Title: Efficient Distributed Quantum Computing

Date: Mar 11, 2013 04:00 PM

URL: http://pirsa.org/13030104

Abstract: <div id="Cleaner">I will explain how to simulate arbitrary quantum circuits on a distributed quantum computer (DQC), in which the pairs of qubits that are allowed to interact are restricted to the edges of some (connected) graph G. Even for graphs with only a modest number of long-range qubit interactions, such as the hypercube, this simulation is, in fact, efficient. Furthermore, for all graphs, the emulation scheme is very close to being optimal. <div id="Cleaner">Secondly I will present an efficient quantum algorithm for parallel unrestricted memory look-up. As an application, I will show that the space-time trade off for Element Distinctness and Collision Finding can be improved. <div id="Cleaner">Both results arise from applying the ideas of reversible sorting networks to quantum computing.<div id="Cleaner">http://arxiv.org/abs/1207.2307</div></div></div></div>

Efficient Distributed Quantum Computing

Bob Beals, Stephen Brierley[†], Oliver Gray[†], Aram Harrow, Sandy Kutin, Noah Linden, Dan Shepherd, Mark Stather

[†]Heilbronn Institute for Mathematical Research
University of Bristol, UK

February 19, 2013

Distributed quantum computing

We often design (and cost) algorithms in the **circuit model** (Q Circuit):

- can perform up to $\frac{N}{2}$ 2-qubit gates per time step;
- in each time step, any pair of qubits could be involved in a gate.

More realistic: **distributed quantum computer** (DQC)[vM06, vM09]:

[van Meter, Oskin, "Architectural implications of quantum computing technologies" (2006)]
[van Meter, Munro, Nemoto, Itoh, "Arithmetic on a distributed-memory quantum multicomputer" (2009)]

Distributed quantum computing

We often design (and cost) algorithms in the **circuit model** (Q Circuit):

- can perform up to $\frac{N}{2}$ 2-qubit gates per time step;
- in each time step, any pair of qubits could be involved in a gate.

More realistic: **distributed quantum computer** (DQC)[vM06, vM09]:

[van Meter, Oskin, "Architectural implications of quantum computing technologies" (2006)]
[van Meter, Munro, Nemoto, Itoh, "Arithmetic on a distributed-memory quantum multicomputer" (2009)]

Distributed quantum computing

We often design (and cost) algorithms in the **circuit model** (Q Circuit):

- can perform up to $\frac{N}{2}$ 2-qubit gates per time step;
- in each time step, any pair of qubits could be involved in a gate.

More realistic: **distributed quantum computer** (DQC)[vM06, vM09]:

- many (N) nodes, each comprising a small number (Q) of qubits;
- limited number of connections between nodes, given by graph $\mathcal{G}$;

[van Meter, Oskin, "Architectural implications of quantum computing technologies" (2006)]
[van Meter, Munro, Nemoto, Itoh, "Arithmetic on a distributed-memory quantum multicomputer" (2009)]

9

Distributed quantum computing

We often design (and cost) algorithms in the **circuit model** (Q Circuit):

- can perform up to $\frac{N}{2}$ 2-qubit gates per time step;
- in each time step, any pair of qubits could be involved in a gate.

More realistic: **distributed quantum computer** (DQC)[vM06, vM09]:

- many (N) nodes, each comprising a small number (Q) of qubits;
- limited number of connections between nodes, given by graph $\mathcal{G}$;
- can perform arbitrary 2-qubit gates at each time step, as long as all pairs lie within a node or in neighbouring nodes.

[van Meter, Oskin, "Architectural implications of quantum computing technologies" (2006)]
[van Meter, Munro, Nemoto, Itoh, "Arithmetic on a distributed-memory quantum multicomputer" (2009)]

11

Distributed quantum computing

Fix Q and fix a connected graph $\mathcal{G}$.

Problem

How do we run general algorithms (designed in the circuit model with graph K_N) on a distributed computer with graph $\mathcal{G}$ and Q qubits per node?

Distributed quantum computing

Fix Q and fix a connected graph $\mathcal{G}$.

Problem

How do we run general algorithms (designed in the circuit model with graph K_N) on a distributed computer with graph $\mathcal{G}$ and Q qubits per node?

Solution

- *We will give a recipe that maps* **any circuit** *on $N \cdot Q$ qubits to a circuit that respects the graph $\mathcal{G}$.*
- *The depth overhead is* **almost optimal**.

1D nearest neighbours

- Suppose we wish to simulate the circuit model on the 1D nearest neighbour graph. In each time step there might be $O(N)$ 2-qubit gates, each one acting on qubits distance $O(N)$ apart.

