Quantum computing promises new opportunities for solving hard computational problems, but harnessing this novelty will require breakthrough concepts in the design, operation, and application of computing systems. In this talk, we define some of the challenges facing the development of quantum computing systems as well as software-based approaches that can be used to overcome these challenges.
The Future of Computing

- Social concerns driving persistent demand for computational power
  - Security, Commerce, Situational Awareness
  - Sensing, Data Mining, Artificial Intelligence
  - Scientific Discovery, Research, Engineering

- New computational power from advanced design, manufacturing, programming
  - Multicore and parallel programming paradigms
  - New materials and device concepts
  - Hardware-software co-design

- Future devices must control quantum effects
  - Device features already at nanoscale dimensions; 10 nm CMOS in 2016, 2017
  - Quantum perturbations of short channel effects
After a glorious 50 years, Moore’s law—which states that computer power doubles every two years at the same cost—is running out of steam…

…Quantum computers could offer a giant leap in speed—but only for certain applications.

.. A working quantum computer would be a boon—but no one is sure how much of one.

http://www.economist.com/technology-quarterly/2016-03-12/after-moores-law
Timeline of Quantum Computing

• Early 1990’s, quantum computing was codified to harness capabilities of quantum physics
  – Compute with atomic, molecular, optical coherence
  – Use “inherent parallelism” of quantum systems
  – Exponential speed ups over select classical algorithms

• For 20 years, most quantum technologies have remained in the proof of concept phase
  – R&D with significant basic research investments
  – Only a few mature examples, QRNG, QKD, D-Wave
  – Left with a large, diverse quantum technology base

• In the 2010’s, research and development began to address system-level concerns
  – Microarchitecture: instruction sets, layout
  – Programming: logical, physical
  – Macroarchitecture: technology, integration
  – Performance: costs, efficiency, stability
Scientific Applications of Quantum Computing

- Physical Modeling and Simulation
  - Computational Chemistry, Material Science
  - High-energy Physics, Quantum Field Theory

- Applied Mathematics
  - Optimization, Graph Theory, Linear Algebra
  - Placement, Scheduling, Routing
  - Sampling, Search, Random Numbers

- Analysis
  - Situational Awareness, Pattern Matching
  - Machine Learning, Deep Belief Networks
  - Scientific Data Mining, Anomaly Detection

\[ H = -\sum_{i=1}^{n} \frac{\hbar^2}{2\mu_i} \nabla_i^2 - \sum_{i \neq j}^{n} e_i e_j r_{ij} \]

Electronic Structure Calculations

\[ E(\sigma) = J_0 + \sum_i J_i S_i(\sigma) + \sum_{i<j} J_{ij} S_i(\sigma) S_j(\sigma) + \ldots \]

Ground state sampling
Progress in Quantum Processing Units

• Programmable chips have been realized
  – Proof-of-principle demonstrations
  – Superconducting Josephson junctions
  – Doped semiconductors quantum dots
  – Electromagnetically trapped Ions

• Quantum algorithms have been implemented
  – Search, Factoring, Chemistry, Machine Learning
  – Multiple operations acting on multiple qubits
  – Correct results obtained for simple cases with moderate statistical errors
  – Logical operations not yet fault-tolerant

• Low-level quantum instruction control
  – Early infrastructure for translating high-level languages into intermediate representations
  – Variety of programming approaches
Resource Hierarchy: The Temple of Doom

• Advances in ‘system’ integration have begun
  – Scaling up to multi-qubit, fault-tolerant operations
  – Battling noise and physical imperfections
  – Quantum error correction for fault-tolerant operation

• Daunting task to abstract interfaces, layers
  – Resources balloon with conservative estimates
  – Wide open area of design and trade off studies
  – Verification and validation of behavior difficult

• We need methods to measure performance
  – Benchmark emerging devices
  – Verify and validate claims
  – Relativize performance of components

Layered Architecture

Proposed stack for a fault-tolerant circuit-based system
Credit: Van Meter and Horsman (2013)
HPC as a Use Case for QPU’s

- **Massive parallel processing**
  - Support complex, large-scale problems that have significant time and memory demands
  - Application codes need to be parallelized and capable of utilizing distributed resources

