GPU Acceleration of Conformational Stability Computation for CD4+ T-cell Epitope Prediction

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

Samuel J. Landry
Department of Biochemistry and Molecular Biology
Tulane University School of Medicine
New Orleans, LA, USA
landry@tulane.edu

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

2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) | 979-8-3503-8622-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/BIBM62325.2024.10821831

***Abstract*—CD4+ T cells play a crucial role in adaptive immunity and are a significant component of immunological response in many settings. Computational prediction of which antigenic peptides are presented and bind to T cells is a problem that has been studied for several decades. Current efforts apply supervised learning methods to predict peptide-MHCII binding, but do not incorporate the role of antigen processing. To address this, our group developed the Antigen Processing Likelihood (APL) algorithm, which relies on a free energy-based conformational stability metric known as COREX. COREX requires the analysis of a potentially large conformational ensemble and is thus computationally intensive. In recent prior work, we parallelized this analysis with an algorithm we called *pCOREX*. pCOREX reduced the computation time from hours/days to minutes, demonstrating a near-ideal speedup on 192 CPU cores. In this paper, we achieve an even more substantial acceleration of the COREX algorithm by making use of GPU cores, and demonstrate a reduction in computation time from minutes to seconds.**

***Index Terms*—CD4+ T cell epitope prediction, antigen processing, free energy computation, GPU parallelization**

I. Introduction

Innate and adaptive immunity offer the primary defenses against illness caused by pathogens and cancers [1]. In this paper, we focus on a fundamental mechanism for launching CD4+ T cells, or "helper" T cells. CD4+ T cell roles include immune response initiation, maintenance, and memory [2]. CD4+ T cell immune response is initiated through the MHCII pathway, in which a peptide-MHC complex binds to a T cell receptor (TCR) on the surface of an antigen-presenting cell (APC) [3], [4]. In the MHCII pathway, antigens are first taken up by APCs and then undergo processing within the endosome. During processing, antigens are cleaved into peptides concurrently with binding to MHC-II molecules. The resulting peptide-MHCII complexes are transported to the APC surface and recognized by CD4+ T cells. The problem of *computational epitope prediction* asks us to take antigen sequence/structure as input, and predict the constituent peptides (i.e., epitopes) that will initiate a CD4+ T cell response. Consequently, computational epitope prediction has been an active area of research. Existing methods make use primarily of sequence-based, MHCII allele-specific machine learning methods to predict peptide-MHC binding (e.g., SMM [5], [6], NetMHC [7], [8],NetMHCpan [9], [10], NetMHCcons [11]). These supervised methods are developed based on the training data collected from MHC binding assay experiments, which do not consider the antigen structure and thus do not make use of the fact that proteolytic cleavage and MHCII binding occur concurrently.

To utilize antigen 3D structure for epitope prediction, our group has designed the *Antigen Processing Likelihood* (APL) algorithm [12]. Because dominant CD4+ T-cell epitopes are often located near conformation flexible regions of antigens [13]–[15], this method aggregates several conformational stability metrics with trainable weights to locate epitopes [12]. We have in fact shown that combining APL with MHCII binding prediction surpasses previous state-of-the-art T-cell epitope prediction tools and achieves higher accuracy on various benchmarks [12], [16].

A key requirement for APL is COREX (CORrelation with hydrogen EXchange protection factors) [17], a conformational stability metric that, depending on antigen size, consumes hours to days of computation time. In recent work, our group has developed a parallelized COREX algorithm (pCOREX) that has near-ideal parallel speedup. Using 192 CPU cores, pCOREX can reduce the worst-case computation time for an antigen to about 30 minutes [18]. However, pCOREX is still 1-2 orders of magnitude slower than the time required for MHCII binding prediction (e.g. using NetMHC).

