Microsoft

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

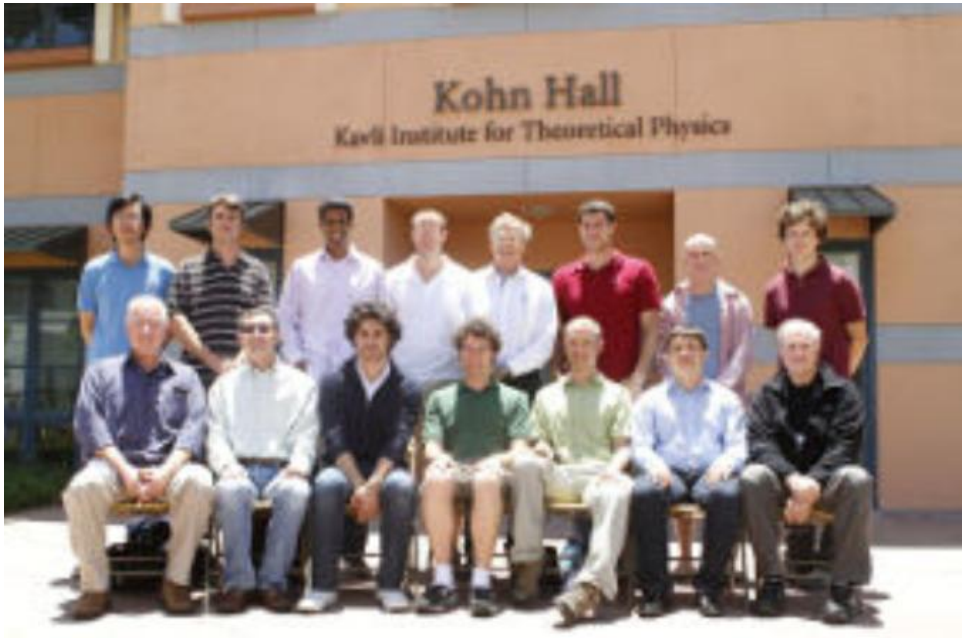
Matthias Troyer

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

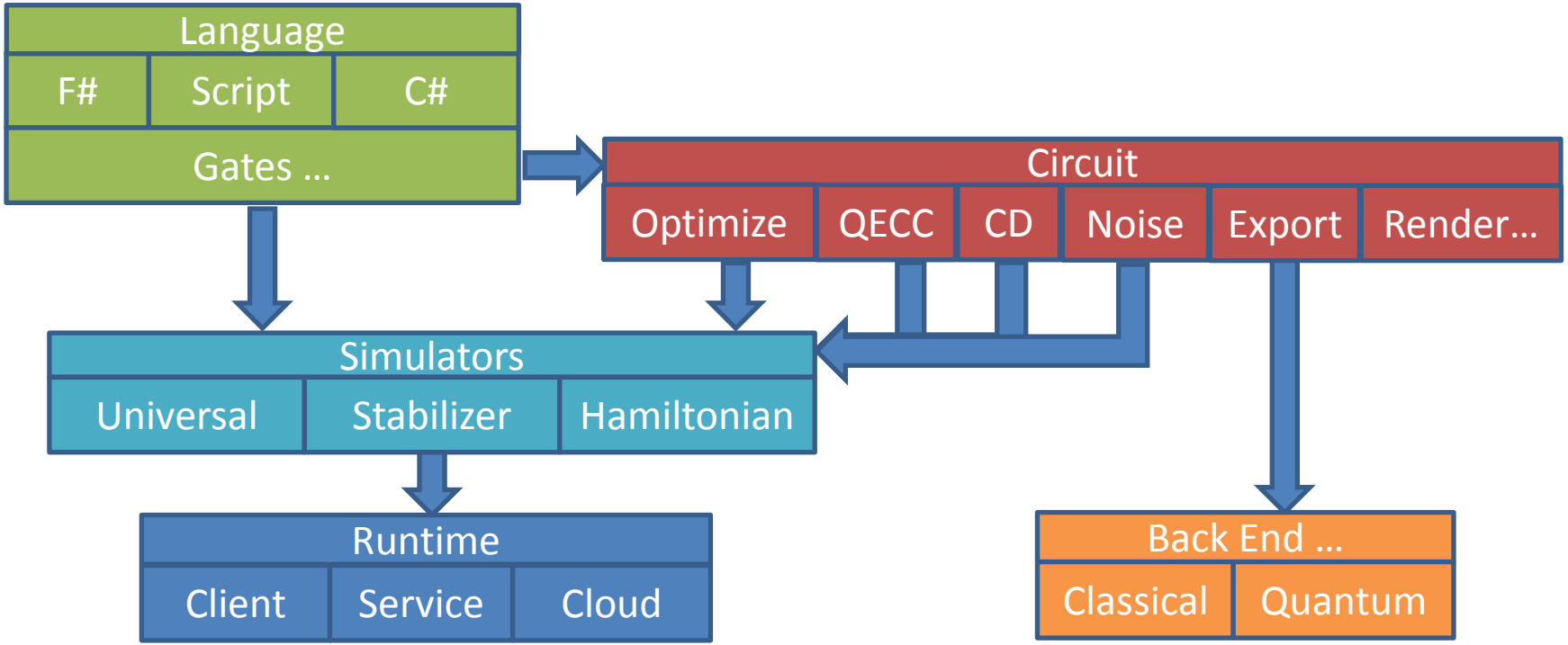
QuArC Areas of Research

- 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**
  - **LIQ$Ui|\rangle$ (Wecker, Geller, Smith, Svore, Bocharov)**
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)

LIQ$Ui|\rangle$ 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

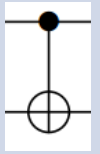
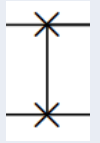
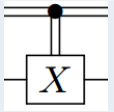
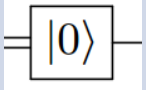
The LIQ$Ui|\rangle$ Platform



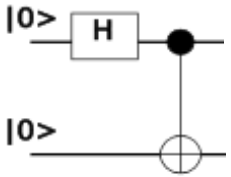
Quantum Gates

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

Type	Basis	U	Name	Sym
Pauli	$\{|0\rangle, |1\rangle\}$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	X	X
	$\{|0\rangle, |1\rangle\}$	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	Y	Y
Z Rotation	$\{|0\rangle, |1\rangle\}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	Z	Z
$e^{i\pi/2}$	$\{|0\rangle, |1\rangle\}$	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	S	S
	$\{|0\rangle, |1\rangle\}$	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$	T	T
	$\{|0\rangle, |1\rangle\}$	$\begin{bmatrix} 0 & 1 \\ 1 & e^{i\pi/8} \end{bmatrix}$	R4	$R4$
Identity	$\{|0\rangle, |1\rangle\}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	I	I
Hadamard	$\{|0\rangle, |1\rangle\}$	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	H	H

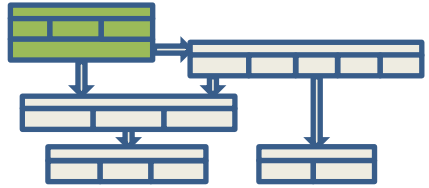
Type	Basis	U	Name	Sym
Controlled Not	$\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	CNOT (CX)	
	$\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	SWAP	
Measure	$\{|0\rangle, |1\rangle\}$	Qubit to Bit	M	
Binary Control	$\{|0\rangle, |1\rangle\}$	Conditional Application	BC	
Restore	$\{|0\rangle, |1\rangle\}$	Bit to Qubit	Reset	

Teleport Example



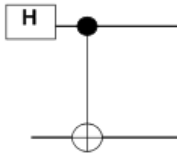
- Alice Entangles two qubits
- Bob takes one of them far away
- Alice is given a new qubit with a message
- Alice entangles it with her local part of the Bell pair
- Alice measure the local qubits, yielding 2 classical bits
- Alice transmits the two bits via classical channels
- Remotely, Bob applies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

Teleport: User Code



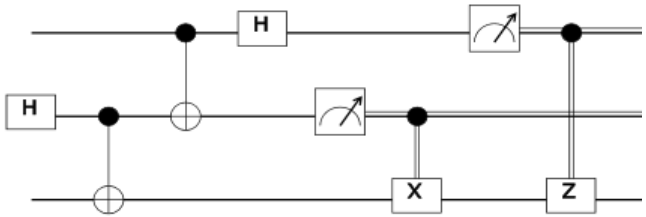
- Define a function to perform entanglement:

```
let EPR (qs:Qubits) = H qs; CNOT qs
```

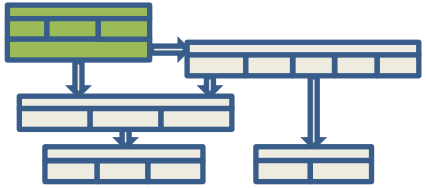


- Rest of the algorithm:

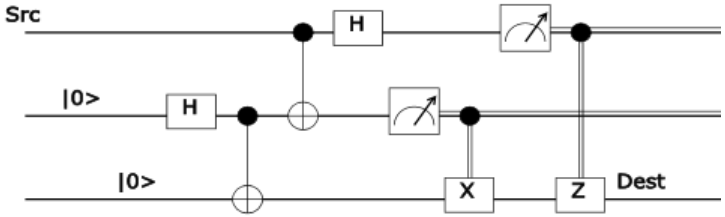
```
let teleport (qs:Qubits) =
    let q0,q1,q2 = qs.[0],qs.[1],qs.[2]

    EPR[q1;q2]; CNOT qs; H qs
    M[q1]; BC X [q1;q2]
    M[q0]; BC Z [q0;q2]
```

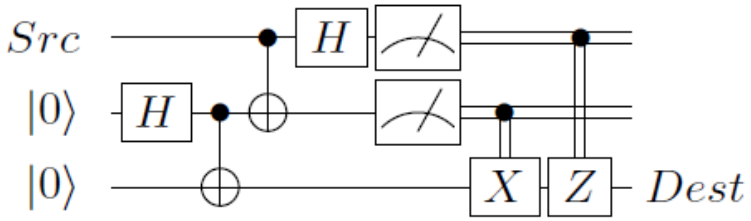


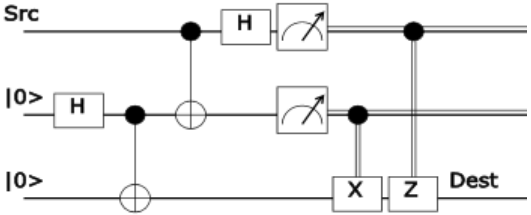
Teleport: Full Circuit



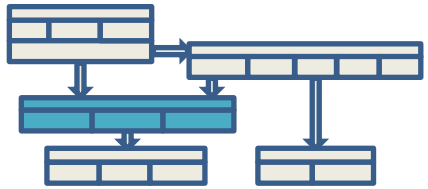
```
let teleport (qs:Qubits) =
    let qs'     = qs.Tail                  // Skip first qubit
    Label >!< (["Src";"|0>";"|0>"],qs) // Label the first 3 qubits
    EPR qs'; CNOT qs; H qs                 // EPR 1,2, then CNOT 0,1 and H 0
    M qs'; BC X qs'                        // Conditionally apply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







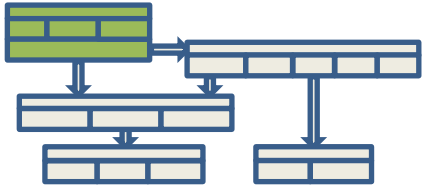
Teleport: Running the code



```
loop N times:
        … create 3 qubits
        … init the first one to a random state
        … print it out
        teleport qs
        … print out the result
```

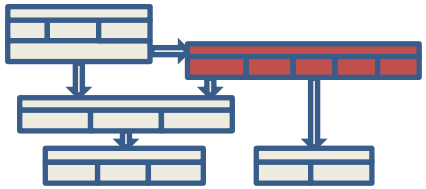
```
0:0000.0/Initial State: (  0.3735-0.2531i)|0>+( -0.4615-0.7639i)|1>
0:0000.0/Final   State: (  0.3735-0.2531i)|0>+( -0.4615-0.7639i)|1> (bits:10)
0:0000.0/Initial State: ( -0.1105+0.3395i)|0>+(   0.927-0.1146i)|1>
0:0000.0/Final   State: ( -0.1105+0.3395i)|0>+(   0.927-0.1146i)|1> (bits:11)
0:0000.0/Initial 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)
```

Gate Definition: Standard



