RESEARCH-ARTICLE

GPU Acceleration for Markov Chain Monte Carlo Sampling

JIARUI LI, Tulane University, New Orleans, LA, United States

SAMUEL J LANDRY, Tulane University School of Medicine, New Orleans, LA, United States

RAMGOPAL REDDY METTU, Tulane University, New Orleans, LA, United States

Published: 08 October 2024

AIMLSystems 2024: The 4th International Conference on AI-ML Systems
October 8 - 11, 2024
Louisiana, Baton Rouge, USA

https://doi.org/10.1145/3703412.3703428
ISBN: 9798400711619

GPU Acceleration for Markov Chain Monte Carlo Sampling

Jiarui Li
Department of Computer Science
Tulane University
New Orleans, LA, United States
jli78@tulane.edu

Samuel Landry
Department of Biochemistry
Tulane University School of Medicine
New Orleans, LA, United States
landry@tulane.edu

Ramgopal R Mettu
Department of Computer Science
Tulane University
New Orleans, LA, USA
rmettu@tulane.edu

Abstract

In areas such as molecular biology, computer vision, and natural language processing, graphs are commonly used to represent the structure of probability distributions (or their equivalents) that are too large to consider explicitly. In these settings, we are commonly interested in sampling from the distribution for various inference tasks. A typical approach is the use of Markov Chain Monte Carlo (MCMC) methods. The classic, uniprocessor, approach to MCMC still results in poor performance. While parallel approaches on CPU or GPU devices have been proposed, they are often designed for specific tasks and/or do not effectively utilize the inter-device communication capabilities of GPUs. We propose a novel, GPU-parallelizable MCMC method for this setting. Our approach makes use of a partitioning approach to divide the graph and dispatch the resulting sub-graphs to GPUs. Using the communication capabilities of GPUs (e.g., NVLink), we give a way to coordinate information between the computations over adjacent subgraphs and subsequently merge them. This approach takes advantage of the high degree of GPU parallelism while maintaining the generality of MCMC sampling. We demonstrate the performance of our approach for estimating protein conformational stability. Over four different benchmarks and two GPU platforms we show that our method achieves up to a 400% speedup over an adaptive Monte Carlo sampling method.

CCS Concepts

• **Computing methodologies** → **Self-organization**; • **Applied computing** → **Molecular structural biology**; • **Mathematics of computing** → **Metropolis-Hastings algorithm**.

Keywords

Markov Chain Monte Carlo, Bayesian networks, GPU acceleration, parallel algorithms, protein conformational stability

ACM Reference Format:
Jiarui Li, Samuel Landry, and Ramgopal R Mettu. 2024. GPU Acceleration for Markov Chain Monte Carlo Sampling. In *4th International Conference on AI-ML Systems (AIMLSystems 2024), October 08–11, 2024, Baton Rouge, LA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3703412.3703428

AIMLSystems 2024, October 08–11, 2024, Baton Rouge, LA, USA

ACM ISBN 979-8-4007-1161-9/24/10
https://doi.org/10.1145/3703412.3703428

1 Introduction

Markov chain Monte Carlo (MCMC) methods are commonly used in settings where standard Monte Carlo sampling does not suffice to provide reasonable estimates for an underyling distribution. MCMC methods are widely utilized in various fields such as molecular biology [2, 19, 24], computer vision [6, 8], and natural language processing [22, 27] to estimate predictive factors. More generally they are also used in Bayesian inference to generically sample the posterior distribution. Approaches taken by methods to improve MCMC performance include exploiting the structure of the target distribution (e.g., Hamiltonian Monte Carlo [5, 11, 15]) and breaking the problem into subproblems that can be solved concurrently [20]. In this paper we will focus on the latter, motivated by high-dimensional distributions that require MCMC to generate a large number of samples for the desired solution quality. The Markov chain aspect of MCMC would seem to impose a bottleneck on parallelization but in fact this topic is well-studied.

Existing methods can generally be placed into two categories: data-level parallelization and algorithmic parallelization. Data-level parallelization splits a large dataset into multiple sub-components and dispatches them to different agents to calculate them in parallel [17, 23, 28]. Algorithmic parallelization decomposes the task of MCMC sampling into multiple smaller instances that can be conducted in parallel [10, 14, 26], and then aggregated appropriately. Both of these approaches to parallelize MCMC introduce non-negligible challenges. In algorithmic parallelization, communication between different computational agents introduces challenges. Although data-level parallelization does not face this issue, it cannot accelerate the burn-in process. Recent methods for algorithmic parallelization (e.g., [4, 16]) have managed to achieve success but are primarily designed for CPU cores.

GPU platforms offer orders of magnitude more parallelism than CPUs, but have traditionally had little to no coordination between devices. Recently, however, features such as NVLink on NVIDIA platforms allows different GPUs to communicate more efficiently [7] and allow the possibility of algorithmic parallelism for MCMC on GPUs. Prior to this capability, parallelism on GPUs could only be achieved for tasks (e.g., such as image processing [12]) that did not require coordination between GPUs.