- **State of the art in HPC**
  - Current systems 10-20 petaflops
  - Plans indicate exaflops by early ‘20’s

<table>
<thead>
<tr>
<th>System</th>
<th>Peta-flops</th>
<th>Memory (TiB)</th>
<th>Power (MW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tianhe-2</td>
<td>33.8</td>
<td>1,375</td>
<td>17.6</td>
</tr>
<tr>
<td>Titan</td>
<td>17.5</td>
<td>693.5</td>
<td>8.2</td>
</tr>
<tr>
<td>Sequioa</td>
<td>17.1</td>
<td>1,500</td>
<td>7.9</td>
</tr>
<tr>
<td>K</td>
<td>10.5</td>
<td>1,500</td>
<td>12.6</td>
</tr>
<tr>
<td>Mira</td>
<td>8.5</td>
<td>768</td>
<td>3.9</td>
</tr>
</tbody>
</table>

Top 5 HPC systems ranked by performance on LINPACK benchmark
Hybrid High Performance Computing

- **HPC Accelerator Architectures**
  - Heterogeneous, hierarchical systems use specialized processors to *accelerate*
  - GPU’s – thousands of highly efficient cores
  - Xeon Phi’s – hundreds of coherent vector units
  - Other considerations: cost, power, acceptance

- **Quantum processing units (QPU’s) may be suitable as HPC accelerators**
  - This creates a hybrid computational model
  - What functions do quantum processors excel at and when should they be used?
  - Overhead to switch computational models?
  - What are the behavioral and functional requirements placed on the processor?
  - How will we benchmark these systems?
High-Performance Computing with QPUs

• Integrating QPU’s with conventional HPC creates a mixed computational model
  – A deterministic RAM-based controller driving a probabilistic processor

- **Shared Memory Model**
  - Asymmetric multiprocessing system
  - Simpler scheduling and programming models

- **Shared Resource Model**
  - Few QPU’s shared by multiple nodes
  - Distributed but synchronized programs
  - Balance between quantum and HPC designs

- **Quantum Accelerator Model**
  - Each Node has a dedicated QPU
  - Fits existing distributed accelerator design
  - Supplemented with quantum network
Forecasting System Performance

• Key characteristics for evaluating QPUs are the same as for other HPC components

\[ \text{rate} \times \text{time} = \text{money} \]

• Absolute measures are the same, but relative measures are different
  – Probabilistic success is a statistical measure that requires sampling
  – Quantum computing does not always have a good metric for operations, e.g., Q-FLOPS?

<table>
<thead>
<tr>
<th>Quality</th>
<th>Absolute</th>
<th>Relative</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Time to solution</td>
<td>FLOPS</td>
<td>Logical operations per sec (LOPS)</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Power Consumption</td>
<td>FLOP / Watt</td>
<td>LOP / Watt</td>
</tr>
<tr>
<td>Cost</td>
<td>Budget</td>
<td>FLOP / Dollar</td>
<td>LOP / Dollar</td>
</tr>
<tr>
<td>Scaling</td>
<td>Size/Time</td>
<td>T1/(N*TN)</td>
<td>Control line bottleneck</td>
</tr>
<tr>
<td>Portability</td>
<td>User Adoption</td>
<td>Lines of code</td>
<td>Lines of code, Language constructs</td>
</tr>
</tbody>
</table>

• How do we quantify QPU performance for HPC stakeholders?
  – Identify levels of abstraction, technology dependencies, use cases, bottlenecks
ORNL Quantum Computing Institute

• ORNL interaction point for resources in quantum computing
  – “to fosters collaborations that promote the use of theory, computation, and experiment for research and development of quantum computing system”

• The QCI leverages expertise across ORNL in the following disciplines:
  
  Computer Science  Material Science
  Mathematics  Physics
  Modeling and Simulation  Imaging and Characterization
  Experimental Studies  Electrical Engineering

• Membership includes over 50 staff, associates, and students

• Seminars, mailing list, newsletter, solicitations, conferences, publications
Software Ecosystems for Quantum Computing

