Quantum Computing Research at Microsoft

Dave Wecker (wecker@microsoft.com)
Microsoft QuArC Group
QuArC Team

Alex Bocharov  Alan Geller  Yuri Gurevich  Matt Hastings

Martin Roetteler  Burton Smith  Krysta Svore

Matthias Troyer  David Tuckerman  Dave Wecker  Nathan Wiebe
Station Q Team – Theoretical Physics
QuArC Goal

To design real-world quantum algorithms for implementation on small-, medium-, and large-scale quantum computers

To design quantum circuits for efficient implementation of quantum algorithms

To design a comprehensive system architecture for a scalable, fault-tolerant, programmable quantum computer
QuArC Areas of Research

• **Quantum circuit synthesis**
  - Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  - Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  - Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  - Characterization of quantum state transformations using ancilla (Blass, Gurevich)
  - A canonical form for \{H,T\} single-qubit circuits (Bocharov, Svore)

• **Quantum algorithms**
  - Faster phase estimation (Svore, Hastings, Freedman)
  - Hubbard model (Wecker, Troyer, Hastings, Nayak, Clark)
  - Quantum chemistry (Wecker, Troyer)
  - Hamiltonian simulation (Wiebe, Wecker, Troyer)
  - 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
  - Classically simulating adiabatic algorithms (Hastings, Freedman, Troyer, Wecker)
  - Computational Complexity (Hastings, Freedman)

• **Quantum error correction and distillation**
  - Noise threshold for small-distance surface codes (Tomita, Svore)
  - Noise threshold for magic state distillation on topological architectures (Chen, Svore)
  - State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
  - Topological Computational Power (Hastings, Nayak, Freedman)

• **Quantum languages and platforms**
  - LIQubit (Wecker, Geller, Smith, Svore, Bocharov)
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)
LIQ\textsubscript{U}i Goals

- Create a simulation environment that makes it easy to create complicated quantum circuits
- The simulation should be as efficient as possible with as large a number of entangled qubits (and sets of them) as possible
- Circuits should be re-targetable for many purposes including: Rendering, Optimization and Export
- Provide multiple simulators targeting tradeoffs between universality, large numbers of qubits and physical simulations
- Allow user extensibility for maximum flexibility
The LIQâ„¢ Platform

<table>
<thead>
<tr>
<th>Language</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F#</td>
<td>Script</td>
<td>C#</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gates ...</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Simulators</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal</td>
<td>Stabilizer</td>
<td>Hamiltonian</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Runtime</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Service</td>
<td>Cloud</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimize</td>
<td>QECC</td>
<td>CD</td>
<td>Noise</td>
<td>Export</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Back End ...</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical</td>
<td>Quantum</td>
<td></td>
</tr>
</tbody>
</table>
# Quantum Gates

**Evolution:** \( |\psi'\rangle = U|\psi\rangle \), this may be realized by a Hamiltonian \( H = \frac{\ln(U)}{\Delta t} \)

<table>
<thead>
<tr>
<th>Type</th>
<th>Basis</th>
<th>U</th>
<th>Name</th>
<th>Sym</th>
</tr>
</thead>
</table>
| Pauli            | \{\ket{0}, \ket{1}\}  | \[
\begin{pmatrix}
1 & 0 \\
0 & 1 
\end{pmatrix}
\] | X    | \(X\) |
|                  | \{\ket{0}, \ket{1}\}  | \[
\begin{pmatrix}
0 & -i \\
i & 0 
\end{pmatrix}
\] | Y    | \(Y\) |
| Z Rotation       | \{\ket{0}, \ket{1}\}  | \[
\begin{pmatrix}
1 & 0 \\
0 & -1 
\end{pmatrix}
\] | Z    | \(Z\) |
| \(e^{i\pi/2}\)   | \{\ket{0}, \ket{1}\}  | \[
\begin{pmatrix}
1 & 0 \\
0 & i 
\end{pmatrix}
\] | S    | \(S\) |
|                  | \{\ket{0}, \ket{1}\}  | \[
\begin{pmatrix}
1 & 0 \\
0 & e^{i\pi/4} 
\end{pmatrix}
\] | T    | \(T\) |
| Identity         | \{\ket{0}, \ket{1}\}  | \[
\begin{pmatrix}
1 & 0 \\
0 & 1 
\end{pmatrix}
\] | I    | \(I\) |
| Hadamard         | \{\ket{0}, \ket{1}\}  | \[
\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1 \\
1 & -1 
\end{pmatrix}
\] | H    | \(H\) |

<table>
<thead>
<tr>
<th>Type</th>
<th>Basis</th>
<th>U</th>
<th>Name</th>
<th>Sym</th>
</tr>
</thead>
</table>
| Controlled Not   | \{\ket{00}, \ket{01}\} | \[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 
\end{pmatrix}
\] | CNOT (CX)    |     |
| SWAP             | \{\ket{00}, \ket{01}, \ket{10}, \ket{11}\} | \[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 
\end{pmatrix}
\] | SWAP       |     |
| Measure          | \{\ket{0}, \ket{1}\}  | Qubit to Bit      | M                |     |
| Binary Control   | \{\ket{0}, \ket{1}\}  | Conditional Application | BC            |     |
| Restore          | \{\ket{0}, \ket{1}\}  | Bit to Qubit      | Reset            |     |
Teleport Example

- Alice Entangles two qubits
- Bob takes one of them far away
- Alice is given a new qubit with a message
- Alice entangles it with her local part of the Bell pair
- Alice measure the local qubits, yielding 2 classical bits
- Alice transmits the two bits via classical channels
- Remotely, Bob applies gates as determined by the 2 bits
- Bob recovers the sent message
Teleport: User Code

- Define a function to perform entanglement:
  ```
  let EPR (qs:Qubits) = H qs; CNOT qs
  ```

- Rest of the algorithm:
  ```
  let teleport (qs:Qubits) =
    let q0,q1,q2 = qs.[0],qs.[1],qs.[2]
    EPR[q1;q2]; CNOT qs; H qs
    M[q1]; BC X [q1;q2]
    M[q0]; BC Z [q0;q2]
  ```
Teleport: Full Circuit

```haskell
let teleport (qs:Qubits) =
  let qs' = qs.Tail
  // Skip first qubit
  Label >>= (["Src";"|0>";"|0>"],qs)
  // Label the first 3 qubits
  EPR qs'; CNOT qs; H qs
  // EPR 1,2, then CNOT 0,1 and H 0
  M qs'; BC X qs'
  // Conditionally apply X
  M qs; BC Z !!(qs,0,2)
  // Conditionally apply Z
  Label "Dest" !! (qs,2)
  // Label output

let circ2 = circ.Fold()
circ2.Render("teleport.svg")
```

