The Quantum Computing Company™

Simulating a quantum annealer with GPU-based Monte Carlo algorithms

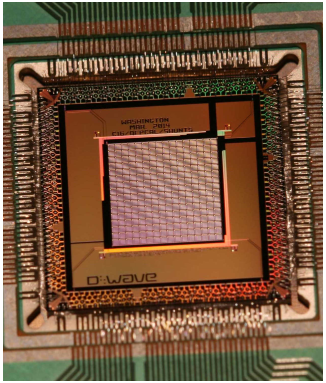
Mayssam Mohammadi Nevisi
Mani Ranjbar
James King
Sheir Yarkoni
Jeremy P. Hilton
Catherine C. McGeoch

April 6, 2016

Introduction

D-Wave QPU

- Quantum annealing chip
- Highly specialized co-processor
- Physical implementation of an NP-hard optimization problem
- Physical heuristic algorithm runs on the chip



Ising Minimization

Given:

- A graph $G = (V, E)$
- A collection of weights $h = \{h_i : i \in V\}$ and $J = \{J_{ij} : (i, j) \in E\}$ (the Hamiltonian)

Assign:

- Values from $\{-1, +1\}$ to n *spin variables* $s = \{s_i\}$

Such that we minimize the *energy function*:

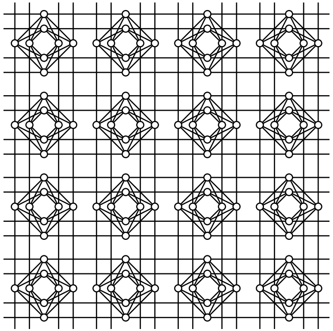
$$E(s) = \sum_{i \in V} h_i s_i + \sum_{(i,j) \in E} J_{ij} s_i s_j.$$

Chimera topology

- C_k is a $k \times k$ grid of dense $K_{4,4}$ "unit cells"

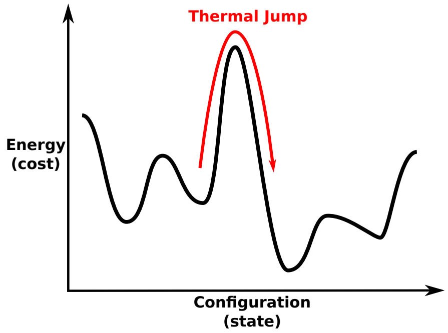
Processor	Topology	Qubits
D-Wave One	C_4	128
D-Wave Two	C_8	512
D-Wave 2X	C_{12}	1152

- Chimera topologies are *bipartite*
- Any graph can be embedded in a Chimera graph via minor embedding



Simulated (Thermal) Annealing

- Heuristic optimization algorithm that simulates classical thermal annealing
- System of spins moves randomly in state space
- Cools slowly from hot (random/explorative) to cold (greedy/exploitative)
- Uses thermal activation to jump over energy barriers

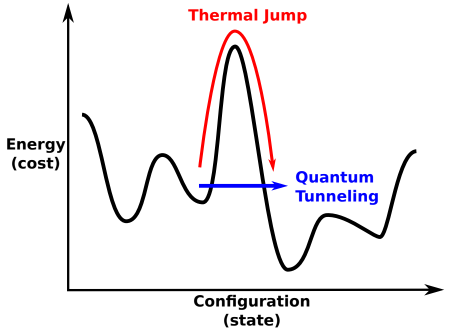


Quantum Annealing

- Quantum annealing (QA) is related to adiabatic quantum computing (AQC)

$$\mathcal{H}(t) = A(t) \cdot \mathcal{H}_{\text{init}} + B(t) \cdot \mathcal{H}_{\text{prob}}.$$

- Takes advantage of thermal activation just like classical annealing
- Also has a new complementary resource: quantum tunneling.

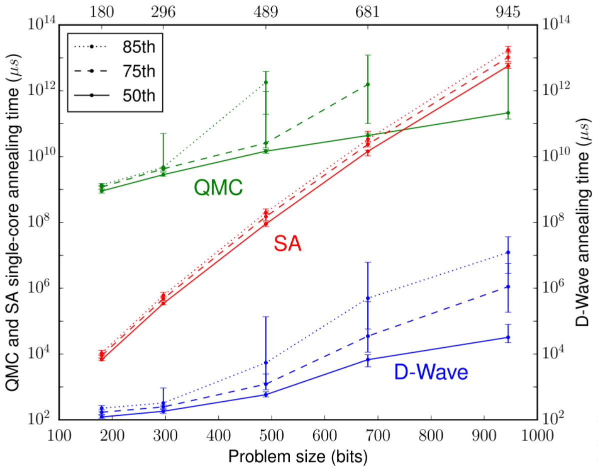


Motivation for GPU Solvers

Why develop optimized GPU implementations?

