GPU-Accelerated Tensor Networks
Harnessing the power of GPUs

arXiv: 2306.00358, in collaboration with Raghav G. Jha
Abhishek Samlodia, Syracuse University
asamlodia@gmail.com
01 August 2023
Tensor Networks in Physics

Motivation

- **Goal**: Estimating the Partition Function and Expectation of the subsequent Observables for a system

\[ Z = \sum_{\{s_i\}} \exp(-\beta H_{s_i}) \] (1)

- Standard Monte Carlo Methods sample states \( s_i \) using various techniques but can only compute expectation of the observable
- Tensor Networks can estimate the Partition Function in the Thermodynamic Limit as well as Handle Sign Problems
Tensor Renormalization Group (TRG)
A Tensor Networks Method

- More natural to lattice field theory, based on the Lagrangian of the system.
- First introduced to study discrete spin models and now it has also been used to study spin models with continuous symmetry and gauge theories in two and higher dimensions. (arxiv: 2202.10051, 2010.06539)
- Primary reason to use TRG methods is their ability to deal with complex-actions involving terms such as topological-\(\theta\), chemical potentials, etc. because no sampling technique is used here.
• Assuming that the system is represented by a network of 4-rank tensors, $A_0$. First perform singular value decomposition of each tensor $A_0$.
The original network configuration after SVD looks like,
Now contract the four $F$ tensors from SVD to get new network configuration with each tensor $A_1$.
• Keep repeating this process until only 1 tensor is left in the network configuration and then take the tensor trace which gives the partition function.

And that is how we get the Partition Function!
Higher Order Tensors Renormalization Group
TRG + HOSVD = HOTRG

- Higher Order TRG (HOTRG) truncates the diagonal matrix $S$ obtained from SVD at a finite $D$ which reduces the errors in the numerical estimations.
- Truncation means choosing a part of the diagonal matrix consisting of maximum number of non-zero elements with $D$ as an upper bound to this number.
Table of Contents

2 Using GPU

- Introduction
- Using GPU
- Results
- Summary
How to use Graphical Processing Units?
Tensor Initialization, Contraction and Linear Algebra Operations on a GPU

• We will use Python Programming Language.
• Define all the tensors using **PyTorch** package. These tensors are all CPU tensors.
• Use **opt_einsum_torch** package which has a compatibility with PyTorch on GPU for tensor contractions.
• This package performs Einstein Summations significantly faster than other python modules such as **ncon**, **einsteinpy**, **numpy.einsum**, etc.
• To perform SVD and other Linear Algebra Operations, **torch.linalg** sub-module was used.
Runtime Comparison for HOTRG
Time Complexity Orders on CPU v/s GPU

<table>
<thead>
<tr>
<th>Device</th>
<th>Scaling</th>
<th>Scaling for $d = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>$O(D^{4d-1})$</td>
<td>$O(D^7)$</td>
</tr>
<tr>
<td>GPU (This Work)</td>
<td>-</td>
<td>$O(D^{5-6})$</td>
</tr>
</tbody>
</table>

Table: $d$ is the number of Euclidean Dimensions for a system, $D$ is the truncation parameter

- In HOTRG, the dominant contribution in execution time for CPU and GPU both, is from Tensor Contractions.
- HOSVD gets a boost in compute time on a GPU whereas for a CPU the time scaling is given as $O(D^6)$
- The absolute run-time reduces significantly when GPU is used over a CPU.
**GPU v/s CPU**

**How GPUs are better**

- **Number of Cores**: A GPU has large number of cores (in thousands) as compared to a CPU which has only 20-60.

- **Heavy Parallelization**: 32 threads per core for GPU as compared to 2 threads per core for a CPU.

- **Load Management**: Unlike a CPU, a GPU can reduce memory subsystem load by dynamically changing the number of available registers (from 64 to 256 per thread).

- **Shared Memory**: GPUs have shared memory management which is significantly faster than CPU’s L1 Cache memory.
For 2d Ising Model, we compared the Free Energy with the exact known results to check the accuracy of the HOTRG Algorithm.