1D nearest neighbours

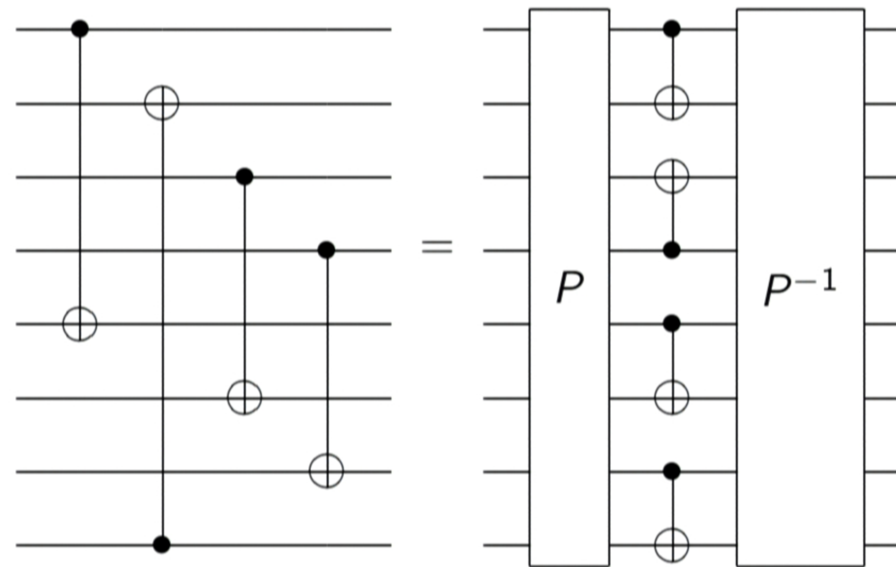
- Suppose we wish to simulate the circuit model on the 1D nearest neighbour graph. In each time step there might be $O(N)$ 2-qubit gates, each one acting on qubits distance $O(N)$ apart.
- We could bring a pair together using SWAP gates, perform the gate locally and return them in time $O(N)$. Total time $O(N^2)$.

1D nearest neighbours

- Suppose we wish to simulate the circuit model on the 1D nearest neighbour graph. In each time step there might be $O(N)$ 2-qubit gates, each one acting on qubits distance $O(N)$ apart.
- We could bring a pair together using SWAP gates, perform the gate locally and return them in time $O(N)$. Total time $O(N^2)$.
- [HNYN11] use a sorting network (Insertion) to do all $O(N)$ gates in time $O(N)$.

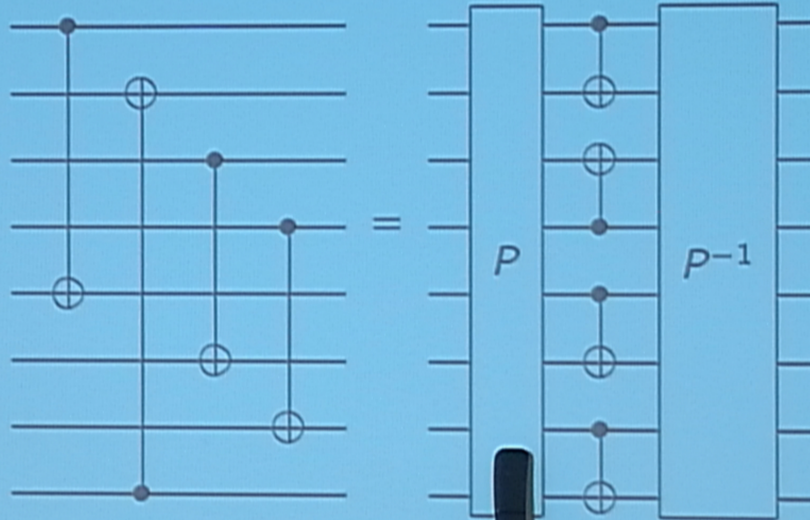
[Hirata, Nakanishi, Yamashita, Nakashima, "An efficient conversion of quantum circuits to a linear nearest neighbor architecture" (2011)]

1D nearest neighbours



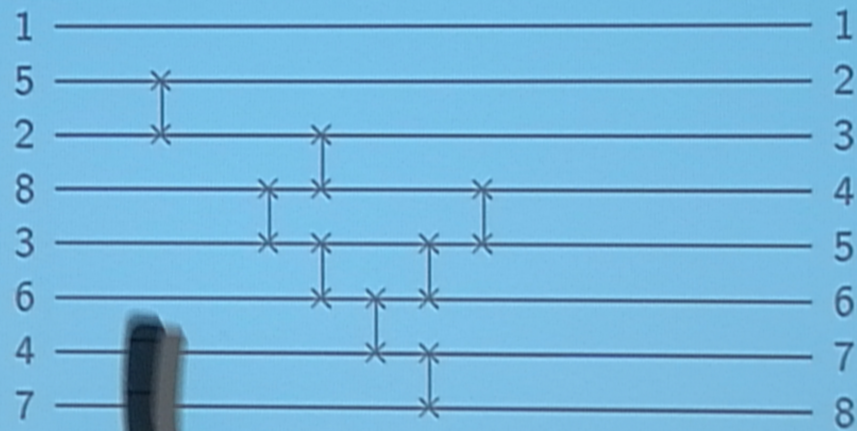
$$P = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 5 & 7 & 2 & 6 & 8 & 4 \end{pmatrix}$$

1D nearest neighbours

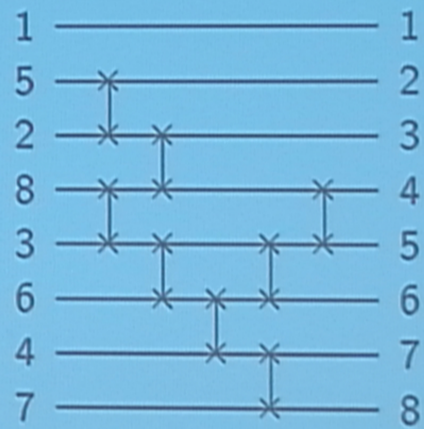


$$P = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 5 & 7 & 2 & 6 & 8 & 4 \end{pmatrix}$$

Insertion/Bubble Sort



Insertion/Bubble Sort



Quantum data moving

Suppose we have

- N pieces of quantum data $|x_1\rangle, \ldots, |x_N\rangle$, stored at the N nodes of a graph $\mathcal{G}$;

Quantum data moving

Suppose we have

- N pieces of quantum data $|x_1\rangle, \ldots, |x_N\rangle$, stored at the N nodes of a graph $\mathcal{G}$;
- a permutation $i \mapsto j_i$.

