

# Quantum Circuit Placement: Optimizing Qubit-to-qubit Interactions through Mapping Quantum Circuits into a Physical Experiment

Dmitri Maslov  
University of Waterloo  
Waterloo, ON, Canada  
dmitri.maslov@gmail.com

Sean M. Falconer  
University of Victoria  
Victoria, BC, Canada  
seanf@uvic.ca

Michele Mosca  
University of Waterloo  
Waterloo, ON, Canada  
mmosca@iqc.ca

## ABSTRACT

We study the problem of the practical realization of an abstract quantum circuit when executed on quantum hardware. By practical, we mean adapting the circuit to particulars of the physical environment which restricts/complicates the establishment of certain direct interactions between qubits. This is a quantum version of the classical circuit placement problem. We study the theoretical aspects of the problem and also present empirical results that match the best known solutions that have been developed by experimentalists. Finally, we discuss the efficiency of the approach and scalability of its implementation with regards to the future development of quantum hardware.

## Categories and Subject Descriptors

17[New, Emerging, or Specialized Design Technologies]:17.3 Quantum Computing

## General Terms

Quantum Circuits, Circuit Placement

## Keywords

Quantum Circuits, Quantum Circuit Placement, Time Optimization

## 1. INTRODUCTION

Quantum algorithm design is usually done in an abstract model of computation with the primary interest being the development of *efficient* algorithms (e.g. polynomial versus exponential sized circuits) independently of the details of the potential physical realization of the related quantum circuits. Theoretical quantum computational architectures usually, for convenience, assume that one can directly interact any two physical qubits. It is well known that in physical implementations<sup>1</sup> direct interactions between two distinct qubits may sometimes be hard if not impossible to establish. For instance, it is not uncommon for an interaction to have less than 0.2 Hz frequency (e.g. carbons C2 and C6 from [13]), while the decoherence may take around one second. This means that the interaction between such qubits will be essentially seen as noise. A physical realization of a quantum processor can of course *indirectly* couple any two qubits, with some overhead cost. However, in practice this can be very costly (and, for example, make the implementation infeasible with available technology), and so

<sup>1</sup> Trapped ions [5] and liquid NMR (Nuclear Magnetic Resonance) [3] are two of the most developed quantum technologies targeted for computation (as opposed to communication or cryptography). Other state of the art quantum information processing proposals are described in [6].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2007, June 4–8, 2007, San Diego, California, USA.

Copyright 2007 ACM ACM 978-1-59593-627-1/07/0006 ...\$5.00.

one needs to find a way to minimize this overhead.

Some authors [11] try to avoid long interactions by considering the linear nearest neighbor (LNN) architecture where all qubits are lined up in a chain (such as trapped ions [5]) and only the nearest neighbors are allowed to interact. Such an architecture, however, *does not necessarily* represent reality. For instance, in NMR (Nuclear Magnetic Resonance) technologies the fastest interactions will not usually form a chain. The LNN architecture can apply to some quantum dot based architectures and certain variations of trapped ion [5] quantum technologies, but does not represent the reality behind other technologies, including those based on NMR, Josephson junctions, optical technology, and certain other variations of trapped ions [12]. Furthermore, we are unaware of any automated procedure for optimizing the interactions even if the latter are restricted to the LNN architecture.

It has been recently proposed to use EPR pairs to teleport logic qubits in a Kane model when long-distance swapping of a *single* value becomes too expensive [1]. However, this type of “technology mating” may not always be possible. For instance, it seems hard if not impossible to mate NMR with an optic EPR source while doing a computation on an NMR machine. It is not obvious how congested such an architecture gets if simultaneous teleporting of a large number of qubits is required. We concentrate on an approach that does not require mating different technologies, and is *technology independent*.

