for reversible functions with no more than 10 inputs. The RWC and KGF are synthesized heuristically and they also scale much worse than the presented method.

Quantum cost comparison can be done accurately but in this paper we just mention that the quantum cost of RWC is at least n times reversible design cost, quantum cost of KGF implementation is at least $2n$ times higher than its reversible cost and quantum cost of GT is at least n times higher than its reversible cost, where n is the number of inputs of a function. The quantum cost for our model is given in previous section, and it cannot exceed 5 times the reversible cost. Clearly, the quantum cost of the presented approach is much better.

To illustrate how good the quantum cost is, compare the results to the ones presented in [13]. [13] gives an example of a circuit for $rd53$ function which has a cost of 12 gates, which seem to be the best among all known in reversible logic synthesis. The generalized Toffoli gates used in [13] are expensive (but no more expensive than the gates in RWC, KGF and GT) and the quantum complexity calculation based on results of [2] gives the quantum cost of 132 for that realization. Although the realization of $rd53$ presented in this paper has 18 gates in the reversible model, its quantum cost is only 36. Relation of the quantum costs 11:3 clearly shows the benefits of using the dynamic algorithm reversible synthesis method even for small functions. Since our method produces better results for larger functions, the quantum cost comparison for them will be even more beneficial.

Another interesting comparison can be made using $2of5$ function. GT realization claims 7 gates only in comparison to 12 in the presented paper. However, the quantum cost of GT realization is 158 in comparison to 32 for our circuit with 12 gates.

5. CONCLUSION

In this paper we presented a general approach to the synthesis of reversible circuits for the dynamic programming problems. As an illustration of its efficiency we applied it to the set of all multiple output symmetric functions and analyzed the three cost factors: reversible model cost, garbage cost and quantum cost. We showed that almost in every possible comparison our method produces better results (except for garbage comparison with [12] and [13]). The garbage in the presented method is close to the theoretical minimum. The quantum cost of the circuits produced by our algorithm is always significantly better. Finally, due to the small size of the gates used (maximum 3 inputs and 3 outputs), the levels of the network can be compressed which will result in a further simplification. On contrary, it is unlikely that level compression operation will give good results for the models that use large gates.

6. REFERENCES

[1] IBM's test-tube quantum computer makes history. Technical report, http://researchweb.watson.ibm.com/resources/news/20011219_quantum.shtml, Dec. 2001.

[2] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *The American Physical Society*, 52:3457–3467, 1995.

[3] C. H. Bennett. Logical reversibility of computation. *IBM J. Research and Development*, 17:525–532, November 1973.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[5] 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.

[6] R. Feynman. Quantum mechanical computers. *Optic News*, 11:11–20, 1985.

[7] P. Kerntopf. Maximally efficient binary and multi-valued reversible gates. In *International Workshop on Post-Binary ULSI Systems*, pages 55–58, Warsaw, Poland, May 2001.

[8] 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.

function			number of gates				garbage			
name	in	out	RWC	KGF	GT	We	RWC	KGF	GT	We
2of5	5	1	N/A	N/A	7	12	N/A	N/A	5	6
rd53	5	3	14	17	13	12	19	19	4	5
rd73	7	3	36	43	37	20	43	47	6	7
rd84	8	4	58	64	N/A	28	66	68	7	11
6sym	6	1	N/A	N/A	13	20	N/A	N/A	6	9
9sym	9	1	52	52	N/A	28	61	60	9	11
xor5	5	1	5	5	4	4	10	9	4	4

Table 1: Comparison of the results to RWC, KGF and GT

[9] 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.

[10] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Research and Development*, 5:183–191, 1961.

[11] D. Maslov, G. Dueck, and N. Scott. Reversible logic synthesis benchmarks page. Technical report, http://www.cs.unb.ca/profs/gdueck/quantum/, 2003.

[12] D. Maslov and G. W. Dueck. Garbage in reversible design of multiple output functions. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 162–170, March 2003.

[13] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of the Design Automation Conference*, pages 318–323, June 2003.

[14] A. Mishchenko and M. Perkowski. Logic synthesis of reversible wave cascades. In *International Workshop on Logic Synthesis*, pages 197–202, June 2002.

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

[16] A. Peres. Reversible logic and quantum computers. *Physical Review A*, 32:3266–3276, 1985.

[17] M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al-Rabadi, L. Joswiak, A. Coppola, and B. Massey. Regularity and symmetry as a base for efficient realization of reversible logic circuits. In *International Workshop on Logic Synthesis*, pages 245–252, 2001.

[18] M. Perkowski, M. Lukac, M. Pivtoraiko, P. Kerntopf, M. Folgheraiter, D. Lee, H. Kim, W. Hwangbo, J.-W. Kim, and Y. W. Choi. A hierarchical appoach to computer-aided desin of quantum circuits. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 201–209, March 2003.

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