![Teleport Circuit Diagram]
Teleport: Running the code

loop N times:

... create 3 qubits
... init the first one to a random state
... print it out
teleport qs
... print out the result

0:0000.0/Initial State: ( 0.3735-0.2531i)|0>+(-0.4615-0.7639i)|1>
0:0000.0/Final State: ( 0.3735-0.2531i)|0>+(-0.4615-0.7639i)|1> (bits:10)
0:0000.0/Initial State: (-0.1105+0.3395i)|0>+(-0.927-0.1146i)|1>
0:0000.0/Final State: (-0.1105+0.3395i)|0>+(-0.927-0.1146i)|1> (bits:11)
0:0000.0/Initial State: (-0.3882-0.2646i)|0>+(-0.8092+0.3528i)|1>
0:0000.0/Final State: (-0.3882-0.2646i)|0>+(-0.8092+0.3528i)|1> (bits:01)
0:0000.0/Initial State: ( 0.2336+0.4446i)|0>+(-0.8527+0.1435i)|1>
0:0000.0/Final State: ( 0.2336+0.4446i)|0>+(-0.8527+0.1435i)|1> (bits:10)
0:0000.0/Initial State: ( 0.9698+0.2302i)|0>+(-0.03692+0.0717i)|1>
0:0000.0/Final State: ( 0.9698+0.2302i)|0>+(-0.03692+0.0717i)|1> (bits:11)
0:0000.0/Initial State: (-0.334-0.3354i)|0>+(-0.315-0.8226i)|1>
0:0000.0/Final State: (-0.334-0.3354i)|0>+(-0.315-0.8226i)|1> (bits:01)
/// <summary>
/// Controlled NOT gate
/// </summary>
/// <param name="qs">Use first two qubits for gate</param>

let CNOT (qs:Qubits) =
    let gate =
        Gate.Build("CNOT", fun () ->
            new Gate(
                Name = "CNOT",
                Help = "Controlled NOT",
                Mat = CSMat(4,
                    [(0,0,1.,0.);(1,1,1.,0.);
                    (2,3,1.,0.);(3,2,1.,0.)]),
                Draw = "\ctrl{#1}\go[\#1]\targ"
            )))
    gate.Run qs
Teleport: Circuit Compilation

```plaintext
let ket = Ket(3)
let circ = Circuit.Compile teleport ket.Qubits
circ.Dump showLogInd 0
```

