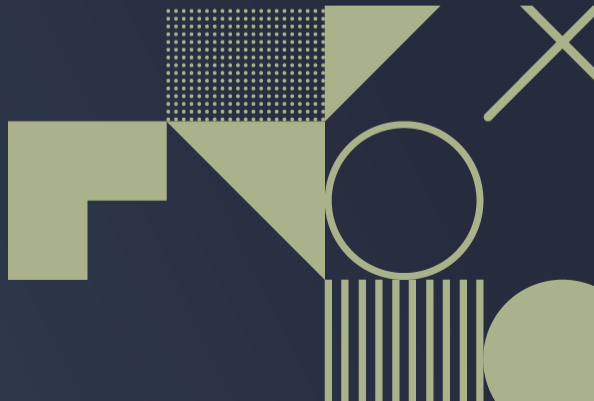
Overcoming Barriers to Scalability in Variational Quantum Monte Carlo

Tianchen Zhao[1], Saibal De[1], Brian Chen[1], James Stokes[2], Shravan Veerapaneni[1,2]

University of Michigan[1]

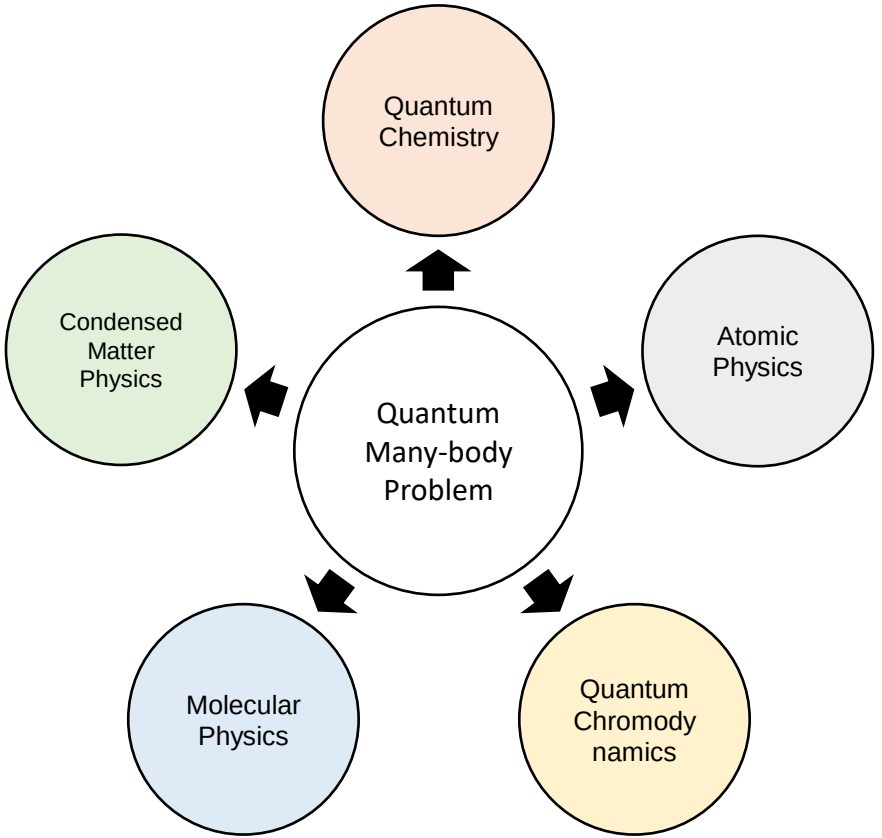
Flatiron Institute - Simons Foundation[2]