In this paper, we provide a novel algorithm for computing COREX using GPU cores that we call *gpuCOREX*. This allows us to scale up parallelization to make use of thousands of GPU cores (on a desktop machine or larger server), versus hundreds of CPU cores on a cluster. To achieve this level of performance we introduce several novel algorithmic methods: memoization between conformational states, a novel adaptive Monte Carlo sampling method, and joint CPU/GPU scheduling. We demonstrate our algorithm on two benchmark sets: a set of 18 pathogenic antigens, and a set of 64 antigens from a large-scale immunopeptidomics study. We find that gpuCOREX achieves near-identical predictions to pCOREX, but still improves performance by over an order of magnitude. With an Amazon AWS cloud GPU instance with 8 V100 cards, computation time is about 30 seconds for the vast majority of antigens. With this level of performance APL computation is now on par with MHCII binding prediction.

II. Background

In this section, we provide a high-level description of our APL algorithm as well as the free-energy approximation approach of the COREX algorithm. We provide more detail on the COREX algorithm as that is the focus of this paper. A detailed exposition and comprehensive experimental validation of the APL algorithm can be found in our works that focus on epitope prediction [12], [16], [19].

A. APL

The Antigen Processing Likelihood (APL) algorithm aims to generate an epitope likelihood profile by incorporating structural constraints of antigens that influence antigen processing. The algorithm utilizes four sources of conformational stability data: sequence entropy, crystallographic B-factors, COREX stability constants, and solvent-accessible surface area. An aggregate stability profile is created from these factors. Regions identified as transitional in terms of conformational stability are assigned higher or lower antigen processing scores based on this profile [12], [16], [19]. The main approach is to upweight parts of the protein that are adjacent to regions of conformational instability, using the fact that proteolytic cleavage is likely to make those parts available for MHCII binding. In practice, we have shown that APL, along with MHCII binding can together significantly improve epitope prediction [12], [18].

B. COREX

The COREX metric quantifies the degree of instability at each residue by considering the structural ensemble of a protein under equilibrium conditions [20]. The high-level approach is to conduct a free-energy approximation over a coarse-grained structural ensemble. To define the coarse-grained ensemble, COREX partitions the given antigen protein into *folding units* of a prescribed size w. For convenience, we will also assume each folding unit has at most W atoms. Then, with various offsets of partition beginning residue from the first residue to the w-th residue, the given protein is partitioned to w different partition schemes. Each folding unit then is allowed to be in either a folded (F) or unfolded ($\bar{F}$) state. For an antigen with N amino acid residues in the primary sequence, we would have $n = \lceil N/w \rceil$ folding units per partition scheme. A *microstate* is defined as a particular setting of folding units in a given partition scheme; this is meant to represent a partially folded state of the protein. When convenient we will represent a microstate as a binary vector with a 0 denoting an unfolded folding unit and 1 denoting a folded folding unit.

The stability at any particular residue j is then defined by the microstates in which j is folded versus in which j in unfolded. More concretely, for each residue j, the conformation stability κ_j is defined as

$$\kappa_j = \frac{P_j^F}{P_j^{\bar{F}}}, \tag{1}$$

where P_j^F and $P^{\bar{F}}$ denote the probability that residue j appears in the folding unit which is folded or unfolded respectively. The probability of any microstate can then be defined as

$$P_C = \frac{e^{-\frac{\Delta G_C}{RT}}}{Q} \tag{2}$$

where $e^{-\frac{\Delta G_C}{RT}}$ is the statistical weight of states. ΔG_C, R, and T denote Gibbs energies, universal gas constant, and absolute temperature separately. Here, Q is the sum of the statistical weights.

$$Q = \sum_{\text{state } C} e^{-\frac{\Delta G_C}{RT}}. \tag{3}$$

The Gibbs energy term can be computed with enthalpy (ΔH_C) and entropy(ΔS_C) of state C at temperature is T:

$$\Delta G_C = \Delta H_C - T\Delta S_C. \tag{4}$$

The COREX algorithm does not actually compute enthalpy and entropy terms, but instead estimates them using a linear approximation parameterized by solvent-accessible surface area [20]. Specifically,

$$\Delta S_C = \Delta S_C^{\text{Solv}} + W\Delta S_C^{\text{Conf}} \tag{5}$$

where ΔS_C^{Solv} and ΔS_C^{Conf} are solvent entropy and conformation entropy respectively and W is entropy weighting. The ΔS_C^{Solv} can be estimated from apolar (ΔCp_{apol}) and polar (ΔCp_{pol}) heat capacity following (6), which can be determined by changes in solvent-accessible surface area.