- Quantum computers are hard to simulate
- Even approximate simulations via Monte Carlo methods can be slow

Between some quantiles and system sizes we observe a prefactor advantage [for D-Wave] as high as 10^8*.*
- Denchev et al. (2015)



Why develop optimized GPU implementations?

- Software solvers slow down our experiments

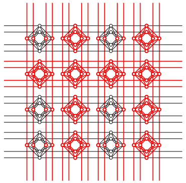
  "This experiment occupied millions of processor cores for several days to tune and run the classical algorithms for these benchmarks."

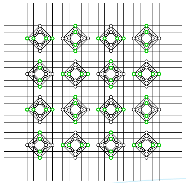
  - Denchev et al. (2015)

- Faster solvers $\longrightarrow$ faster experimental cycle $\longrightarrow$ improved understanding of our chips
- Fast GPU simulation leads to better quantum computers!

Algorithms and GPU suitability

- Good/interesting classical solvers for Chimera Ising problems fall into two categories:
  - Low-treewidth local search
  - Single-spin Monte Carlo algorithms
- Low-treewidth local search is not suitable.
  - Memory requirements are too high
  - Limited parallelizability.
- Single-spin Monte Carlo algorithms are ideal!
  - Very low memory requirements
  - Highly parallelizable.





Algorithms

Simulated Annealing

- Single-spin updates
- Flipping this spin would lead to a change in energy ΔE
- Probability of accepting the spin flip is $\min(1, e^{-\beta \Delta E})$

Algorithm 1 Simulated Annealing

```
for each sample to be taken do
    for $i = 1$ to num_sweeps do
        $\beta$ := betas[i]
        for spin in spins do
            calculate $\Delta E_{\text{spin}}$
            flip spin with probability $\min(1, e^{-\beta \Delta E_{\text{spin}}})$
        end for
    end for
end for
```

samples

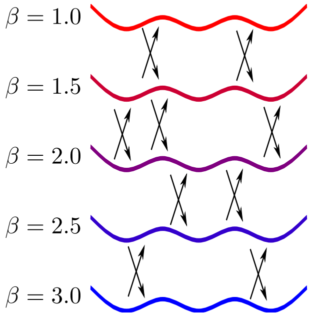
sweeps

spins

bipartite graph means half of the spin updates can be done in parallel

Parallel Tempering

- Instead of one Markov chain that slowly goes from high to low temperature:
  - Use an ensemble of fixed-temperature Markov chains ("replicas")
  - Replicas form a "temperature ladder"
  - Replicas can exchange temperatures with neighbouring chains on the ladder with probability $\min(1, e^{(E_i - E_j)(\beta_i - \beta_j)})$



Approximate Simulations of Quantum Annealing

Quantum Monte Carlo†

- Many replicas of the system (Trotter slices) representing different points in imaginary time
- Path-integral Monte Carlo method
- We implement the 'discrete time' variant

† QMC can reproduce QA equilibrated statistics, but doesn't simulate its dynamics.

Spin Vector Monte Carlo

- Mean-field approximation
- Simulates coherence but no entanglement
- Each spin is represented by an angle

GPU Simulated Annealing Implementation

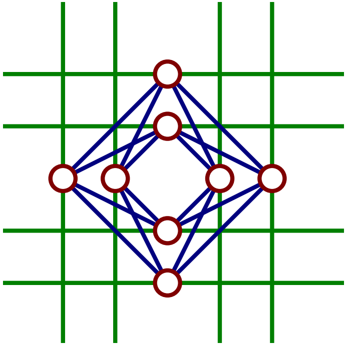
Thread Structure — Hamiltonian

- One unit cell per thread
- Cell Hamiltonian stored as floats in 40 registers

	8	fields (h)
	16	in-tile couplings (J)
+	16	inter-tile couplings[†] (J)
	40	registers

- Compiler uses additional 39 registers per thread

† Each inter-tile coupling is stored in two threads



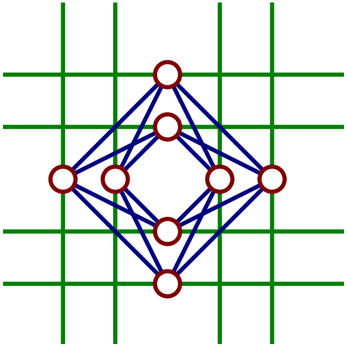
Thread Structure — States

- Each state is $+1$ or -1
- Each state is accessed by multiple threads for energy calculation

States must be stored in shared memory!