Unlike in the quantum case, the classical circuit placement problem is very well known and extensively studied [9]. Classical circuit placement is concerned primarily with minimizing the total wirelength, power, and congestion. Timing minimization is considered to be a secondary optimization objective. In quantum technologies, placement is equivalent to timing optimization under the natural assumption that gate fidelities [14] are inversely proportional to the coupling strength/gate runtime, otherwise, a function of both may be considered. However, timing minimization for quantum versus CMOS technologies are two distinct and dissimilar problems. This is not surprising given the conceptual differences in the two computational models. A quantum circuit placer cannot be constructed by reusing the existing EDA tools and thus, must be built from scratch.

In practice and at present, the mapping of qubits in a circuit into physical qubits is done by hand, but as the quantum computations scale, a more robust (automated) technique for such a mapping must be employed. In this paper, we study the problem of mapping a circuit into a physical experiment where we wish to optimize the interactions. We formulate this problem mathematically and study its complexity. Given the problem of finding an optimal assignment of qubits is unlikely to have a polynomial time optimal solution, as it is NP-Complete, and exhaustive search requires  $n!$  tries for an  $n$ -qubit system, we develop a heuristic-based algorithm. We verify the latter on data taken from existing quantum experiments.

Quantum circuits are cascades composed with quantum gates. While our placement algorithms/theory and program implementation *readily support any gate library*, we next introduce some of the particular gates used in the Experimental Results section of this paper (these are

liquid NMR technology primitives; more generally, elementary gates for a quantum system described by Ising type Hamiltonian). For a detailed introduction into quantum computing, please see [14].

- rotation gates  $R_x(\theta) := \begin{pmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$ ,
  - $R_y(\theta) := \begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$ , and  $R_z(\theta) := \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$ ;
  - two qubit interaction gate
- $$ZZ(\theta) := \begin{pmatrix} e^{-i\theta/2} & 0 & 0 & 0 \\ 0 & e^{i\theta/2} & 0 & 0 \\ 0 & 0 & e^{i\theta/2} & 0 \\ 0 & 0 & 0 & e^{-i\theta/2} \end{pmatrix}.$$

In liquid NMR, single qubit  $R_x$  and  $R_y$  gates are implemented by RF pulses in the  $X$ - $Y$  plane.  $R_z$  gates can be implemented by the change of the rotating reference frame and thus require no action/waiting. Finally,  $ZZ$  gates are a part of the drift Hamiltonian and must be waited for until completion. Note that  $ZZ(\pi/2)$  is equivalent to CNOT up to the single qubit rotations. This means that every circuit with a single qubit and CNOT gates can be easily rewritten in terms of single qubit rotations and  $ZZ(\pi/2)$ , and such an operation does not change a particular instance of the placement problem.

## 2. FORMALIZATION OF THE PROBLEM

Circuits devised by researchers in the area of quantum computing are very often written in terms of single and two qubit gates. We assume such a circuit as input. Next, most of the practical circuits are levelled, that is the gates that can be applied in parallel appear at one logic level (in circuit diagrams: directly below or above other gates from the same level). Levelization helps to reduce the overall runtime of the circuit, and thus it is a desired operation at the time the circuit is synthesized.

In a practical setting and according to how it is done in current experiments the *first* step is to efficiently assign logical qubits of the circuit to be implemented to physical nuclei. In the existing NMR tools, the timing optimization is built into a compiler that takes in a circuit and a refocusing scheme and outputs a sequence of (timed) pulses ready to be executed. This is the last step before the circuit gets executed and before it is done the logical qubits must be assigned to the nuclei.

In practice, it is a natural assumption that gates from the next level can start being executed before execution of the current level has completed. The total **runtime** is defined as the time spent by a circuit between the input initialization and measurement of the output. With the assumptions made, runtime of a circuit composed of gates  $G[1], G[2], \dots, G[k]$  can be found by the following dynamic programming algorithm.

```
Create array Time[1..n] and initiate its elements to 0;
for i=1..k
  if G[i] is a two-qubit gate built on qubits t and c
    time[c]:=max{time[c],time[t]}+GateOperatingTime(G[i]);
    time[t]:=time[c];
  else // else means gate G[i] is a single qubit gate
    // that operates on qubit t
    time[t]:=time[t]+GateOperatingTime(G[i]);
  end if;
end for;
return max{time[1..n]}; // maximally busy qubit is the
// one that finishes the job last
```

Circuit runtime calculation in a computational model where logic levels need to be executed sequentially is also supported by the theory we develop and our program implementation. We next introduce the following notations/definitions.