```
/// <summary>
/// Controlled NOT gate
/// </summary>
/// <param name="qs"> Use first two qubits for gate</param>
[<LQD>]
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

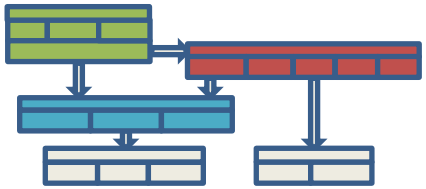


```
let ket      = Ket(3)
let circ     = Circuit.Compile teleport ket.Qubits
circ.Dump showLogInd 0
```

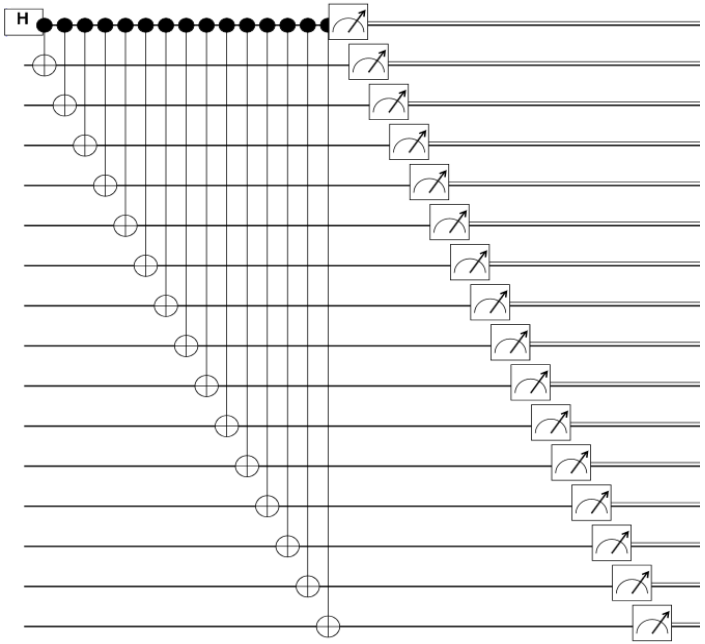
```
SEQ
   APPLY
    GATE H is a  (Normal)
       0.7071 0.7071
       0.7071-0.7071
    WIRE(Id:1)
  APPLY
    GATE CNOT is a Controlled NOT (Normal)
            1      0      0      0
            0      1      0      0
            0      0      0      1
            0      0      1      0
    WIRE(Id:1)
    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)
```

```
APPLY
   GATE Measure is a Collapse State (Measure)
           1      0
           0      1
   WIRE(Id:1)
 Modify
   GATE BitContol is a Bit control Qubit operator (BitOne)
           0      1
           1      0
     WIRE(Id:1)
     WIRE(Id:2)
   APPLY
     GATE X is a Pauli X flip (Normal)
            0      1
            1      0
     WIRE(Id:2)
 APPLY
   GATE Measure is a Collapse State (Measure)
           1      0
           0      1
   WIRE(Id:0)
 Modify
   GATE BitContol is a Bit control Qubit operator (BitOne)
           1      0
           0     -1
     WIRE(Id:0)
     WIRE(Id:2)
   APPLY
     GATE Z is a Pauli Z flip (Normal)
            1      0
            0     -1
     WIRE(Id:2)
```

Fully Entangled: Simple Version

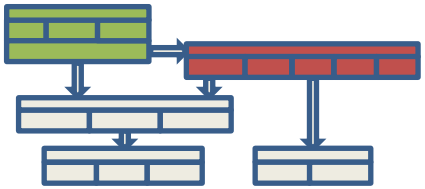


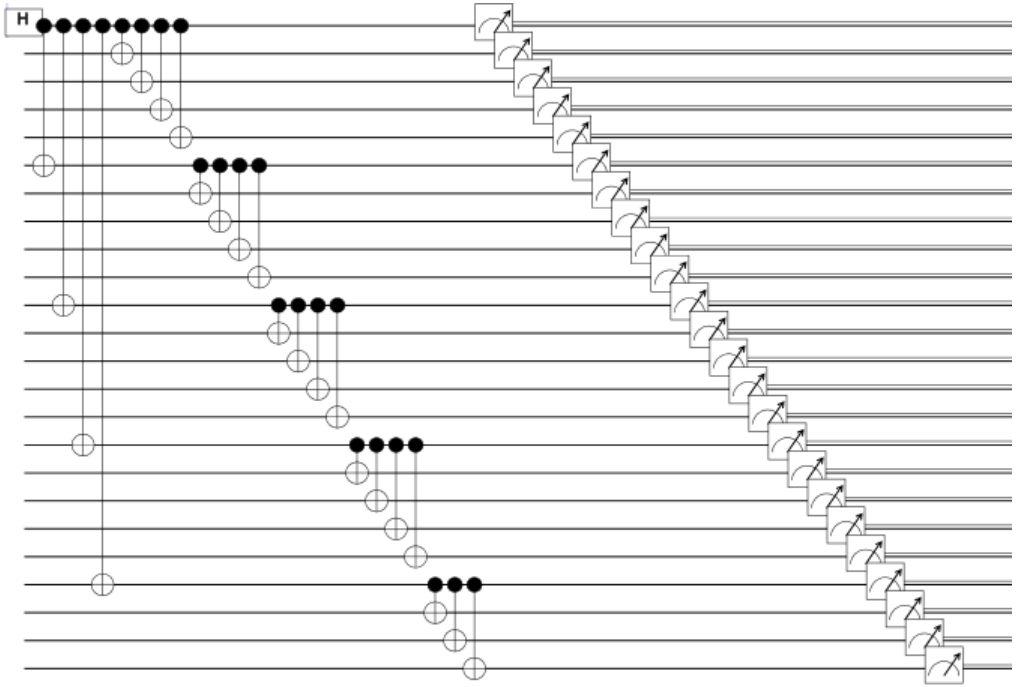
```
let entangle (qs:Qubits) =
    H qs; let q0  = qs.Head
    for q in qs.Tail do CNOT[q0;q]
    M >< qs
```

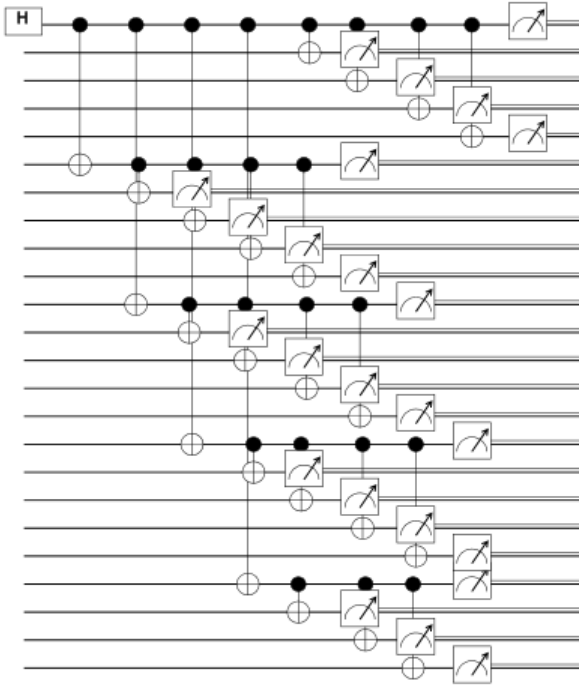



```
0:0000.0/#### Iter   0 [  0.2030]: 0000000000000
0:0000.0/#### Iter   1 [  0.1186]: 0000000000000
0:0000.0/#### Iter   2 [  0.0895]: 0000000000000
0:0000.0/#### Iter   3 [  0.0749]: 0000000000000
0:0000.0/#### Iter   4 [  0.0664]: 1111111111111
0:0000.0/#### Iter   5 [  0.0597]: 0000000000000
0:0000.0/#### Iter   6 [  0.0550]: 1111111111111
0:0000.0/#### Iter   7 [  0.0512]: 0000000000000
0:0000.0/#### Iter   8 [  0.0484]: 0000000000000
0:0000.0/#### Iter   9 [  0.0463]: 0000000000000
0:0000.0/#### Iter  10 [  0.0446]: 0000000000000
0:0000.0/#### Iter  11 [  0.0432]: 1111111111111
0:0000.0/#### Iter  12 [  0.0420]: 0000000000000
0:0000.0/#### Iter  13 [  0.0410]: 0000000000000
0:0000.0/#### Iter  14 [  0.0402]: 0000000000000
0:0000.0/#### Iter  15 [  0.0399]: 0000000000000
0:0000.0/#### Iter  16 [  0.0392]: 1111111111111
0:0000.0/#### Iter  17 [  0.0387]: 1111111111111
0:0000.0/#### Iter  18 [  0.0380]: 0000000000000
0:0000.0/#### Iter  19 [  0.0374]: 1111111111111
```

Fully Entangled: Parallel Version







48 time steps

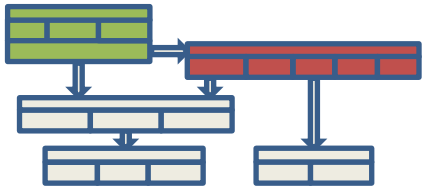
10 time steps

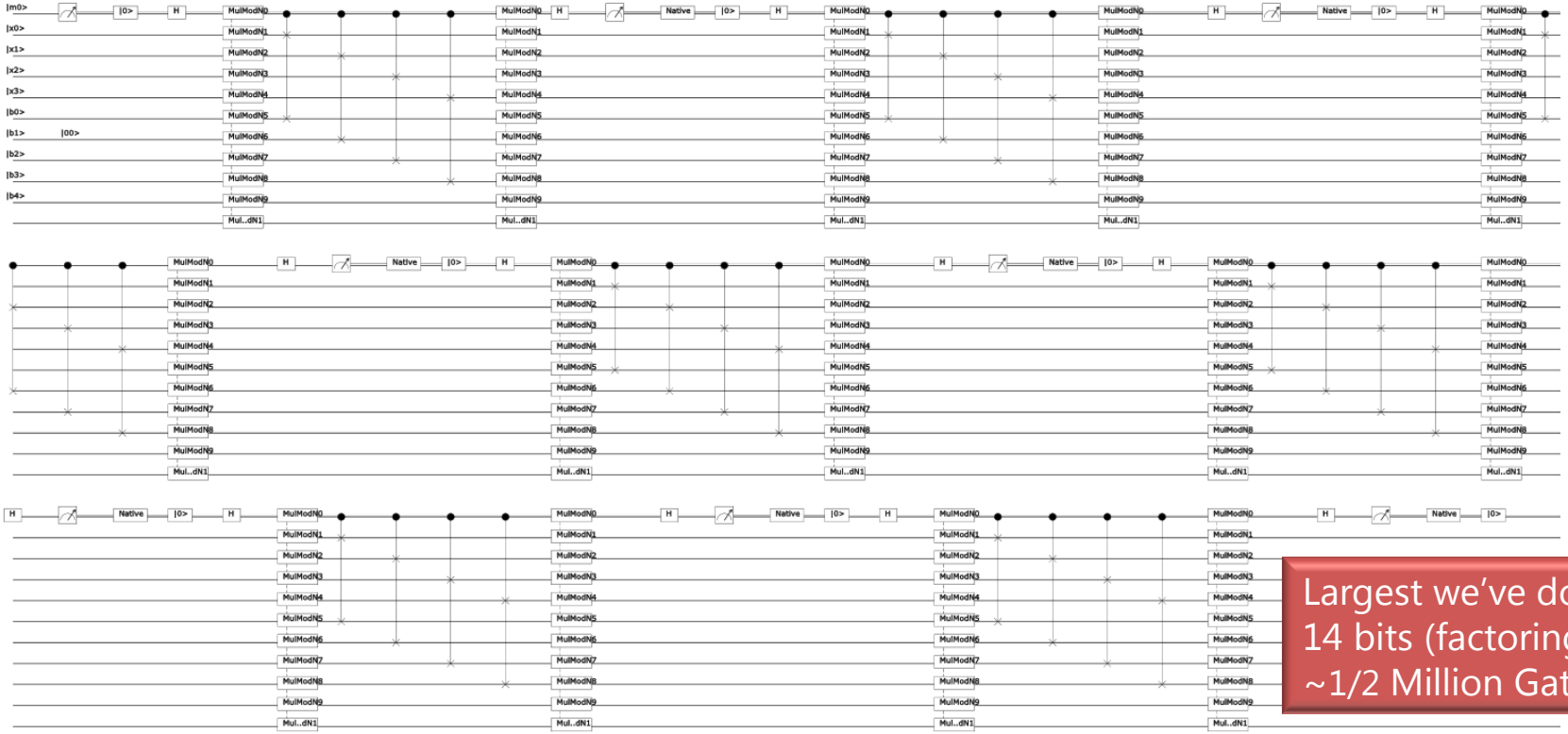
QuArC Areas of Research

- 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
  - LIQ$Ui|\rangle$ (Wecker, Geller, Smith, Svore, Bocharov)
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)