In this paper, we propose a new MCMC approach on distributions that can be represented as graphs (e.g., Bayesian networks). Our approach is to use the conditional dependency structure in the graph to construct sampling subtasks that can be dispatched to GPUs. Then, we use GPU communication to conduct a coordination step that exchanges posteriors to check for convergence. Once convergence is achieved, the subtasks are merged to arrive at the final set of samples. Bayesian networks are used in many machine learning applications, but to our knowledge this is the first

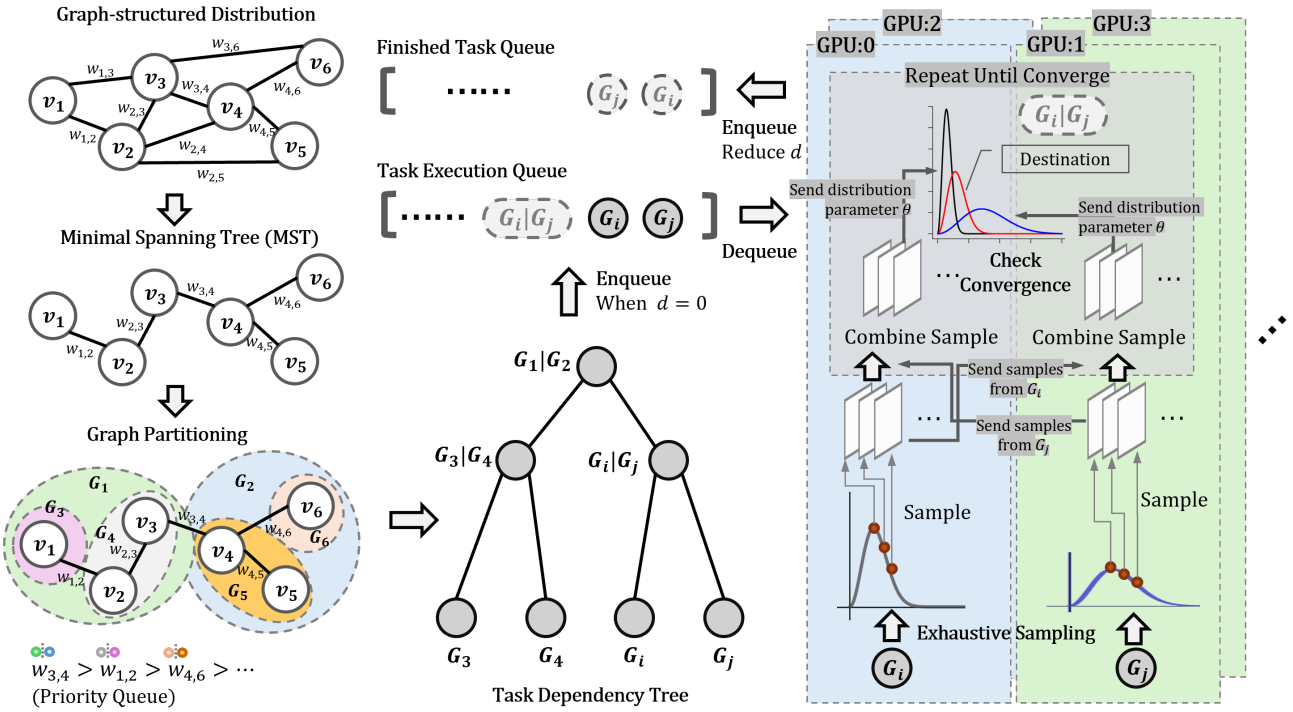

Figure 1: Schematic of our approach. We make use of the fact that we have a graph-structured distribution, and partitions sampling tasks according to the graph. Our approach can utilize GPU communication and thus can conduct computation, aggregation and convergence checking without moving work to CPU cores. CPUs are primarily responsible for managing task dispatching and result collection.

application to MCMC parallelization. Moreover, the computation, coordination and convergence checking steps of our algorithm are conducted on GPUs without the need for synchronization on CPU cores. In prior work, GPU cores were only used for simple computations while coordination steps took place on CPU cores, resulting in a performance bottleneck.

We demonstrate the effectiveness of MCMC sampling in a free-energy approximation scheme [25] we have used to conduct computational epitope prediction (e.g. [1–3, 13]. These free-energy approximations are done on protein structures, which are naturally represented by graph-structured distributions. We demonstrate the performance of our approach on this particular task in comparison with standard Monte Carlo methods and show a significant improvement in performance both in terms of solution quality as well as efficiency in both the desktop and server settings. On a desktop environment (2-GPU NVIDIA A2000) our algorithm achieves up to a 4.0x speedup, and on a server platform (8-GPU V100 NVIDIA) we see up to a 2.7x speedup. Moreover our method is able to achieve speedups of almost 30x in some cases over standard CPU-based Monte Carlo methods.

2 Our Approach

In this section we detail our approach, which we call *GPU-MCMC*. First, we define the problem of sampling from a graph-based distribution. While our approach is general we describe the problem using our application of sampling a protein conformational ensemble. Second, we give a method to partition the graph distribution into different subgraphs to allocated to GPUs for concurrent sampling. We make use of a task dependency tree and cache queues to manage tasks to prevent blocking. Finally, we will introduce how to leverage the communication capability between different GPUs to combine subgraphs and test for convergence.

2.1 Graph Definition

To setup our method for sampling a graph-based distribution, we consider our application of calculating protein conformational stability. A protein can be represented as a graph in which each vertex corresponds to a folding unit (i.e., a block of contiguous amino acids) and is either folded or unfolded (i.e., a binary random variable), with edge weights $w_{i,j}$ defined according to the distance between folding unit i and j and each edge represents the relationship between two folding units. Then, for any combination of states of the vertices (i.e. a particular partially folded conformation of the protein), a probability $p_k \in [0, 1]$ describes the likelihood of that folding state combination occurring. By leveraging MCMC, we can sample the distribution of all possible states for the entire graph or any subgraph. For our application, the input to GPU-MCMC is a graph-structured distribution derived from a protein and our goal is to estimate the distribution of conformations for the protein.