**DEFINITION 1.** A **physical environment (molecule)** is a complete non-oriented graph with a finite set of vertices (nuclei)  $\{v_1, v_2, \dots, v_m\}$  and weighted edges  $\{(v_i, v_j)\}_{1 \leq i < j \leq m}$  with  $W(v_i, v_j) \geq 0$ .

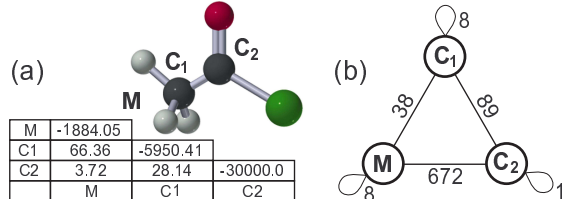
Weights  $W(v_i, v_j)$  indicate how long it takes to apply a fixed angle 2-qubit gate (in case of edge  $(v_i, v_j)$ ,  $i \neq j$ ) or a fixed angle single qubit gate (in case of edge  $(v_i, v_i)$ ). An example of a molecule used in quantum experiments [8] can be found in Figure 1.

**DEFINITION 2.** A **quantum circuit** is built on  $n$  qubits  $q_1, q_2, \dots, q_n$  and consists of a finite number of levels  $L_1, L_2, \dots, L_k$ , where each level  $L_i$  is a set of one or two qubit gates  $G_i^1, G_i^2, \dots, G_i^{l_i}$ . Each gate has a weight function  $T(G_i^j)$  that indicates how long it takes this gate to use the interaction defined by the qubits it operates on <sup>2</sup>.

**DEFINITION 3.** The **quantum circuit placement problem** is to construct an injective (one-to-one) function  $P: \{q_1, q_2, \dots, q_n\} \mapsto \{v_1, v_2, \dots, v_m\}$  such that with this mapping the runtime of a given circuit is minimized. The gate's  $G(v_i, v_j)$  execution cost is defined by the mapping  $q_i \mapsto v_i, q_j \mapsto v_j$  according to the formula

$$\text{GateOperatingTime}(G(q_i, q_j)) := W(v_i, v_j) * T(G(q_i, q_j)).$$

There are  $m!/(m-n)!$  candidates for an optimal matching. It is possible to show that quantum circuit placement problem defined above is NP-Complete via reduction to the Hamiltonian cycle problem [10].



**Figure 1:** (a) acetyl chloride molecule with three spin- $\frac{1}{2}$  nuclei used as qubits:  $M$  (three hydrogens that share the same chemical environment and thus appear practically indistinguishable), carbons  $C_1$  and  $C_2$  and (b) graph representation of interactions between qubit nuclei of this molecule.

## 3. HEURISTIC SOLUTION

Before we begin the description of our algorithm, we first make several observations about qubit assignment. Firstly, in real world experiments, such as [7, 13], the qubit-to-qubit interactions are usually aligned along the chemical bonds, or otherwise using only the fastest interactions allowed by the physical environment. Secondly, most often a single qubit gate can be executed much faster than a two-qubit gate. Third, it may turn out that a circuit is composed of a number of computations/stages each of which has its own best mapping into a physical environment. If the circuit is treated as a whole, no placement will be efficient because at least one part of such a placement will be inefficient. Below, we describe an approximate placement algorithm that uses heuristics derived from these observations.

### 3.1 Algorithm

**Preprocessing.** We first take the physical environment and establish which interactions are considered the fastest. One method for doing this is to choose a *Threshold* such that if a value of a qubit-to-qubit interaction is below the *Threshold*, the interaction is considered to be fast, otherwise it is considered too slow and will not be used. The value of the *Threshold* may be chosen to be the minimal value such that the graph associated with fastest interactions is connected, or may be taken directly from the experimentalists. Either way, we treat the value of the *Threshold* as a known parameter for our algorithm. Using the *Threshold* value insures that our algorithm comes up with a circuit implementation that uses only the fastest interactions.