Shor's algorithm: Full Circuit:

4 bits ~ 8200 gates

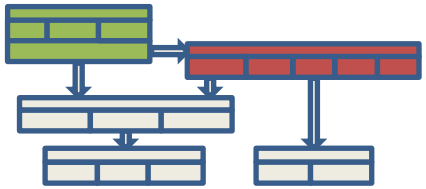


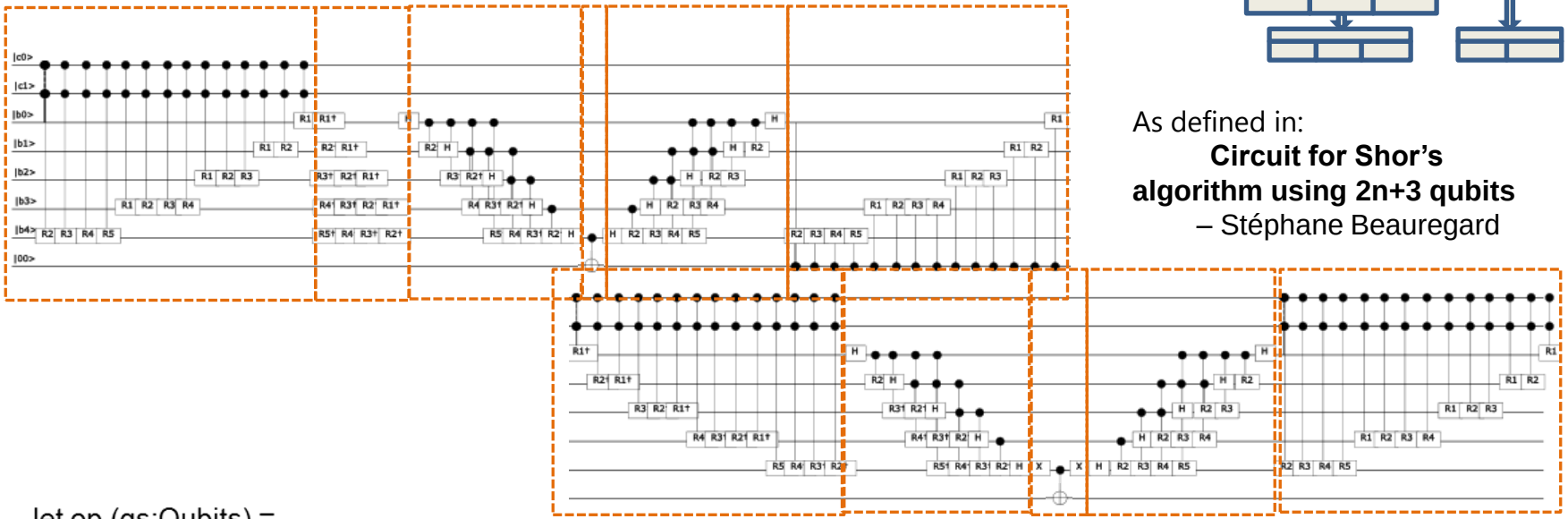


Largest we've done:
14 bits (factoring 8193)
~1/2 Million Gates

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

Shor's algorithm: Modular Adder





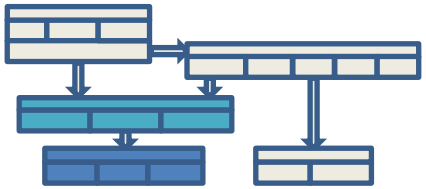
As defined in:

Circuit for Shor's algorithm using 2n+3 qubits

– Stéphane Beauregard

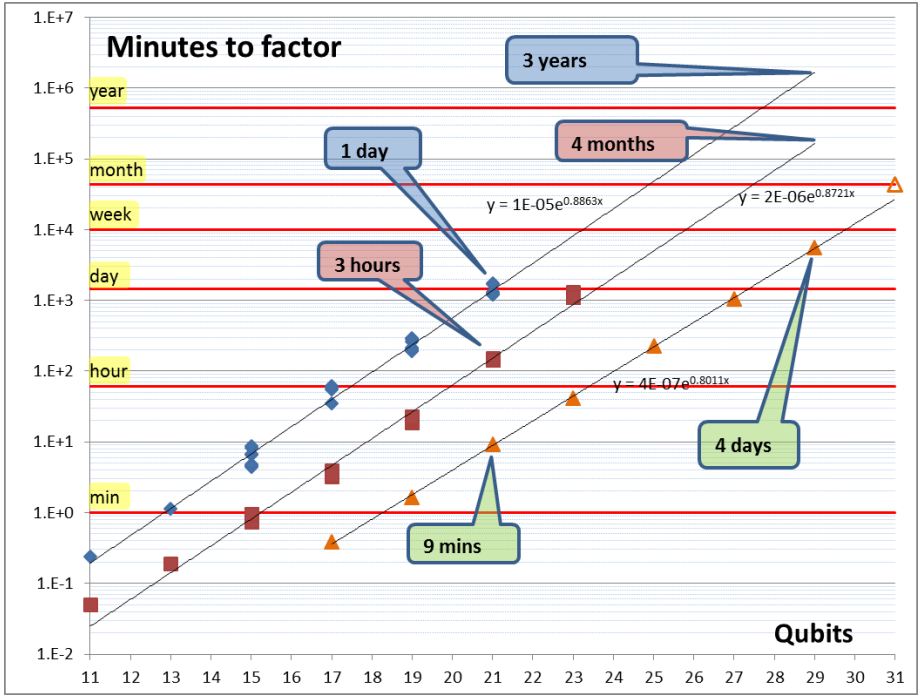
```
let op (qs:Qubits) =
  CCAdd a cbs          // Add a to Φ|b⟩
   AddA' N bs          // Sub N from Φ|a + b⟩
   QFT' bs             // Inverse QFT of Φ|a + b − N⟩
   CNOT [bMx;anc]      // Save top bit in Ancilla
   QFT bs              // QFT of a+b-N
   CAddA N (anc :: bs) // Add back N if negative
   CCAdd' a cbs        // Subtract a from Φ|a + b mod N⟩
   QFT' bs             // Inverse QFT
   X [bMx]             // Flip top bit
   CNOT [bMx;anc]      // Reset Ancilla to |0⟩
   X [bMx]             // Flip top bit back
   QFT bs              // QFT back
   CCAdd a cbs         // Finally get Φ|a + b mod N⟩
```

Shor's algorithm results



N	f1	f2	coPrime	n	Qubits	min total
45	9	5	8	6	15	0.12
51	17	3	8	6	15	0.09
55	5	11	24	6	15	0.14
57	19	3	52	6	15	0.11
63	3	21	23	6	15	0.13
63	3	21	58	6	15	0.22
65	5	13	21	7	17	0.13
69	3	23	19	7	17	0.14
69	3	23	10	7	17	0.24
85	5	17	49	7	17	0.2
99	9	11	53	7	17	0.2
105	35	3	94	7	17	0.18
115	5	23	51	7	17	0.2
117	3	39	59	7	17	0.22
201	3	67	194	8	19	0.49
203	7	29	43	8	19	0.47
207	9	23	169	8	19	0.77
213	3	71	92	8	19	0.68
219	3	73	7	8	19	0.64
237	3	79	211	8	19	0.83
245	7	35	57	8	19	0.89
247	19	13	184	8	19	0.88
249	3	83	29	8	19	0.61
255	5	51	236	8	19	0.86
459	9	51	224	9	21	1.84
465	15	31	136	9	21	2.06
469	7	67	111	9	21	2.3
473	11	43	111	9	21	1.39
475	95	5	286	9	21	1.53
477	9	53	250	9	21	2.64
485	5	97	93	9	21	1.83
501	3	167	274	9	21	1.99

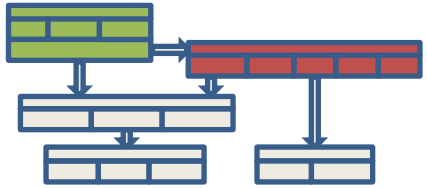
N	f1	f2	coPrime	n	Qubits	min total
507	3	169	68	9	21	2.35
969	3	323	109	10	23	40.75
975	3	325	697	10	23	32.62
981	3	327	248	10	23	44.28
985	5	197	813	10	23	36.22
987	21	47	190	10	23	42.99
993	3	331	892	10	23	36.48
999	9	111	140	10	23	37.44
1001	11	91	288	10	23	39.75
1011	337	3	622	10	23	39.75
1023	3	341	169	10	23	36.53
1995	7	285	1397	11	25	273.63
2005	5	401	483	11	25	263.94
2013	3	671	1960	11	25	270.72
2019	3	673	533	11	25	256.2
2025	9	225	1792	11	25	267.78
2031	3	677	1640	11	25	267.4
2035	5	407	1281	11	25	261.74
2041	13	157	649	11	25	262.29
2043	9	227	1009	11	25	268.68
4047	3	1349	3251	12	27	2200
4053	3	1351	425	12	27	1858
4061	31	131	150	12	27	1931
4063	17	239	832	12	27	2108
4065	15	271	1219	12	27	2278
4071	3	1357	3670	12	27	2073
4077	3	1359	2111	12	27	2562
4081	7	583	1560	12	27	1361
4083	3	1361	3059	12	27	2268
4089	87	47	958	12	27	2247
4095	3	1365	2972	12	27	2393
8189	19	431	6803	13	29	7455



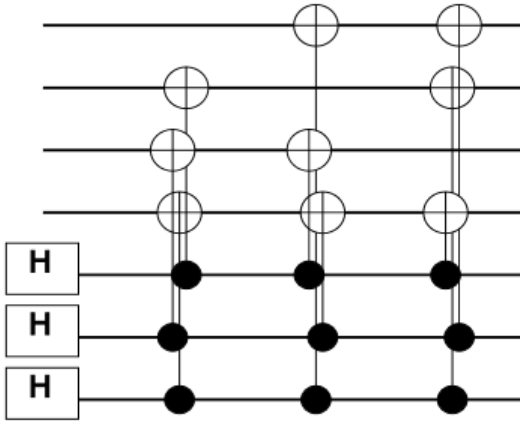
QuArC Areas of Research

- 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
  - LIQ$Ui|\rangle$ (Wecker, Geller, Smith, Svore, Bocharov)
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)