Algorithm 1 Graph Partitioning

Input: t ▷ The minimum number of subgraph vertices
Input: V, E ▷ Vertices and edges of the graph distribution
$(V', E') \leftarrow \text{MST}(V, E)$ ▷ Compute MST of the graph
$Q \leftarrow \text{PriorityQueue}(E')$ ▷ Build priority queue on edges
$\mathcal{G} \leftarrow \emptyset$ ▷ Subgraph set
parfor $e \leftarrow Q$ **do**
  $\mathcal{G}' \leftarrow \text{Partition}((V, E), e)$
  if $\min_i |\mathcal{G}'_i| > t$ **then**
    $\mathcal{G} = \mathcal{G} \cup \mathcal{G}'$
return $\mathcal{G}$

2.2 Graph Partitioning

To parallelize the graph-structured sampling task on different GPUs, we need to construct subtasks that can be dispatched across GPU cores. To do this we will partition the task graph into multiple subgraphs that have minimal dependence on one other. To partition the input graph into subgraphs, we take the following simple approach. First, we generate a minimal spanning tree (MST). Using the fact that any edge in the MST will partition the graph into two independent subgraphs, to partition the MST into least-dependent subgraphs we do the following. First, all edges of the MST are enqueued to a priority queue and ranked by their weights. This priority queue can dequeue the edges from the edge with maximum weight to minimum weight. Second, we dequeue the edge priority queue to fetch ordered edges and utilize the edge to partition the MST. If after a partitioning operation, the number of vertices of one of the subgraphs is less than a threshold t, this partitioning operation is going to be dropped and the algorithm moves to the next edge fetched from the queue.

2.3 Tasks Management Queues

Partitioning the graph constructs sampling subtasks, but we must manage the (ideally sparse) dependencies between subtasks. To manage the dependencies efficiently we built a task dependency tree, which is a binary tree defined as follows. The leaves of this tree are the subtasks without any dependencies, which means that they computed independently at the beginning. The nodes of this tree are the subtasks with dependency requirements. Therefore, the dependency $d \in \mathbb{N}$ of the node can be defined as the number of its child nodes/leaves. When the d of a node is 0, this denotes the node does not have any dependency and can be enqueue to the execution queue, which is used to cache all tasks that are being or waiting to be computed. After a subtask is completed, it will be dequeued from the execution queue and enqueued to the finished task queue, which is used to cache all finished tasks. Additionally, the d of all tasks that are depended on this task are decreased 1. When the task execution queue is empty and d of all nodes in the task dependency tree are 0, the graph-structured distribution estimation has been finished.

2.4 Subgraph Sampling on GPUs

According to the dependency tree, the subgraphs can be categorized into two types. First, any task in the dependency tree that is a leaf can be computed independently. Tasks with dependencies require

Algorithm 2 Combine Subgraphs Across GPUs

Input: D_k ▷ The GPU sampling the neighbor subgraph
Input: $\mathcal{N}, S$ ▷ Subtask distribution and samples
Input: t_c ▷ Distribution convergence threshold
$\theta' \leftarrow D_k.\theta$ ▷ Get distribution from the neighbor subgraph
$\theta \leftarrow \emptyset$ ▷ Empty parameters for the distribution
$S_c \leftarrow \emptyset$
parfor $diff(\theta, \theta') > t_c$ **do**
  $S', S'' \leftarrow \text{Sample}(S, \mathcal{N})$
  $S''' \leftarrow D_k.S''$
  $S_c \leftarrow \text{UniformSample}(\{(s_i, s_j) | s_i \in S', s_j \in S'''\}) \cup S_c$
  $\theta \leftarrow E_{\mathcal{N}}(S_c)$
return $\mathcal{N}(\theta), S_c$

us to combine neighboring subgraphs. For independent tasks, we can sample them exhaustively to estimate the distribution. For the tasks with dependencies, we must provide a way to sample pairs of subgraphs. We give a way to conduct the sampling of states for pairs of subgraphs on GPUs. For convenience, we assume there are n GPUs in total, they are denoted as $\{D_k\}_{k=1}^{n}$. For example, as shown in Fig.1, we want to compute G_i, G_j and the $G_i|G_j$ depending on them. First, the subtasks G_i and G_j are dispatched to two different GPUs and are exhaustively sampled to estimate their distributions respectively. Second, we assign the same GPUs utilized to sample G_i and G_j to combine $G_i|G_j$ cooperatively. For the GPU sampled G_i, it uniformly chooses G_i samples. Then it keeps half of the samples and send the other half to the GPU sampled G_j that will execute the sample procedure on G_j. For the both GPUs, they have half samples from G_i and half samples from G_j, which can be uniformly combined to be the samples for $G_i|G_j$. Based on these samples, each GPU can estimate the distribution, which represented $G_i|G_j$. Then, they exchange the estimated distribution parameters. The difference between the distribution parameters calculated following:

$$diff(\theta_1, \theta_2) = \frac{1}{n} \sum_{i=0}^{n} \frac{|\theta_{1,i} - \theta_{2,i}|}{\max(\theta_{1,i}, \theta_{2,i}) - \min(\theta_{1,i}, \theta_{2,i})} \tag{1}$$

where $\mathcal{N}_1$ and $\mathcal{N}_2$ denote two distributions and their parameters are $\theta_1 \in R^n$ and $\theta_2 \in R^n$. If the difference is less than a threshold t, the two distributions are converged, which means task for combining $G_i|G_j$ has been well sampled. If the two distribution have not converged, we repeat these steps.