**Body.** The algorithm consists of two stages: basic placement and fine tuning, repeated iteratively. In the basic mapping stage we make sure that in some part of the circuit all two qubit gates can be aligned along the fastest interactions of the physical environment. In the fine tuning stage we consider a given subcircuit for which a mapping of circuit qubits into the molecule's nuclei along the fastest interactions exists. We fine tune this matching by shuffling the solution and taking the actual numbers that represent the length of each gate (including single qubit gates) into account.

<sup>2</sup>For example,  $T(R_x(180)) = 2 * T(R_x(90))$  because it takes twice the time to do a 180-degree rotation as compared to a 90-degree rotation.

We start by reading in the 2-qubit gates from the circuit into a workspace  $C$  as long as these gates can be arranged along the fastest interactions provided by the physical environment. As soon as we reach a gate whose addition prohibits the alignment along the fastest interactions, we stop and concentrate on those gates already in  $C$ . We know that the two qubit gates in  $C$ ,  $g_1, g_2, \dots, g_s$ , can be aligned along the fastest interactions. Take one of such alignments in the case when multiple alignments are possible. The alignment itself can be found by any of the known graph monomorphism techniques, such as [2]. If all monomorphisms can be found reasonably fast, we calculate the circuit runtime for each of them and take the best found. Otherwise, we apply a simple hill climbing algorithm, where for every qubit  $q_i$  from the circuit such that there exists a two qubit gate  $g_j$  among those in  $C$  that operates on this qubit, try to map it to any of  $\{v_1, v_2, \dots, v_m\}$  and see if this new placement assignment is better than the one provided by the initial matching. Such an operation can be repeated until no improvement can be found or for a set number of iterations. We refer to this step as fine tuning. After the matching is fine tuned, we return to the circuit, erase all the gates from it up to the 2-qubit gate  $g_{s+1}$  that prevented alignment along the fastest interactions and repeat the above procedure.

The result of the above algorithm is a finite set of finely tuned placement assignments  $P_1, P_2, \dots, P_t$ , but they apply to different subcircuits  $C_1, C_2, \dots, C_t$  of the target circuit. The overall computation looks as follows  $C_1 E_{1,2} C_2 E_{2,3} \dots E_{t-1,t} C_t$ , where  $E_{i,i+1}$  is a circuit composed with SWAP gates such that  $E_{i,i+1}$  transforms mapping given by  $P_i$  into the mapping given by  $P_{i+1}$ , and initially all qubits are placed according to  $P_1$ .

### 3.2 Fast permutation circuits

The algorithm from the previous subsection describes all details of the circuit placement, except how to construct circuits  $E_{i,i+1}$  that would swap the placement assignment of subcircuit  $C_i$  into that for  $C_{i+1}$ . In this subsection we discuss how to compose such circuits efficiently—linear in number of qubits in the physical environment and using only the fastest interactions.

**Problem.** For a given permutation (defined as the transformation of placement assignment of subcircuit  $C_i$  into that for  $C_{i+1}$ ) and a graph defining which two qubits can be interchanged through a single SWAP (such an *adjacency graph* has an edge between two qubits if it is possible to SWAP their values directly) compose a set of SWAPs that realize this permutation.

**Assumptions.** For simplicity, we assume that all SWAP gates applied to the qubits joined by the edges of adjacency graph  $G$  require the same time. Modification of the algorithm presented below that accounts for the actual costs of SWAPs is possible, but it is not discussed here. Next, assume that the graph  $G$ , as well as its subgraphs, are *well separable*. That is, there exists a constant  $s \in (0, 1)$  such that  $G$  can be recursively divided into two connected components  $G_1$  and  $G_2$ , and the ratio of the number of vertices in the smaller subgraph to the number of vertices in the larger subgraph is never less than  $s$ . This is certainly true for the scalable interaction architectures proposed in the literature: LNN architecture (1D lattice,  $s = 1/2$ ), and 2D lattices ( $s \geq 1/2$ ). In our experiments, we found that the interaction graphs for liquid state NMR molecules have  $s = 1/2$ . We also assume that it is possible to execute non-intersecting gates in parallel.