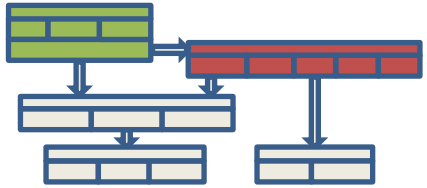
QECC Gate Definition: Steane7 prep

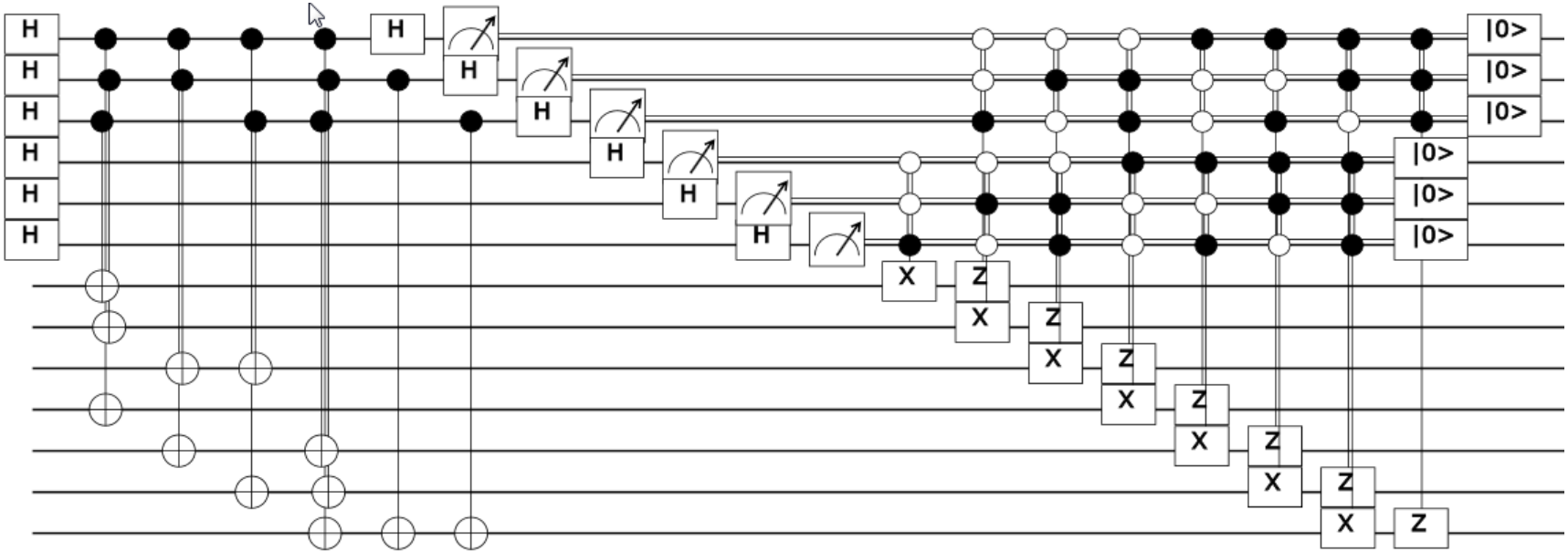


```
/// <summary>
/// Steane7 prep gate (create logical |0>).
/// </summary>
/// <param name="qs"> Physical qubits to use</param>
[<LQD>]
    let prep (qs:Qubits) =
        let nam = "S7_Prep"
        let nam2= "S7\nPrep"
        let gate (qs:Qubits)    =

            // 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
```

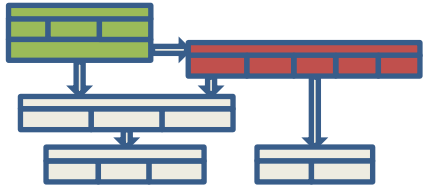


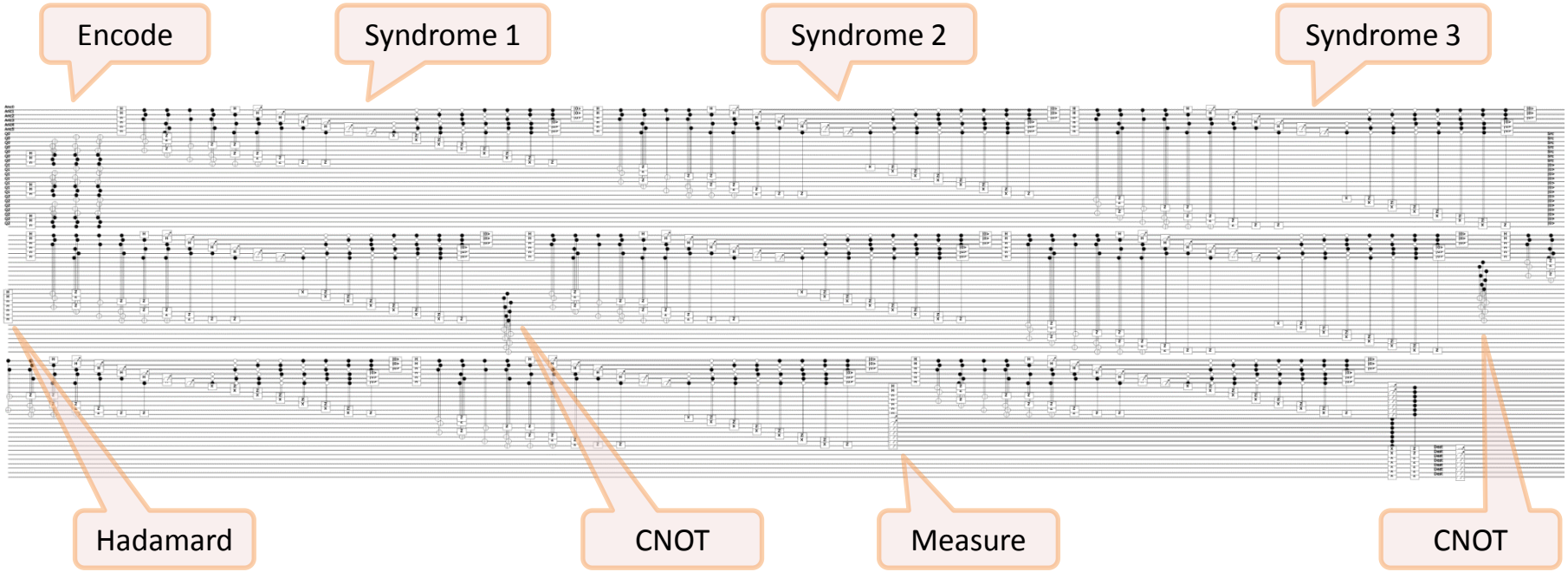
QECC Gate Definition: Steane7 syndrome



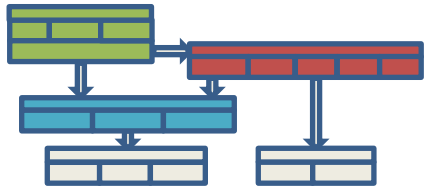


Full Teleport Circuit in a Steane7 Code





Stabilizers



```
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
```

```
      > LIQUiD /l __QECC

0:0000.0/LOOP[Zer0]: InjectedXYZ(0,0,0) Fixes=0 (Zero, One,Zero) dist=(0,0,0)
0:0000.0/LOOP[Zer1]: InjectedXYZ(0,1,0) Fixes=2 ( One,Zero,Zero) dist=(0,0,0)
0:0000.0/LOOP[Zer2]: InjectedXYZ(0,1,1) Fixes=2 (Zero,Zero,Zero) dist=(0,0,0)
0:0000.0/LOOP[Zer3]: InjectedXYZ(0,1,1) Fixes=0 (Zero,Zero,Zero) dist=(0,0,1)
0:0000.0/LOOP[Zer4]: InjectedXYZ(1,1,0) Fixes=3 (Zero, One,Zero) dist=(0,0,0)
0:0000.0/LOOP[Zer5]: InjectedXYZ(1,0,0) Fixes=2 ( One,Zero,Zero) dist=(0,0,0)
0:0000.0/LOOP[Zer6]: InjectedXYZ(0,1,0) Fixes=2 (Zero, One,Zero) dist=(0,0,0)
0:0000.1/LOOP[Zer7]: InjectedXYZ(1,0,0) Fixes=1 ( One,Zero,Zero) dist=(0,0,0)
0:0000.1/LOOP[Zer8]: InjectedXYZ(1,1,0) Fixes=3 ( One,Zero,Zero) dist=(0,0,0)
0:0000.1/LOOP[Zer9]: InjectedXYZ(0,1,0) Fixes=1 ( One, One,Zero) dist=(0,0,0)

0:0000.1/LOOP[One0]: InjectedXYZ(0,0,0) Fixes=0 (Zero, One, One) dist=(0,0,0)
0:0000.1/LOOP[One1]: InjectedXYZ(1,0,0) Fixes=1 (Zero, One, One) dist=(0,0,0)
0:0000.1/LOOP[One2]: InjectedXYZ(0,0,1) Fixes=1 ( One, One, One) dist=(0,0,0)
0:0000.1/LOOP[One3]: InjectedXYZ(0,1,1) Fixes=3 (Zero,Zero, One) dist=(0,0,0)
0:0000.1/LOOP[One4]: InjectedXYZ(0,0,1) Fixes=1 ( One, One, One) dist=(0,0,0)
0:0000.1/LOOP[One5]: InjectedXYZ(0,0,1) Fixes=1 ( One,Zero, One) dist=(0,0,0)
0:0000.1/LOOP[One6]: InjectedXYZ(1,0,0) Fixes=1 (Zero,Zero, One) dist=(0,0,0)
0:0000.1/LOOP[One7]: InjectedXYZ(0,1,0) Fixes=1 (Zero, One, One) dist=(0,0,0)
0:0000.1/LOOP[One8]: InjectedXYZ(1,0,1) Fixes=0 (Zero,Zero, One) dist=(0,0,1)
0:0000.1/LOOP[One9]: InjectedXYZ(1,1,0) Fixes=3 (Zero, One, One) dist=(0,0,0)
```

Final Tableau:

(after Gaussian)

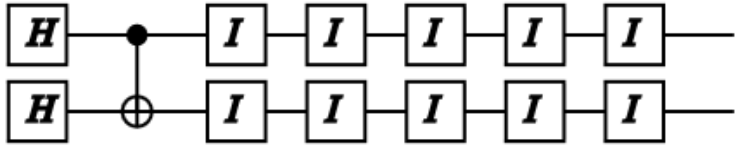
```
-x..............zzz....zzz
-xx.............z......z.
-xxx............z......z
-x..x..........zzzz....zzz
+.x..x.........z..z.z....z.z
+.xx.xx.........z.zz.....zz.
+..x..xx..........zz.....zz
+.x..x..x.........z.z....z.z
+.xx.xx..x........zz.....zz.
-x..x.....x.......zzz....zzz
-x.xx.x....x......z......z..
-xx.xx......x......z......z.
-xxxxxx......x......z......z
+..x..........x.............
+.x..zzzz..zz..x............
+.xx..zz.z.z.z..x...........
+x..z.....zzzz...x..........
-yzx...........zzzy.........
-yxz..........z.zz.y........
+yyy..........zz.z..y.......
+....................x......
+....................x.....
+.....................x....
+......................x...
+zz............zzzz...xxxx..
+.zz..........zz..zz.xx..xx.
+..z..........z.z.z.zx.x.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.............
+...............z............
+................z...........
-................z..........
+.................z.....zz...
+..................z..z.z....
+...................z.zz.....
+....................zzzz....
-.....................zzz....
-......................z.....
+.....................z......
```

