SAPI:

Solver Application Programming Interface

LANL / D-Wave Quantum Programming

June 9, 2016

D-Wave Systems Inc.

Denny Dahl

SAPI Overview

- Available for C, MATLAB or Python programmers
- Available on Windows, OS X or Linux
- Lowest-level supported interface for interacting with D-Wave 2X
- Provides synchronous and asynchronous QMI execution
- SAPI 2.0 released in 2015 – adds support for post-processing
- Download from Qubist includes language & OS specific packages containing programmer reference manual & examples
- Current revision level of SAPI is 2.2
- Anticipate SAPI 2.3 in spring 2016

Basic SAPI functionality

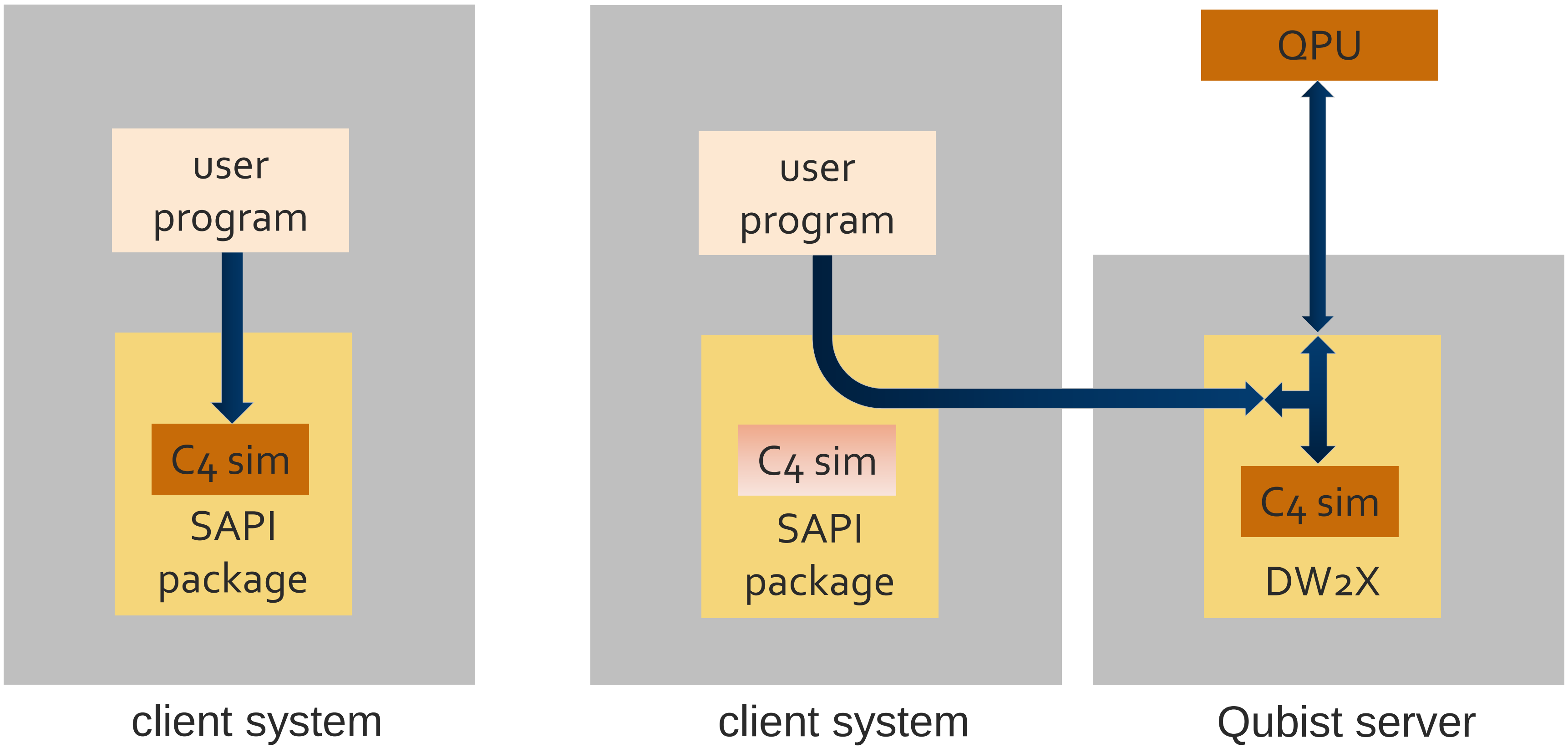
- Local & remote connections
- Access to available solvers
- Accessors to examine solver properties
- QMI creation data structures
- QMI visualization
- QMI execution

Advanced SAPI functionality

- Asynchronous execution
- Embedding
- Order reduction
- Spin reversal transforms
- Post-processing
- Ising/QUBO translation

D-Wave The Quantum Computing Company™

Local versus remote connection



Local

Remote: token required!