**Goal.** Our target is to minimize the number of logic levels one needs in order to realize a given permutation as a circuit. Each logic level is composed with a set of non-intersecting SWAPs operating along the fastest interactions defined by the adjacency graph.

**Solution.** Cut adjacency graph  $G$  (with  $n$  vertices) into two connected subgraphs  $G_1$  and  $G_2$  with the number of vertices equal to or as close to  $n/2$  as possible. To do such a cutting we would have to cut a few edges, each called a *communication channel*.

Consider subgraphs  $G_1$  and  $G_2$ . Color each vertex white if ultimately (according to the permutation that needs to be realized) we

want to see it in  $G_1$  and black if we want to see it in  $G_2$ . The task is to permute the colors by swapping colors in the adjacent vertices such that all white colored vertices go to the  $G_1$  part and all black ones move to  $G_2$ . The bottleneck is in how many edges join the two halves  $G_1$  and  $G_2$ —the number of such edges represents the *carrying capacity* of the communication channel. This is because all white elements from  $G_2$  and all black elements from  $G_1$  must, at some point, use one of such communication channels to move to their subgraph.

If all colored vertices can be brought to their subgraph with a linear cost,  $a * n$ , then we iterate the algorithm and have the following figure for the complexity of the entire algorithm:  $C(n) \leq a * n + C(\frac{n}{s+1})$ , where  $C(n)$  is the complexity of the algorithm for a size  $n$  problem.

The solution for such a recurrence is  $C(n) = \frac{a(s+1)n}{s}$ . If the graph has separability factor  $s = 1/2$ , the recurrence can be rewritten as  $C(n) \leq a * n + C(2n/3)$ . The latter has solution

$$C(n) = 3an + const, \quad (1)$$

providing a linear upper bound for the entire algorithm.

We next discuss how to move all white vertices into the  $G_1$  subgraph and, simultaneously, black ones into the  $G_2$  subgraph in linear time  $a * n$ . Consider a single subgraph, e.g.  $G_1$ , and the edge(s) of  $G$  that we cut to create  $G_1$  and  $G_2$ . First, let us bring all black vertices as close to the communication channel as possible. To do so, we suppose that the communication channel consists of a single edge, otherwise, choose a single edge. We next cut all loops in  $G_1$ —in the following solution we allow swapping values along some but not all edges. At this point, we have a rooted tree with the root being the unique vertex that belongs to  $G_1$  and is the end of the communication channel. A rooted tree has a natural partial order to its vertices, induced by the minimal length path from a given vertex to the root.

We next apply the following algorithm. Suppose  $G_1$  has  $k + 1$  vertices. At step  $i = 1..k$ , we look at every depth  $k - i$  vertex and its parent at depth  $k - i - 1$ . If this vertex is colored black and its parent is white, we call such a parent a *bubble* and interchange it with the child via applying a SWAP. For all other bubbles, if such a bubble has a black child, SWAP the two. Otherwise, all children of the bubble are white, and as a result, this vertex is no longer a bubble. Let us note that once a bubble started “moving”, it will move each step of the algorithm until it is no longer a bubble, which happens only when all its children are white. By the step  $i = k$  all potential bubbles have started moving or have already finished their trip. It may take an additional  $k$  steps for moving bubbles to finish their movement. By the time step  $i = 2k$  is completed, every path from a vertex in  $G_1$  to the root will change color at most once (if color changes it can only change from white to black).