3 Results

To evaluate the quality and efficiency of the proposed GPU-MCMC, we consider a molecular modeling application in which we must sample a large conformational ensemble in order to compute a free energy known as *COREX*, which we describe below. As described earlier, we have designed an algorithm for computational epitope prediction that makes use of antigen structure. The COREX metric is a critical component of our approach, which allows us to use conformational stability to model the likely location of epitopes in an antigen. COREX is the most time-intensive component of our algorithm, requiring 6-8 hours of computation time on a single processor. COREX performance depends primarily on the effectiveness of Monte Carlo sampling; we will show in this section that

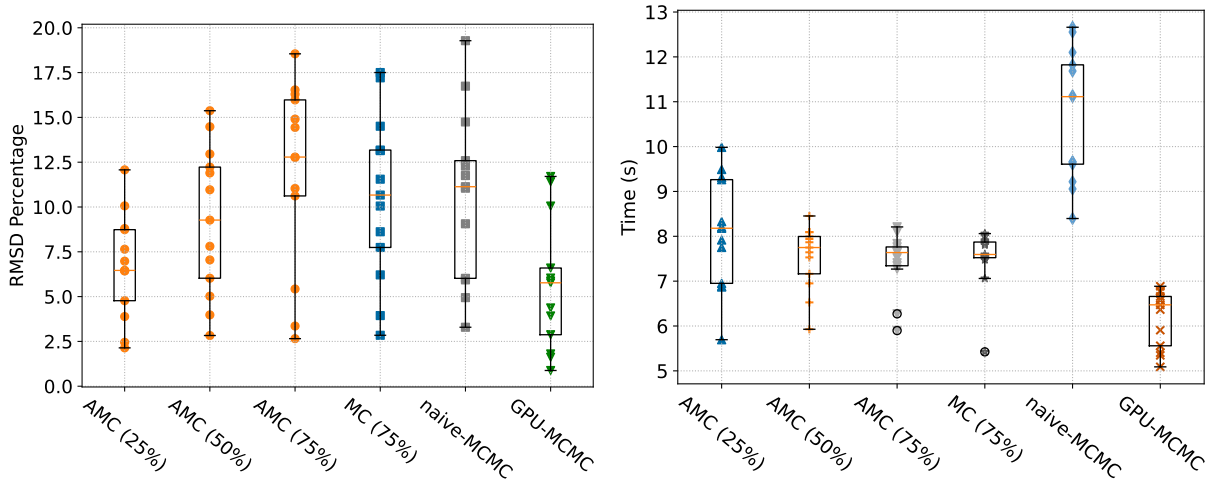

Figure 2: Residue stability constant calculation RMSD Percentage and runtime for the benchmark of 13 small proteins. We compared the solution quality and runtime of our GPU-MCMC against MC (Monte Carlo), AMC (adaptive Monte Carlo), and naive-MCMC (naive Markov Chain Monte Carlo) for proteins with known ground-truth results. GPU-MCMC easily achieves the best quality-time tradeoff.

our approach can make real-time computational epitope prediction from antigen structure a potential reality.

To test our algorithm we assess sampling quality and efficiency using several sets of proteins. First, we conduct a baseline evaluation of our algorithm with respect to a standard adaptive Monte Carlo method on a set of small proteins for which the conformational ensembles can be exhaustively enumerated. Then, we consider the efficiency of GPU-MCMC sampling on a three distinct sets of proteins. For comparison, we test the performance of GPU-MCMC against pCOREX (an adaptive Monte Carlo (AMC) method) as well as a simple GPU-based naive Markov Chain Monte Carlo (naive-MCMC) sampling method. Overall, GPU-MCMC yields higher quality samples compared to these methods and achieves a significant speedup.

3.1 COREX Algorithm and Monte Carlo Sampling

COREX (CORelation with hydrogen EXchange protection factors) is an algorithm used to measure the stability of each residue by analyzing the protein's structural ensemble when it is in equilibrium [25]. At a high level, the algorithm accepts a protein structure as input and yields stability constants that characterize the likelihood of unfolding at each amino acid residue. The computation of stability constants is done through a coarse-grained approximation of the free energy of the system. The algorithm first partitions the protein into different *folding units*, which is just a contiguous block of residues (usually 10 amino acids). A conformational *microstate* is then defined according to whether its folding units are folded or unfolded. The COREX free energy approximation is defined according the full ensemble of microstates, but even the coarse-grained approach of using folding units the ensemble is too large to consider in its entirety. Effectively sampling a high-quality ensemble is the key computational bottleneck to computing the COREX metric, since we must evaluate the energetics of each microstate that is considered.

The original COREX method utilizes a straightforward, fixed threshold Monte Carlo (MC) sampling on a single processor. However, the naive MC method is limited in efficiency and can take hours to compute a single protein. To tackle this, we developed pCOREX, which implements data-parallel naive Monte-Carlo sampling on multiple CPUs [2]. As an alternative to naive Monte-Carlo sampling, one can also utilize data-parallel adaptive Monte Carlo ("AMC") sampling on multiple GPUs that makes use of a floating threshold on conformational energies to maintain a stable acceptance rate. It is also natural to ask if a naive MCMC approach also suffices for improved performance. To test this, we consider a GPU-based naive Markov Chain Monte Carlo (MCMC) algorithm. Rather than generating new samples uniformly, this approach uniformly alters several bits of microstates randomly sampled from the pool of accepted microstates. Thus for our comparisons we compare our method ("GPU-MCMC") against naive Monte Carlo (MC), adaptive Monte Carlo ("AMC"), and naive Markov Chain Monte Carlo ("naive-MCMC") methods.

3.2 Experimental Setup

We implemented GPU-MCMC in PyTorch, an open-source, GPU-supported scientific computation library. We set the folding unit size is set to 10 residues with a minimum size of 4 residues, resulting in a total of 10 partition schemes. For the pathogen, melanoma, and immunopeptidomics benchmarks, MC, AMC, and naive-MCMC sample 10,000 samples for each partition scheme, yielding a total of 100,000 samples that must be gathered for each protein. For the small protein benchmark, due to the ensemble size, MC, AMC, and naive-MCMC uniformly sample 50 samples for each partition scheme, resulting in 500 samples in total. For naive-MCMC, we