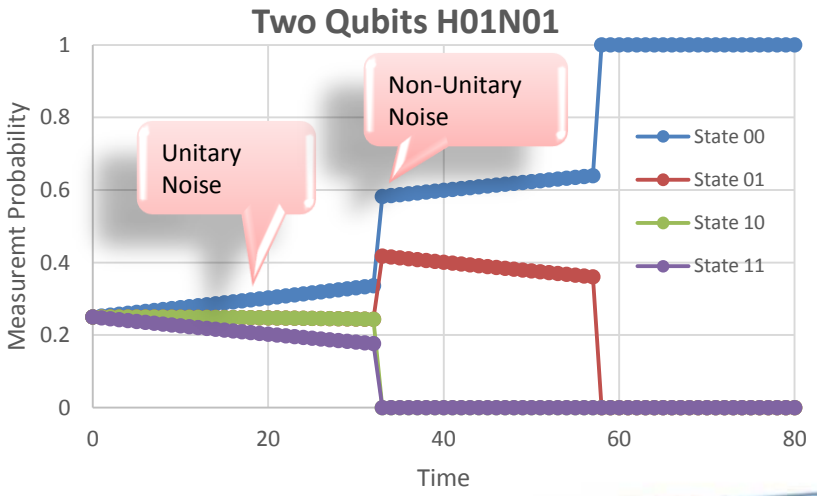
QuArC Areas of Research

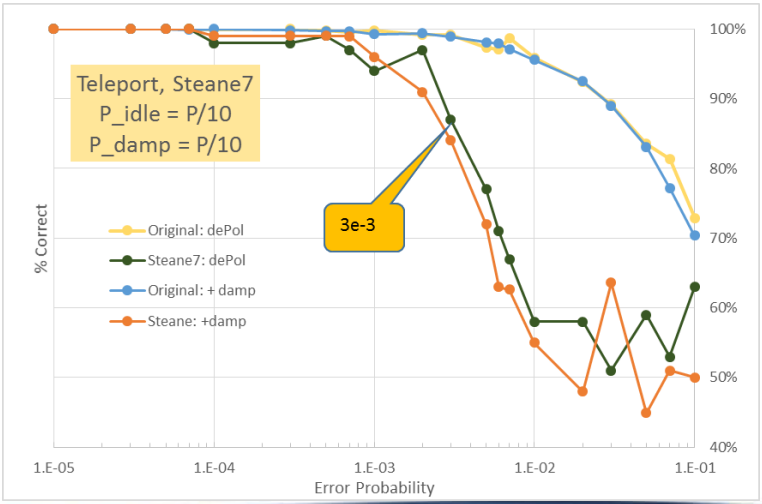
- Quantum circuit synthesis
  - Efficient decomposition into Fibonacci anyon braids (Kliuchnikov, Bocharov, Svore)
  - Repeat-until-success circuits for extremely low-depth synthesis (Paetznick, Svore)
  - Efficient decomposition into V basis circuits (Bocharov, Gurevich, Svore)
  - Characterization 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
  - LIQ$Ui|\rangle$ (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
```



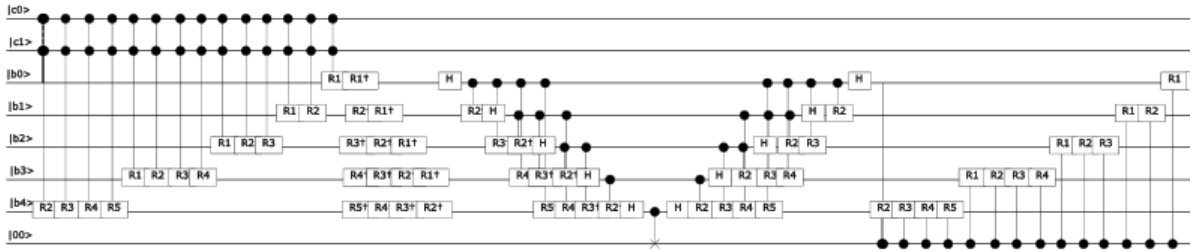


QuArC Areas of Research

- **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
  - LIQ$Ui|\rangle$ (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.



$$R_{64} = HTHTHSHTHTHTHTHTHTHSTHTHTHTHSHTHTHTHTHTHTHTHTHSHS$$

Single-qubit Gate Decomposition

Efficient Decomposition of Single-Qubit Gates into V Basis Circuits

Alex Bocharov,[1,*] Yuri Gurevich,[2,†] and Krysta M. Svore[1,‡]

[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

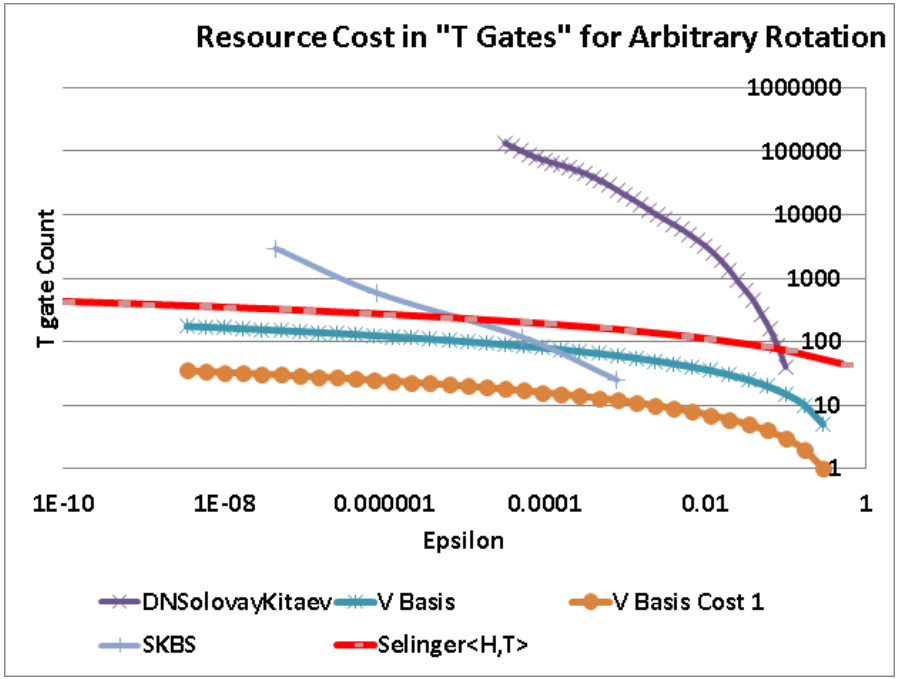
http://arxiv.org/abs/1303.1411

Single-qubit Gate Decomposition

- Standard basis of decomposition is $\{T, Clifford\}$ a.k.a. $\{H, T\}$
- Consider the *efficiently universal basis* $\{V, Clifford\}$

$$V = \frac{1}{\sqrt{5}}\begin{bmatrix} 1+2i & 0 \\ 0 & 1-2i \end{bmatrix}$$

- V-basis offers 25 - 75% improvement over $\{H, T\}$ in T count
- V-basis offers 10x improvement if V is a physical gate



Bocharov, Gurevich, Svore, 2013; arXiv:1303.1411

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
  - LIQ$Ui|\rangle$ (Wecker, Geller, Smith, Svore, Bocharov)
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)

Faster Phase Estimation

Faster Phase Estimation

Krysta M. Svore,[1,*] Matthew B. Hastings,[2,†] and Michael Freedman[2,‡]

[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)

Type	Kitaev's Phase Estimation [11]			Fast Phase Estimation		
	Depth	Width	Size	Depth	Width	Size
Sequential	$O(m\log(m))$	$O(m\log(m))$	$O(m\log(m))$	$O(m\log^*(m))$	$O(m\log^*(m))$	$O(m\log^*(m))$
Parallel	$O(\log(m))$	$O(m\log(m))$	$O(m\log(m))$	$O(\log(m))$	$O(m\log^*(m))$	$O(m\log^*(m))$
Cluster	$O(\log(m))$	$O(m^2)$	$O(m\log(m))$	$O(\log^*(m))$	$O(m^2)$	$O(m\log^*(m))$

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
  - LIQ$Ui|\rangle$ (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*

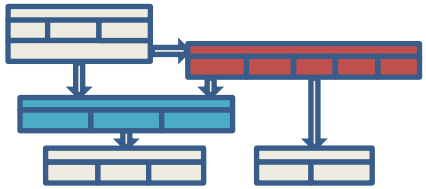
http://arxiv.org/abs/1304.4595

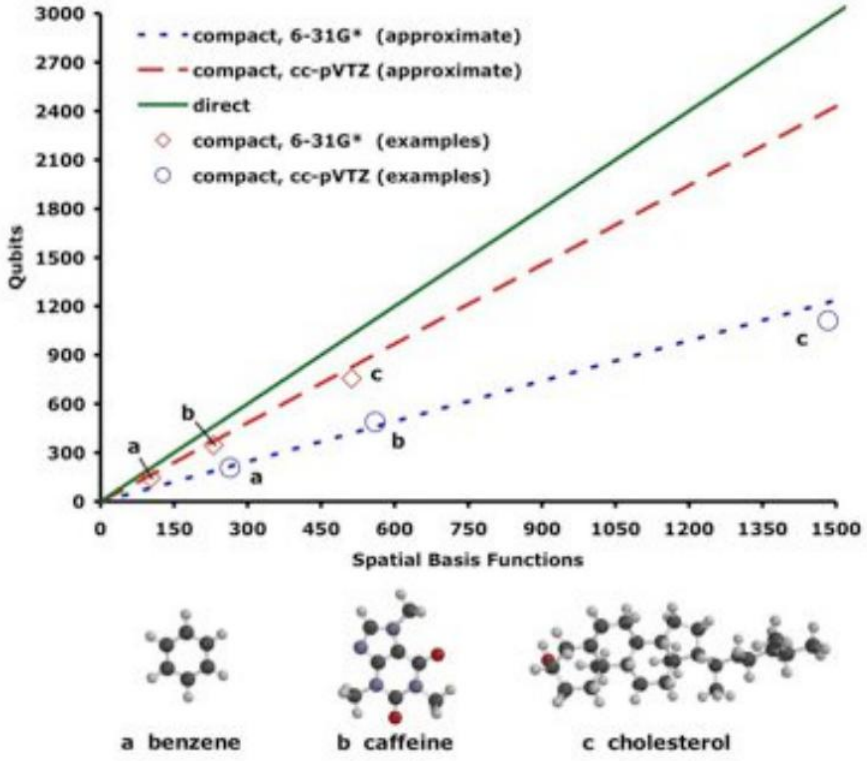
QuArC Areas of Research

- 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
  - LIQ$Ui|\rangle$ (Wecker, Geller, Smith, Svore, Bocharov)
  - Quantum control architecture (Smith, Wecker, Geller)
  - Cold classical systems architecture, design and implementation (Smith, Wecker, Geller)

Hamiltonians (Fermion Model)

$$H = \sum_{pq} h_{pq} a_p^\dagger a_q + \frac{1}{2} \sum_{pqrs} h_{pqrs}\, a_p^\dagger a_q^\dagger a_r a_s$$

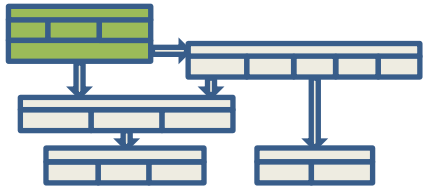




- Very interesting molecules can be modeled with high fidelity in less than 1000 qubits
- Opportunity to design new materials, including ones for next generation QC (bootstrapping)
- Fits naturally into the gate and circuit model that we're using

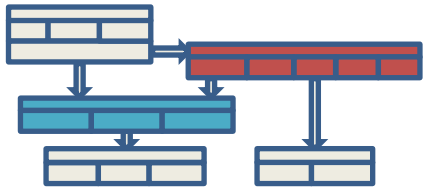
http://arxiv.org/abs/quant-ph/0604193

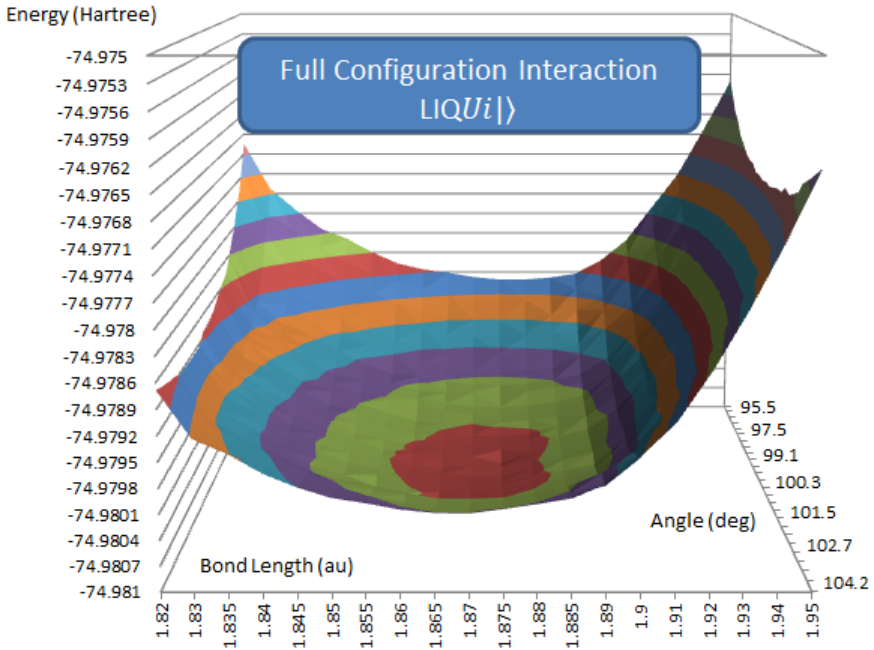
Defining the Water Molecule



