Los Alamos NATIONAL LABORATORY

Integrating High-Performance Computing with Quantum Computing

Scott Pakin

11 January 2023

LA-UR-22-29744

Focus of This Talk

- Experience and opportunities for using quantum computing to accelerate high-performance computing applications
- Experience and opportunities for using high-performance computing to help develop quantum computing applications

Complementary Roles

- High-performance computing is good for computations that…
  - Input, manipulate, and output large amounts of data
  - Involve many tasks cooperating to solve a large problem
  - May be floating-point intensive
- Quantum computing is good for computations that…
  - Input and output minimal amounts of data but perform extreme amounts of work on that data
  - Reveal global properties of data
  - Work with discrete data

More general-purpose

At best constant speedup over sequential, classical execution (but possibly a very large constant)

More specialized

At best exponential speedup over sequential, classical execution (but typically coming with a constant-time performance penalty)

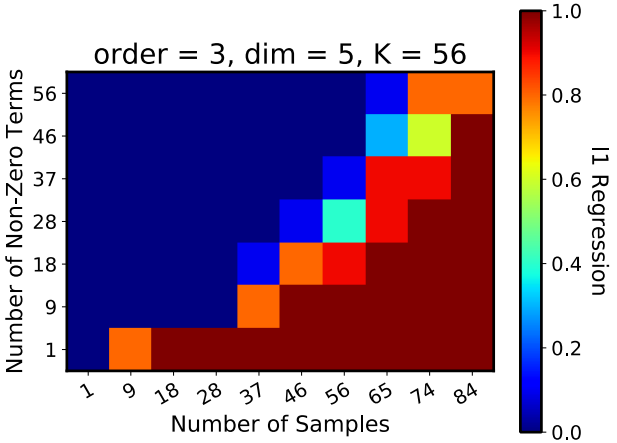
QC for HPC

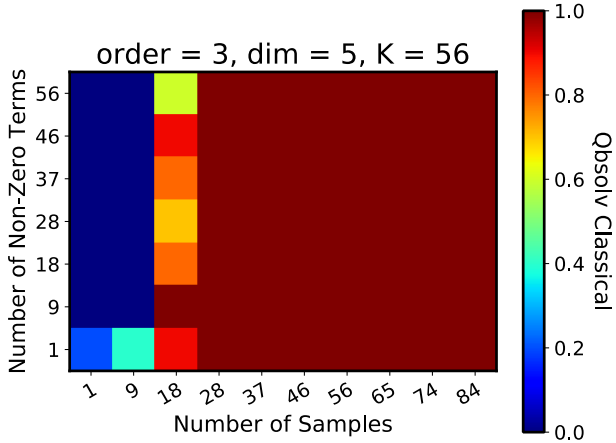
Quantum Optimization for Uncertainty Quantification

- Problem
  - Want the $\bar{x}$ that minimizes $f(\bar{x})$
  - $f(\bar{x})$ is expensive to evaluate, typically requiring a long-running HPC simulation
  - How to evaluate only those $\bar{x}$ with a good chance of minimizing $f(\bar{x})$?
- Solution
  - Use a 0–1 linear combination of many basis functions as a surrogate for $f(\bar{x})$
  - Have a quantum computer find which combination of basis functions best fits the known $\{\bar{x}, f(\bar{x})\}$
  - Minimize the surrogate function also using a quantum computer
  - Evaluate the real f at the point that minimizes the surrogate
  - Repeat the process including the new $\{\bar{x}, f(\bar{x})\}$ pair

Quantum Optimization for Uncertainty Quantification (cont.)