We compute error in free energy as,

\[ \frac{|\delta f|}{f} = \frac{|f_{\text{TRG}} - f_{\text{E}}|}{f_{\text{E}}} \]  \hspace{1cm} (2)

The starting tensor \( A_0 \) for the Ising Model is (By Expanding Boltzmann Weights),

\[ A_0 = T_{ijkl} = W_{ia}W_{ja}W_{ka}W_{la}, \]  \hspace{1cm} (3)

\[ W_{ia} = \begin{bmatrix} \sqrt{\cosh(\beta)} & \sqrt{\sinh(\beta)} \\ \sqrt{\cosh(\beta)} & -\sqrt{\sinh(\beta)} \end{bmatrix} \]  \hspace{1cm} (4)
2d Ising Model
Free Energy v/s Temperature

<table>
<thead>
<tr>
<th>(\delta f)/f</th>
<th>\times 10^{-9}</th>
</tr>
</thead>
<tbody>
<tr>
<td>D = 64</td>
<td></td>
</tr>
</tbody>
</table>

\(\delta f\)
**2d Ising Model**

*Run-time Comparison Table*

- Comparison the execution time for a single $T = T_c$ run on CPU and GPU

| $D$    | $\left| \frac{\delta f}{f} \right|$ | A100 | RTX 2080 | 4 CPUs |
|--------|------------------------------------|------|----------|--------|
| 84     | $6.6 \times 10^{-10}$              | 6004 | 9171     | 11714  |
| 94     | $4.4 \times 10^{-10}$              | 11960| 19305    | 29376  |
| 104    | $2.9 \times 10^{-10}$              | 21376| 36159    | 58715  |
| 109    | $2.4 \times 10^{-10}$              | 28942| 46350    | 80578  |

*Table:* Timings (in seconds) for the HOTRG algorithm for Ising model for $2^{20} \times 2^{20}$ lattice at $T = T_c$.

- CPU-time scaling is $\sim D^7$ while the GPU-time is $\sim D^{5.9}$.
- Absolute execution time reduces as the GPU architecture improves.
2d Ising Model
Run-time Comparison Plot

Runtime (in seconds) vs. $D$

- 4 CPUs
- 1 NVIDIA RTX2080
- 1 NVIDIA A100

Abhishek Samlodia, Syracuse University, asamlodia@gmail.com | GPU-Accelerated Tensor Networks
The generalized XY (GXY) model is a standard XY model with a spin nematic deformation term in the Hamiltonian,

\[ H = -\Delta \sum_{\langle ij \rangle} \cos(\theta_i - \theta_j) - (1 - \Delta) \sum_{\langle ij \rangle} \cos(2(\theta_i - \theta_j)) - h \sum_i \cos \theta_i, \]

where \( \langle ij \rangle \) denote the nearest neighbours and \( \theta_i \in [0, 2\pi) \).

We performed tensor computations for a fixed value of \( \Delta = 0.5 \) and for different \( D \), keeping \( h = 0 \).

GPU run-time scales as \( \sim D^{5.4} \) as compared to CPU run-time which scales as \( \sim D^7 \).
2d Generalized-XY Model

System Size: $2^{30} \times 2^{30}$

**Abhishek Samlodia, Syracuse University, asamlodia@gmail.com**

**GPU-Accelerated Tensor Networks**
Summary

Take-home message

• TRG Methods get a significant boost in compute time if GPUs are used, where absolute execution time reduces as the GPU Architecture improves.

• Run-time for HOTRG Method scales as $O(D^{(5-6)})$ on a GPU as compared to $O(D^7)$ on a CPU for models with 2 Euclidean Dimensions, where the leading contribution comes from tensor contractions.

• Computing Critical Exponents to study system’s behaviour around critical point requires higher values of $D$. Using GPU-Code such exponents can be computed in a reasonable amount of time and memory.

• Large number of cores and large number of threads launched per core enables GPU to perform tensor computations remarkably better than an CPU.
GPU-Accelerated Tensor Networks

Thank you for listening!

Any questions?