Introduction

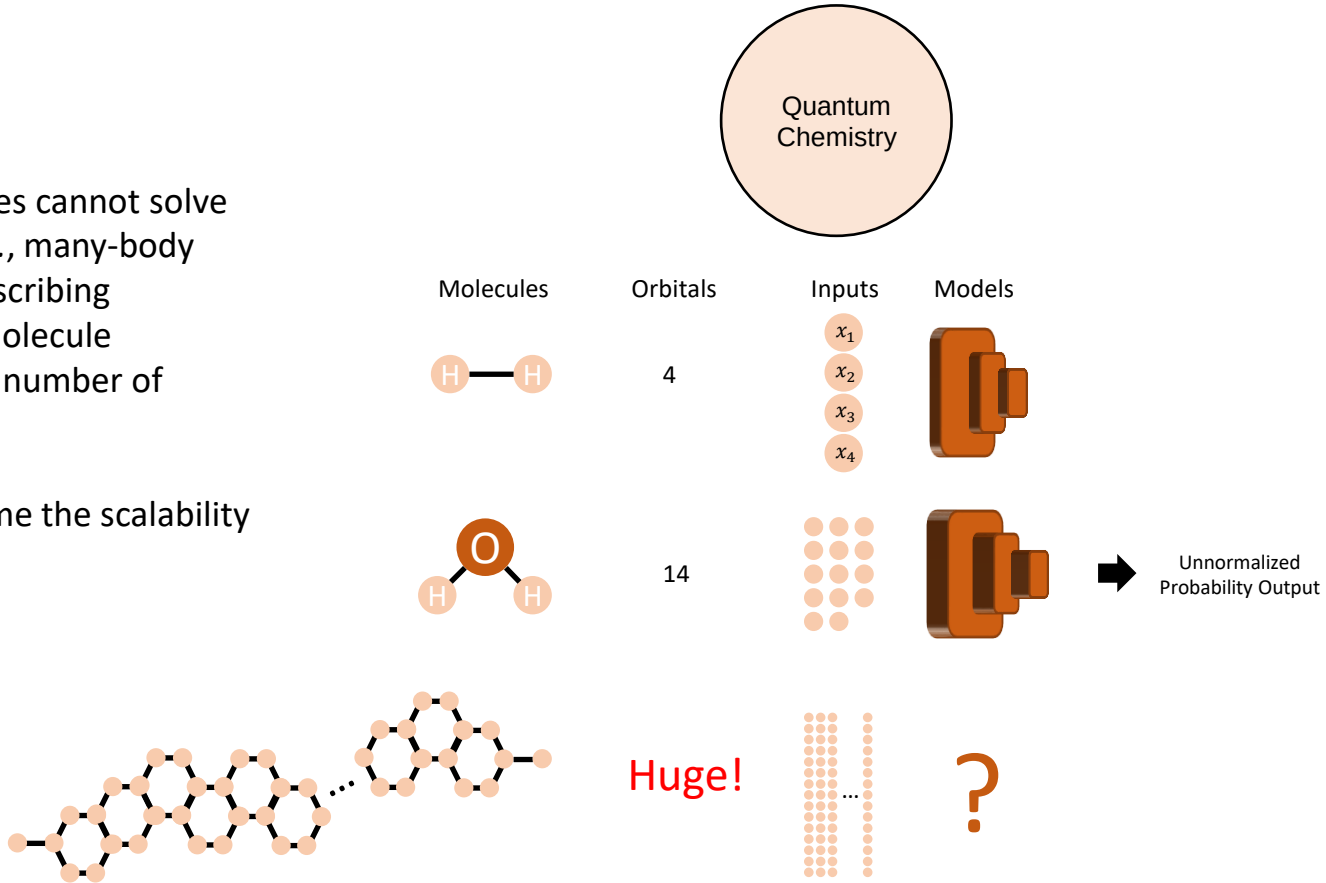
- Quantum many-body system describes a vast category of physical problems involving many interacting particles.
- Variational Quantum Monte Carlo[1] (VQMC) overcomes the curse of dimensionality by sampling and optimizing parametrized quantum state.

[1]Carleo and Troyer, 2017



Motivation

- Existing VQMC approaches cannot solve large-scale problems, *e.g.*, many-body Schrodinger equation describing interactions within the molecule consisting of a very large number of orbiting electrons.
- We would like to overcome the scalability barrier of VQMC.



Variational Quantum Monte Carlo

Sampling

- VQMC transforms the eigenvalue problem into a stochastic optimization problem.
- Direct computation of the energy functional is impractical.
- Approximation is done by averaging over finite samples from the probability distribution.

Schrodinger Equation

$$H\psi=\mathrm{E}\psi$$

Optimization Problem

$$\min_{\psi} L(\psi),\ L(\psi) = \frac{\langle\psi|H\psi\rangle}{\langle\psi|\psi\rangle}$$

Energy Functional

Finite Approximation

$$\mathbb{E}_{x\sim\pi}\left[\frac{(H\psi)(x)}{\psi(x)}\right] \approx \sum_{i=1}^{k} \frac{(H\psi)(x_i)}{\psi(x_i)}$$

$$\pi(x) = \psi(x)^2/\langle\psi|\psi\rangle$$

Variational Quantum Monte Carlo

Optimization

- The wave function is approximated by a neural network ψ parametrized by θ.
- ψ computes the unnormalized amplitudes of wavefunction for all possible xs.
- Approximation is done by MCMC sampling from ψ.

Schrodinger Equation

$$H\psi = \mathrm{E}\psi$$

Optimization Problem

$$\min_{\theta} L(\theta),\ L(\theta) = \frac{\langle \psi_\theta | H \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle}$$

Energy Functional

Finite Approximation

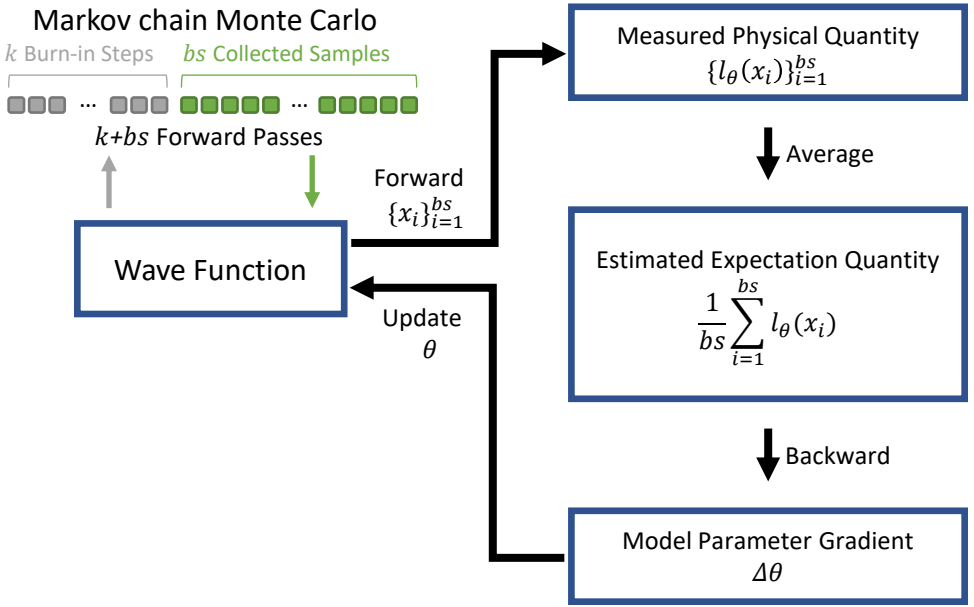
$$\mathbb{E}_{x \sim \pi_\theta}\left[\frac{(H\psi_\theta)(x)}{\psi_\theta(x)}\right] \approx \sum_{i=1}^{k} \frac{(H\psi_\theta)(x_i)}{\psi_\theta(x_i)}$$