We wish to move the data according to the permutation j:

$$V_N : |x_1, \ldots, x_N\rangle \mapsto |x_{j_1}, \ldots, x_{j_N}\rangle .$$

Theorem

There exists an algorithm for V_N, restricted to $\mathcal{G}$, with depth $O(D_\mathcal{G})$, where $D_\mathcal{G}$ is the depth of a sorting network over $\mathcal{G}$.

Sorting networks

Definition

- A comparator is a device that takes two elements as input and outputs them in the correct order.

Sorting networks

Definition

- A comparator is a device that takes two elements as input and outputs them in the correct order.
- A sorting network is a network on T wires built from comparators that corrects sorts any inputs taken from an ordered set of T elements.

To make a sorting network reversible, we add an extra bit for each comparator to record the did-it-swap data.

Sorting networks

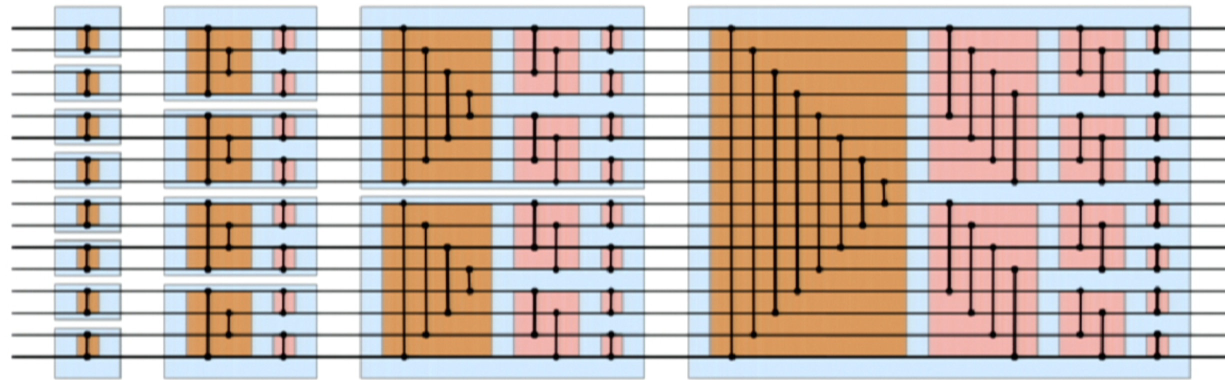
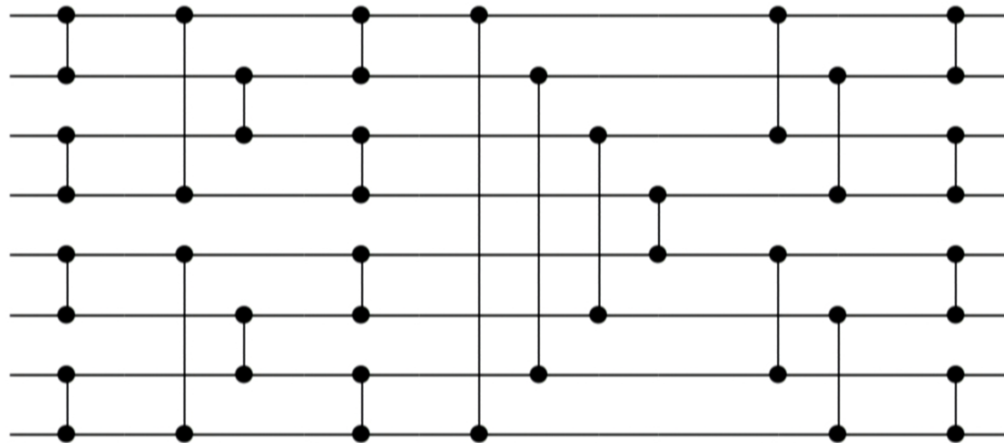
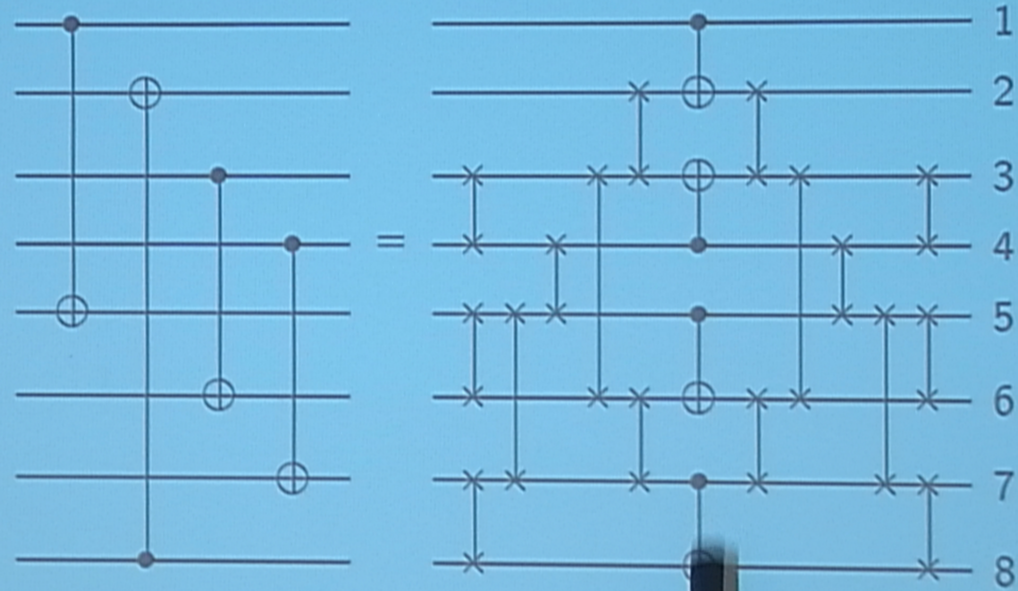