SEQ
- APPLY GATE H is a (Normal)  
  0.7071 0.7071 
  0.7071 0.7071 
  WIRE(Id:1)
- APPLY GATE CNOT is a Controlled NOT (Normal)  
  1 0 0 0 
  0 1 0 0 
  0 0 0 1 
  0 0 1 0 
  WIRE(Id:1) WIRE(Id:2)
- APPLY GATE CNOT is a Controlled NOT (Normal)  
  1 0 0 0 
  0 1 0 0 
  0 0 0 1 
  0 0 1 0 
  WIRE(Id:0) WIRE(Id:1)
- APPLY GATE H is a (Normal)  
  0.7071 0.7071 
  0.7071 0.7071 
  WIRE(Id:0)
```
let entangle (qs:Qubits) =
    H qs; let q0 = qs.Head
    for q in qs.Tail do CNOT[q0;q]
    M >>= qs
Quantum circuit synthesis
- Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
- Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
- Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
- Characterization of quantum state transformations using ancilla (Blass, Gurevich)
- A canonical form for \{H,T\} single-qubit circuits (Bocharov, Svore)

Quantum algorithms
- Faster phase estimation (Svore, Hastings, Freedman)
- Hubbard model (Wecker, Troyer, Hastings, Nayak, Clark)
- Quantum chemistry (Wecker, Troyer)
- Hamiltonian simulation (Wiebe, Wecker, Troyer)
- 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
- Classically simulating adiabatic algorithms (Hastings, Freedman, Troyer, Wecker)
- Computational Complexity (Hastings, Freedman)

Quantum error correction and distillation
- Noise threshold for small-distance surface codes (Tomita, Svore)
- Noise threshold for magic state distillation on topological architectures (Chen, Svore)
- State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
- Topological Computational Power (Hastings, Nayak, Freedman)

Quantum languages and platforms
- LiQUiD (Wecker, Geller, Smith, Svore, Bocharov)
- Quantum control architecture (Smith, Wecker, Geller)
- Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)
Circuit for Shor's algorithm using 2n+3 qubits – Stéphane Beauregard

Largest we’ve done: 14 bits (factoring 8193) ~1/2 Million Gates
Shor’s algorithm: Modular Adder

As defined in:

Circuit for Shor’s algorithm using 2n+3 qubits
– Stéphane Beauregard

let op (qs:Qubits) =

CCAdd a cbs  // Add a to \( \Phi|b\rangle \)
AddA' N bs   // Sub N from \( \Phi|a + b\rangle \)
QFT' bs      // Inverse QFT of \( \Phi|a + b - N\rangle \)
CNOT [bMx;anc]  // Save top bit in Ancilla
QFT bs       // QFT of a+b-N
CCAddA N (anc :: bs) // Add back N if negative
CCAdd' a cbs  // Subtract a from \( \Phi|a + b \mod N\rangle \)

QFT' bs       // Inverse QFT
X [bMx]       // Flip top bit
CNOT [bMx;anc] // Reset Ancilla to |0\rangle
X [bMx]       // Flip top bit back
QFT bs        // QFT back
CCAdd a cbs   // Finally get \( \Phi|a + b \mod N\rangle \)
### Shor’s algorithm results

<table>
<thead>
<tr>
<th>N</th>
<th>f1</th>
<th>f2</th>
<th>coPrime</th>
<th>n</th>
<th>Qubits</th>
<th>min total</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>6</td>
<td>15</td>
<td>0.12</td>
</tr>
<tr>
<td>51</td>
<td>17</td>
<td>3</td>
<td>8</td>
<td>6</td>
<td>15</td>
<td>0.09</td>
</tr>
<tr>
<td>55</td>
<td>11</td>
<td>24</td>
<td>6</td>
<td>15</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>19</td>
<td>3</td>
<td>52</td>
<td>6</td>
<td>15</td>
<td>0.11</td>
</tr>
<tr>
<td>63</td>
<td>21</td>
<td>23</td>
<td>6</td>
<td>15</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>21</td>
<td>58</td>
<td>6</td>
<td>15</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>13</td>
<td>21</td>
<td>7</td>
<td>17</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>23</td>
<td>19</td>
<td>7</td>
<td>17</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>3</td>
<td>23</td>
<td>10</td>
<td>7</td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>5</td>
<td>17</td>
<td>49</td>
<td>7</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>9</td>
<td>11</td>
<td>53</td>
<td>7</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>35</td>
<td>94</td>
<td>7</td>
<td>17</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>5</td>
<td>23</td>
<td>51</td>
<td>7</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>3</td>
<td>39</td>
<td>59</td>
<td>7</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>3</td>
<td>67</td>
<td>194</td>
<td>8</td>
<td>0.49</td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>7</td>
<td>29</td>
<td>43</td>
<td>8</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>9</td>
<td>23</td>
<td>169</td>
<td>8</td>
<td>0.77</td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>3</td>
<td>71</td>
<td>92</td>
<td>8</td>
<td>0.68</td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>3</td>
<td>73</td>
<td>7</td>
<td>8</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>3</td>
<td>79</td>
<td>211</td>
<td>8</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>7</td>
<td>35</td>
<td>57</td>
<td>8</td>
<td>0.89</td>
<td></td>
</tr>
<tr>
<td>247</td>
<td>19</td>
<td>13</td>
<td>184</td>
<td>8</td>
<td>0.89</td>
<td></td>
</tr>
<tr>
<td>249</td>
<td>3</td>
<td>83</td>
<td>29</td>
<td>8</td>
<td>0.61</td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>5</td>
<td>51</td>
<td>236</td>
<td>8</td>
<td>0.86</td>
<td></td>
</tr>
<tr>
<td>459</td>
<td>9</td>
<td>51</td>
<td>224</td>
<td>9</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>465</td>
<td>15</td>
<td>31</td>
<td>136</td>
<td>9</td>
<td>2.06</td>
<td></td>
</tr>
<tr>
<td>469</td>
<td>7</td>
<td>67</td>
<td>111</td>
<td>9</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>473</td>
<td>11</td>
<td>43</td>
<td>111</td>
<td>9</td>
<td>1.39</td>
<td></td>
</tr>
<tr>
<td>475</td>
<td>9</td>
<td>286</td>
<td>9</td>
<td>21</td>
<td>1.53</td>
<td></td>
</tr>
<tr>
<td>477</td>
<td>9</td>
<td>53</td>
<td>250</td>
<td>9</td>
<td>2.64</td>
<td></td>
</tr>
<tr>
<td>485</td>
<td>5</td>
<td>97</td>
<td>93</td>
<td>9</td>
<td>1.83</td>
<td></td>
</tr>
<tr>
<td>501</td>
<td>3</td>
<td>167</td>
<td>274</td>
<td>9</td>
<td>1.99</td>
<td></td>
</tr>
</tbody>
</table>

**Minutes to factor**

- **1 day**: $y = 1E-05e^{0.001x}$
- **3 hours**: $y = 2E-06e^{0.003x}$
- **4 days**: $y = 4E-07e^{0.005x}$
- **9 mins**: $y = 1E-04e^{0.007x}$
- **3 months**: $y = 2E-06e^{0.01x}$
- **3 years**: $y = 2E-07e^{0.015x}$

**Qubits**

- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31

**Graph**

- 1.0e+7
- 1.0e+6
- 1.0e+5
- 1.0e+4
- 1.0e+3
- 1.0e+2
- 1.0e+1
- 1.0e+0
- 1.0e-1
- 1.0e-2
- 1.0e-3
- 1.0e-4
- 1.0e-5
- 1.0e-6
- 1.0e-7
- 1.0e-8
- 1.0e-9
- 1.0e-10
- 1.0e-11
- 1.0e-12

- 1 day
- 3 hours
- 4 days
- 3 months
- 9 mins
- 3 years
QuArC Areas of Research

• Quantum circuit synthesis
  • Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  • Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  • Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  • Characterization of quantum state transformations using ancilla (Blass, Gurevich)
  • A canonical form for \{H,T\} single-qubit circuits (Bocharov, Svore)

• Quantum algorithms
  • Faster phase estimation (Svore, Hastings, Freedman)
  • Hubbard model (Wecker, Troyer, Hastings, Nayak, Clark)
  • Quantum chemistry (Wecker, Troyer)
  • Hamiltonian simulation (Wiebe, Wecker, Troyer)
  • 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
  • Classically simulating adiabatic algorithms (Hastings, Freedman)
  • Computational Complexity (Hastings, Freedman)

• Quantum error correction and distillation
  • Noise threshold for small-distance surface codes (Tomita, Svore)
  • Noise threshold for magic state distillation on topological architectures (Chen, Svore)
  • State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
  • Topological Computational Power (Hastings, Nayak, Freedman)

• Quantum languages and platforms
  • LiQUi\d (Wecker, Geller, Smith, Svore, Bocharov)
  • Quantum control architecture (Smith, Wecker, Geller)
  • Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)
QECC Gate Definition: Steane7 prep

/// <summary>
/// Steane7 prep gate (create logical |0>).
/// </summary>
/// <param name="qs">Physical qubits to use</param>

let prep (qs:Qubits) =
  let nam = "S7_Prep"
  let nam2 = "S7\nPrep"
  let gate (qs:Qubits) =
    // Create logical |0> prep circuit
    let op (qs:Qubits) =
      let xH i = H [qs.[i]]
      let xc i j = CNOT [qs.[i];qs.[j]]
      xH 6; xc 6 3; xH 5; xc 5 2; xH 4
      xc 4 1; xc 5 3; xc 4 2; xc 6 0; xc 6 1
      xc 5 0; xc 4 3
    Gate.Build(nam, fun () ->
      new Gate(
        Qubits = qs.Length,
        Name = nam,
        Help = "Prepare logical 0 state",
        Draw = sprintf "\multigate{#d}{%s}"
          (qs.Length-1) nam,
        Op = WrapOp op
      )
    )
  (gate qs).Run qs
QECC Gate Definition: Steane7 syndrome
Full Teleport Circuit in a Steane7 Code

Encode
Syndrome 1
Syndrome 2
Syndrome 3

Hadamard
CNOT
Measure
CNOT
Stabilizers