Normalizing Factor

$$\pi_\theta(x) = \psi_\theta(x)^2 / \langle \psi_\theta | \psi_\theta \rangle$$

Algorithm Overview

- Perform $k + bs$ forward passes to obtain samples (last bs samples are kept)
- Estimate energy functional with the samples
- Update the model

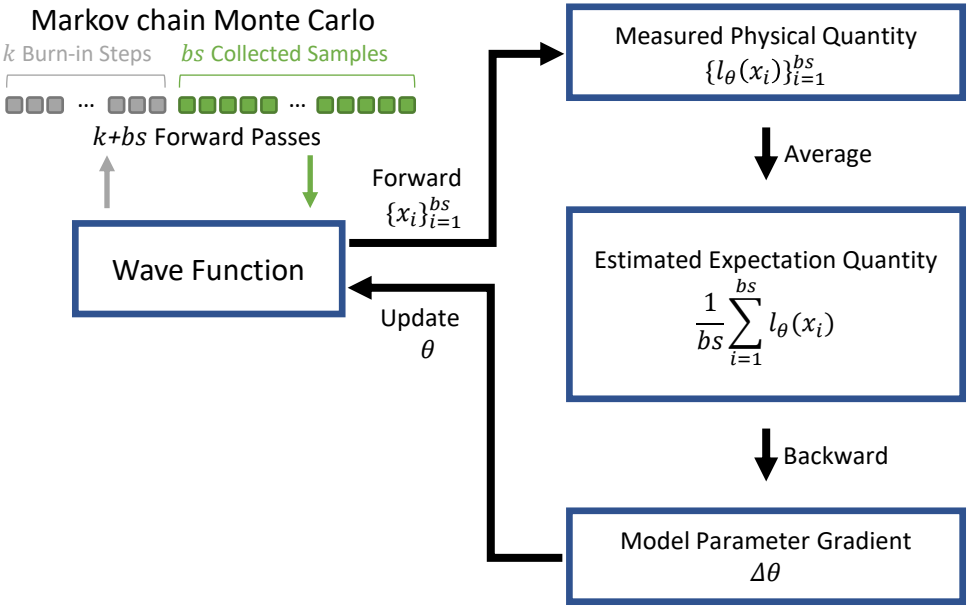


k: Number of Burn-in steps
bs: Batch size

Sampling with MCMC

Cons

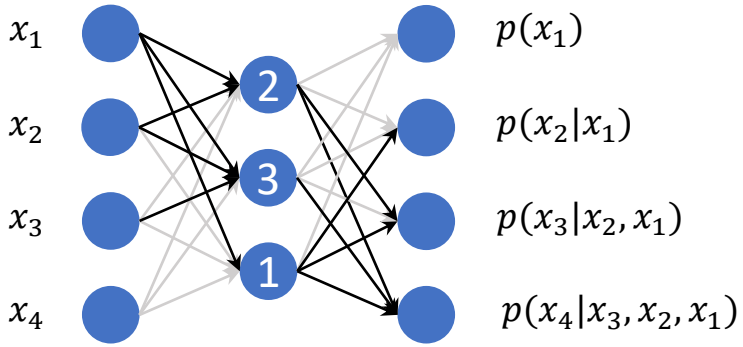
- The burn-in process in MCMC is an inherently sequential task.
- Sampling precise and uncorrelated samples become increasingly difficult for large input dimension.



k: Number of Burn-in steps
bs: Batch size

Autoregressive Sampling with MADE

- Autoregressive sampling supports **efficient** and **exact** sampling.
- MADE models are built on the autoregressive property.