Figure: The bitonic sort on 16 wires

Bitonic Sort



Bitonic Sort



Simulating the circuit model

Theorem

Let $\mathcal{G}$ be a connected graph on N nodes, with $Q = O(\log N)$ qubits per node. Then the DQC model on $\mathcal{G}$ can simulate the circuit model with depth overhead factor $O(D_{\mathcal{G}})$.

Simulating the circuit model

Theorem

Let $\mathcal{G}$ be a connected graph on N nodes, with $Q = O(\log N)$ qubits per node. Then the DQC model on $\mathcal{G}$ can simulate the circuit model with depth overhead factor $O(D_{\mathcal{G}})$.

Proof.

At each time step, use V_N to bring interacting pairs together, perform the 2-qubits unitaries locally, then return qubits by unsorting V_N^{-1}. □

44

Simulating the circuit model

Theorem

Let $\mathcal{G}$ be a connected graph on N nodes, with $Q = O(\log N)$ qubits per node. Then the DQC model on $\mathcal{G}$ can simulate the circuit model with depth overhead factor $O(D_{\mathcal{G}})$.

Proof.

At each time step, use V_N to bring interacting pairs together, perform the 2-qubits unitaries locally, then return qubits by unsorting V_N^{-1}. □

Theorem

The cost of emulating arbitrary circuits of width $O(N \log N)$ on $\mathcal{G}$ is $\Omega(D_{\mathcal{G}} / \log N \log \log N)$.

Simulating the circuit model

Theorem

Let $\mathcal{G}$ be a connected graph on N nodes, with $Q = O(\log N)$ qubits per node. Then the DQC model on $\mathcal{G}$ can simulate the circuit model with depth overhead factor $O(D_{\mathcal{G}})$.

Proof.

At each time step, use V_N to bring interacting pairs together, perform the 2-qubits unitaries locally, then return qubits by unsorting V_N^{-1}. □

Theorem

The cost of emulating arbitrary circuits of width $O(N \log N)$ on $\mathcal{G}$ is $\Omega(D_{\mathcal{G}} / \log N \log \log N)$.

Examples

- [HNYN11] 1D nearest neighbour with bubble/insertion sort: val($\mathcal{G}$) = 2, $D_{\mathcal{G}} = O(N)$.

[Hirata, Nakanishi, Yamashita, Nakashima, "An efficient conversion of quantum circuits to a linear nearest neighbor architecture" (2011)]

Examples

- [HNYN11] 1D nearest neighbour with bubble/insertion sort: $\mathrm{val}(\mathcal{G}) = 2$, $D_{\mathcal{G}} = O(N)$.
- 2D nearest neighbour with bubble/insertion sort: $\mathrm{val}(\mathcal{G}) = 4$, $D_{\mathcal{G}} = O(\sqrt{N})$.

[Hirata, Nakanishi, Yamashita, Nakashima, "An efficient conversion of quantum circuits to a linear nearest neighbor architecture" (2011)]

Examples

- [HNYN11] 1D nearest neighbour with bubble/insertion sort: $\text{val}(\mathcal{G}) = 2,\ D_{\mathcal{G}} = O(N)$.
- 2D nearest neighbour with bubble/insertion sort: $\text{val}(\mathcal{G}) = 4,\ D_{\mathcal{G}} = O(\sqrt{N})$.
- hypercube with bitonic sort: $\text{val}(\mathcal{G}) = \log N,\ D_{\mathcal{G}} = O(\log^2 N)$.
- hypercube with AKS sort: $\text{val}(\mathcal{G}) = \log N,\ D_{\mathcal{G}} = O(\log N)$.

[Hirata, Nakanishi, Yamashita, Nakashima, "An efficient conversion of quantum circuits to a linear nearest neighbor architecture" (2011)]

Examples

- [HNYN11] 1D nearest neighbour with bubble/insertion sort: $\mathrm{val}(\mathcal{G}) = 2$, $D_{\mathcal{G}} = O(N)$.
- 2D nearest neighbour with bubble/insertion sort: $\mathrm{val}(\mathcal{G}) = 4$, $D_{\mathcal{G}} = O(\sqrt{N})$.
- hypercube with bitonic sort: $\mathrm{val}(\mathcal{G}) = \log N$, $D_{\mathcal{G}} = O(\log^2 N)$.
- hypercube with AKS sort: $\mathrm{val}(\mathcal{G}) = \log N$, $D_{\mathcal{G}} = O(\log N)$.
- complete graph with any sorting algorithm: $\mathrm{val}(\mathcal{G}) = N - 1$, $D_{\mathcal{G}} = O(\log N)$.