This brings us to the next part: interchanging white and black nodes in  $G_1$  and  $G_2$  and actually using the communication channel. We use the communication channel every odd step in this part of the algorithm to bring a single black vertex from  $G_1$  over to  $G_2$  and a single white vertex from  $G_2$  to  $G_1$ . Once a white vertex arrives in  $G_1$  we call it a bubble and rise it until all its children are white. A similar operation is performed with graph  $G_2$ . Every even step the root vertex of  $G_1$  is white (and thus is a bubble) and one of its children is black (if all children are white, the algorithm is finished;  $G_1$  is guaranteed to contain only white vertices, and all of them). We propagate the bubble to one of its children to have a black vertex ready for the next transfer. To get all white nodes to their side we need at most  $2k$  swaps.

Hence, the total cost of bringing white nodes to  $G_1$  and black ones to  $G_2$  is  $2k + 2k = 4k$ . Since in the graphs we consider  $s \geq 1/2$ , the above expression gives an upper bound of  $a = 8/3$ . Substituting this into the recurrence solution (1) gives the upper bound of  $8n + const$  for the number of levels in a quantum circuit composed with SWAPs working with the fastest interactions and realizing a given permutation. By considering permutation  $(n, 2, 3, \dots, n - 1, 1)$  and the LNN architecture we can show that the above algorithm is asymptotically optimal. In our implementation of the above algorithm, we do not block the communication channel in hopes of achieving a faster solution.

**Table 1: Mapping experimentally constructed circuits into their physical environment.**

Circuit			Environment		Estimated	Search
name	# gates	# qubits	name	# qubits	circuit runtime	space size
err. corr. encod. [8]	9	3	acetyl chloride [8], Figure 1	3	.0136 sec	6
5 bit err. corr. ([7], Fig. 1)	25	5	trans-crotonic acid ([7], Fig. 3)	7	.0779 sec	2,520
pseudo-cat state prep. ([13], Fig. 1)	54	10	histidine ([13], Fig. 2)	12	.5170 sec	239,500,800

### 3.3 Implementation and its scalability

We implemented the above algorithms in C++ using the VFLib graph matching library [2]. The bottleneck of our algorithm and its implementation is the efficiency of computing a solution to the subgraph monomorphism problem. This is because in order to solve the placement problem for a circuit with  $k$  gates, among which  $s$  are two-qubit gates, the subgraph monomorphism routine is called  $s$  times. All other operations performed by our algorithm and its implementation are at most quadratic in  $k$ . A study of runtime for aligning a particular type of graphs with around 1000 vertices showed that the alignment took close to one second on average [2]. We thus conclude that our implementation should not take more than an hour (3600 seconds, in other words, 3600 calls to subgraph isomorphism subroutine) to place quantum circuits built on a few hundred qubits and containing a few thousand gates.

## 4. EXPERIMENTAL RESULTS

In this section we present two types of experiments. First, we take circuits that were executed experimentally on an existing hardware. We erase the placement assignment of the circuit qubits into the physical environment and try to reconstruct it. Our tool finds the best solutions found by hand by experimentalists (meaning the tool creates only one workspace and the matching made is equivalent to that by experimentalists). Table 1 summarizes the results. The first three columns list properties of the circuit—a short description of what the circuits does and its source, the number of gates and qubits in it. The next two columns present properties of the environment—the name of the environment and its source, and the maximal number of qubits it supports. The *Estimated circuit runtime* column shows a result of the running time our algorithm formulated—an estimate (actual runtime, depending on the particulars of technology, may be slightly different to account for the second order effects) of how long it will take to execute a given circuit. The last column presents the size of the search space for placing a circuit as a whole (considering  $k$  subcircuits results in power  $k$  blow up of the search space size).

We next consider a circuit ([14], page 219) for a 6 qubit Quantum Fourier Transformation (QFT) and map it into a 7-qubit molecule. Unlike previous circuits considered in this paper the QFT circuit is not composed of liquid NMR primitives. This circuit is inconvenient for quantum architectures since it contains a 2-qubit gate for every pair of qubits. The circuit for 6-qubit QFT is used for the study of the relation between the value of *Threshold*, the number of subcircuits our software aligns individually and the overall cost of the alignment. Results are illustrated in Table 2, with the *Threshold* value appearing in the first column, the number of individual subcircuits aligned in second column, the time spent by the mapped circuit swapping qubits and the actual computation in the third and fourth columns, and the total cost of the alignment in the last column. As expected, usage of a large *Threshold* value resulted in a single working space, with the time spent on useful calculations decreasing when reducing the value of *Threshold*. However, the number of individually aligned subcircuits grows, and the time spent in swapping the values increases. For *Threshold* value 50 (in this case, adjacency graph of the molecule is unconnected, which, in fact, is an indication that the value of *Threshold* is already too small), the mapped circuit does too much swapping, resulting in high overall cost. The best result is achieved with *Threshold* = 200. This indicates that the *quantum circuit place-*