Germain *et al.*, ICML 2015

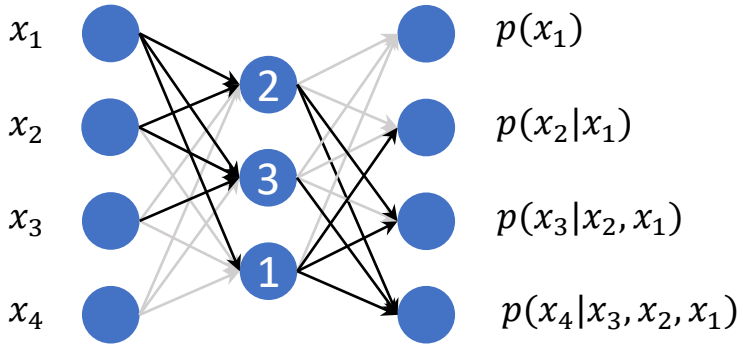
$$p(x) = p(x_4|x_3,x_2,x_1)p(x_3|x_2,x_1)p(x_2|x_1)p(x_1)$$

k: Number of Burn-in steps
bs: Batch size
n: Input dimension

Application of MADE to Quantum Problems

Consider spin configuration $x = \{x_1, x_2, \dots, x_n\}$, then $p(x_k|x_{k-1}, \dots, x_2, x_1)$

- is a number representing the probability of being 1, for spin-1/2 particles;
- is a d-vector representing the probability of the possible values, for system of d local degree of freedom;
- is a 2-vector μ, σ assuming the value is Gaussian(μ, σ^2), for continuous systems.



Germain *et al.*, ICML 2015

$$p(x) = p(x_4|x_3, x_2, x_1)p(x_3|x_2, x_1)p(x_2|x_1)p(x_1)$$

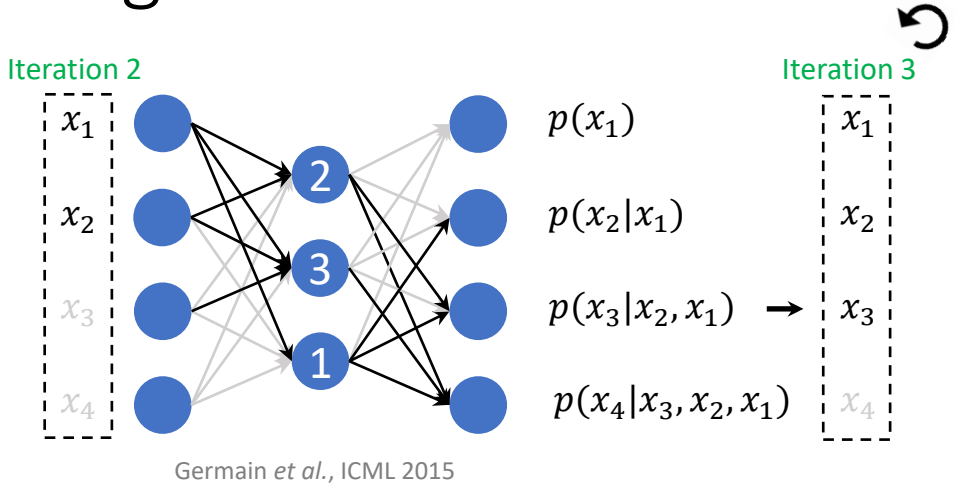
k: Number of Burn-in steps
bs: Batch size
n: Input dimension

Sampling Quantum Configuration

At **Third** Iteration

Current configuration $x = \{x_1, x_2, x_3, x_4\}$

1. Compute $p(x_3|x_2, x_1)$
2. Sample x_3
3. Update $x = \{x_1, x_2, x_3, x_4\}$



Germain *et al.*, ICML 2015

$$p(x) = p(x_4|x_3, x_2, x_1)p(x_3|x_2, x_1)p(x_2|x_1)p(x_1)$$

Taking n iterations to get configurations of dimension n.

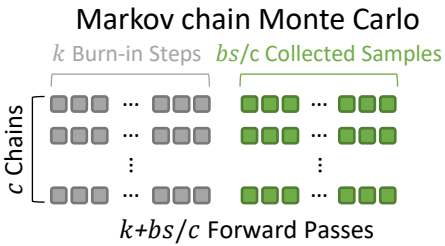
k: Number of Burn-in steps
bs: Batch size
n: Input dimension

Autoregressive Sampling

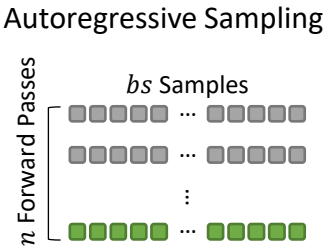
Cons?