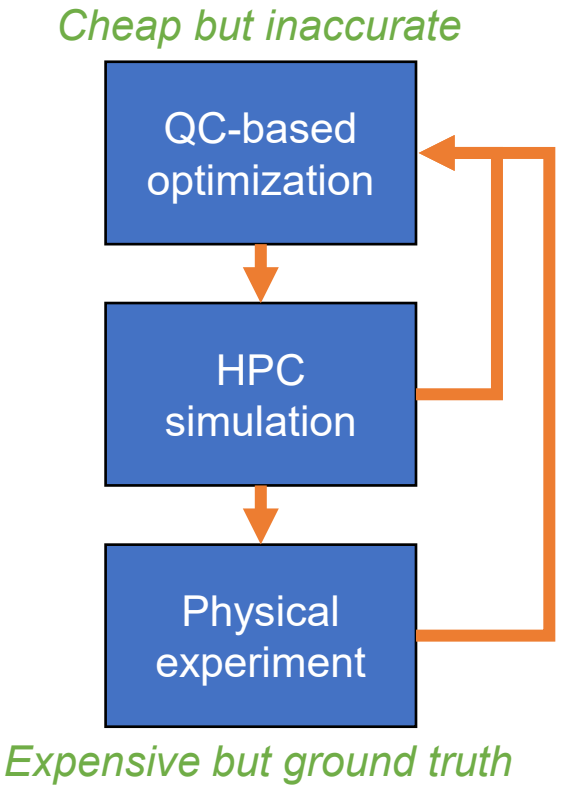
- Work by Bert Debusschere, Khachik Sargsyan, and Ojas Parekh (Sandia National Laboratories) on LANL's D-Wave System
- Better recovery when using 0–1 coefficients on D-Wave (right) than the more traditional real-valued coefficients on a classical computer (left)





A Variation: Model Parameterization

- (A project that unfortunately never got off the ground)
- Problem
  - Need to find values for a parameterized model that best fit the experimental data
  - Experimental data are expensive and time-consuming to acquire and therefore in short supply
  - HPC simulations are faster but not 100% accurate
- Solution
  - As in the UQ example, use quantum computing to optimize a surrogate function
  - Based on the quantum computer's recommendation, select HPC simulations to run
  - Only when finding model parameters that look very promising, gather more experimental data