*ment tool has to use some rounds of SWAPs to achieve best results.* This is because the circuit with *Threshold* value 10000 is placed optimally as whole (i.e. when no SWAPs are allowed). The time spent in swapping suggests that our next goal should be the optimization of the swapping algorithm and the relation between time spent in swapping and doing useful calculation.

**Table 2: Relation between *Threshold*, number of subcircuits, and the cost of the alignment.**

<i>Threshold</i>	Subcirc	SWAP	Compute	Total
10000	1	0 sec	.4137 sec	.4137 sec
1000	2	.2460 sec	.1949 sec	.4409 sec
500	5	.4398 sec	.0327 sec	.4725 sec
200	5	.3324 sec	.0298 sec	.3622 sec
100	5	.4515 sec	.0321 sec	.4836 sec
50	9	.6846 sec	.0201 sec	.7048 sec

## 5. CONCLUSIONS

In this paper we presented an algorithm for the quantum circuit placement problem and its implementation. We tested our implementation on experimental data and computed, within a fraction of a second, the best results that have been reported by physicists. Further analysis indicates that our tool can handle very large quantum circuits (hundreds of qubits and thousands of gates) in a reasonable time (hours). We found that considering subcircuits and swapping their mappings is essential in achieving the best results. Our future work will be dedicated towards optimizing swapping circuits (in particular, we plan to explore look-ahead and peep-hole optimization techniques), study of the trade-off between increasing the size of the working space  $C$  and reducing SWAPs, and running larger (artificial) experiments to further address efficiency and scalability of our approach.

## Acknowledgements

This work was supported by NSERC, DTO-ARO, CFI, ORDCF, CIAR, CRC, ORF, and Ontario-MRI.

## 6. REFERENCES

- [1] D. Copley *et al.* *IEEE Journal of Selected Topics in Quantum Electronics*, 9(6):1552–1569, 2003.
- [2] L. P. Cordella *et al.* *10th ICIAP*, vol. 2, pages 1172–1178, 1999. The VFLib graph matching library, <http://amalfi.dis.unina.it/graph/>.
- [3] D. G. Cory *et al.* quant-ph/0004104, April 2000.
- [4] H. K. Cummins *et al.* *Physical Review Letters*, 88:187901, 2002.
- [5] H. Häffner *et al.* *Nature*, 438:643–646, 2005.
- [6] R. Hughes *et al.* [http://qist.lanl.gov/qcomp\\_map.shtml](http://qist.lanl.gov/qcomp_map.shtml), 2004.
- [7] E. Knill *et al.* *Physical Review Letters*, 86(5811), 2001.
- [8] M. Laforest *et al.* *Physical Review A*, 75(012331), 2007.
- [9] L. Lavagno, G. Martin, and L. Scheffer, editors. *Electronic Design Automation for Integrated Circuits Handbook*. Taylor & Francis, 2006.
- [10] D. Maslov, S. M. Falconer, and M. Mosca. quant-ph/0703256.
- [11] R. Van Meter and K. M. Itoh. *Physical Review A*, 71(052320), May 2005.
- [12] R. Van Meter and M. Oskin. *ACM Journal of Emerging Technologies in Computing Systems*, 2(1):31–63, 2006.
- [13] C. Negrevergne *et al.* *Physical Review Letters*, 96(170501), 2006.
- [14] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
- [15] P. W. Shor. *SIAM Journal of Computing*, 26(5):1484–1509, 1997.