- Possibly Greater complexity

$c \times k + bs$ samples computed



n× bs samples computed



k: Number of Burn-in steps
bs: Batch size
n: Input dimension

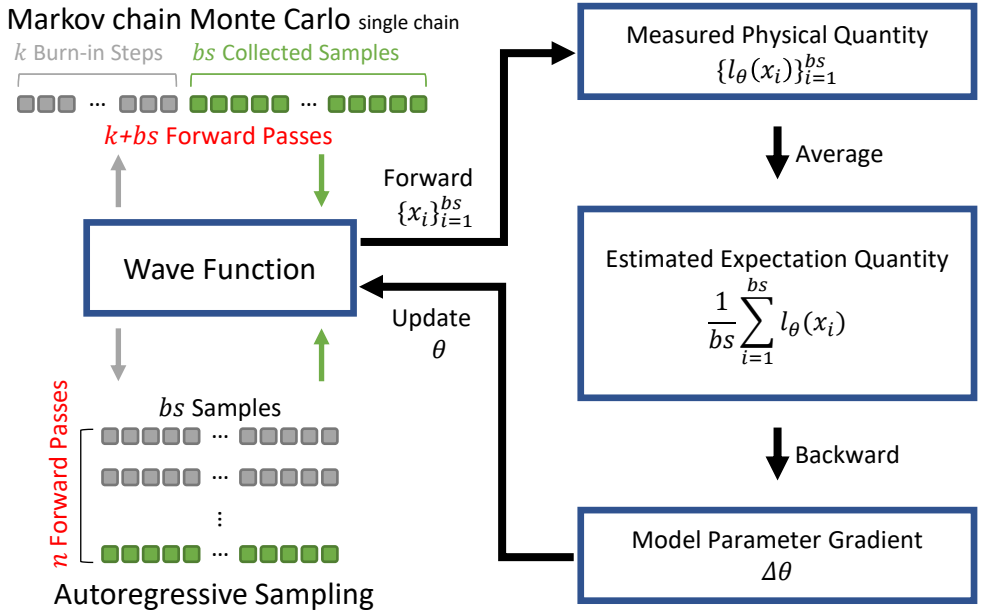
Autoregressive Sampling (AUTO) on GPUs

- MCMC is not guaranteed to converge after $k + bs$ iterations.

  ➡ Fail for large problems

- AUTO is guaranteed to converge after n iterations.

  ➡ Work well for large problems



k: Number of Burn-in steps
bs: Batch size
n: Input dimension

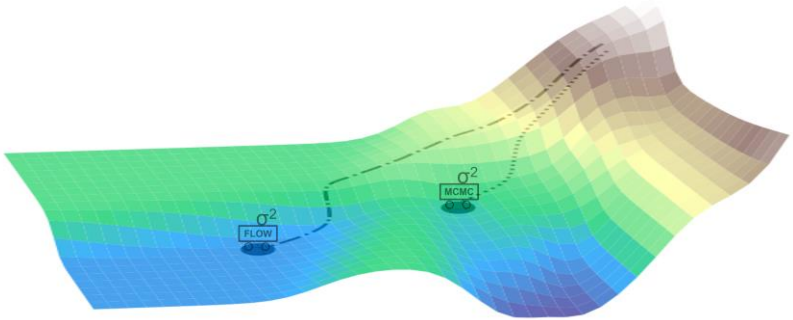
Autoregressive Sampling (AUTO) on GPUs

- MCMC is not guaranteed to converge after $k + bs$ iterations.

  ➡ Fail for large problems

- AUTO is guaranteed to converge after n iterations.

  ➡ Work well for large problems

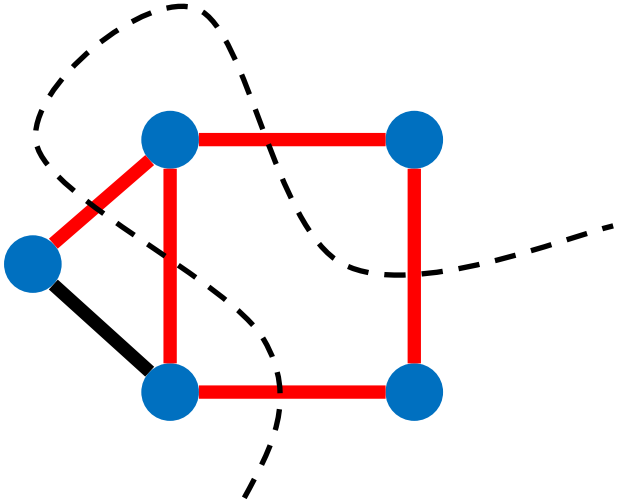


AUTO

1. achieves superior running time efficiency with GPU
2. and (more importantly) works better for large-scale problems!

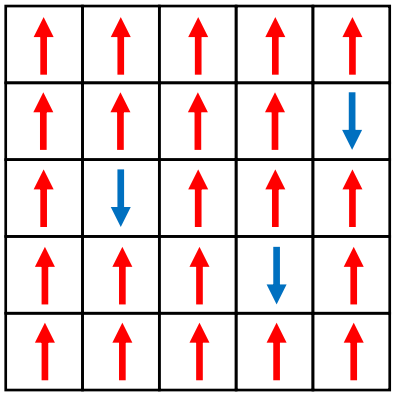
Problems

- **MaxCut** (Classical Problem for which SDP solver exists in the literature)