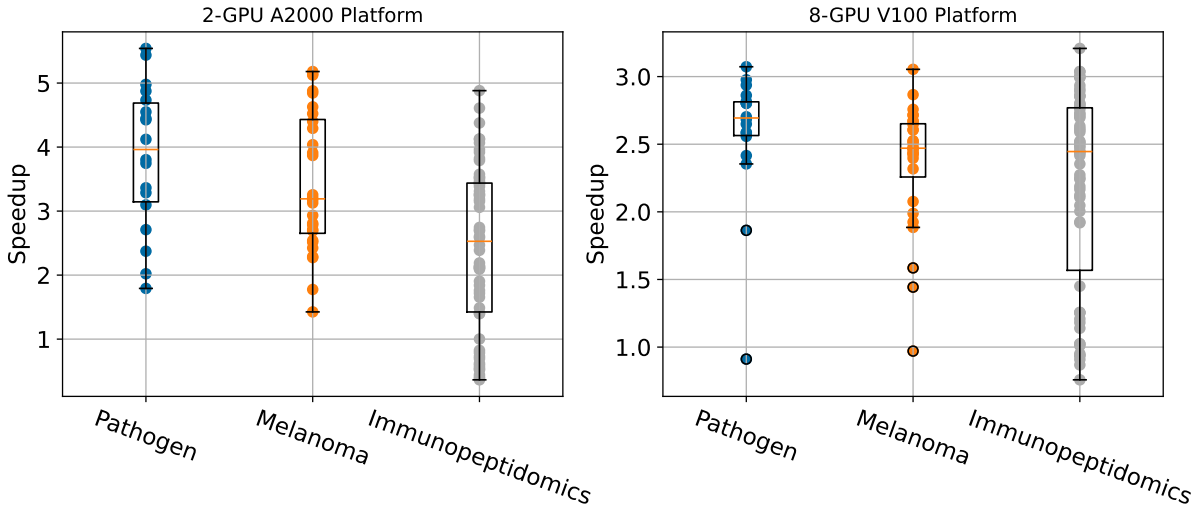

Figure 3: Performance overview of GPU-MCMC speedups over standard GPU Monte Carlo methods. On a 2-GPU A2000 platform, GPU-MCMC achieves 4.0x, 3.2x, and 2.5x speedups on pathogen, melanoma, and immunopeptidomics benchmarks respectively. On 8-GPU V100 platform, we achieve speedups of 2.7x, 2.5x and 2.4x, respectively.

uniformly flip 90% of the bits of selected sampled microstates to generate new samples. GPU-MCMC gathers samples until convergence so the number of samples will differ by protein.

Two GPU environments are considered for our experiments. The first environment is a standard desktop server, which has two A2000 NVIDIA GPUs with 24GB of CUDA memory and two Intel Xeon W-2245 3.9 GHz CPUs with 64GB memory. The second configuration is an Amazon Web Service (AWS) EC2 P3.x16large instance[9]. It comprises of eight NVIDIA V100 GPUs connected by NVLink with 128GB CUDA memory and 96 virtual Intel E5 2628 3 GHz CPU cores with 488 GB memory. For the CPU environment used for MC, we make use of AWS EC2 C6a instance with 192 virtual 3rd gen AMD EPYC CPU cores and 384 GB memory.

3.3 Small Protein Benchmark

To evaluate the sampling quality of the novel GPU-MCMC sampling approach, we compiled a benchmark of 13 small proteins varying in size from 63 to 117 residues. These proteins were chosen because we can compute COREX stability by enumerating the full micro-state ensemble to establish ground-truth values. Then, we can compare the quality of the different MC sampling methods to this baseline by measuring the percentage root-mean-square deviation (RMSD%):

$$RMSD\%(y,\hat{y}) = \frac{\sqrt{\sum^{i}(\hat{y_i} - y_i)^2} \times 100}{\max(\hat{y} \cup y) - \min(\hat{y} \cup y)}\% \tag{2}$$

where y and $\hat{y}$ denote the ground-truth (baseline) and sampling values separately.

We compare the quality and runtime of GPU-MCMC to naive-MCMC and AMC with different thresholds including 25%, 50%, and 75%. Fig.2 shows the quality and speed of these methods. Additionally, we used MC with a 75% sampling threshold as the baseline. We can clearly see that GPU-MCMC sampling accuracy surpasses all other methods in terms of sampling quality as well as runtime. Importantly we can see that not only is naive-MCMC is the slowest algorithm, since the sampling approach limits concurrency, but it also produces poor quality results with respect to the ground truth. In the following sections, in evaluating speedup over existing methods, we will use AMC with a 50% threshold since it is best tradeoff between quality and performance. As we will see, the concurrency of naive-MCMC is so poor that for the larger benchmark tests we do not even provide experimental results since it requires on the order of hours to complete per antigen.

3.4 Experimental Validation on Benchmark Sets of Antigens

In this section consider three different sets of protein antigens, motivated by different epitope prediction tasks. First, we consider a set of pathogenic antigens taken from viruses and bacteria [2]. The second set of proteins is taken from a melanoma cancer protein study [21] which sought to identify potential immunotherapy targets. Finally we consider a large benchmark of proteins used to study the MHCII pathway [18].

In general, the efficiency of GPU-MCMC surpasses AMC on all three benchmarks and both desktop and high-performance server platforms. As shown in Fig.3, on 2 A2000 platform, it achieves 4.0x, 3.2x, and 2.5x speedup against AMC on pathogen, melanoma, and immunopeptidomics benchmarks respectively. On 8 V100 platform, it achieves 2.7x, 2.5x and 2.4x speedup against AMC separately. We will introduce the details about the experiments on the three benchmarks respectively.