```ocaml
let tele1 (qs:Qubits) = X qs; teleport qs; M [qs.[2]]
let tgtC1 = Circuit.Compile tele1 qs
let s7 = Steane7(tgtC1)
let s7C = s7.Circuit
let stab = Stabilizer(s7C,s7.Ket)
stab.Run()
let bit0,dist0  = s7.Log2Phys 0 |> s7.Decode
let bit1,dist1  = s7.Log2Phys 1 |> s7.Decode
let bit2,dist2  = s7.Log2Phys 2 |> s7.Decode
```

---

Final Tableau:
(after Gaussian)

```
25
- X................ZZZ....ZZZ
- XX................Z......Z.
- XXX................Z......Z
- X..X............ZZZZ....ZZZ
+..X..X.........Z..Z.Z....Z.Z
+..XX.XX.........Z.ZZ.....ZZ.
+..X..XX...........ZZ.....ZZ
+..X..X..X.........Z.Z....Z.Z
+..XX.XX..X........ZZ.....ZZ.
- X..X.....X.......ZZZ....ZZZ
- X.XX.X....X......Z......Z..
- XX.XX......X......Z......Z.
- XXXXX......X......Z......Z.
+..X.........X............
+..X..ZZZZ..ZZ..X............
+..XX..ZZ.Z.Z.Z..X...........
+X..Z.....ZZZZ...X..........
- YZX.............ZZZY........
- YZZ.........ZZZZ........
- Z....................X......
- Z.....................X....
- Z......................X....
- Z.......................X...
- ZZ...............ZZZZ...XXXX..
- ZZ.............ZZ..ZZ.XX..XX.
- ZZ...........ZZ.Z.Z.ZX.X.X.
----------------------------
+ZZ.ZZ.........ZZZZ........
+ZZ.ZZ.......ZZ..ZZ........
+..Z..Z.......Z.Z.Z.Z.......
+...Z.....ZZZZ..............
+....ZZZZ..ZZ...............
+.....ZZ.Z.Z.Z..............
+......Z....................
+.......Z...................
+........Z..................
+.........Z................
+..........Z................
+...........Z...............
+............Z..............
+.............Z.............
+..............Z............
- Z.........................ZZ
- ZX......................ZZ
- YXZ..........Z.ZZ.Y........
- YYY..........ZZ.Z..Y.......
- X....................X......
+..X.............X............
+..X..ZZZZ..ZZ..X............
+..XX..ZZ.Z.Z.Z..X...........
+X..Z.....ZZZZ...X..........
```
QuArC Areas of Research

• Quantum circuit synthesis
  • Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  • Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  • Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  • Characterization of quantum state transformations using ancilla (Blass, Gurevich)
  • A canonical form for \{H,T\} single-qubit circuits (Bocharov, Svore)

• Quantum algorithms
  • Faster phase estimation (Svore, Hastings, Freedman)
  • Hubbard model (Wecker, Troyer, Hastings, Nayak, Clark)
  • Quantum chemistry (Wecker, Troyer)
  • Hamiltonian simulation (Wiebe, Wecker, Troyer)
  • 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
  • Classically simulating adiabatic algorithms (Hastings, Freedman, Troyer, Wecker)
  • Computational Complexity (Hastings, Freedman)

• Quantum error correction and distillation
  • Noise threshold for small-distance surface codes (Tomita, Svore)
  • Noise threshold for magic state distillation on topological architectures (Chen, Svore)
    - State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
    - Topological Computational Power (Hastings, Nayak, Freedman)

• Quantum languages and platforms
  • LiQUiD (Wecker, Geller, Smith, Svore, Bocharov)
  • Quantum control architecture (Smith, Wecker, Geller)
  • Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)
Advanced Noise Modeling