[Hirata, Nakanishi, Yamashita, Nakashima, "An efficient conversion of quantum circuits to a linear nearest neighbo

Parallel memory look-up

Sometimes it is useful to augment the circuit model by allowing simultaneous queries to a shared quantum RAM – we call this the **Q PRAM** model.

Parallel memory look-up

Sometimes it is useful to augment the circuit model by allowing simultaneous queries to a shared quantum RAM – we call this the **Q PRAM** model.

For example, to use Grover's algorithm to search a database, we need to be able to query the database in superposition. What if we wanted to run several Grover searches concurrently, with many processors all querying the same memory at once?

Parallel memory look-up

Sometimes it is useful to augment the circuit model by allowing simultaneous queries to a shared quantum RAM – we call this the **Q PRAM** model.
For example, to use Grover's algorithm to search a database, we need to be able to query the database in superposition. What if we wanted to run several Grover searches concurrently, with many processors all querying the same memory at once?

Solution

- *We show that Q Circuit can* **efficiently simulate** *Q PRAM;*

Parallel memory look-up

Sometimes it is useful to augment the circuit model by allowing simultaneous queries to a shared quantum RAM – we call this the **Q PRAM** model.
For example, to use Grover's algorithm to search a database, we need to be able to query the database in superposition. What if we wanted to run several Grover searches concurrently, with many processors all querying the same memory at once?

Solution

- *We show that Q Circuit can* **efficiently simulate** *Q PRAM;*

55

Parallel memory look-up

Sometimes it is useful to augment the circuit model by allowing simultaneous queries to a shared quantum RAM – we call this the **Q PRAM** model.
For example, to use Grover's algorithm to search a database, we need to be able to query the database in superposition. What if we wanted to run several Grover searches concurrently, with many processors all querying the same memory at once?

Solution

- *We show that Q Circuit can* ***efficiently simulate*** *Q PRAM;*
- *Parallel memory look-ups are* ***almost as cheap*** *as a single memory look-up.*

Single memory look-up

Look up single piece (d bits) of quantum data in list $|x_1\rangle, \ldots, |x_N\rangle$:

$$U_{(1.N)} : |j\rangle |y\rangle |x_1, \ldots, x_N\rangle \mapsto |j\rangle |y \oplus x_j\rangle |x_1, \ldots, x_N\rangle .$$

Single memory look-up

Look up single piece (d bits) of quantum data in list $|x_1\rangle, \ldots, |x_N\rangle$:

$$U_{(1.N)} : |j\rangle |y\rangle |x_1, \ldots, x_N\rangle \mapsto |j\rangle |y \oplus x_j\rangle |x_1, \ldots, x_N\rangle .$$

Theorem

In the circuit model, this requires width $\Omega(Nd)$ and depth $\Omega(\log N)$.

59

Parallel memory look-up

We now wish to look up N pieces of data in parallel:

$$U_{(N,N)} : |j_1, \ldots, j_N\rangle |y_1, \ldots, y_N\rangle |x_1, \ldots, x_N\rangle$$
$$\mapsto |j_1, \ldots, j_N\rangle |y_1 \oplus x_{j_1}, \ldots, y_N \oplus x_{j_N}\rangle |x_1, \ldots, x_N\rangle$$

Parallel memory look-up

We now wish to look up N pieces of data in parallel:

$$U_{(N,N)} : |j_1, \ldots, j_N\rangle |y_1, \ldots, y_N\rangle |x_1, \ldots, x_N\rangle$$
$$\mapsto |j_1, \ldots, j_N\rangle |y_1 \oplus x_{j_1}, \ldots, y_N \oplus x_{j_N}\rangle |x_1, \ldots, x_N\rangle$$

Parallel memory look-up

We now wish to look up N pieces of data in parallel:

$$U_{(N,N)} : |j_1, \ldots, j_N\rangle |y_1, \ldots, y_N\rangle |x_1, \ldots, x_N\rangle$$
$$\mapsto |j_1, \ldots, j_N\rangle |y_1 \oplus x_{j_1}, \ldots, y_N \oplus x_{j_N}\rangle |x_1, \ldots, x_N\rangle$$

Theorem
There exists a circuit for $U_{(N.N)}$ with width $O(N(\log N + d))$ and depth $O(\log N \log(d \log N))$.

60

Parallel memory look-up

We now wish to look up N pieces of data in parallel:

$$U_{(N,N)} : |j_1, \ldots, j_N\rangle |y_1, \ldots, y_N\rangle |x_1, \ldots, x_N\rangle$$
$$\mapsto |j_1, \ldots, j_N\rangle |y_1 \oplus x_{j_1}, \ldots, y_N \oplus x_{j_N}\rangle |x_1, \ldots, x_N\rangle$$

Theorem

There exists a circuit for $U_{(N.N)}$ with width $O(N(\log N + d))$ and depth $O(\log N \log(d \log N))$.