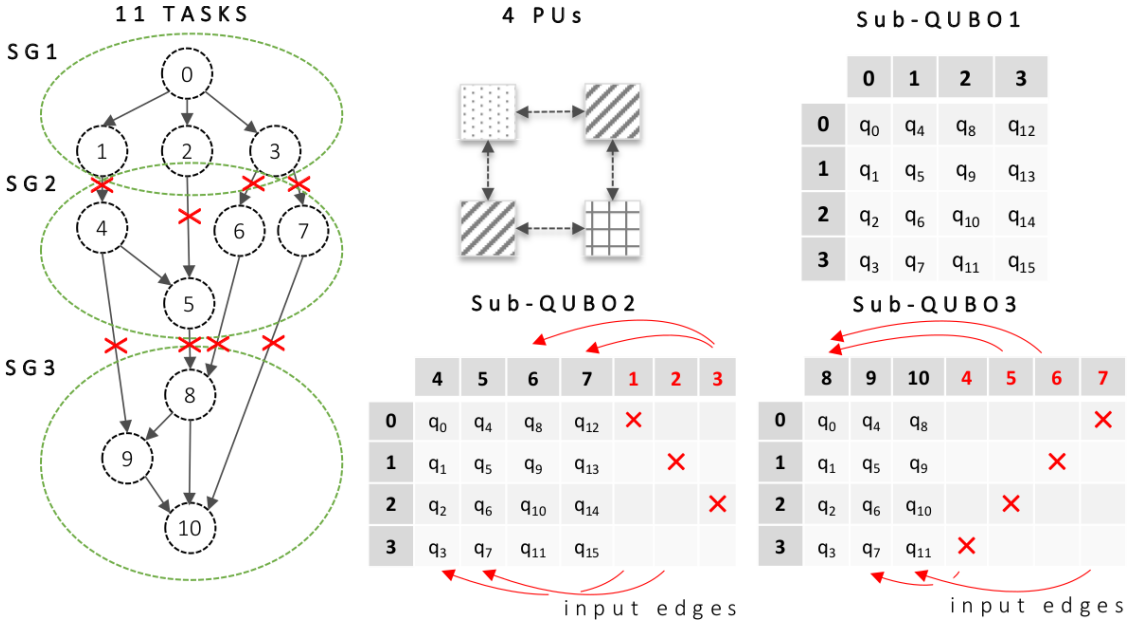


Mapping Compute Tasks to HPC Hardware

- Problem
  - Have many, possibly small, communicating tasks that need to run on heterogeneous HPC hardware (e.g., including CPUs and GPUs)
  - If tasks are maximally spread across the system, parallelism is maximized (good), but communication overhead can dominate execution time (bad)
  - If tasks are packed onto few processors, communication is minimized (good), but available parallelism is not exploited (bad)
- Solution
  - Encode a QUBO that maps each task to exactly one processing unit
  - Partition the task graph based on dependency levels
  - Represent inter-task communication with a quadratic coefficient proportional to the communication cost
  - Use a quantum annealer to find the mapping that maximizes performance

Mapping Tasks to HPC Hardware (cont.)

- Work by Anastasiia Butko (Lawrence Berkeley National Laboratory) on LANL's D-Wave System
- Found promising performance and scalability relative to classical solvers



Partitioning a task communication graph and mapping it to multiple QUBOs

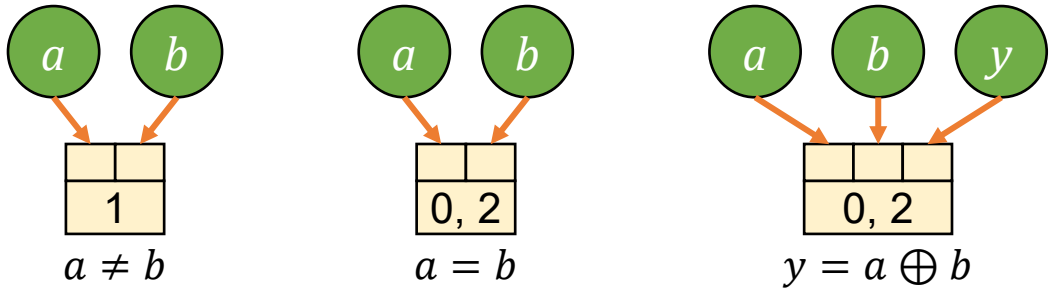
HPC for QC

My Current Project: A Classical Programming Model for Quantum Computers

- Goals
  - Relatively easy to use by traditional HPC developers
  - Applicable to a range of problems
  - Portable across different QCs and (for development) even classical computers
  - Able, at least potentially, to deliver a performance benefit
- Work in progress: **NchooseK**
  - New constraint-programming system
  - Designed for simplicity of expressing problems and of compiling code to both circuit-model and annealing-model QCs

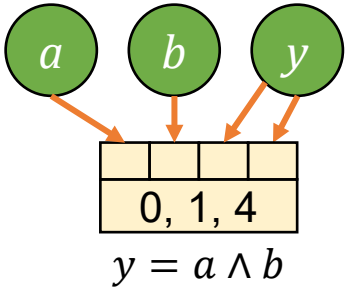
NchooseK Semantics

- Only one basic primitive: $nck(N, K)$
  - N is a multiset of variables, e.g., $\{a, a, b\}$
  - K is a set of numbers, e.g., $\{1, 3\}$
- Interpretation: "I want any $k \in K$ of the Boolean variables listed in N to be TRUE"
- Examples:
  - $nck(\{a, b\}, \{1\})$: "I want exactly one of a and b to be TRUE (i.e., $a \neq b$)"
  - $nck(\{a, b\}, \{0, 2\})$: "I want either both or neither of a and b to be TRUE (i.e., $a = b$)"
  - $nck(\{a, b, y\}, \{0,2\})$: "I want either zero or two of a, b, and y to be TRUE (i.e., $y = a \oplus b$)"
    - Only FFF, FTT, TFT, and TTF have either 0 or 2 TRUE