• **Software is needed to address multiple aspects of quantum computing**
  – Applications and programming
  – Execution and run-time
  – Device and Architecture design

• **Modeling and simulation provides a useful proxy for hardware QPU’s**
  – Separation of concerns isolates functional requirements, manages complexity
  – Quantum mechanics limits the range of exact simulation
Models of hybrid HPC architectures help stakeholders set expectations, planning

- Priority is performance models at the node-level
- This works for classical parallelism, domain decomposition methods
- Quantum parallelism across nodes is more complicated scaling

Two problems need to be solved

- **Forward problem**: What is the performance of a fixed system design? Time-to-solution?
- **Inverse problem**: What system meets given performance requirements?
Device Modeling and Simulation

- Phosphorous donor atoms are embedded in a matrix of isotopically pure silicon
  - Qubit encoded in the nuclear spin state of the donor atom
- $^{31}\text{P}$ nuclear state is effectively isolated in the vacuum of a zero-spin $^{28}\text{Si}$ matrix
  - Dominant loss from isotopic impurities, magnetic field fluctuations, charge-traps
- High-fidelity model of a two-qubit device
  - Integrate electromagnetic field solvers with quantum chemistry calculations
  - Density function theory (DFT) orbital calculations to recover hyperfine coupling

Electron orbitals in phosphorous doped silicon

Integrated nanoscale electronic control gates

Illustration: Bryan Christie Design
Device Modeling and Simulation

- Phosphorous donor atoms are embedded in a matrix of isotopically pure silicon
  - Qubit encoded in the nuclear spin state of the donor atom
- $^{31}$P nuclear state is effectively isolated in the vacuum of a zero-spin $^{28}$Si matrix
  - Dominant loss from isotopic impurities, magnetic field fluctuations, charge-traps
- High-fidelity model of a two-qubit device
  - Integrate electromagnetic field solvers with quantum chemistry calculations
  - Density function theory (DFT) orbital calculations to recover hyperfine coupling
Device Modeling and Simulation

- Electron orbital density for $^{31}\text{P}$ in $^{28}\text{Si}$ nanocrystal (3 nm)
  - DFT calculations
  - 80% charge isosurface
  - Uniform field Z-field
  - Hydrogen-like orbital
Device Design Sequence

- Flow Chart of Design Process

1. System Specification

2. Architectural Design

3. Functional Design

4. Logic Design

5. Circuit Design

6. Physical Design

7. Fabrication

8. Testing

Layout Verification

Logic Verification

Circuit Verification
Device Design Sequence

- Flow Chart of Design Process

1. System Specification
   - Requirements Management Tool

2. Architectural Design
   - Design Capture Tool

3. Functional Design
   - Behavioral Simulator
   - Logic Synthesis
   - Circuit Synthesis

4. Logic Design
   - Logic Verification
   - Circuit Simulator
   - Circuit Synthesis

5. Circuit Design
   - Circuit Verification
   - Layout Synthesis Tool

6. Physical Design
   - Layout Verification
   - Circuit Extractor Tool

7. Fabrication

8. Testing
Device Design Sequence

- Flow Chart of Design Process

1. System Specification
   - Requirements Management Tool

2. Architectural Design
   - Design Capture Tool

3. Functional Design
   - Behavioral Simulator

4. Logic Design
   - Logic Verification
   - Circuit Simulator

5. Circuit Design
   - Circuit Verification
   - Layout Synthesis Tool

6. Physical Design
   - Layout Verification
   - Circuit Extractor Tool

7. Fabrication

8. Testing
Device Design Sequence

- Flow Chart of Design Process

1. System Specification
   - Requirements Management Tool
2. Architectural Design
   - Design Capture Tool
3. Functional Design
   - Behavioral Simulator
   - Logic Synthesis
   - Circuit Synthesis
4. Logic Design
   - Logic Verification
   - Circuit Simulator
5. Circuit Design
   - Circuit Verification
   - Layout Synthesis Tool
6. Physical Design
   - Layout Verification
   - Circuit Extractor Tool
7. Fabrication
8. Testing
Abstract Machine Models for Hybrid HPC