```
Noise(circ:Circuit, ket:Ket, models: NoiseModels)
type NoiseModel = {
  gate: string // Gate name (ending with "*" for wildcard match)
  maxQs: int // Max qubits that gate uses
  time: float // floating duration of gate (convention Idle = 1.0)
  func: NoiseFunc // Noise Model to execute
  gateEvents: NoiseEvents // Stats for normal gates
  ecEvents: NoiseEvents // Stats for EC gates
}
member n.DampProb // Get/Set damping probability on a qubit
```

Two Qubits H01N01

- **Unitary Noise**
- **Non-Unitary Noise**

<table>
<thead>
<tr>
<th>State</th>
<th>Measurement Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0.2</td>
</tr>
<tr>
<td>01</td>
<td>0.4</td>
</tr>
<tr>
<td>10</td>
<td>0.6</td>
</tr>
<tr>
<td>11</td>
<td>0.8</td>
</tr>
</tbody>
</table>

- Teleport, Steane7
  - P_idle = P/10
  - P_damp = P/10

![Graph showing measurement probabilities over time](image)

- [Graph showing error probability vs. percentage correct](image)
QuArC Areas of Research

- **Quantum circuit synthesis**
  - Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  - Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  - Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  - Characterization of quantum state transformations using ancilla (Blass, Gurevich)
  - A canonical form for \{H,T\} single-qubit circuits (Bocharov, Svore)

- **Quantum algorithms**
  - Faster phase estimation (Svore, Hastings, Freedman)
  - Hubbard model (Wecker, Troyer, Hastings, Nayak, Clark)
  - Quantum chemistry (Wecker, Troyer)
  - Hamiltonian simulation (Wiebe, Wecker, Troyer)
  - 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
  - Classically simulating adiabatic algorithms (Hastings, Freedman)
  - Computational Complexity (Hastings, Freedman)

- **Quantum error correction and distillation**
  - Noise threshold for small-distance surface codes (Tomita, Svore)
  - Noise threshold for magic state distillation on topological architectures (Chen, Svore)
  - State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
  - Topological Computational Power (Hastings, Nayak, Freedman)

- **Quantum languages and platforms**
  - LiQUiD (Wecker, Geller, Smith, Svore, Bocharov)
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)
Approximation of Quantum Circuits

• Some gates may not be available in a fault-tolerant physical implementation.

• Need to efficiently decompose a quantum circuit into fault-tolerant, implementable gates.

\[ R_{64} = HTHTHSHTHTHTHTHTHTSHTHTHTHTHTHTHTHTHTHTHTHTHSHS \]
Efficient Decomposition of Single-Qubit Gates into $V$ Basis Circuits

Alex Bocharov, 1 * Yuri Gurevich, 2 Y and Krysta M. Svore 1, 4
1 Quantum Architectures and Computation Group, Microsoft Research, Redmond, WA 98052 USA
2 Research In Software Engineering Group, Microsoft Research, Redmond, WA 98052 USA

We develop the first constructive algorithms for compiling single-qubit unitary gates into circuits over the universal $V$ basis. The $V$ basis is an alternative universal basis to the more commonly studied $\{H,T\}$ basis. We propose two classical algorithms for quantum circuit compilation: the first algorithm has expected polynomial time (in precision $\log(1/\epsilon)$) and offers a depth/precision guarantee that improves upon state-of-the-art methods for compiling into the $\{H,T\}$ basis by factors ranging from 1.86 to $\log_2(5)$. The second algorithm is analogous to direct search and yields circuits a factor of 3 to 4 times shorter than our first algorithm, and requires time exponential in $\log(1/\epsilon)$; however, we show that in practice the runtime is reasonable for an important range of target precisions.

PACS numbers: 03.67.Lx, 03.65.Fd
Keywords: quantum gate decomposition, quantum compilation

http://arxiv.org/abs/1303.1411
Single-qubit Gate Decomposition

- Standard basis of decomposition is $\{T, \text{Clifford}\}$ a.k.a. $\{H, T\}$
- Consider the efficiently universal basis $\{V, \text{Clifford}\}$
  \[ V = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 + 2i & 0 \\ 0 & 1 - 2i \end{bmatrix} \]
- $V$-basis offers 25 - 75% improvement over $\{H, T\}$ in T count
- $V$-basis offers 10x improvement if $V$ is a physical gate

QuArC Areas of Research

• Quantum circuit synthesis
  • Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  • Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  • Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  • Characterization of quantum state transformations using ancilla (Blass, Gurevich)
  • A canonical form for \{H,T\} single-qubit circuits (Bocharov, Svore)

• Quantum algorithms
  • Faster phase estimation (Svore, Hastings, Freedman)
  • Hubbard model (Wecker, Troyer, Hastings, Nayak, Clark)
  • Quantum chemistry (Wecker, Troyer)
  • Hamiltonian simulation (Wiebe, Wecker, Troyer)
  • 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
  • Classically simulating adiabatic algorithms (Hastings, Freedman)
  • Computational Complexity (Hastings, Freedman)

• Quantum error correction and distillation
  • Noise threshold for small-distance surface codes (Tomita, Svore)
  • Noise threshold for magic state distillation on topological architectures (Chen, Svore)
  • State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
  • Topological Computational Power (Hastings, Nayak, Freedman)

• Quantum languages and platforms
  • LiQUiD (Wecker, Geller, Smith, Svore, Bocharov)
  • Quantum control architecture (Smith, Wecker, Geller)
  • Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)
Faster Phase Estimation

Krysta M. Svore,† Matthew B. Hastings,‡ and Michael Freedman§

1 Quantum Architectures and Computation Group
Microsoft Research, Redmond, WA 98052 USA
2 Station Q, Microsoft Research, Santa Barbara, CA 93106 USA
(Dated: April 3, 2013)

We develop several algorithms for performing quantum phase estimation based on basic measurements and classical post-processing. We present a pedagogical review of quantum phase estimation and simulate the algorithm to numerically determine its scaling in circuit depth and width. We show that the use of purely random measurements requires a number of measurements that is optimal up to constant factors, albeit at the cost of exponential classical post-processing; the method can also be used to improve classical signal processing. We then develop a quantum algorithm for phase estimation that yields an asymptotic improvement in runtime, coming within a factor of log* of the minimum number of measurements required while still requiring only minimal classical post-processing. The corresponding quantum circuit requires asymptotically lower depth and width (number of qubits) than quantum phase estimation.

PACS numbers: 03.67.Lx, 03.65.Fd
Keywords: quantum phase estimation, inference

http://arxiv.org/abs/1304.0741
Faster Phase Estimation

- Use basic measurement and classical post-processing
- Perform inference across multiple qubits
- Technique requires asymptotically fewer measurements than Kitaev-style phase estimation
- Achieve within a factor of $\log^*$ of the minimum number of measurements required (with minimal classical post-processing)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Depth</td>