Shared Variables

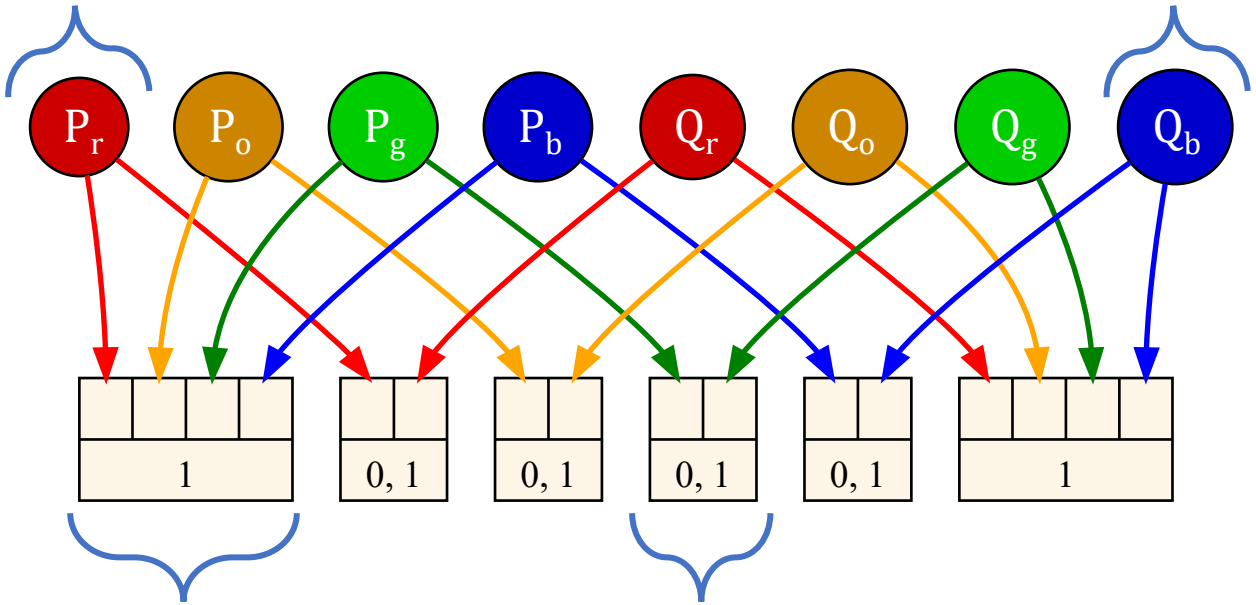
- Variables that are shared within or across nck constraints will be assigned the same value
- Example of using a variable more than once within a constraint:
  - $nck(\{a, b, y, y\}, \{0,1,4\})$: "I want exactly 0, 1, or 4 of a, b, y, and y to be TRUE, and the two ys must have the same value (i.e., $y = a \wedge b$)"
    - Only FFFF, FTFF, TFFF, and TTTT honor 0, 1, or 4 TRUE and have the last two values equal
- Example of using a variable more than once across constraints:
  - $nck(\{P_r, P_o, P_g, P_b\}, \{1\}) \wedge nck(\{P_r, Q_r\}, \{0,1\}) \wedge nck(\{P_o, Q_o\}, \{0,1\}) \wedge nck(\{P_g, Q_g\}, \{0,1\}) \wedge nck(\{P_b, Q_b\}, \{0,1\}) \wedge nck(\{Q_r, Q_o, Q_g, Q_b\}, \{1\})$
    - Two adjacent regions, P and Q, of a map four-coloring problem



Two Regions from a Map Four-Coloring Problem

Region P is red

Region Q is blue



Region P must have exactly one of the four colors

Regions P and Q cannot both be green

Implementation Overview

NchooseK constraints

Given an upper-triangular $Q \in \mathbb{R}^{N \times N}$, find $\arg\min_x(x^T Q x)$ over all $x \in \{0,1\}^N$

QUBO

(via QAOA)

Quantum annealers (using D-Wave's Ocean)

Circuit-model quantum computers (using IBM's Qiskit)

Classical computers (using Microsoft's Z3 SMT solver)

Why High-Performance Computing is Required

- Convert constraint → truth table → QUBO
  - Use a constraint-programming solver to find the QUBO matrix
- May need to augment the truth table with additional columns
  - Want to add as few columns as possible because columns cost qubits
  - Minimum number of additional columns is unknown
  - Boolean values with which to populate the additional columns are unknown
- Challenge
  - There are an exponential number of ways to populate those additional columns
  - Specifically, $2^{c \cdot 2^{|N|}}$ possibilities for a constraint $nck(N, K)$ with c additional columns
  - Intractable with brute force

Why High-Performance Computing is Required (cont.)