61

Algorithm for $U_{(N,N)}$

1. Format the data:

$$|j_1,\ldots,j_N\rangle\,|y_1,\ldots,y_N\rangle\,|x_1,\ldots,x_N\rangle\bigotimes_{i=1}^{N}|(0,0,0,0)\rangle_{2i}\,|(0,0,0,0)\rangle_{2i+1}$$

$$\mapsto |0,\ldots,0\rangle\,|0,\ldots,0\rangle\,|0,\ldots,0\rangle\bigotimes_{i=1}^{N}|(i,a,0,x_i)\rangle_{2i}\,|(j_i,q,y_i,0)\rangle_{2i+1}$$

in depth 1.

Algorithm for $U_{(N,N)}$

2. Sort the packets (by first two registers):

$$|0\rangle \bigotimes_{k=1}^{2N} |(i_k, f_k, y_k, x_k)\rangle_i \mapsto |\sigma\rangle \bigotimes_{k=1}^{2N} \left|(i_{\sigma(k)}, f_{\sigma(k)}, y_{\sigma(k)}, x_{\sigma(k)})\right\rangle_k$$

in depth $D_{\mathcal{G}}$.

Algorithm for $U_{(N,N)}$

2. Sort the packets (by first two registers):

$$|0\rangle \bigotimes_{k=1}^{2N} |(i_k, f_k, y_k, x_k)\rangle_i \mapsto |\sigma\rangle \bigotimes_{k=1}^{2N} |(i_{\sigma(k)}, f_{\sigma(k)}, y_{\sigma(k)}, x_{\sigma(k)})\rangle_k$$

in depth $D_{\mathcal{G}}$.

Algorithm for $U_{(N,N)}$

2. Sort the packets (by first two registers):

$$|0\rangle \bigotimes_{k=1}^{2N} |(i_k, f_k, y_k, x_k)\rangle_i \mapsto |\sigma\rangle \bigotimes_{k=1}^{2N} |(i_{\sigma(k)}, f_{\sigma(k)}, y_{\sigma(k)}, x_{\sigma(k)})\rangle_k$$

in depth $D_{\mathcal{G}}$.
We obtain a sequence of packets

$$\dots |(i, q, y_*, 0)\rangle \dots |(i, q, y_*, 0)\rangle\, |(i, a, 0, x_i)\rangle \dots$$

where $*$ labels processors that have queried processor

Algorithm for $U_{(N,N)}$

3. Cascade: we have a sequence of packets

$$\ldots|(i,q,y_*,0)\rangle \ldots |(i,q,y_*,0)\rangle\,|(i,a,0,x_i)\rangle \ldots$$

We copy the data x_i leftwards into the empty data registers by performing CNOTs on the fourth register at distance 1, then 2, then 4, ... in depth $\lceil \log 2N \rceil$ to obtain the sequence

$$\ldots|(i,q,y_*,x_i)\rangle \ldots |(i,q,y_*,x_i)\rangle\,|(i,a,0,x_i)\rangle \ldots$$

Algorithm for $U_{(N,N)}$

3. Cascade: we have a sequence of packets

$$\ldots |(i, q, y_*, 0)\rangle \ldots |(i, q, y_*, 0)\rangle \, |(i, a, 0, x_i)\rangle \ldots$$

We copy the data x_i leftwards into the empty data registers by performing CNOTs on the fourth register at distance 1, then 2, then 4, ... in depth $\lceil \log 2N \rceil$ to obtain the sequence

$$\ldots |(i, q, y_*, x_i)\rangle \ldots |(i, q, y_*, x_i)\rangle \, |(i, a, 0, x_i)\rangle \ldots$$

Algorithm for $U_{(N,N)}$

4. Copy: Every packet CNOTs the memory-data into the target-data register

$$\ldots |(i, q, y_*, x_i)\rangle \ldots |(i, q, y_*, x_i)\rangle \, |(i, a, 0, x_i)\rangle \ldots$$
$$\mapsto \ldots |(i, q, y_* \oplus x_i, x_i)\rangle \ldots |(i, q, y_* \oplus x_i, x_i)\rangle \, |(i, a, x_i, x_i)\rangle \ldots$$

in depth 1.

Algorithm for $U_{(N,N)}$

6. Unsort: (the first two registers are unchanged). Running the sorting network in reverse we obtain

$$\bigotimes_{i=1}^{N} |(i, a, x_i, x_i)\rangle \, |(j_i, q, y_i \oplus x_{j_i}, 0)\rangle$$

in depth $D_{\mathcal{G}}$.

Algorithm for $U_{(N,N)}$

6. Unsort: (the first two registers are unchanged). Running the sorting network in reverse we obtain

$$\bigotimes_{i=1}^{N} |(i, a, x_i, x_i)\rangle\, |(j_i, q, y_i \oplus x_{j_i}, 0)\rangle$$

in depth $D_{\mathcal{G}}$.

Algorithm for $U_{(N,N)}$

7. Unformat:

$$|0,\ldots,0\rangle\,|0,\ldots,0\rangle\,|0,\ldots,0\rangle \bigotimes_{i=1}^{N} |(i,a,x_i,x_i)\rangle_{2i}\,|(j_i,q,y_i\oplus x_{j_i},0)\rangle_{2i+1}$$

$$\mapsto |j_1,\ldots,j_N\rangle\,|y_1\oplus x_{j_1},\ldots,y_N\oplus x_{j_N}\rangle\,|x_1,\ldots,x_N\rangle \bigotimes_{i=1}^{N} |(0,0,0,0)\rangle_{2i}\,|(0,0,0,0)\rangle_{2i+1}$$

using Pauli X and SWAP in depth 2.