```
// 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 OO=-32.696652545 ...
```

Water





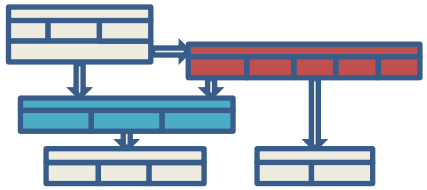
	Ground State		
	RHF	DFT	LIQ$Ui\|\rangle$
Egs	-74.965832	-76.399089	-74.980538
Angle	100.5	102	101.5
Bond Len	1.86	1.84	1.87

$$34{,}000\ gates \times 2^{14} bits \times 2^{10} Trotter \times 50\ samples \times 546\ pts = 1.5 \times 10^{16}\ gate\ ops$$

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

	H_2O,STO-3g
Spin Orbitals	14
SO Constants	434
Primitive Gates	3.4E+04
Qubits	15
9 bit accuracy	2.E+07
Sampling	9.E+08
Trotterization	9.E+11
Full Graph	9.E+13
SK Rotations	9.E+15
QECC	9.E+18

Multi-Resolution Trotterization



Towards realistic quantum algorithms: the case of quantum chemistry

Dave Wecker,[1] Bela Bauer,[2] Bryan K. Clark,[2] and Matthias Troyer[3]

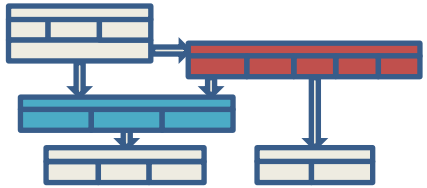
[1] *Quantum Architectures and Computation Group, Microsoft Research, Redmond, WA 98052, USA*
[2] *Station Q, Microsoft Research, Santa Barbara, CA 93106-6105, USA*
[3] *Theoretische Physik, ETH Zurich, 8093 Zurich, Switzerland*

				Sequential		Parallel		Parallel adaptive		
Molecule	Basis	Spin orbitals	Basis size	Rotations	Total	Rotations	Total	Rotations	Total	Reduction
H_2O	STO3G	14	441	1615	20494	1615	6438	338	810	0.13
	P321	26	1.66×10^6	16253	381208	16253	72536	1005	2805	0.039
	DZVP	38	1.35×10^8	63863	2124678	63863	295430	1794	4859	0.016
	P631SS	50	2.8×10^9	177205	7694840	177205	837480	3134	8728	0.010
	P631PPSS	54	6.5×10^9	244123	11404322	244123	1159082	6995	19736	0.017
	P6311SS	62	2.9×10^{10}	419219	22330186	419219	2003978	6308	18465	0.009
CO_2	STO3G	30	1.86×10^6	19639	531926	19639	89062	1550	3713	0.042
	P321	54	1.70×10^{14}	134231	6187878	134231	630006	2750	7496	0.012
	DZVP	90	1.03×10^{20}	1394669	106047976	1394669	6791752	8036	22918	0.0033
	P631SS	90	1.03×10^{20}	1023013	76179240	1023013	4950192	7181	19959	0.0040
Fe_2S_2	STO3G	112	3.4×10^{25}	7441260	630767773	7441260	35865821	3220	8763	0.00024
	P321	168	2.8×10^{48}	45074552	5840151333	45074552	220771169	12571	37489	0.00017

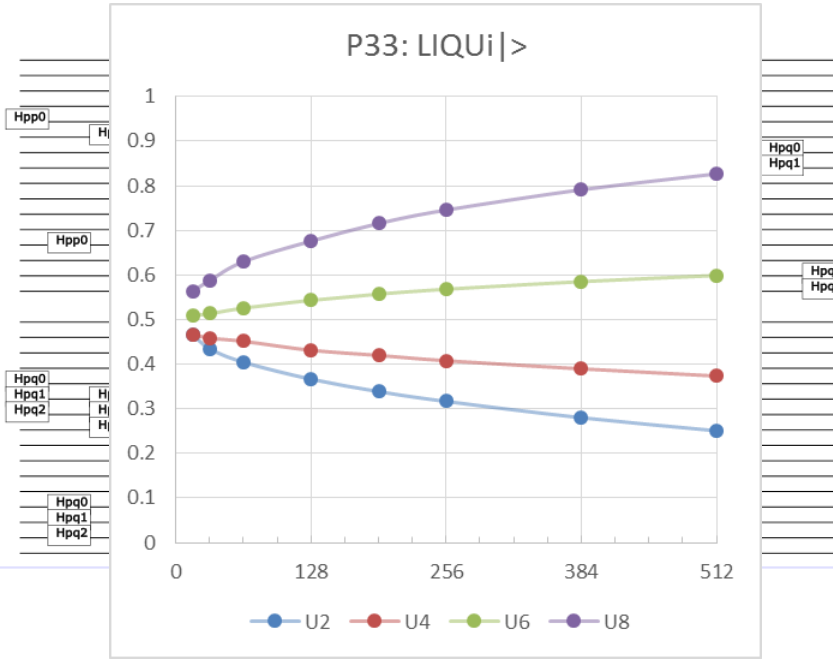
QuArC Areas of Research

- 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
  - LIQ$Ui|\rangle$ (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



- $H = -\sum_{\langle i,j \rangle} \sum_{\sigma} t_{ij} \left(c^{\dagger}_{i,\sigma} c_{j,\sigma} + c^{\dagger}_{j,\sigma} c_{i,\sigma} \right) + U \sum_{i} n_{i,\uparrow} n_{i,\downarrow} + \sum_{i} \epsilon_i \eta_i$



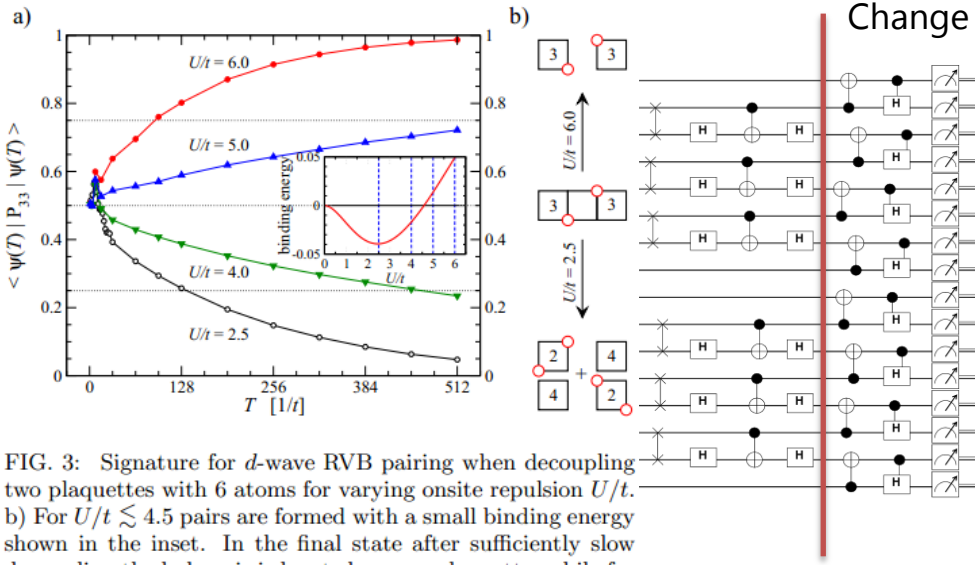


FIG. 3: Signature for d-wave RVB pairing when decoupling two plaquettes with 6 atoms for varying onsite repulsion U/t. b) For $U/t \lesssim 4.5$ pairs are formed with a small binding energy shown in the inset. In the final state after sufficiently slow decoupling the hole pair is located on one plaquette, while for large repulsion $U/t \gtrsim 4.5$ the unpaired holes separate into 3 atoms on each plaquette (right panel). a) Projection of the final state onto the subspace with 3 atoms on each plaquette.

urement

See: d-wave resonating valence bond states of fermionic atoms in optical lattices

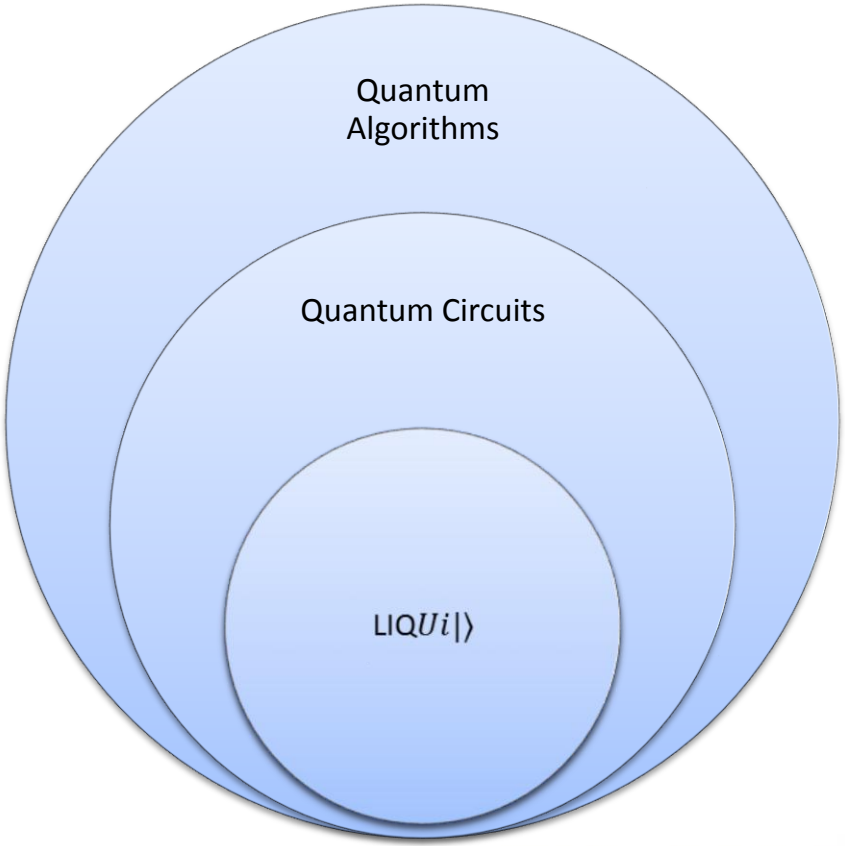
Microsoft

Thank You!

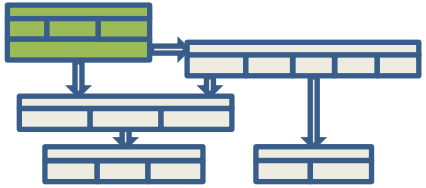
Microsoft

Backup Slides

QuArC Research Activities

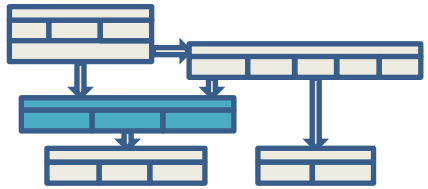


A little bit of F# syntax



- 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 LIQ$Ui|\rangle$ 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: **!!**

Script Mode (.fsx file)

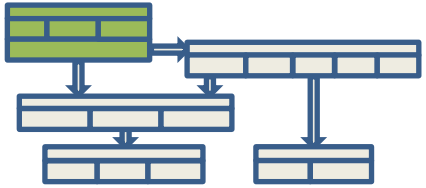


```
#if INTERACTIVE
#r "Liquid.dll"
#else
namespace Microsoft.Research.Liquid // For incremental compile
#endif

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>]                         // Callable from the command line
    let Main() =                    // 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 LIQ$Ui|\rangle$ and call Main()
- **`Liquid /l script.dll Main()`** load previously compiled script into LIQ$Ui|\rangle$ and call Main()

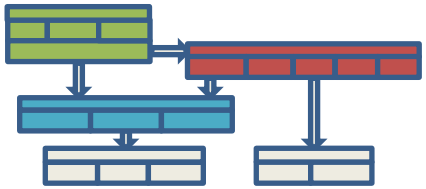
Library of Gates



- **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
```

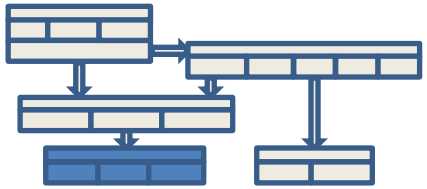
Fully Entangled: Parallel Version