- Very tail-heavy distribution of execution times for converting constraints to optimal QUBOs, even when using a sophisticated CP solver
  - In most cases, QUBO generation is fast (a fraction of a second)
  - In a few cases, QUBO generation takes seconds or minutes
  - In rare cases, QUBO generation takes many, many hours
- Solution (rather, workaround): Use an HPC system to precompute many QUBOs in parallel
  - Store results in a database for future use
  - For the rare, super-slow cases, have each core work on the same problem but with a different ordering of the search space

Why High-Performance Computing is Required (cont.)

- Example: Convert $nck([A,B,C,D,E,F,G,H],\{0,3,4,5,6,7\})$ to a QUBO
- After 25 hours (!) running on 300+ cores, the following solution was found:

$$
Q = \begin{array}{c}
\begin{matrix} A & B & C & D & E & F & G & H & \alpha & \beta & \gamma \end{matrix} \\
\begin{pmatrix}
-9 & 15 & 4 & 4 & 4 & 3 & 4 & 4 & -17 & 11 & -22 \\
0 & -14 & 15 & 15 & 15 & 11 & 15 & 15 & -63 & 43 & -83 \\
0 & 0 & -9 & 4 & 4 & 3 & 4 & 4 & -17 & 11 & -22 \\
0 & 0 & 0 & -9 & 4 & 3 & 4 & 4 & -17 & 11 & -22 \\
0 & 0 & 0 & 0 & -9 & 3 & 4 & 4 & -17 & 11 & -22 \\
0 & 0 & 0 & 0 & 0 & -7 & 3 & 3 & -13 & 8 & -16 \\
0 & 0 & 0 & 0 & 0 & 0 & -9 & 4 & -17 & 11 & -22 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -9 & -17 & 11 & -22 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 83 & -47 & 93 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -15 & -62 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 122
\end{pmatrix}
\end{array}
$$

Circuit Synthesis

- Given a unitary matrix, generate a high-quality quantum circuit
  - E.g., short circuit depth
- Given a quantum circuit, generate a superior quantum circuit that performs the same function
- Extremely time-consuming task
- Use HPC resources to accelerate the search for quality circuits

BQSKit

- **QSearch**: Optimal depth synthesis up to four qubits
- **LEAP**: Best quality of solution synthesis up to six qubits
- **QFAST**: Scales good solution quality synthesis up to eight qubits
- **QGO**: Optimizing compiler combining partitioning and synthesis
- **QUEST**: Scalable circuit approximations
- **QFactor**: Fastest quantum circuit optimizer using tensor networks

Debugging Quantum Applications

- Not possible to stop quantum execution, inspect and manipulate state, and resume execution
  - Measurement collapses the wave function
- Instead, co-opt quantum simulators for use as quantum debuggers
  - Add support for setting breakpoints and watchpoints, single-stepping, querying/modifying the state vector, and other features helpful for understanding
- As qubit counts increase, time and/or memory requirements increase exponentially
- HPC-based quantum simulators can enable debugging of larger quantum systems than desktop-based quantum simulators

Summary

- Quantum computing and high-performance computing are mutually beneficial
- Symbiosis is not limited to using quantum computing as an accelerator for certain subroutines in an HPC program
- Examples considered
  - Quantum optimization to suggest the most fruitful HPC-based simulation to run next
  - Quantum optimization for improving parallel task placement in an HPC application
  - Using HPC resources to precompute costly code transformations for use in compilation of code for quantum computers
  - Using HPC resources to generate highly optimized quantum circuits
  - Simulating quantum circuits on an HPC system to debug quantum applications
- As high-performance computers and quantum computers become increasingly integrated, more opportunities for joint usage will assuredly arise