Solver properties

Property	Description
supported_problem_types	QUBO or Ising
num_qubits	Total number of qubits, both working and non-working, in the QPU
qubits	List of qubit indices of working qubits
couplers	List of working couplers, represented as pairs of qubit indices

D:WAVE
The Quantum Computing Company™

Solving parameters for all solvers

Parameter	Description
num_reads	A positive integer that indicates the number of samples (output solutions) to read from the solver
answer_mode	Return a histogram of answers sorted in order of energy (‘histogram’) or return all answers individually in the order they were read (‘raw’)
max_answers	Maximum number of answers returned from the solver in histogram mode

D:Wave
The Quantum Computing Company™

Solving parameters specific to QPU

Parameter	Description
auto_scale	Multiply all weights and strengths by an overall scalar to maximally fill range (enabled by default)
annealing_time	Duration in microseconds of annealing time (20 usec default)
beta	Inverse temperature of Boltzmann distribution in post-processing
chains	Lists of qubits that represent the same logical variable in post-processing

D:Wave
The Quantum Computing Company™

 |

Solving parameters: QPU (cont.)

Parameter	Description
num_spin_reversal_transforms	Do (1) or do not (0) apply spin-reversal transforms
postprocess	Either empty string, “sampling”, or “optimization”
programming_thermalization	Duration in microseconds of post-programming cool-down interval
readout_thermalization	Duration in microseconds of post read-out cool-down interval

D:WAVE The Quantum Computing Company™

SAPI initialization & clean-up

The C SAPI library maintains some internal global state that you must initialize and clean up.

C	sapi_globalInit() sapi_globalCleanup()
MATLAB	NONE
Python	NONE

Connections

SAPI uses different function calls for local and remote connections

C	sapi_localConnection(...) sapi_remoteConnection(...)
MATLAB	sapiLocalConnection(...) sapiRemoteConnection(...)
Python	local_connection RemoteConnection

D:Wave The Quantum Computing Company™

Solvers

- Quantum hardware typically supports a single solver
- Software simulators typically implement several solvers

C	sapi_listSolvers(...) sapi_getSolver(...)
MATLAB	sapiListSolvers(...) sapiSolver(...)
Python	*.solver_names *.get_solver

D:Wave
The Quantum Computing Company™

Properties

C	sapi_solverProperties(...)
MATLAB	sapiSolverProperties(...)
Python	*.supported_problem_types *.*

D-Wave The Quantum Computing Company™

QMI data structure

C	sapi_Problem
MATLAB	Ising: h,J QUBO: Q
Python	Ising: h,J QUBO: Q

D-Wave
The Quantum Computing Company™

QMI execution

C	sapi_solveIsing(...) sapi_solveQUBO(...) sapi_asyncSolveIsing(...) sapi_asyncSolveQubo(...)
MATLAB	sapiSolveIsing(...) sapiSolveQubo(...) sapiAsyncSolveIsing(...) sapiAsyncSolveQubo(...)
Python	solve_ising solve_qubo async_solve_ising async_solve_qubo

