Topics

• Introduction to Quantum Computing
• D-Wave Background
• D-Wave System
• Programming Environment
• Potential Applications
<table>
<thead>
<tr>
<th><strong>QUBIT</strong></th>
<th>$q_i$</th>
<th>Quantum bit which participates in annealing cycle and settles into one of two possible final states: ${0,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COUPLER</strong></td>
<td>$q_i q_j$</td>
<td>Physical device that allows one qubit to influence another qubit</td>
</tr>
<tr>
<td><strong>WEIGHT</strong></td>
<td>$a_i$</td>
<td>Real-valued constant associated with each qubit, which influences the qubit’s tendency to collapse into its two possible final states; controlled by the programmer</td>
</tr>
<tr>
<td><strong>STRENGTH</strong></td>
<td>$b_{ij}$</td>
<td>Real-valued constant associated with each coupler, which controls the influence exerted by one qubit on another; controlled by the programmer</td>
</tr>
<tr>
<td><strong>OBJECTIVE</strong></td>
<td>$Obj$</td>
<td>Real-valued function which is minimized during the annealing cycle</td>
</tr>
</tbody>
</table>

$$Obj(a_i, b_{ij}; q_i) = \sum_i a_i q_i + \sum_{ij} b_{ij} q_i q_j$$

The system samples from the $q_i$ that minimize the objective.
The QMI for the 1000-qubit chip has (nominally):

<table>
<thead>
<tr>
<th>qubit weights</th>
<th>intracell coupler strengths</th>
<th>intercell coupler strengths</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 per cell x 12 x 12</td>
<td>16 per cell x 12 x 12</td>
<td>2 directions x 4 x 12 x 11</td>
</tr>
<tr>
<td>1152</td>
<td>2304</td>
<td>1056</td>
</tr>
</tbody>
</table>

Total size of the QMI is 1152 + 2304 + 1056 = 4412 parameters

Each parameter can be specified to about 4-5 bits of precision
<table>
<thead>
<tr>
<th>Generation</th>
<th>L,M,N</th>
<th>Qubits</th>
<th>Couplers</th>
<th>QMI coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulator</td>
<td>4,4,4</td>
<td>128</td>
<td>352</td>
<td>480</td>
</tr>
<tr>
<td>Rainier</td>
<td>4,4,4</td>
<td>128</td>
<td>352</td>
<td>480</td>
</tr>
<tr>
<td>Vesuvius</td>
<td>4,8,8</td>
<td>512</td>
<td>1472</td>
<td>1984</td>
</tr>
<tr>
<td>W1K</td>
<td>4,12,12</td>
<td>1K</td>
<td>3K</td>
<td>4K</td>
</tr>
<tr>
<td>W2K</td>
<td>4,16,16</td>
<td>2K</td>
<td>6K</td>
<td>8K</td>
</tr>
<tr>
<td>Chimera</td>
<td>L,M,N</td>
<td>2LMN</td>
<td>$L^2MN + L(M - 1)N + LM(N - 1)$</td>
<td>$2LMN + L^2MN + L(M - 1)N + LM(N - 1)$</td>
</tr>
</tbody>
</table>

**Diagram:**
- **Unit Cell (UC):**
  - M columns
  - N rows
- **Cross:**
  - L nodes
- **Column:**
  - L nodes
- **Unit Cell:**
  - Nodes arranged in a 2D grid with M columns and N rows.
Programming Environment

• Operates in a hybrid mode with a HPC System or Data Analytic Engine acting as a co-processor or accelerator
• D-Wave system is “front-ended” on a network by a standard server
• User formulates problem as a series of Quantum Machine Instructions (QMI)
• Front end sends QMI to quantum processor (QP)
• QP starts to sample from the distribution of bit-strings defined by the QMI
• Results are returned to the front-end and on to the user
Example: 4-coloring Canada’s provinces
Canada represented as a graph
Needle & Haystack: Coloring Canada

<table>
<thead>
<tr>
<th># of colors</th>
<th>Needle</th>
<th>Haystack</th>
<th>N/H</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1728</td>
<td>$3^{13} = 1.6 \times 10^6$</td>
<td>0.0011</td>
</tr>
<tr>
<td>4</td>
<td>653184</td>
<td>$4^{13} = 6.7 \times 10^7$</td>
<td>0.0097</td>
</tr>
</tbody>
</table>