3.4.1 Pathogen Benchmark. To evaluate the efficiency of the novel proposed GPU-MCMC, we compare the performance among GPU-MCMC, naive-MCMC, AMC, and MC on a set of 19 pathogenic antigen proteins that induce immune response in C57BL/6 mice and human subjects. Sizes of the proteins range from 216 to 1,065

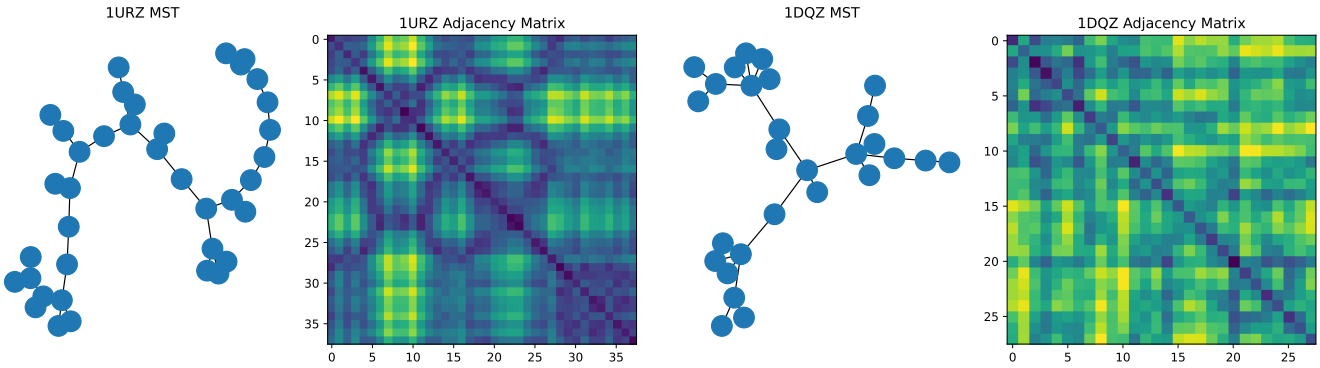

Figure 4: The special case study of GPU-MCMC sampling for the pathogen benchmark. 1DQZ (M.t. Ag85A) protein case is the case where the rumtime of GPU-MCMC increase abnormally. 1URZ (TBE Ebv) protein case is a normal case selected to compare to 1DQZ case. It demonstrates that, 1DQZ has density connection and complex graph, which cause the MST cannot partition the subgraphs effectively. This causes the distribution converge between different GPUs more difficult than the normal cases.

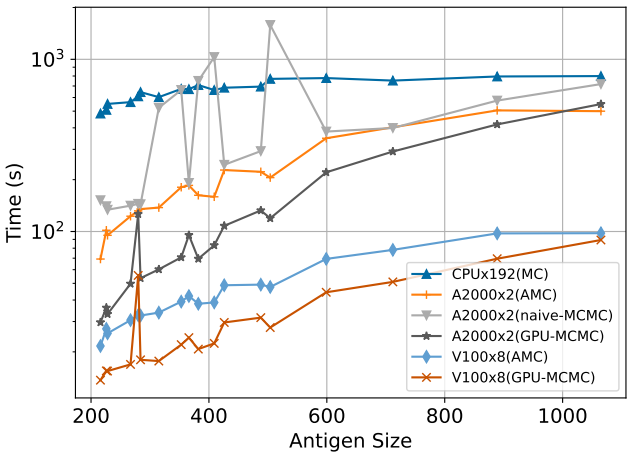

Figure 5: GPU-MCMC performance on the benchmark of 18 pathogenic antigens. When compared to AMC, the median runtime of our GPU-MCMC sampling method is significantly lower. The median runtime of 171.6s and 38.9 for AMC can be reduced 89.2s and 23.3s on 2 A2000 GPUs and 8 V100 GPUs. Notably the performance of naive-MCMC is quite poor and its runtime exceeds the CPU-based approach for several antigens.

residues. The MC with a fixed threshold of 75% is run on CPUs (as in [2]); we use this as a baseline for absolute performance. AMC with threshold 50% run on 2 A2000 GPUs and 8 V100 GPUs is considered as the baseline for GPU performance. Performance results for this set of proteins are shown in Fig. 5. AMC executing on 2 A2000 GPUs and 8 V100 GPUs takes a median 171.6 and 38.9 seconds respectively and 1.9x and 8.3x speedup comparing to the MC baseline. Notability, the COREX with GPU-MCMC achieves runtimes of 89.20 and 23.30 seconds on 2 A2000 and 8 V100 respectively, resulting in 7.5x and 28.7x speedup compared to the CPU-based MC baseline. The GPU-MCMC surpassed the naive-MCMC method, the median time of which is 336.8 seconds. For several cases, the MCMC is slower than CPU-based MC method. Therefore, it is more appropriate to compare the performance of GPU-MCMC to AMC. In the following subsections we do not even report the performance of naive-MCMC.

We note that in Fig. 5 we can observe an anomalous runtime for a protein with 280 residues (1DQZ). To study this case more closely we show the graph, MST and adjacency matrix for 1DQZ along with another protein (1URZ, 382 residues) for comparison in Fig. 4. Notably the graph structure of 1DQZ is much more dense, and we believe this affects the ability of GPU-MCMC to converge on the partitioned subgraphs. We discuss this issue further in Sec. 4.

Overall, however this experiment demonstrates GPU-MCMC remarkably improved the sampling efficiency on both desktop platform such as the machine with 2 A2000 GPUs and high performance computation platform such as the server with 8 V100 GPUs.

3.4.2 Melanoma Benchmark. Motivated by the identification of proteins for use in cancer immunotherapy[21], we compiled a benchmark of 28 cancer proteins that were identified as candidates for melanoma immunotherapy [1]. This dataset ranges in size from 172 to 1,245 residues. Following the same configuration as previous experiments, the GPU-MCMC compare to AMC on both 2 A2000 and 8 V100 platforms and MC, which is run on CPU platform [2].