Algorithm for $U_{(N,N)}$

7. Unformat:

$$
\begin{aligned}
&|0,\ldots,0\rangle\,|0,\ldots,0\rangle\,|0,\ldots,0\rangle \\
&\qquad \bigotimes_{i=1}^{N} |(i,a,x_i,x_i)\rangle_{2i}\,|(j_i,q,y_i\oplus x_{j_i},0)\rangle_{2i+1} \\
\mapsto\ &|j_1,\ldots,j_N\rangle\,|y_1\oplus x_{j_1},\ldots,y_N\oplus x_{j_N}\rangle\,|x_1,\ldots,x_N\rangle \\
&\qquad \bigotimes_{i=1}^{N} |(0,0,0,0)\rangle_{2i}\,|(0,0,0,0)\rangle_{2i+1}
\end{aligned}
$$

using Pauli X and SWAP in depth 2.

Grover searching a database

- Grover's search algorithm solves $f(j) = 1$ where $f : \{1, \ldots, N\} \to \{0, 1\}$. It searches through the set $\{1, \ldots, N\}$ using

$$U_f : |j\rangle |y\rangle \mapsto |j\rangle |y \oplus f(j)\rangle .$$

Grover searching a database

- Grover's search algorithm solves $f(j) = 1$ where $f : \{1, \ldots, N\} \to \{0, 1\}$. It searches through the set $\{1, \ldots, N\}$ using

$$U_f : |j\rangle |y\rangle \mapsto |j\rangle |y \oplus f(j)\rangle .$$

Grover searching a database

- Grover's search algorithm solves $f(j) = 1$ where $f : \{1, \ldots, N\} \to \{0, 1\}$. It searches through the set $\{1, \ldots, N\}$ using

$$U_f : |j\rangle |y\rangle \mapsto |j\rangle |y \oplus f(j)\rangle .$$

- Suppose we want to search a database instead: find j such that $\alpha(x_j) = 1$ where $\alpha : X = \{x_j \mid j = 1, \ldots N\} \to \{0, 1\}$.

Grover searching a database

- Grover's search algorithm solves $f(j) = 1$ where $f : \{1, \dots, N\} \to \{0, 1\}$. It searches through the set $\{1, \dots, N\}$ using

$$U_f : |j\rangle |y\rangle \mapsto |j\rangle |y \oplus f(j)\rangle .$$

- Suppose we want to search a database instead: find j such that $\alpha(x_j) = 1$ where $\alpha : X = \{x_j \mid j = 1, \dots N\} \to \{0, 1\}$.
- Construct oracle using $U_{1,N} \circ U_\alpha \circ U_{1,N}$:

$$\begin{aligned} |j\rangle |0\rangle |y\rangle |x_1, \dots, x_N\rangle &\mapsto |j\rangle |x_j\rangle |y\rangle |x_1, \dots, x_N\rangle \\ &\mapsto |j\rangle |x_j\rangle |y \oplus \alpha(x_j)\rangle |x_1, \dots, x_N\rangle \\ &\mapsto |j\rangle |0\rangle |y \oplus \alpha(x_j)\rangle |x_1, \dots, x_N\rangle \end{aligned}$$

Grover searching a database

- Grover's search algorithm solves $f(j) = 1$ where $f : \{1, \dots, N\} \to \{0, 1\}$. It searches through the set $\{1, \dots, N\}$ using

$$U_f : |j\rangle |y\rangle \mapsto |j\rangle |y \oplus f(j)\rangle .$$

- Suppose we want to search a database instead: find j such that $\alpha(x_j) = 1$ where $\alpha : X = \{x_j \mid j = 1, \dots N\} \to \{0, 1\}$.
- Construct oracle using $U_{1,N} \circ U_\alpha \circ U_{1,N}$:

$$\begin{aligned} |j\rangle |0\rangle |y\rangle |x_1, \dots, x_N\rangle &\mapsto |j\rangle |x_j\rangle |y\rangle |x_1, \dots, x_N\rangle \\ &\mapsto |j\rangle |x_j\rangle |y \oplus \alpha(x_j)\rangle |x_1, \dots, x_N\rangle \\ &\mapsto |j\rangle |0\rangle |y \oplus \alpha(x_j)\rangle |x_1, \dots, x_N\rangle \end{aligned}$$

Grover searching a database

This easily generalises to search for r-tuples in a database:

- let $\alpha : X^r \to \{0,1\}$ and find $\mathbf{j} = (j^1, \ldots, j^r)$ such that $\alpha(x_{j^1}, \ldots, x_{j^r}) = 1$.
- Construct the oracle $U^r_{1,N} \circ U_\alpha \circ U^r_{1,N}$.
- Requires width $\Omega(Nd)$ and depth $\Omega(\sqrt{(N^r/M)}(r \log N + D_\alpha))$.