<td>Width</td>
</tr>
<tr>
<td>Sequential</td>
<td>$O(m \log^*(m))$</td>
<td>$O(m \log^*(m))$</td>
</tr>
<tr>
<td>Parallel</td>
<td>$O(m \log^*(m))$</td>
<td>$O(m \log^*(m))$</td>
</tr>
<tr>
<td>Cluster</td>
<td>$O(m \log^*(m))$</td>
<td>$O(m \log^*(m))$</td>
</tr>
</tbody>
</table>
QuArC Areas of Research

- Quantum circuit synthesis
  - Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  - Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  - Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  - Characterization of quantum state transformations using ancilla (Blass, Gurevich)
  - A canonical form for \{H,T\} single-qubit circuits (Bocharov, Svore)

- Quantum algorithms
  - Faster phase estimation (Svore, Hastings, Freedman)
  - Hubbard model (Wecker, Troyer, Hastings, Nayak, Clark)
  - Quantum chemistry (Wecker, Troyer)
  - Hamiltonian simulation (Wiebe, Wecker, Troyer)
  - 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
  - Classically simulating adiabatic algorithms (Hastings, Freedman, Troyer, Wecker)
  - Computational Complexity (Hastings, Freedman)

- Quantum error correction and distillation
  - Noise threshold for small-distance surface codes (Tomita, Svore)
  - Noise threshold for magic state distillation on topological architectures (Chen, Svore)
  - State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
  - Topological Computational Power (Hastings, Nayak, Freedman)

- Quantum languages and platforms
  - LiQUiD (Wecker, Geller, Smith, Svore, Bocharov)
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)
Quantum annealing with more than one hundred qubits

Sergio Boixo,¹ Troels F. Rønnow,² Sergei V. Isakov,² Zhihui Wang,³ David Wecker,⁴ Daniel A. Lidar,⁵ John M. Martinis,⁶ and Matthias Troyer*²

¹ Information Sciences Institute and Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089, USA
² Theoretische Physik, ETH Zurich, 8093 Zurich, Switzerland
³ Department of Chemistry and Center for Quantum Information Science & Technology, University of Southern California, Los Angeles, California 90089, USA
⁴ Quantum Architectures and Computation Group, Microsoft Research, Redmond, WA 98052, USA
⁵ Departments of Electrical Engineering, Chemistry and Physics, and Center for Quantum Information Science & Technology, University of Southern California, Los Angeles, California 90089, USA
⁶ Department of Physics, University of California, Santa Barbara, CA 93106-9530, USA

http://arxiv.org/abs/1304.4595
QuArC Areas of Research

• Quantum circuit synthesis
  • Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  • Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  • Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  • Characterization of quantum state transformations using ancilla (Blass, Gurevich)
  • A canonical form for {H,T} single-qubit circuits (Bocharov, Svore)

• Quantum algorithms
  • Faster phase estimation (Svore, Hastings, Freedman)
  • Hubbard model (Wecker, Troyer, Hastings, Nayak, Clark)
  • Quantum chemistry (Wecker, Troyer)
  • Hamiltonian simulation (Wiebe, Wecker, Troyer)
  • 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
  • Classically simulating adiabatic algorithms (Hastings, Freedman, Troyer, Wecker)
  • Computational Complexity (Hastings, Freedman)

• Quantum error correction and distillation
  • Noise threshold for small-distance surface codes (Tomita, Svore)
  • Noise threshold for magic state distillation on topological architectures (Chen, Svore)
  • State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
  • Topological Computational Power (Hastings, Nayak, Freedman)

• Quantum languages and platforms
  • LiQUiD (Wecker, Geller, Smith, Svore, Bocharov)
  • Quantum control architecture (Smith, Wecker, Geller)
  • Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)
Hamiltonians (Fermion Model)

\[ H = \sum_{pq} h_{pq} a^+_p a_q + \frac{1}{2} \sum_{pqrs} h_{pqrs} a^+_p a^+_q a_r a_s \]

- Very interesting molecules can be modeled with high fidelity in less than 1000 qubits
- Opportunity to design new materials, including ones for next generation QC (bootstrapping)
- Fits naturally into the gate and circuit model that we’re using

// Invoke by picking which test to run:
// Liquid /s H2O.fsx Main(245)

module Script =  // The script module allows for incremental loading

let dic = Dictionary<string,string>()  // Parameters to Fermion

dic.["Test"] <- "245"  // Test to process
dic.["Bits"] <- "20"  // Bit accuracy
dic.["Trotter"] <- "128"  // Trotter number
dic.["Thresh"] <- "-83.7"  // Max threshold to accept as an answer (w/o NR)
dic.["Emin"] <- "-85.1"  // Min possible energy (without nuclear repulsion)
dic.["Emax"] <- "-35.0"  // Max possible energy (without nuclear repulsion)
dic.["Ecnt"] <- "10"  // Electron count
dic.["Sos"] <- "14"  // Spin orbitals
dic.["Parity"] <- "1"  // Enforce parity between rows and columns?
dic.["Diff"] <- "0"  // Spin Up/Down enforced difference (default is none)
dic.["HalfUp"] <- "0"  // These are interleaved
dic.["Single"] <- "1"  // Use single Unitary?
dic.["Preps"] <- "[1;2;3;4;5;6;7;8;9;10]"  // Prepared states to start in (list of lists)
dic.["Alter"] <- "0.0"  // Alter angle by factor (0.0< "don't alter"
dic.["AltCnt"] <- "1"  // Count of altered circuits to use

let data = [|
  "tst=0 info=95.5,1.820 nuc=9.162349762 Ehf=74.962999077 00=-32.696652545 ...|
]
Water

### Full Configuration Interaction

\( |\text{LIQUid}\rangle\)

#### Ground State

<table>
<thead>
<tr>
<th></th>
<th>RHF</th>
<th>DFT</th>
<th>LIQUid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Egs</td>
<td>-74.965832</td>
<td>-76.399089</td>
<td>-74.980538</td>
</tr>
<tr>
<td>Angle</td>
<td>100.5</td>
<td>102</td>
<td>101.5</td>
</tr>
<tr>
<td>Bond Len</td>
<td>1.86</td>
<td>1.84</td>
<td>1.87</td>
</tr>
</tbody>
</table>

- Conserve Electrons
- Conserve Total Spin
- Up spins = Down spins
- \(2^{15} \times 2^{15} \rightarrow 441 \times 441\)
- \(2^k = k\) matrix multiples

34,000 gates \(\times 2^{14}\) bits \(\times 2^{10}\) Trotter \(\times 50\) samples \(\times 546\) pts = \(1.5 \times 10^{16}\) gate ops
## Multi-Resolution Trotterization

Towards realistic quantum algorithms: the case of quantum chemistry

Dave Wecker,1 Bela Bauer,2 Bryan K. Clark,2 and Matthias Troyer3

1Quantum Architectures and Computation Group, Microsoft Research, Redmond, WA 98052, USA
2Station Q, Microsoft Research, Santa Barbara, CA 93106-6105, USA
3Theoretische Physik, ETH Zurich, 8093 Zurich, Switzerland

<table>
<thead>
<tr>
<th>Molecule</th>
<th>Basis</th>
<th>Spin orbitals</th>
<th>Basis size</th>
<th>Sequential Rotations</th>
<th>Sequential Total</th>
<th>Parallel Rotations</th>
<th>Parallel Total</th>
<th>Parallel adaptive Rotations</th>
<th>Parallel adaptive Total</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2O</td>
<td>STO3G</td>
<td>14</td>
<td>441</td>
<td>1615</td>
<td>20494</td>
<td>1615</td>
<td>6438</td>
<td>338</td>
<td>810</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>P321</td>
<td>26</td>
<td>1.66 × 10^6</td>