Fig.6 provides the performance results for this test set, and we see that once again GPU-MCMC achieves superior performance. Compared to AMC, on the 2 A2000 GPU platform GPU-MCMC decreases the median runtime from 160.9 to 97.0 seconds for a 2.0x, and achieves a speedup of 3.5x over MC on the CPU platform. On the 8 V100 GPU platform, GPU-MCMC decreases the median runtime from 37.3 to 24.6 seconds for a speedup of 8.9x and achieves a speedup of 11.1x over the CPU-based MC baseline.

From Fig.6 we do see artifacts for three cases in which GPU-MCMC time are increased. As with the pathogen benchmark, these are proteins for which the associated graphs are dense and make it difficult for subgraph sampling to converge. Despite these increases however, GPU-MCMC still has superior performance over the vast majority of the test proteins.

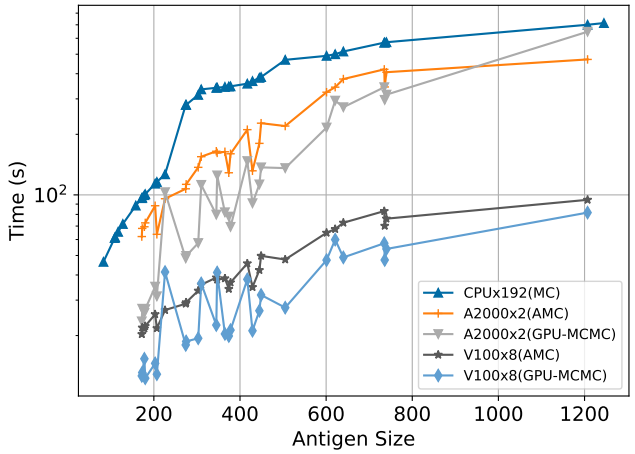

Figure 6: GPU-MCMC performance on the melanoma benchmark of 28 proteins. When compared to AMC, the median runtime of our GPU-MCMC sampling method is significantly lower. The median runtimes of 160.9s and 37.3s for AMC can be reduced 97.0s and 24.6s on 2 A2000 GPUs and 8 V100 GPUs.

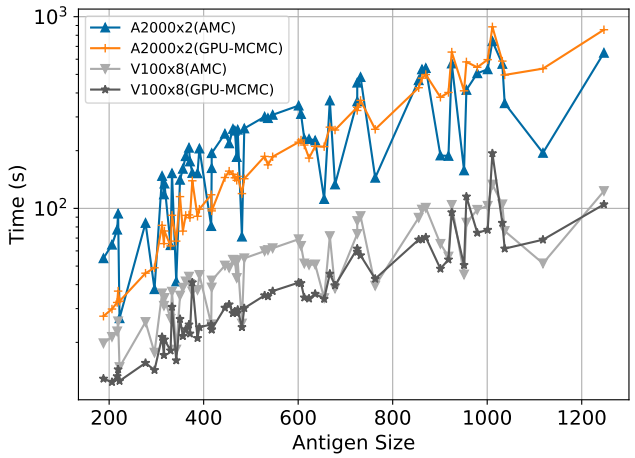

Figure 7: GPU-MCMC performance on a benchmark of 62 proteins taken from an immunopeptidomics study. When compared to AMC, the median runtime of our GPU-MCMC sampling method is significantly lower. The median runtime of 200.0s and 49.7s for AMC can be reduced to 153.5s and 32.6s on 2 A2000 GPUs and 8 V100 GPUs respectively.

3.4.3 Immunopeptidomics Benchmark. Finally we consider a benchmark of 62 proteins from an immunopeptidomics study [18] with sizes ranges from 188 to 1,247 residues. As with the other experiments above, we evaluate GPU-MCMC on both 2 A2000 and 8 V100 GPUs platforms. For this dataset the MC method on a CPU platform was too time-consuming to run and thus we just compare AMC and GPU-MCMC on the two GPU platforms.

On the 2-GPU A2000 platform, GPU-MCMC reduces the median runtime over AMC from 200.0s to 153.5s. On the 8-GPU V100 platform, GPU-MCMC reduces the median runtime from from 49.7s to 32.6s over AMC.

Notably on this benchmark, the AMC runtime fluctuated sharply as well. This is caused by the method of updating the adaptive threshold. After several epochs of updates, if the accept rate is still extremely low then virtually all samples will be accepted, leading to a very fast runtime. In contrast, GPU-MCMC utilizes a convergence criterion and is thus more stable in comparison. In our epitope prediction work we found these AMC-based still adequate for our purposes, but the GPU-MCMC results are likely much higher quality.

4 Discussion

In this paper, we have introduced a novel Markov chain Monte Carlo sampling method for the GPU setting. Our algorithm is designed to efficiently sample distributions that can be structured as graphs and takes advantage of GPU-level communication to achieve a high degree of parallelism.

In our experimental evaluations for a free energy approximation task, on a typical desktop platform our approach achieves 4.0x, 3.2x, and 2.5x speedup against the adaptive Monte Carlo method on pathogen, melanoma, and immunopeptidomics benchmarks respectively. On a high-performance server platform, it achieves 2.7x, 2.5x and 2.4x speedup against the adaptive Monte Carlo method. Overall our GPU-MCMC approach significantly outperforms standard Monte Carlo approaches.

As noted, for distributions represented by densely connected graphs we observed that method of subgraph construction as not effective and resulted in longer convergence times. In future work we plan to explore the many alternative partitioning scheme available to determine whether this issue can be addressed.

Acknowledgments

This research is supported by NIH U54-CA260581 to James E. Robinson, an award from the Tulane Center for Emerging and Re-emerging Infectious Diseases and the Harold L. and Heather E. Jurist Center of Excellence for Artificial Intelligence at Tulane University. We also acknowledge support from the Amazon AWS Cloud Credit for Research program for providing necessary GPU resources.