- Models of QPU’s and controllers
  - Use SST to model nodes, memory hierarchy, network, execution
  - Custom QPU, QRAM components

- Discrete event simulations to account for data movement, latency, power
  - Profile application performance against small scale kernels
  - Extrapolate run-time behavior using scaling tools, e.g. xSim

- Early observations stress memory management
  - Fault-tolerant operations emphasize data movement, classical processing

```c
#ifdef USING_CLASSICAL

  ChemPS2::FCI(Ham, Nel_up, Nel_down, Irreps[counter],
               maxMemWorkMB, FCIverbose);

  ...
#endif

#elifdef USING_SSTMAC

  /* pretend computation takes 0.1us per iteration */
  nloop = nintervals/size + ((rank < max_rank);
  SSTMAC_compute(nloop * 1e-7);

#endif
```
Interconnect Design and Simulation

- Quantum interconnects are essential for advanced QPU-QPU programming
  - Distributed QPU’s for fault-tolerant controllability, infrastructure limitations
  - Software-defined networking may help manage heterogeneous devices

```
<table>
<thead>
<tr>
<th>QNIC 1</th>
<th>QNIC 2</th>
<th>QNIC 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 1</td>
<td>Node 2</td>
<td>Node 3</td>
</tr>
<tr>
<td>CNIC 1</td>
<td>CNIC 2</td>
<td>CNIC 3</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Classical Switch</th>
<th>Controller</th>
<th>Quantum Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical network traffic</td>
<td>Control messages</td>
<td>Quantum network traffic</td>
</tr>
<tr>
<td>Switching messages</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Quantum Network Simulation

- Node and switch software and middleware layers operate independently of the hardware, so we use numerical simulations to test them.

Mininet environment with nodes and switches connected to quantum network simulator

Wireshark running on mininet nodes and DSPY running on quantum simulator capture complete quantum network behavior

```json
"topic": "client_tx", "data": {"method": "tx", "parameters": [167772682, "chpext", "init 2\n 0\n 0\n 0\n 1\n 0", "\n"], "time": "2016-03-04 14:43:13.360873"}

"topic": "sabot_response", "data": {"result": 636}, "time": "2016-03-04 14:43:13.361554"

"topic": "sabot_request", "data": {"method": "measure_state", "parameters": [1, 636, "chpext", "c 0\n 1\n 0\n 0\n 0\n 1", 10]}, "time": "2016-03-04 14:43:13.361609"

"topic": "client_tx", "data": {"method": "tx", "parameters": [167772682, "chpext", "init 2\n 0\n 0\n 0\n 0\n 0", "\n"], "time": "2016-03-04 14:43:13.362130"}
```
Split-execution Computing Model

• **Software execution split across different hardware**
  - Example: CPU-GPU accelerators
  - Off-load certain computations to performance optimal platform
  - Requires code refactoring to utilize quantum primitives
  - What software requirements emerge for the application?

• **DW2X is a third generation quantum processor**
  - Programmable superconducting integrated circuit
  - 1152-qubit register in a 2D Chimera layout
  - EM shielding, UHV, cooled to 14mK
  - A special-purpose optimization solver

*Unit cell of flux qubits Credit: Harris et al., Phys Rev B (2010)*
QPU Integration Design

- Loose integration vs. tight integration
QPU Integration Design

- Loose integration vs. tight integration
Programming the Processor

- Logical problem statements are cast as quadratic optimization problems
  - QUBO problems are mapped to Ising models
- Computation by quantum annealing
  - Prepares the energetic ground state of $H(t)$
  - Dynamics driven by energetic minimization

$$f(x_1, \ldots, x_n) = c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{1 \leq i < j \leq n} c_{ij} x_i x_j$$

$$H(t) = -\sum_i h_i Z_i - \sum_{i,j> i} J_{ij} Z_i Z_j - \sum_i f(t) X_i$$

*Schematic of Ising model energy landscape*
Simple Split-Execution Example

- Simplest Example: A single thread pushes a task to a QCP (quantum coprocessor)

1. Send program
2. Parse program
3. Compile program to ISA
4. Issue instructions
5. Parse instructions to gates
6. Execute gates
7. Receive readout
8. Parse readout
9. Parse results
10. Return solution
Simple Split-Execution Example

- Simplest Example: A single thread pushes a task to a QCP (quantum coprocessor)

1. Send program
2. Parse program
3. Compile program to ISA
4. Issue instructions
5. Parse instructions to gates
6. Execute gates
7. Receive readout
8. Parse readout
9. Parse results
10. Return solution
Integrated Development Environments

- Eclipse-based development environment for reducing problems to programs
  - Multi-stage parameter setting, graph processing
  - Map QUBO to ISING,
  - Embed ISING into Hardware
  - Server-client connection with QPU, simulator
  - Remote connection to D-Wave QPU

Compilation sequence for quantum program

1. Define QUBO Problem
2. Map Logical Ising Hamiltonian
3. Embed Physical Ising Hamiltonian
4. Initialize Computational Registers
5. Execute Annealing Schedules
6. Readout Computational Result
7. Return QUBO Solution
ICE-QI

- Interactive programming environment
  - End-to-end workflow for solving with D-Wave
  - Job management using DWS SAPI
  - Efficient collection and storage of results, program data, and trends

The D-Wave CAM Launcher takes a set of memories, embeds them in a D-Wave Chimera graph, and executes them on a D-Wave adiabatic quantum computer.
Model of an Optimization Application

- ASPEN modeling language to represent machine and program
- Machine includes CPU-QPU interactions, clock rates, memory
  - Highlights 3 execution stages
  - Stage 1: Pre-processing
  - Stage 2: Execution
  - Stage 3: Post-processing
- Analysis extracts timing results for problem size, accuracy goals