```
// Parallel entanglement
let entangle (qs':Qubits) =

    // Do the heads of the groups
    H qs'
    for i in 5..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 [  2.0535]: 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

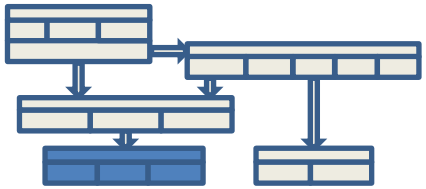


```
<?xml version="1.0" encoding="utf-8"?>
<Ensemble Default="H2O">
    <Pars>
        <Exe>\\machine-00\Liquid\Liquid.exe</Exe>
        <Host>machine-00</Host>
        <Host>machine-01</Host>
        <Host>machine-02</Host>
        <Host>machine-03</Host>
        <Host>machine-04</Host>
        <Host>machine-05</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>
```

Run with:

Liquid /e H2O

Fully Entangled: Ensemble computation



85.46	88.87	96.14	95.32	96.46	98.45	86.73	100.00	81.52	98.71
0.00	1871.95	3958.16	0.00	496.04	0.00	0.00	11568.47	95792.95	0.00
7466.70	388.01	361.41	139.80	196.19	185.48	193.52	211.49	177.01	204.51
...-PCTS-00	...-PCTS-01	...-PCTS-02	...-PCTS-03	...-PCTS-04	...-PCTS-05	...-PCTS-06	...-PCTS-07	...-PCTS-08	...-PCTS-09

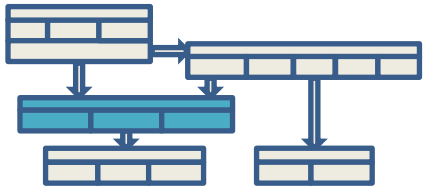
80.98	79.92	86.65	97.15	83.71	85.74	90.19	87.56	87.31	94.59
3838.85	11934.32	1904.94	0.00	0.00	3666.33	0.00	0.00	0.00	0.00
419.64	440.93	401.82	163.41	422.36	181.26	232.18	332.85	398.86	504.57
...-PCTS-10	...-PCTS-11	...-PCTS-12	...-PCTS-13	...-PCTS-14	...-PCTS-15	...-PCTS-16	...-PCTS-17	...-PCTS-18	...-PCTS-19

```
0:0000.2/*1 (  1 0.1%)
1:0000.4/*3*3*3*3*3*3*3*3*3*3*4*3*3*3*3*3*3*3 ( 60 6.1%)
2:0000.6/*6*6*6*6*6*6*6*6*6*6*7*6*6*6*6*6*6*6 ( 60 12.1%)
3:0000.8/*9*9*9*9*9*9*10*9*9*9*9*9*9*9*9*9*9*9 ( 60 18.1%)
1:0001.0/*12*12*12*12*12*12*13*12*12*12*12*12*12*12*12*12*12*12*12 ( 60 24.1%)
3:0001.2/*15*15*15*15*15*15*15*15*15*15*16*15*15*15*15*15*15*15*15 ( 60 30.1%)
4:0001.4/*18*18*18*18*18*18*18*18*18*18*19*18*18*18*18*18*18*18*18 ( 60 36.1%)
5:0001.6/*21*21*21*21*21*21*21*21*21*21*22*21*21*21*21*21*21*21*21 ( 60 42.1%)
6:0001.8/*24*24*24*24*24*24*24*24*24*24*25*24*24*24*24*24*24*24*24 ( 60 48.1%)
1:0002.0/*27*27*27*27*27*27*27*27*27*27*28*27*27*27*27*27*27*27*27 ( 60 54.1%)
7:0002.2/*30*30*30*30*30*30*30*30*30*30*31*30*30*30*30*30*30*30*30 ( 60 60.1%)
0:0002.4/*33*33*33*33*33*33*33*33*33*33*34*33*33*33*33*33*33*33*33 ( 60 66.1%)
8:0002.6/*36*36*36*36*36*36*36*36*36*36*37*36*36*36*36*36*36*36*36 ( 60 72.1%)
0:0002.8/*39*39*39*39*39*39*39*39*39*39*40*39*39*39*39*39*39*39*39 ( 60 78.1%)
9:0003.0/*42*42*42*42*42*42*42*42*42*42*43*42*42*42*42*42*42*42*42 ( 60 84.1%)
a:0003.2/*45*45*45*45*45*45*45*45*45*45*46*45*45*45*45*45*45*45*45 ( 60 90.1%)
4:0003.3/*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*49*49*49 48*49 48 48 (  7 96.7%)
c:0003.5/
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%)
```

1000 Runs of 22 Entangled Qubits in 3 ½ minutes

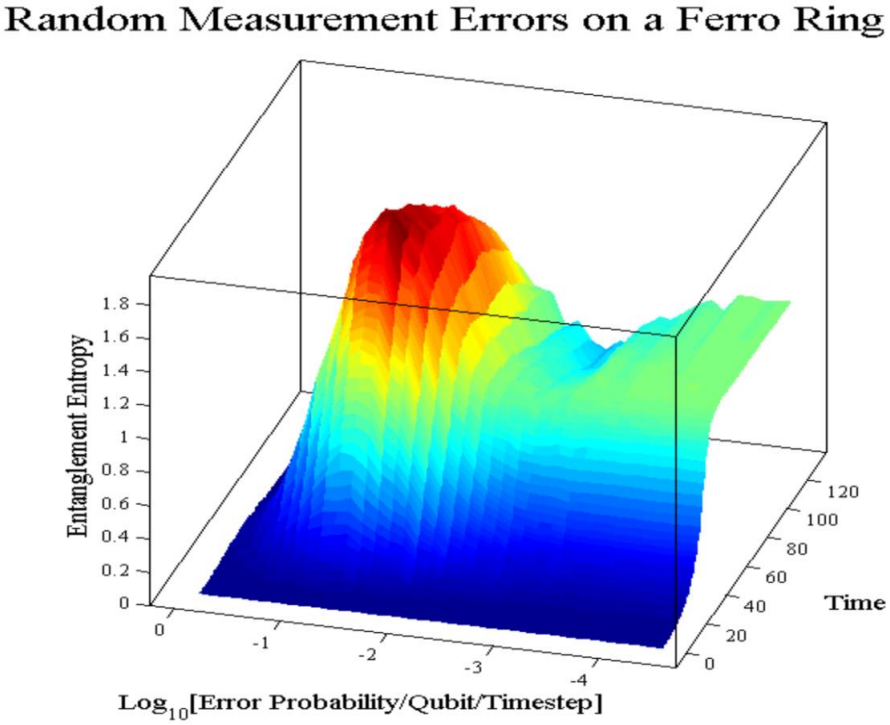
Hamiltonians (Spin Model)

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



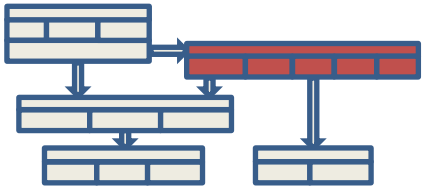
- Built-in Hamiltonian simulator for doing spin-glass models
- Able to reproduce ground states for published D-Wave examples
- Built-in test for doing ferromagnetic chains
- Here's what the circuit looks like…
- Just added a decoherence model and entanglement entropy measurement

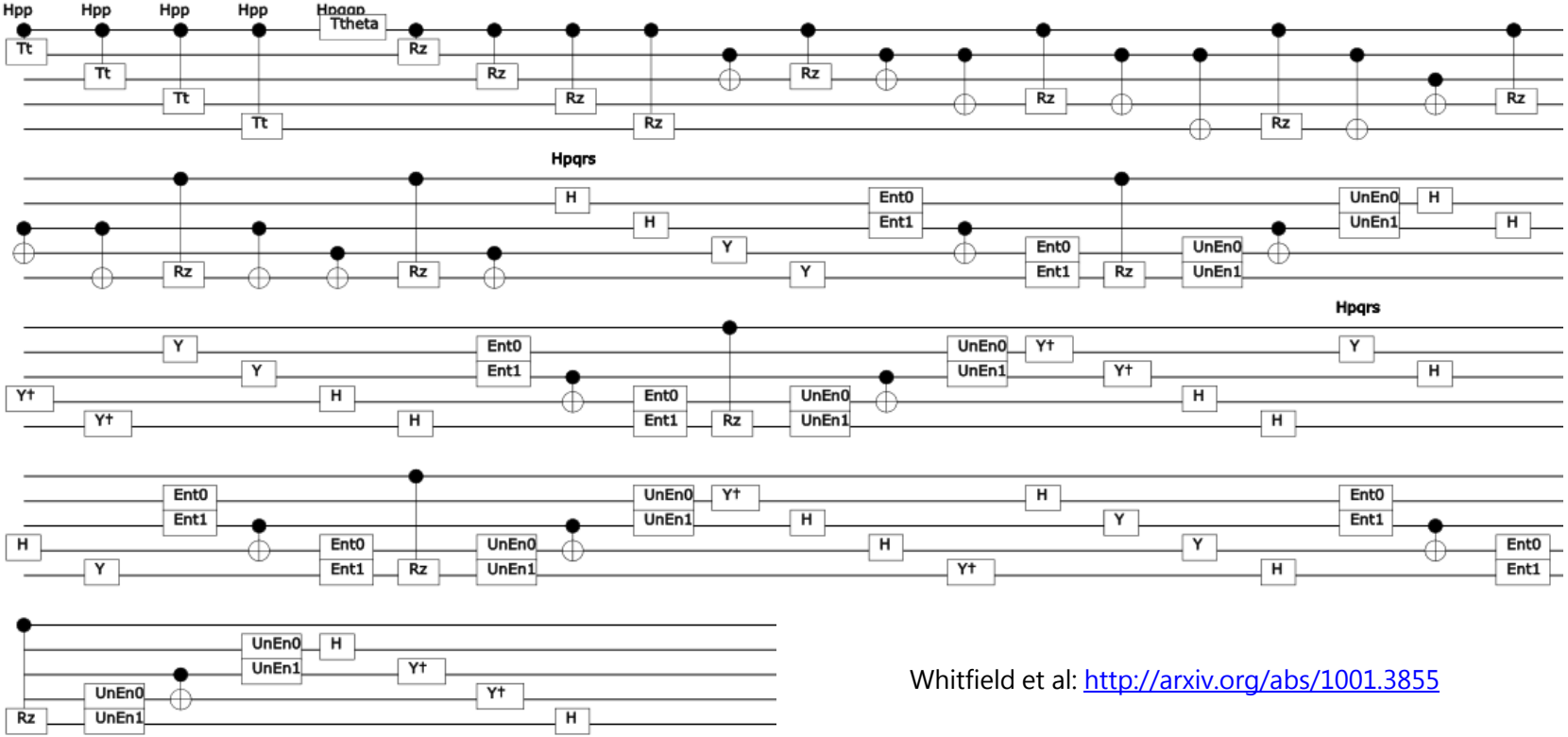
  (thanks to Andrew Das Sarma)



in conjunction with Matthias Troyer, ETH Zurich

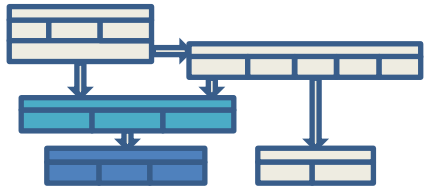
Molecular Hydrogen Circuit (H_2)