References

[1] Avik Bhattacharya, Molly C. Lyons, Samuel J. Landry, and Ramgopal R. Mettu. 2022. Incorporating antigen processing into CD4+ T cell epitope prediction with integer linear programming. In *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics* (Northbrook, Illinois) *(BCB '22)*. Association for Computing Machinery, New York, NY, USA, Article 20, 10 pages. https://doi.org/10.1145/3535508.3545545

[2] Avik Bhattacharya, James O. Wrabl, Samuel J. Landry, and Ramgopal R. Mettu. 2023. Parallel Computation of Conformational Stability for CD4+ T-cell Epitope Prediction. In *2023 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 88–93. https://doi.org/10.1109/BIBM58861.2023.10385333

[3] Tysheena Charles, Daniel L Moss, Pawan Bhat, Peyton W Moore, Nicholas A Kummer, Avik Bhattacharya, Samuel J Landry, and Ramgopal R Mettu. 2022. CD4+ T-Cell Epitope Prediction by Combined Analysis of Antigen Conformational Flexibility and Peptide-MHCII Binding Affinity. *Biochemistry* 61, 15 (2022), 1585–1599.

[4] Daniel A. De Souza, Diego Mesquita, Samuel Kaski, and Luigi Acerbi. 2022. Parallel MCMC Without Embarrassing Failures. In *Proceedings of The 25th International*

Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 151), Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera (Eds.). PMLR, 1786–1804. https://proceedings.mlr.press/v151/de-souza22a.html

[5] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. 1987. Hybrid monte carlo. *Physics letters B* 195, 2 (1987), 216–222.

[6] Ertunc Erdil, Sinan Yildirim, Mujdat Cetin, and Tolga Tasdizen. 2016. MCMC shape sampling for image segmentation with nonparametric shape priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 411–419.

[7] Denis Foley and John Danskin. 2017. Ultra-performance Pascal GPU and NVLink interconnect. *IEEE Micro* 37, 2 (2017), 7–17.

[8] Fredrik K Gustafsson, Martin Danelljan, and Thomas B Schon. 2020. Evaluating scalable bayesian deep learning methods for robust computer vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 318–319.

[9] Amazon Web Services Inc. 2024. Amazon Web Services. https://aws.amazon.com/ec2/instance-types/. Accessed: 2024-06-17.

[10] Kathryn Blackmond Laskey and James W Myers. 2003. Population markov chain monte carlo. *Machine Learning* 50 (2003), 175–196.

[11] David JC MacKay. 2003. *Information theory, inference and learning algorithms*. Cambridge university press.

[12] James Martens and Ilya Sutskever. 2010. Parallelizable sampling of markov random fields. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 517–524.

[13] Ramgopal R Mettu, Tysheena Charles, and Samuel J Landry. 2016. CD4+ T-cell epitope prediction using antigen processing constraints. *Journal of immunological methods* 432 (2016), 72–81.

[14] Lawrence Murray. 2010. Distributed markov chain monte carlo. In *Proceedings of neural information processing systems workshop on learning on cores, clusters and clouds*, Vol. 11.

[15] RM Neal. 2011. Handbook of Markov Chain Monte Carlo, volume 2, chapter MCMC Using Hamiltonian Dynamics.

[16] Willie Neiswanger, Chong Wang, and Eric Xing. 2013. Asymptotically exact, embarrassingly parallel MCMC. *arXiv preprint arXiv:1311.4780* (2013).

[17] David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. 2009. Distributed algorithms for topic models. *Journal of Machine Learning Research* 10, 8 (2009).

[18] Niclas Olsson, Wei Jiang, Lital N Adler, Elizabeth D Mellins, and Joshua E Elias. 2022. Tuning DO: DM ratios modulates MHC class II immunopeptidomes. *Molecular & Cellular Proteomics* 21, 3 (2022).

[19] Bruce Rannala. 2002. Identifiability of parameters in MCMC Bayesian inference of phylogeny. *Systematic biology* 51, 5 (2002), 754–760.

[20] Christian P Robert, Víctor Elvira, Nick Tawn, and Changye Wu. 2018. Accelerating MCMC algorithms. *Wiley Interdisciplinary Reviews: Computational Statistics* 10, 5 (2018), e1435.

[21] Ugur Sahin, Evelyna Derhovanessian, Matthias Miller, Björn-Philipp Kloke, Petra Simon, Martin Löwer, Valesca Bukur, Arbel D Tadmor, Ulrich Luxemburger, Barbara Schrörs, et al. 2017. Personalized RNA mutanome vaccines mobilize poly-specific therapeutic immunity against cancer. *Nature* 547, 7662 (2017), 222–226.

[22] Sameer Singh, Michael Wick, and Andrew McCallum. 2012. Monte carlo MCMC: Efficient inference by sampling factors. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction (AKBC-WEKEX)*. 111–115.

[23] Alexander Smola and Shravan Narayanamurthy. 2010. An architecture for parallel topic models. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 703–710.

[24] Gloria I Valderrama-Bahamóndez and Holger Fröhlich. 2019. MCMC techniques for parameter estimation of ODE based models in systems biology. *Frontiers in Applied Mathematics and Statistics* 5 (2019), 55.

[25] Jason Vertrees, Phillip Barritt, Steve Whitten, and Vincent J Hilser. 2005. COREX/BEST server: a web browser-based program that calculates regional stability variations within protein structures. *Bioinformatics* 21, 15 (2005), 3318–3319.

[26] Darren J Wilkinson. 2006. Parallel bayesian computation. *Statistics Textbooks and Monographs* 184 (2006), 477.

[27] Shaojun Zhao and Daniel Gildea. 2010. A fast fertility hidden Markov model for word alignment using MCMC. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. 596–605.

[28] Martin Zinkevich, John Langford, and Alex Smola. 2009. Slow learners are fast. *Advances in neural information processing systems* 22 (2009).