```plaintext
include memory/ddr3_1066.aspen
include sockets/intel_xeon_e5_2680.aspen
include sockets/nvidia_m2090.aspen
include sockets/dwave_vesuvius_20.aspen

machine SimpleNode
{
  [1] SIMPLE nodes
}

node SIMPLE
{
  [1] intel_xeon_e5_2680 sockets
  [1] nvidia_m2090 sockets
  [1] DwaveVesuvius20 sockets
}
Model of an Optimization Application

- **ASPEN modeling language to represent machine and program**
  - Machine includes CPU-QPU interactions, clock rates, memory
    - Highlights 3 execution stages
    - **Stage 1:** Pre-processing
    - **Stage 2:** Execution
    - **Stage 3:** Post-processing
  - Analysis extracts timing results for problem size, accuracy goals

```plaintext
include memory/ddr3_1066.aspen
include sockets/intel_xeon_e5_2680.aspen
include sockets/nvidia_m2090.aspen
include sockets/dwave_vesuvius_20.aspen

model Stage1
{
  param LPS = 0 // Input Parameter
  param Ising = LPS^2
  param NH = LPS
  param EH = NH*(NH-1) / 2
  param M = 12
  param N = 12
  param NG = 8*M*N
  param EG = 4*(2*M*N - M - N) + 16*M*N
  param EmbeddingOps = (EG+NG*log(NG))*(2*EG)*NH*NG
  param ParameterSetting = LPS^3

  // Hardware constants for DW2 in microseconds
  param StateCon = 252162
  param PMMSW = 33095
  param PMMElec = 0
  param PMMChip = 11264
  param PMMTherm = 10000
  param SWRun = 4000
  param ElecRun = 9052
  param ProcessorInitialize = StateCon+PMMSW+PMMElec+PMMChip+PMMTherm+SWRun+ElecRun
}```
Model of an Optimization Application

- **ASPEN modeling language** to represent machine and program

- **Machine includes CPU-QPU interactions, clock rates, memory**
  - Highlights 3 execution stages
  - Stage 1: Pre-processing
  - Stage 2: Execution
  - Stage 3: Post-processing

- **Analysis extracts timing results for problem size, accuracy goals**