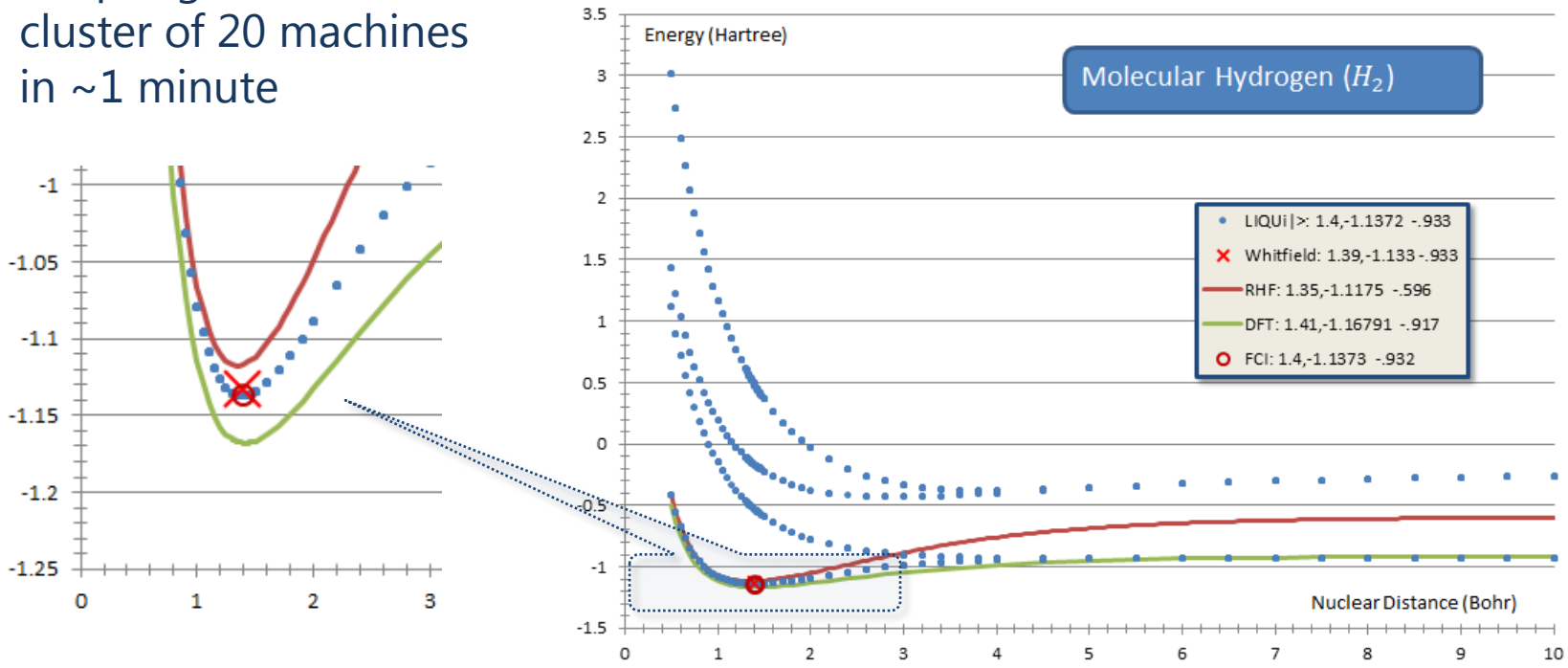


Whitfield et al: http://arxiv.org/abs/1001.3855

Quantum Chemistry Simulation Results

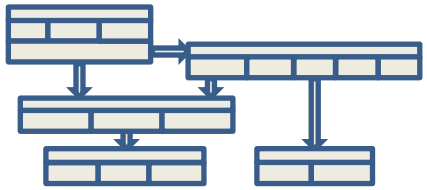


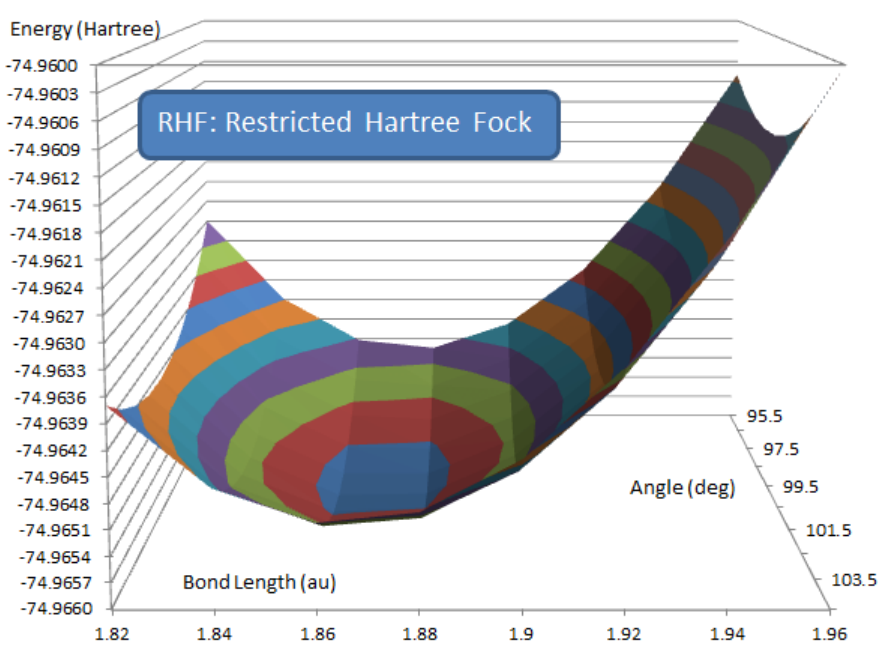
- Output generated on cluster of 20 machines in ~1 minute

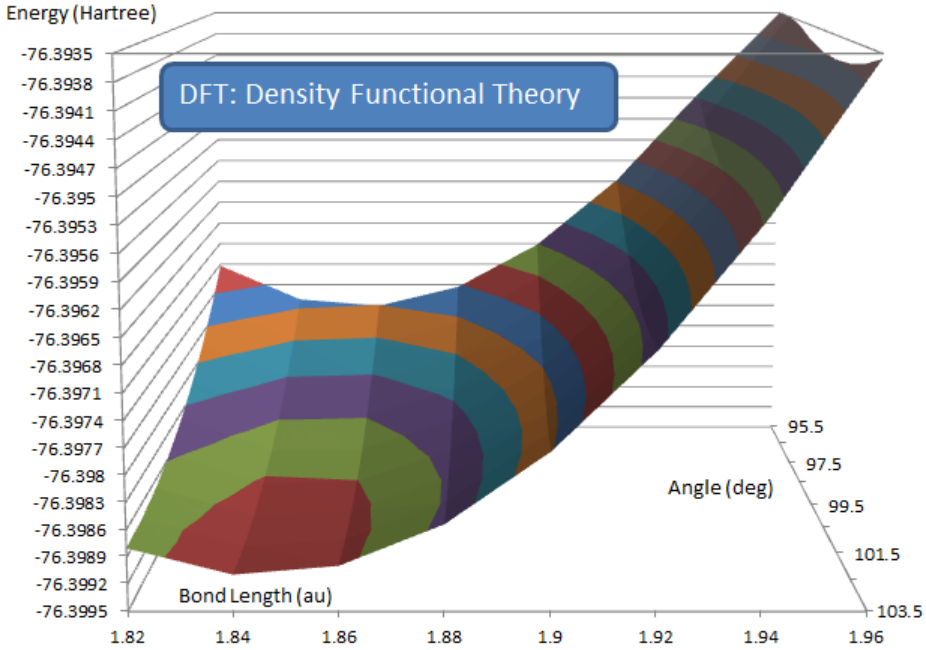


Lanyon et al: http://arxiv.org/abs/0905.0887

Water

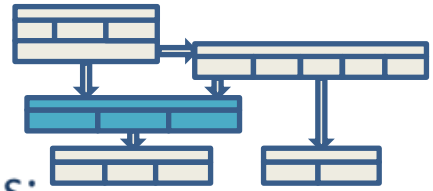






- Simple STO-3g basis (for comparison with simulation)

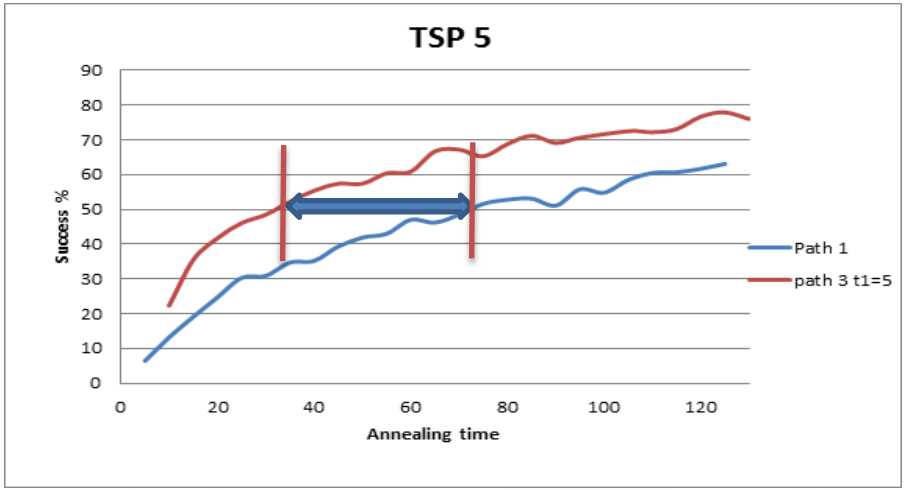
Machine Learning - Traveling Salesman



- 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 2nd quantized form)
- Accumulate average probabilities of evolving state vector
- Example: Synthetic web graph (recursive matrix definition) of 256 pages takes 8 qubits

Adiabatic quantum algorithm for search engine ranking
Garnerone, Zanardi, Lidar (arXiv.org/1109.6546)

LIQ$Ui|\rangle$ Documentation

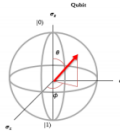
Volume 1

QUANTUM ARCHITECURES AND COMPUTATION

LIQ$Ui|\rangle$ Users Manual

LIQ$Ui|\rangle$ Users Manual

language (e.g., C#) that ha
e LIQ$Ui|\rangle$ library. This inp
run directly on a simulator,
nanipulator that edits the ci
prepares it for either simula
he simulators currently avai
Clients and Servers as well
y machines that share a LA
planned that will allow F# S
Azure cloud for remote execu



ere representation of a Qubit

DATA TYPES

$Ui|\rangle$ relies on a basic set of
led in the host language. T

value that may take on the v
ever, it may also have the
ting the value of a Qubit tha
d (still in a quantum state).

ue that becomes a Bit after
esents a location on a unit sp
ch sphere. A qubit is a supe
which may be viewed as the
3. 1). It may be viewed as a
d by a pair of complex numb
1 (1).

$$a|0\rangle + b|1\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$$

vector representing all the q
his will start out at the size o
in the system (N), but as w
ether, it may grow as large a
ng the state vector and opera
central roles of LIQ$Ui|\rangle$.

rations on Kets, we define G
orm it may be a unitary m
ific operation (e.g., Hadamar
re are also non-unitary gates

LIQUi|> Library

Gate Members

Gate Class Constructors Methods Properties See Also Send Feedback

LIQUi|>

The Gate type exposes the following members.

Constructors

	Name	Description
	Gate	Create a new gate from scratch

Methods

	Name	Description
S	Build	
S	CacheClear	Clear out cache generated by Gate.Build
S	CacheStats	Get gate cache statistics (hits,misses)
	Dump	
	Equals(Object)	Custom equality to another object (Overrides Object.Equals(Object).)
	Equals(Gate)	Custom equality definition
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
S	GetCirc	Get a circuit from a KetMode of CircMode (used internally)
S	GetGate	
	GetHashCode	Hash code for custom equality tests (Overrides Object.GetHashCode().)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	NewMat	Duplicate gate with new matrix
	Run	Heart of the simulator (all gate functions call this). Runs in 1 of 3 modes: Normal: call the gate Gate: If Ket.GateO isn't None then return gate for function Circuit: If Ket.CircuitO isn't None then build circuit
	ToString	Return string output to print gate info (Overrides Object.ToString().)

Properties

	Name	Description
	Arity	Get arity of the gate (based on Qubits or Mat size).
S	CacheDisable	Disable the gate cache (for parallel circuit builds)
S	Dummy	Dummy gate for communications operations
	Help	Get help string
	Mat	Get matrix (if gate has one)
	Name	Get name of gate
	Op	Get gate operation type
	Parent	Get any parent gate
	Qubits	Get number of qubits specified for gate (if specified)
	Render	Get any rendering instructions

See Also

Gate Class

Microsoft.Research.Liquid Namespace