<td>16253</td>
<td>381208</td>
<td>16253</td>
<td>72536</td>
<td>1005</td>
<td>2805</td>
<td>0.039</td>
</tr>
<tr>
<td></td>
<td>DZVP</td>
<td>38</td>
<td>1.35 × 10^8</td>
<td>63863</td>
<td>2124678</td>
<td>63863</td>
<td>295430</td>
<td>1794</td>
<td>4859</td>
<td>0.016</td>
</tr>
<tr>
<td></td>
<td>P631SS</td>
<td>50</td>
<td>2.8 × 10^9</td>
<td>177205</td>
<td>7694840</td>
<td>177205</td>
<td>837480</td>
<td>3134</td>
<td>8728</td>
<td>0.010</td>
</tr>
<tr>
<td></td>
<td>P631PPSS</td>
<td>54</td>
<td>6.5 × 10^9</td>
<td>244123</td>
<td>11404322</td>
<td>244123</td>
<td>1159082</td>
<td>6995</td>
<td>19736</td>
<td>0.017</td>
</tr>
<tr>
<td></td>
<td>P6311SS</td>
<td>62</td>
<td>2.9 × 10^10</td>
<td>419219</td>
<td>22330186</td>
<td>419219</td>
<td>2003978</td>
<td>6308</td>
<td>18465</td>
<td>0.009</td>
</tr>
<tr>
<td>CO2</td>
<td>STO3G</td>
<td>30</td>
<td>1.86 × 10^6</td>
<td>19639</td>
<td>531926</td>
<td>19639</td>
<td>89062</td>
<td>1550</td>
<td>3713</td>
<td>0.042</td>
</tr>
<tr>
<td></td>
<td>P321</td>
<td>54</td>
<td>1.70 × 10^14</td>
<td>134231</td>
<td>6187878</td>
<td>134231</td>
<td>630006</td>
<td>2750</td>
<td>7496</td>
<td>0.012</td>
</tr>
<tr>
<td></td>
<td>DZVP</td>
<td>90</td>
<td>1.03 × 10^20</td>
<td>1394669</td>
<td>106047976</td>
<td>1394669</td>
<td>6791752</td>
<td>8036</td>
<td>22918</td>
<td>0.0033</td>
</tr>
<tr>
<td></td>
<td>P631SS</td>
<td>90</td>
<td>1.03 × 10^20</td>
<td>1023013</td>
<td>76179240</td>
<td>1023013</td>
<td>4950192</td>
<td>7181</td>
<td>19959</td>
<td>0.0040</td>
</tr>
<tr>
<td>Fe2S2</td>
<td>STO3G</td>
<td>112</td>
<td>3.4 × 10^25</td>
<td>7441260</td>
<td>630767773</td>
<td>7441260</td>
<td>35865821</td>
<td>3220</td>
<td>8763</td>
<td>0.00024</td>
</tr>
<tr>
<td></td>
<td>P321</td>
<td>168</td>
<td>2.8 × 10^48</td>
<td>45074552</td>
<td>5840151333</td>
<td>45074552</td>
<td>220771169</td>
<td>12571</td>
<td>37489</td>
<td>0.00017</td>
</tr>
</tbody>
</table>
QuArC Areas of Research

- Quantum circuit synthesis
  - Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  - Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  - Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  - Characterization of quantum state transformations using ancilla (Blass, Gurevich)
  - A canonical form for \{H,T\} single-qubit circuits (Bocharov, Svore)

- Quantum algorithms
  - Faster phase estimation (Svore, Hastings, Freedman)
  - **Hubbard model** (Wecker, Troyer, Hastings, Nayak, Clark)
  - Quantum chemistry (Wecker, Troyer)
  - Hamiltonian simulation (Wiebe, Wecker, Troyer)
  - 2D nearest-neighbor architecture to factor in polylog depth (Pham, Svore)
  - Classically simulating adiabatic algorithms (Hastings, Freedman, Troyer, Wecker)
  - Computational Complexity (Hastings, Freedman)

- Quantum error correction and distillation
  - Noise threshold for small-distance surface codes (Tomita, Svore)
  - Noise threshold for magic state distillation on topological architectures (Chen, Svore)
  - State distillation protocol to implement single-qubit gates (Duclos-Cianci, Svore)
  - Topological Computational Power (Hastings, Nayak, Freedman)

- Quantum languages and platforms
  - LiQUi\(\text{D}\) (Wecker, Geller, Smith, Svore, Bocharov)
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)


Adiabatic Solution of the Hubbard model

\[ H = -\sum_{\langle i,j \rangle} \sum_\sigma t_{ij}(c_{i,\sigma}^\dagger c_{j,\sigma} + c_{j,\sigma}^\dagger c_{i,\sigma}) + U \sum_i n_{i,\uparrow} n_{i,\downarrow} + \sum_i \epsilon_i n_i \]

See: d-wave resonating valence bond states of fermionic atoms in optical lattices
Thank You!
QuArC Research Activities

Quantum Algorithms

Quantum Circuits

\text{LIQUiD}\rangle
A little bit of F# syntax

- Parentheses are just for association (they don’t mean function call)
- Arrays and lists are accessed with: A.[i]
- Lists have square brackets: [1;2;3;4]
  - arrays have fence posts [[1;2;3;4]]
- Function calls use white space between the arguments: F a b c
  - F(a,b,c) calls F with a single tuple argument
- Output can be “piped” between functions: F a b |> G
- An empty argument can be denoted by: ()
- Some LIQUi|)] specific operators:
  - Complex operators: -(negative), ^~ (conjugate), +, -, *
  - Multiply (VV, MV, MM): *, Kronecker Product (VV or MM): *!
  - Map a gate to a list of qubits: >>= with a parameter: >!<
  - Convert any legal object to list of qubits: !!
#if INTERACTIVE
#r "Liquid.dll"
#else
namespace Microsoft.Research.Liquid // For incremental compile
#endif

open System // Open any support libraries
open Microsoft.Research.Liquid // Get necessary Liquid libraries
open Util // General utilities
open Operations // Basic gates and operations

module Script = // Incremental loaded

[<LQD>]
let Main() = // Callable from the command line
   . . . // Name of callable function

#if INTERACTIVE
do Script.Main() // Default start routine
#endif

May be run in several ways:

- **fsi script.fsx** Executes default start routine and exits
- **fsi --use:script.fsx** Executes default start routine and stays in the interpreter
- **Liquid /s script.fsx Main** compiles the script into LIQUi|δ and call Main()
- **Liquid /l script.dll Main** load previously compiled script into LIQUi|δ and call Main()
Library of Gates

- **Standard 1 Qubit**: H: Hadamard, S: Phase, X,Y,Z,I: Pauli, T: $\frac{\pi}{8}$
- **Parameterized 1 Qubit**: R: $\frac{2\pi}{2^k}$ Rotation, Label
- **Standard 2 Qubit**: CNOT: Controlled Not, SWAP
- **Parameterized 2 Qubit**: U: Eigen measure
- **Standard 3 Qubit**: CCNOT: Toffoli
- **Pseudo 1 Qubit**: M: measure, Reset, Restore, Native
- **Meta**: BC: Binary Control, Adj: Adjoint, Cgate, CCgate, Wrap: High level gate, Transverse: QECC, Hamiltonian Gates
- **Note**:
  - ✓ None are “baked in”
  - ✓ User extensible

M qs; BC (Adj T) qs
// Parallel entanglement
let entangle (qs:Qubits) =

  // Do the heads of the groups
  H qs'
  for i in 5..qs'.Length-1 do CNOT !!(qs',0,i)

  // Do each group of upto 4
  let rec doGroup (qs:Qubits) =
    let len = qs.Length
    if len >= 2 then
      let lst = if len <= 5 then len-1 else 4
      for i in [1..lst] do CNOT !!(qs,0,i)
      Seq.skip (lst+1) qs |> Seq.toList |> doGroup
    doGroup qs'

  // Measure all the qubits
  M << qs'

0:0000.0/=============== Entangle 20 qubits, 10 times ===============
0:0000.0/Starting 20 qubits...
0:0000.0/  Entangle Mem: Priv= 110/ 585 MB WS= 59/ 59 MB
0:0000.0/  Measure Mem: Priv= 78/ 585 MB WS= 27/ 60 MB
0:0000.0/#### Iter 0 [ 2.1942]: 00000000000000000000
0:0000.1/#### Iter 1 [ 2.1146]: 00000000000000000000
0:0000.1/#### Iter 2 [ 2.0675]: 00000000000000000000
0:0000.1/#### Iter 3 [ 2.0585]: 11111111111111111111
0:0000.2/#### Iter 4 [ 2.0419]: 00000000000000000000
0:0000.2/#### Iter 5 [ 2.0353]: 11111111111111111111
0:0000.2/#### Iter 6 [ 2.0639]: 11111111111111111111
0:0000.3/#### Iter 7 [ 2.0638]: 00000000000000000000
0:0000.3/#### Iter 8 [ 2.0618]: 00000000000000000000
0:0000.4/#### Iter 9 [ 2.0676]: 11111111111111111111
0:0000.4/Got 4 ones (40.000%)
Simulating Water on a LAN or Cluster

Run with:
Liquid /e H2O