```plaintext
include memory/ddr3_1066.aspen
include sockets/intel_xeon_e5_2680.aspen
include sockets/nvidia_m2090.aspen
include sockets/dwave_vesuvius_20.aspen

definition LPS = 100

model-stage1
  param LPS = 0 // Input Parameter
  param Ising = LPS^2

model-stage2
  param Success = 0.9999
  param Accuracy = 0
  param AnnealReadResults = 320
  param AnnealThermalization = 5

kernel-stage2-processing
  execute mainblock2[1]
    // Number of QPU calls
    QuOps = floor(1/(Accuracy/100))/floor(1-Success)
  execute mainblock3[1]
    // Readout time
    microseconds [AnnealReadResults]
  execute mainblock4[1] { // Initialization time
    microseconds [AnnealThermalization]
  }
```
Model of an Optimization Application

- ASPEN modeling language to represent machine and program
- Machine includes CPU-QPU interactions, clock rates, memory
  - Highlights 3 execution stages
  - Stage 1: Pre-processing
  - Stage 2: Execution
  - Stage 3: Post-processing
- Analysis extracts timing results for problem size, accuracy goals

```cpp
#include memory/ddr3_1066.aspen
#include sockets/intel_xeon_e5_2680.aspen
#include sockets/nvidia_m2090.aspen
#include sockets/dwave_vesuvius_20.aspen

model Stage1
{
  param LPS = 0 // Input Parameter
  param Ising = LPS^2
}

model Stage2
{
  param Success = 0.9999
  param Accuracy = 0
}

model Stage3
{
  param LPS = 0
  param Success = 0.75
  param Accuracy = 0.99
  param Results = ceil(log(1-(Accuracy))/log(1-Success))
  param Length = LPS
  param SortOps = log(Results) * Results

  data R as Array(Results, LPS)

  kernel FindSolution {
    execute sort [1] {
      loads [Results] of size [4*Length]
      flops [SortOps] as sp
      stores [Results] to R
    }
  }

  kernel main {
    FindSolution
  }
```
Simulated Timings using ASPEN

- Predict timings for each stage at sizes larger than currently possible with hardware
  - Full program execution cycle
  - Compare timings at small scales
  - Identify bottlenecks in methods simulated

![Graphs showing stage 1: Preprocessing, stage 2: Execution, and stage 3: Post-processing](image)

- Stage 1: Preprocessing
  - Model: \( O(\text{size}^3) \)
  - Actual time vs size

- Stage 2: Execution
  - Accuracy: \(~10^{-3}\)

- Stage 3: Post-processing
  - Accuracy: \(~10^{-7}\)
Quantum Run-time Environment

• **Software is the leading bottleneck in the previous execution model**
  — It was a simple approach for run-time compilation
  — Faster, inline libraries are needed to mitigate slowdown from embedding
  — Linkable libraries in fast, compiled language

• **Can we push pre-processing off-line, mitigate costs?**
  — Just-in-time compilation for minor embedding
The Next Generation of QPU’s

• D-Wave provides a significant test platform
  – Exposes programming interfaces, raises benchmark questions
  – Hardware constraints impact programming model

• Future processors will be more complicated
  – Examples: Chips from IBM, Google, IARPA
  – Fault-tolerant protocols developed but not yet tested

• Infrastructure will remain, bulky, extensive, bottleneck
  – State-of-the-art chips are limited by precision of classical signal generators
  – Noise in magnetic, electronics, and optical systems limits controllability of qubits

• Emerging software community
  – QxBranch, Artise-QB, Rigetti, 1QBit
Looking Beyond CMOS Technology for Future HPC

Software Ecosystems for Quantum Computing

- Quantum computing offers a compelling but challenging technology for beyond CMOS
  - It integrates with existing computing paradigm but pushes hardware and software changes
  - Aggressive hardware development is not yet able to inform HPC design decisions
  - Currently there is a large uncertainty in performance due to engineering challenges

- Modeling and simulation offer proxies for performance expectations
  - Results highly dependent on design; extrapolation of behavior is risky
  - Sets a big picture to measure progress
  - Creates an ecosystem for software solutions
Looking Beyond CMOS Technology for Future HPC

Software Ecosystems for Quantum Computing

Travis Humble
Quantum Computing Institute
Oak Ridge National Laboratory

Parts of this presentation were developed in collaboration with these outstanding scientists:


THANK YOU

April 5, 2016 Hanover, Maryland