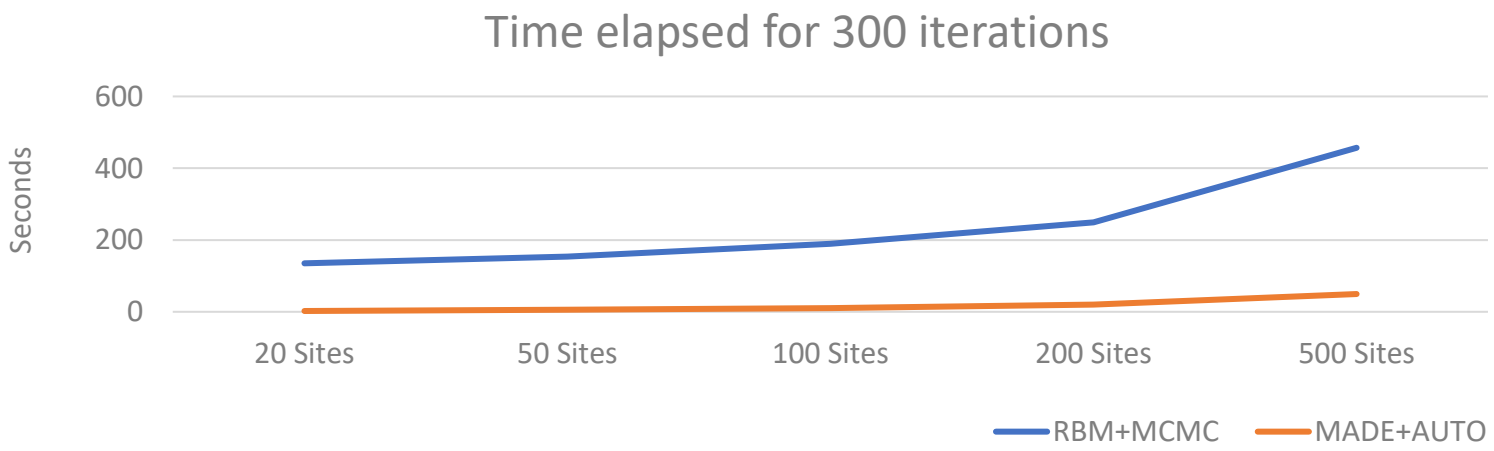


- **Transverse Field Ising** Model[1] (TIM)

[1]Bravyi and Hastings, 2017



Running Time

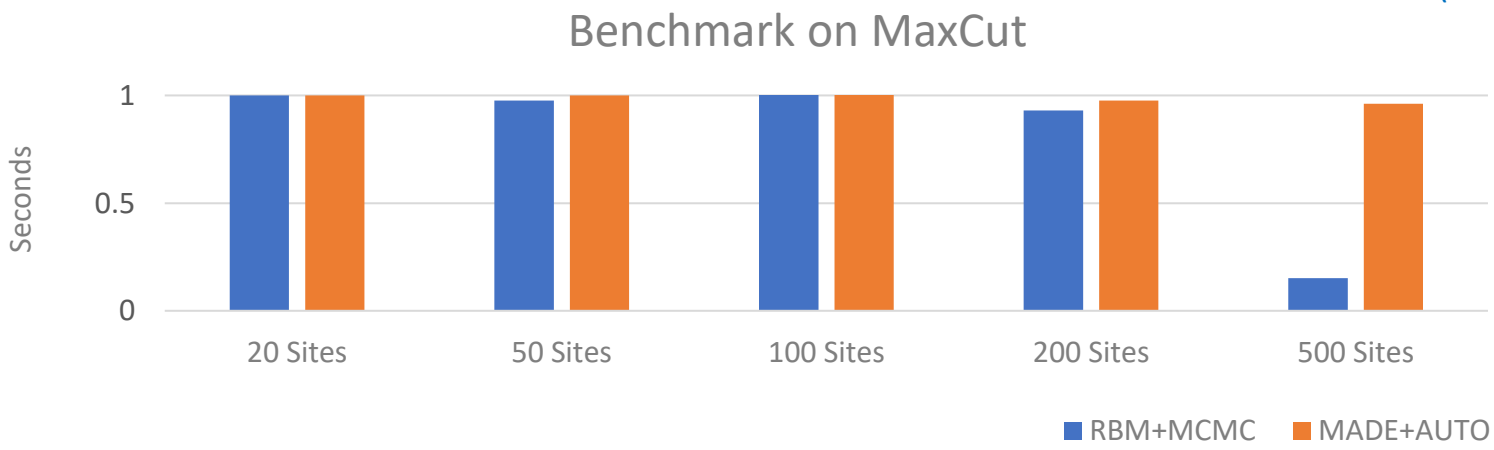


AUTO runs faster with GPU!

RBM – Restricted Boltzmann Machine
MCMC – Markov chain Monte Carlo
Burn-in iteration for MCMC $\sim 3 \times n$
MADE - Masked Autoencoder for Distribution Estimation
AUTO - Autoregressive sampling
Batch size = 1024