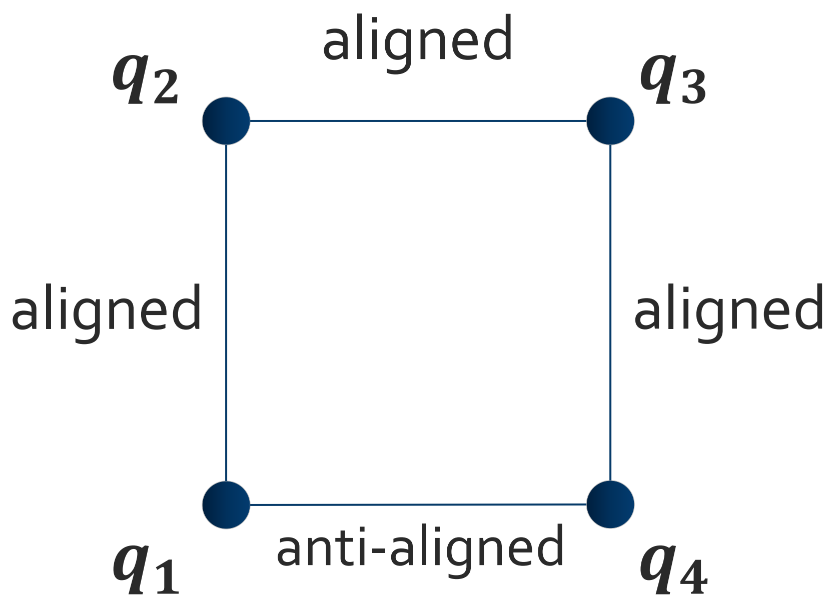
D:WAVE
The Quantum Computing Company™

Solutions

C	sapi_IsingResult
MATLAB	answer = sapi*Solve*
Python	answer = sapi_*

SAPI example: frustrated system

We know how to make aligned and anti-aligned chains.
Combine these two chain types to build a **frustrated system**.



q_1	q_2	q_3	q_4
0	0	0	0
0	0	0	1
0	0	1	1
0	1	1	1
1	1	1	1
1	1	1	0
1	1	0	0
1	0	0	0

D-Wave The Quantum Computing Company™

QUBOs for individual constraints

aligned

q_1	q_2	$q_1 + q_2 - 2q_1q_2$
0	0	0
0	1	1
1	0	1
1	1	0

aligned

q_3	q_4	$q_3 + q_4 - 2q_3q_4$
0	0	0
0	1	1
1	0	1
1	1	0

aligned

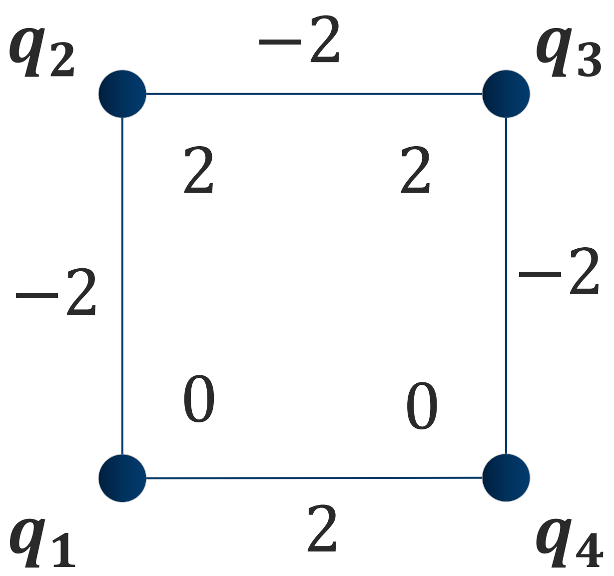
q_2	q_3	$q_2 + q_3 - 2q_2q_3$
0	0	0
0	1	1
1	0	1
1	1	0

anti-aligned

q_4	q_1	$-q_4 - q_1 + 2q_4q_1$
0	0	0
0	1	-1
1	0	-1
1	1	0

Aggregate QUBO

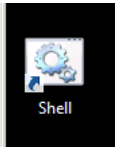
1. Confirm that the QUBO represented here is the sum of the individual QUBOs from the last slide.
2. Input the QUBO below into Quantum Apprentice on the Four Qubits tab.
3. Confirm that you get the desired set of states.