(not to scale)
Encode colors and provinces via qubits

• Pick unary encoding for simplicity:
  – 13 regions
  – 4 colors (Blue, Green, Red, Yellow)
  – Create $13 \times 4 = 52$ logical qubits

• Build QMI with these four tasks:
  1. Turn on exactly one of the four color qubits for each region
  2. Map logical color qubits for a region to physical qubits of a unit cell
  3. Use intercell couplers to enforce neighbor constraints
  4. Clone regions as necessary so that Canada can embed into a planar grid

• Each task contributes a portion of the final QMI
• Add individual contributions to get the total QMI
Task 1: turn on one of four color qubits

Objective: \( O(q_b, q_g, q_r, q_y) = (q_b + q_g + q_r + q_y - 1)^2 \cong -1(q_b + q_g + q_r + q_y) + 2(q_b q_g + q_b q_r + q_b q_y + q_g q_r + q_g q_y + q_r q_y) \)
Task 2: embed logical to physical qubits
Task 3: Intercell couplers constrain neighbors

British Columbia | Alberta

Diagram showing the intercell couplers connecting British Columbia and Alberta.
Task 4: Clone regions for planar embedding

- Northwest Territories
- British Columbia
- Alberta
- BC
- AB
- NT

Diagram showing the cloning process for regions.
Colors encoded in unit cells
Scaling up...

- We cannot fit all the states into unit cells of the chip...
- ...so we adopt a divide-and-conquer strategy

Divide the US map into chunks.
Process the first chunk and get valid colorings for the first chunk of states.
Use these colorings to bias the second chunk.
Repeat.
...and up...

254 counties in Texas
...and up

3108 US counties
Implementations of map coloring

C

```c
void setup_unit_cell(int row, int col)
{
    int i, j;
    if (cell_region[row][col] == UNDEF)
        return;
    /* STEP 1: turn on one of C qubits */
    for (i=0; i<C; ++i)
    {
        weight[DW_QUBIT(row,col,'L',i)] += -0.5;
        weight[DW_QUBIT(row,col,'R',i)] += -0.5;
    }
    for (i=0; i<C; ++i)
        for (j=0; j<C; ++j)
            if (i != j)
                strength[DW_INTRACELL_COUPLER(row,col,i,j)] += 1;
    /* STEP 2: create chains */
    for (i=0; i<C; ++i)
    {
        weight[DW_QUBIT(row,col,'L',i)] += 1;
        weight[DW_QUBIT(row,col,'R',i)] += 1;
        strength[DW_INTRACELL_COUPLER(row,col,i,i)] += -2;
    }
}
```

ToQ

| mbool: | 1, 4, @AB |
| mbool: | 1, 4, @BC |
| mbool: | 1, 4, @MB |
| mbool: | 1, 4, @NB |
| mbool: | 1, 4, @NL |
| mbool: | 1, 4, @NS |
| mbool: | 1, 4, @NT |
| mbool: | 1, 4, @NU |
| mbool: | 1, 4, @ON |
| mbool: | 1, 4, @QC |
| mbool: | 1, 4, @SK |
| mbool: | 1, 4, @YT |

assert: @AB != @BC
assert: @AB != @NT
assert: @AB != @SK
assert: @BC != @NT
assert: @BC != @YT
assert: @MB != @NU
assert: @MB != @ON
assert: @MB != @QC
assert: @NB != @NS
assert: @NB != @QC
assert: @NL != @QC
assert: @NT != @NU
assert: @NT != @SK
assert: @NT != @YT
assert: @ON != @QC

entire program

QMI:

- weights
- strengths
• Algorithms are being discovered for mapping applications to adiabatic quantum computers

• These algorithms are being turned into software tools

• This is greatly improving programmer productivity in the AQC environment...

• ...and closing the gap between applications and AQC power
Quantum computing is real. But it’s also hard. So hard that “D-Wave is driving the hardware forward,” says International president Bo Ewald. “But we need more smart people thinking about applications, and another set thinking about software tools.” That’s where the company’s new software tool Qbsolv comes in. Qbsolv is designed to help developers program D-Wave machines without needing a background in quantum physics. A few of D-Wave’s partners are already using the tool, but today the company released Qbsolv as open source, meaning anyone will be able to freely share and modify the software.
Thank You

Questions?

Denny Dahl
ddahl@dwavesys.com