Performance

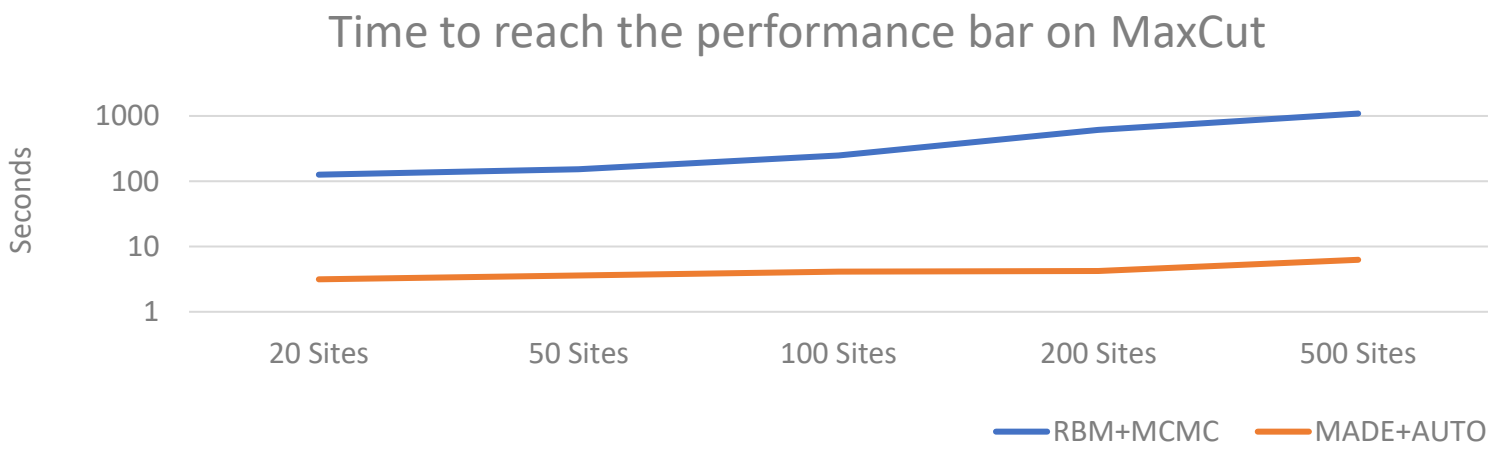
A = Performance of Random Cut
B = Performance of Burer–Monteiro
Performance of X = (X - A) / (X - B)



AUTO gets EXACT samples regardless of problem size!

RBM – Restricted Boltzmann Machine
MCMC – Markov chain Monte Carlo
Burn-in iteration for MCMC $\sim 3 \times n$
MADE - Masked Autoencoder for Distribution Estimation
AUTO - Autoregressive sampling
Batch size - 1024

Time Efficiency

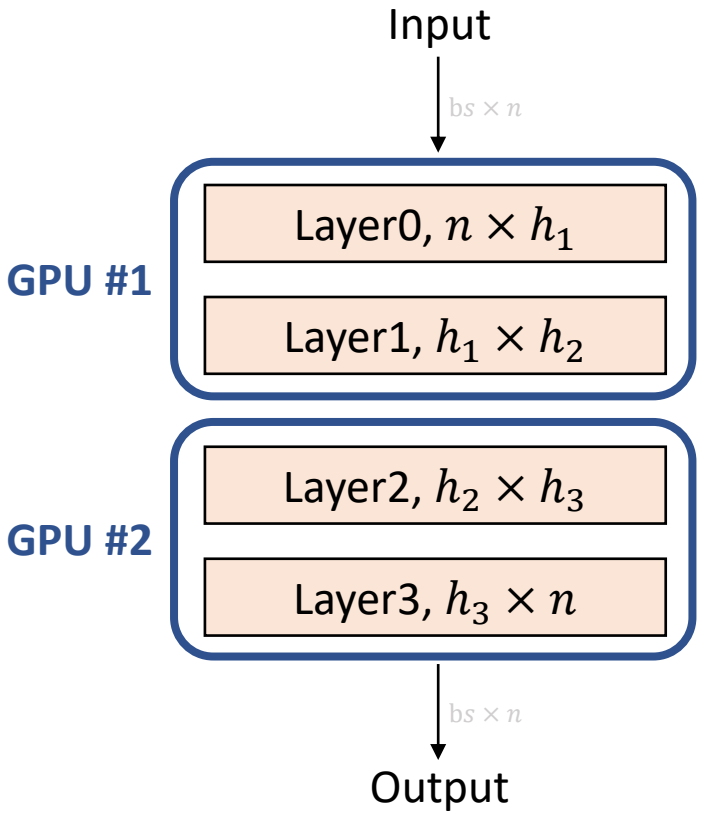


AUTO achieves better time efficiency!

RBM – Restricted Boltzmann Machine
MCMC – Markov chain Monte Carlo
Burn-in iteration for MCMC $\sim 3 \times n$
MADE - Masked Autoencoder for Distribution Estimation
AUTO - Autoregressive sampling
Batch size = 1024

Parallelization

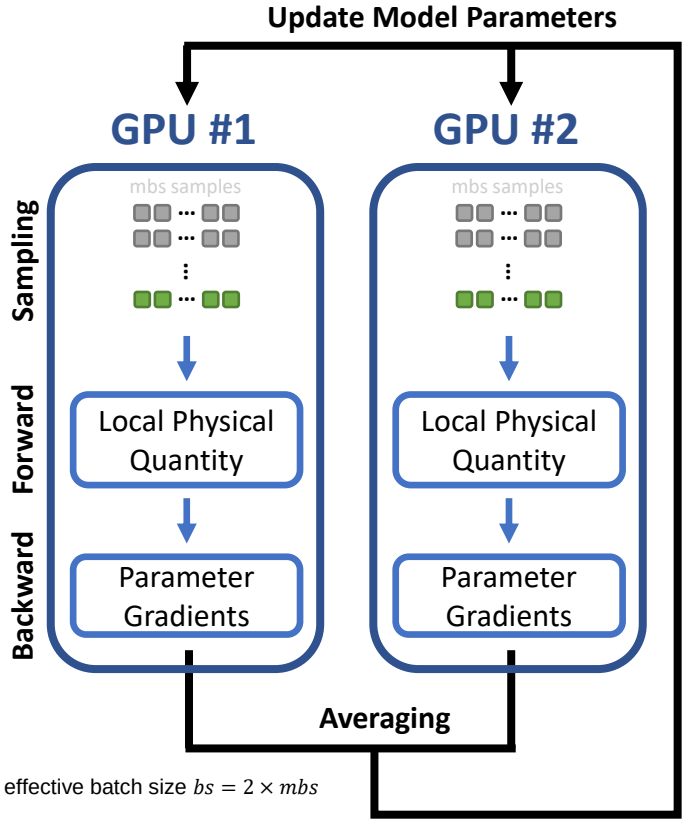
- Model Parallelization: distribute the model parameters across computing units
- Sampling Parallelization: use identical copies of the model across the computing units to generate only a few samples per unit