(M = number of solutions, X contains strings of length d bits)

Grover searching a database

This easily generalises to search for r-tuples in a database:

- let $\alpha : X^r \to \{0,1\}$ and find $\mathbf{j} = (j^1, \ldots, j^r)$ such that $\alpha(x_{j^1}, \ldots, x_{j^r}) = 1$.
- Construct the oracle $U^r_{1,N} \circ U_\alpha \circ U^r_{1,N}$.
- Requires width $\Omega(Nd)$ and depth $\Omega(\sqrt{(N^r/M)}(r \log N + D_\alpha))$.

(M = number of solutions, X contains strings of length d bits)

Multi-Grover search

Theorem

Suppose that we have N functions $\alpha_1, \ldots, \alpha_N : X^r \rightarrow \{0,1\}$, where the database X contains N data-strings of length $d = O(polylog N)$. Then there is a quantum algorithm that returns all N solutions with width $\tilde{O}(N)$ and depth $\tilde{O}(\sqrt{N^r/M})$.

Multi-Grover search

Theorem

Suppose that we have N functions $\alpha_1, \ldots, \alpha_N : X^r \to \{0,1\}$, where the database X contains N data-strings of length $d = O(polylogN)$. Then there is a quantum algorithm that returns all N solutions with width $\tilde{O}(N)$ and depth $\tilde{O}(\sqrt{N^r/M})$.

Element distinctness

- Element distinctness: given $f : \{0,1\}^n \rightarrow \{0,1\}^n$, is f injective?
- Classically $O(N)$ queries, or $ST = O(N^2)$.

Element distinctness

- Element distinctness: given $f : \{0,1\}^n \rightarrow \{0,1\}^n$, is f injective?
- Classically $O(N)$ queries, or $ST = O(N^2)$.
- [Ambainis] gives $\tilde{O}(N^{\frac{2}{3}})$ queries, $S = T = \tilde{O}(N^{\frac{2}{3}})$.

[Ambainis, "Quantum walk algorithm for element distinctness" (2007)]

Element distinctness

- Element distinctness: given $f : \{0,1\}^n \to \{0,1\}^n$, is f injective?
- Classically $O(N)$ queries, or $ST = O(N^2)$.
- [Ambainis] gives $\tilde{O}(N^{\frac{2}{3}})$ queries, $S = T = \tilde{O}(N^{\frac{2}{3}})$.
- Grover search S blocks of size N^2/S, $ST^2 = \tilde{O}(N^2)$.

[Ambainis, "Quantum walk algorithm for element distinctness" (2007)]

Element distinctness

- Element distinctness: given $f : \{0,1\}^n \to \{0,1\}^n$, is f injective?
- Classically $O(N)$ queries, or $ST = O(N^2)$.
- [Ambainis] gives $\tilde{O}(N^{\frac{2}{3}})$ queries, $S = T = \tilde{O}(N^{\frac{2}{3}})$.
- Grover search S blocks of size N^2/S, $ST^2 = \tilde{O}(N^2)$.
- Running the algorithm of [Buhrman et al] coupled with multi-Grover yields the trade-off $ST = \tilde{O}(N)$.

[Ambainis, "Quantum walk algorithm for element distinctness" (2007)]
[Buhrman, Durr, Heiligman, Hoyer, Magniez, Santha, de Wolf "Quantum algorithms for element distinctness" (2005)]

Element distinctness

- Element distinctness: given $f : \{0,1\}^n \to \{0,1\}^n$, is f injective?
- Classically $O(N)$ queries, or $ST = O(N^2)$.
- [Ambainis] gives $\tilde{O}(N^{\frac{2}{3}})$ queries, $S = T = \tilde{O}(N^{\frac{2}{3}})$.
- Grover search S blocks of size N^2/S, $ST^2 = \tilde{O}(N^2)$.
- Running the algorithm of [Buhrman et al] coupled with multi-Grover yields the trade-off $ST = \tilde{O}(N)$.
- This answers the challenge of [Grover-Rudolph].

[Ambainis, "Quantum walk algorithm for element distinctness" (2007)]
[Buhrman, Durr, Heiligman, Hoyer, Magniez, Santha, de Wolf "Quantum algorithms for element distinctness" (2005)]
[Grover, Rudolph, "How significant are the known collision and element distinctness algorithms?" (2004)]

Element distinctness

- Element distinctness: given $f : \{0,1\}^n \to \{0,1\}^n$, is f injective?
- Classically $O(N)$ queries, or $ST = O(N^2)$.
- [Ambainis] gives $\tilde{O}(N^{\frac{2}{3}})$ queries, $S = T = \tilde{O}(N^{\frac{2}{3}})$.
- Grover search S blocks of size N^2/S, $ST^2 = \tilde{O}(N^2)$.
- Running the algorithm of [Buhrman et al] coupled with multi-Grover yields the trade-off $ST = \tilde{O}(N)$.
- This answers the challenge of [Grover-Rudolph].

[Ambainis, "Quantum walk algorithm for element distinctness" (2007)]
[Buhrman, Durr, Heiligman, Hoyer, Magniez, Santha, de Wolf "Quantum algorithms for element distinctness" (2005)]
[Grover, Rudolph, "How significant are the known collision and element distinctness algorithms?" (2004)]

Conclusion

- We showed that a distributed quantum computer can simulate the Q PRAM model;
- on the hypercube, the simulation is efficient.
- Thus quantum memory can be distributed among many processors.
- As a corollary we meet the Grover-Rudolph challenge and improve the space-time trade-off for element distinctness and the collision problem.