```xml
<?xml version="1.0" encoding="utf-8"?>
<Ensemble Default="H2O">
  <Pars>
    <Exe>\machine-00\Liquid\Liquid.exe</Exe>
    <Host>machine-00</Host>
    <Host>machine-01</Host>
    <Host>machine-02</Host>
    <Host>machine-03</Host>
    <Host>machine-04</Host>
    <Host>machine-05</Host>
    <Host>machine-06</Host>
    <Host>machine-07</Host>
    <Host>machine-08</Host>
    <Host>machine-09</Host>
  </Pars>
  <H2O Count="546" Script="\machine-00\Liquid\H2O.fsx">
    <Cmd Range="0,1,545">Main(%d)</Cmd>
  </H2O>
</Ensemble>
```
1000 Runs of 22 Entangled Qubits in 3 ½ minutes

1: 00:00.4/*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48 ( 59 96.0%)
2: 00:03.3/*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48*48 ( 59 96.0%)
b:0003.4/ 48 48 48 48 48 48 48*49 48 48*50 48 48 48*49 48 49 48 49 48 49 48 49 48 49 48   (  7 96.7%)
c:0003.5/==== 1000 Runs of 22 qubits each across 60 silos with 8 threads each =====
c:0003.5/Final tally: All 0s = 489 (48.90%), All 1s = 511 (51.10%)
Hamiltonians (Spin Model)

$$H(t) = \Gamma(t) \sum_{i=1}^{N} \Delta_i \sigma_i^x + \Lambda(t) \left( \sum_{i=1}^{N} h_i \sigma_i^z + \sum_{i,j=1}^{N} J_{ij} \sigma_i^z \sigma_j^z \right)$$

- Built-in Hamiltonian simulator for doing spin-glass models
- Able to reproduce ground states for published D-Wave examples
- Built-in test for doing ferromagnetic chains
- Here’s what the circuit looks like...
- Just added a decoherence model and entanglement entropy measurement (thanks to Andrew Das Sarma)

in conjunction with Matthias Troyer, ETH Zurich
Molecular Hydrogen Circuit ($H_2$)

Quantum Chemistry Simulation Results

- Output generated on cluster of 20 machines in ~1 minute

Water

- Simple STO-3g basis (for comparison with simulation)
Machine Learning - Traveling Salesman

- Base encoding of the problem as a Hamiltonian on edges:
  - \( H = H_l + \eta H_c = \sum_e (d_e - 6\eta) a_e + \eta \sum_{e,e'} c_{e,e'} a_e a_{e'} + H_4 \)

- Higher order constraints are needed to prevent disconnected routes
- Simulator does 8 cities (28 qubits) and solves for the optimal route
- Adding more sophisticated constraints (e.g., edge pair flips):

\[
H_4 = \sum_{i \neq j \neq k \neq l} \sigma_{ij} \sigma_{kl} \sigma_{ik} \sigma_{jl}
\]

~50% reduction in annealing time
Quantum Walks (PageRank example)

• Start with a standard stochastic probability matrix for PageRank \((G)\)
• Define a Hamiltonian: \(\mathcal{H} = (\mathbb{I} - G)^\dagger (\mathbb{I} - G)\)
• Convert to a Unitary: \(U = e^{-i\mathcal{H}}\)
• Evolve from a starting state of the static probabilities (or perform an adiabatic evolution in a 2\(^{nd}\) quantized form)
• Accumulate average probabilities of evolving state vector
• Example: Synthetic web graph (recursive matrix definition) of 256 pages takes 8 qubits
QUANTUM ARCHITECTURES AND COMPUTATION

LIQUiQC Users Manual

DATA TYPES

LIQUiQC relies on a basic set of data types to model the quantum circuit. The

LIQUiQC format is a text-based language that allows for the creation of quantum circuits. The LIQUiQC format is described in the following section.

Properties

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.

LIQUiQC is a language for specifying quantum circuits. It is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is based on the Quantum Circuit Description Language (QCDL), which is a simple and expressive language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits. The LIQUiQC language is designed to be easy to use and understand, and it is used in the LIQUiQC language for specifying quantum circuits.