n: Problem size
h: hidden layer size

Parallelization

- Model Parallelization: distribute the model parameters across computing units
- Sampling Parallelization: use identical copies of the model across the computing units to generate only a few samples per unit

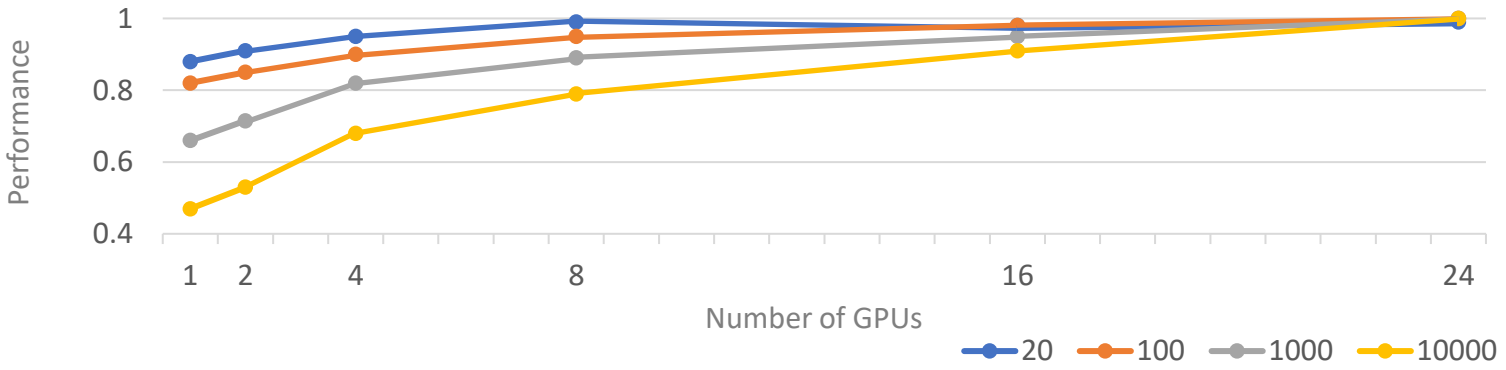


Communication for parameter gradient only!

n: Problem size
h: hidden layer size
mbs: mini batch size

Performance over Effective Batch-sizes

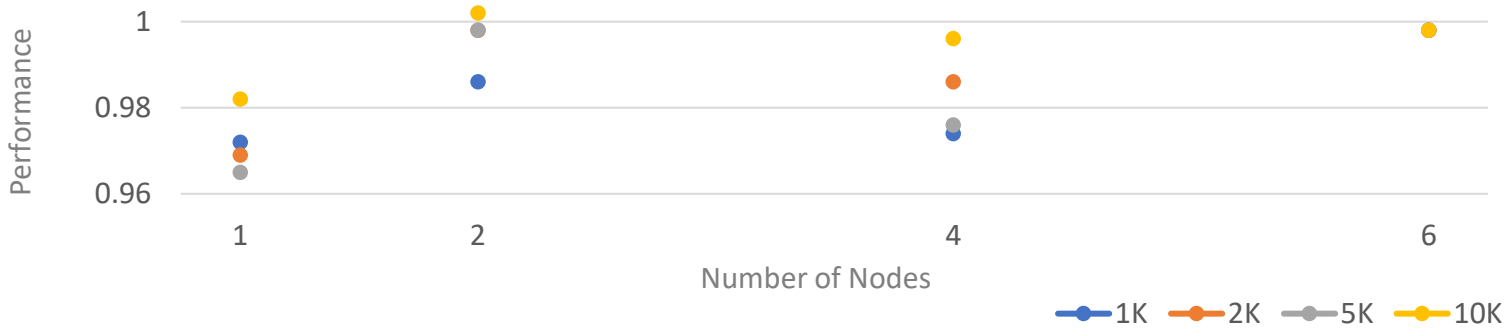
Effective Batch Size = 4 × Number of GPUs
Performance of X = X / Performance with 24 GPUs



We can solve large scale problems with sufficient number of GPUs.

Parallelization Efficiency

Setting: 4 GPUs per node
Performance of X Nodes = Time elapsed with X Nodes / Time elapsed with 6 Nodes



Our parallelization achieves near-optimal weak scaling.

Thank you!