$$Obj = 2q_2 + 2q_3 - 2q_1q_2 - 2q_2q_3 - 2q_3q_4 + 2q_4q_1$$

D-WAVE The Quantum Computing Company™

Warm-up with C program



1. On Darwin, navigate to the **sapi** directory:

```
/home/ddahl> cd sapi
```

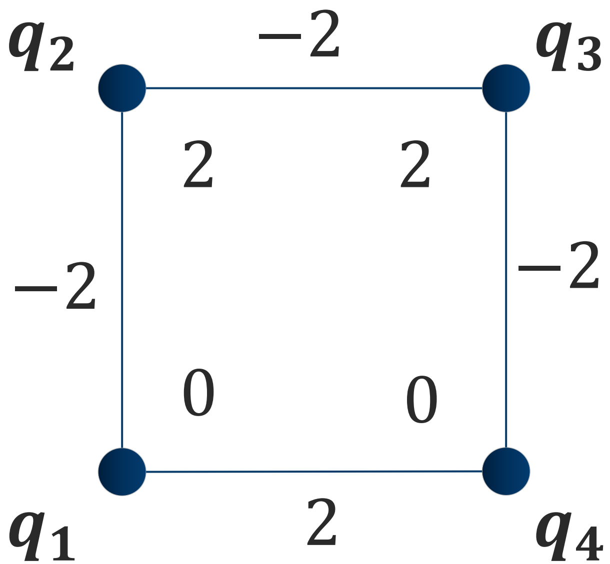
2. Read the README file (using emacs, vi or …)
3. Look at the C program **eq.c** and note lines 230-233

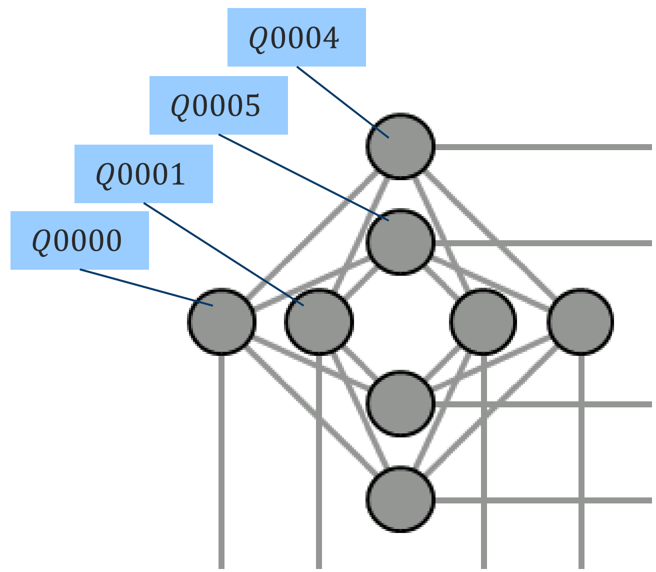
4. Compile, link and run **eq.c** as follows (or use build.bash):

```
> gcc -I $DWAVE_HOME -c eq.c
> gcc -L $DWAVE_HOME -l dwave_sapi eq.o -o eq
> eq
```

5. Change **num_reads** in **eq.c** to 1000 and repeat step 4.

Embed QUBO to unit cell





$$q_1 \Rightarrow Q0000$$
$$q_2 \Rightarrow Q0004$$
$$q_3 \Rightarrow Q0001$$
$$q_4 \Rightarrow Q0005$$

D:Wave
The Quantum Computing Company™

Run kink.c on the local simulator

1. Copy **eq.c** to **kink.c**

```
/home/ddahl/sapi> cp eq.c kink.c
```

2. Edit lines initializing **DW_weight** and **DW_strength** to reflect the embedded logical problem. To determine the correct indices for the **DW_strength** array, click on the couplers in Quantum Apprentice on the **Chimera** tab and note the coupler label in the name box.
3. Compile, link and run **kink.c**

```
> gcc -I $DWAVE_HOME -c kink.c
> gcc -L $DWAVE_HOME -l dwave_sapi kink.o -o kink
> kink
```

4. Did you see all eight valid answers? What fraction of your samples were invalid? How even was the distribution of samples across the valid answers?

Run dw2x.c on DW2x_SYS4

1. Look at the C program **dw2x.c** and edit lines 238-241
2. Insert API token into line 239. Token can be found on line 2 of file ~/.dwrc following the comma
3. Compile, link and run **dw2x.c**

```
> gcc -I $DWAVE_HOME -c dw2x.c
> gcc -L $DWAVE_HOME -l dwave_sapi dw2x.o -o dw2x
> dw2x
```

4. Edit lines 230-233 **dw2x.c** to include the kink QUBO and increase the number of reads to 1000. Repeat step 3.
5. What went wrong? Fix the problem & re-run. Answer the same questions from (4.) on the prior slide.

D:WAVE The Quantum Computing Company™

Summary

- Roughly equivalent functionality is available from all three SAPI interfaces
- Programmer convenience is a reasonable criteria to use in choosing one of these interfaces, but keep in mind:
- Working at this level gives the user the ***most control***…
- …and provides the ***least support*** for mapping high level problems to the system