- $8k^2$ states per sample
- Storing as floats is faster than packing bits; registers are still the limiting factor†

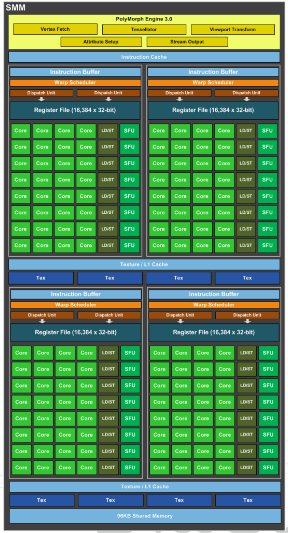
† For parallel tempering and quantum Monte Carlo we pack bits because we have up to 64 replicas



Block Structure

- 79 registers per thread
- k^2 threads per sample
- 65,536 registers per SM (Maxwell)
- Each SM can run $\left\lfloor \frac{65{,}536}{79k^2} \right\rfloor$ samples in parallel

Topology	C_4	C_8	C_{12}
Concurrent samples per SM	51	12	5

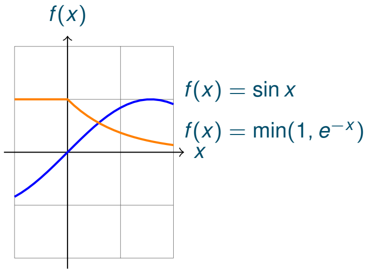


Fast Random Number Generation

- A significant fraction of running time is used to generate random numbers.
- We use xorshift random number generators
  - 2-3 times faster than cuRand
  - Imperfect but still suitable for applications that are not highly sensitive to RNG quality.

Fast Approximations of Mathematical Functions

- Exponentiation is necessary to determine flip probabilities
- Sine and cosine are used in Spin Vector Monte Carlo
- CPU implementations often cache function values in lookup tables
  - Not feasible for GPUs due to memory restrictions
- CUDA to the rescue! Intrinsic fast math functions are:
  - Faster than regular math functions or Taylor approximations
  - Accurate enough for our Monte Carlo algorithms

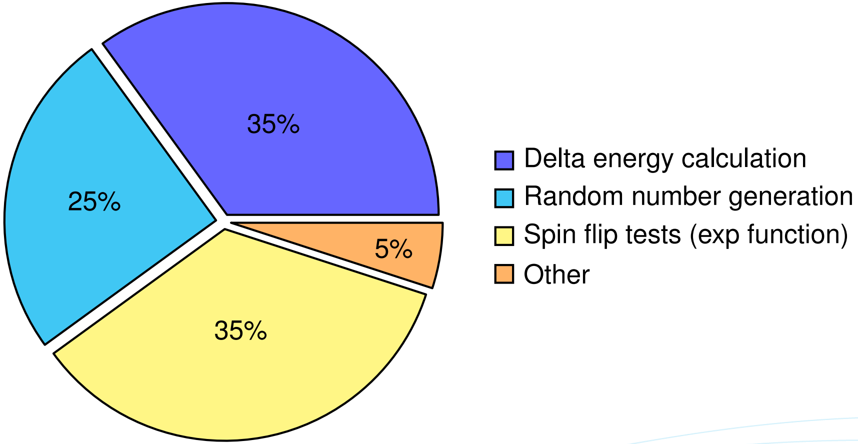


Results

Implementation Speeds

- Code is still being fine-tuned
- Significant speedup over CPU seen in all four algorithms
- Huge spin flip/nanosecond/dollar improvement over CPUs
- Actual numbers to be released in a forthcoming paper

Breakdown of Runtime — Simulated Annealing



Conclusion

Recap

- Quantum processors are very hard to simulate classically
- Monte Carlo algorithms are among the best tractable approximations
- Monte Carlo algorithms with single-spin updates are ideal for GPU
- We can achieve significant speedups even over a more expensive CPU

Looking to the Future

- Future D-Wave chips will be bigger and denser
- Future NVIDIA chips will be bigger and faster (more registers per SM?)
- GPUs should continue to beat CPUs for Monte Carlo algorithms with single-spin updates
- Algorithms with low-treewidth updates unlikely to become feasible for GPUs