$$\Delta S_C^{\text{Solv}} = \Delta Cp_{\text{apol}} \cdot \ln(T/385) - \Delta Cp_{\text{pol}} \cdot \ln(T/335) \tag{6}$$

With the probability of a single microstate defined as above, it is efficient to compute. However, even a modestly-sized protein would have a massive set of microstates even with a folding unit size of 10. To address this, we can the structural ensemble [21], with the Monte Carlo sampling condition that can be described as (7).

$$e^{\frac{\Delta G_C}{\Delta G_U}} \geq P \cdot r \tag{7}$$

where the r is a uniformly chosen value between $[0, 1]$, P denotes probability threshold between $(1, 0]$, and ΔG_U is Gibbs energy for the native state. In [18], we leveraged hundreds of CPU cores to achieve significant speedup. In this paper, we demonstrate that GPU cores can provide an additional 1-2 orders of magnitude speedup.

III. Our Approach

In this section, we will describe our framework for managing parallel computations on GPU cores to evaluate a structural ensemble. Our approach is to use a novel precomputation scheme to parallelize SASA computation. We also introduce a novel Monte Carlo sampling procedure that improves upon the method used in [18].

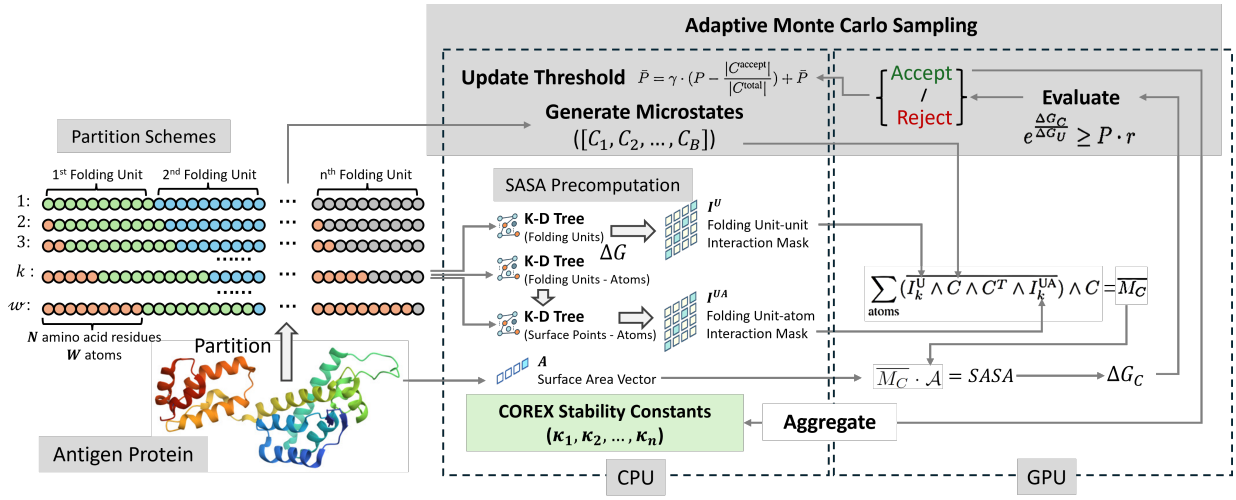

Fig. 1. **Overview of gpuCOREX.** Our algorithm spans both the CPU and GPU. As shown, the CPU carries out precomputation steps while GPU resources carry out the calculation of ΔG_C values for a batch of conformations.

A. *Parallel Computation of Solvent Accessible Surface Area*

As discussed in Section II, the COREX metric is parameterized entirely by the solvent accessible surface area (SASA) associated with a particular microstate. In this section we therefore focus just on the efficient parallel computation of SASA across a large ensemble of partially folded microstates. Once we are able to obtain SASA for a given microstate, we can use (5) to compute the quantities needed to compute stability constants in (1). To define SASA for a partially folded microstate, we use the predefined constant values for the amino acids in unfolded folding units and then must compute the SASA of the remaining folded folding units. While this can be done independently for each microstate, there is potentially a large amount of overlapping, serial computation between microstates if done naively using a standard method such as the Shrake-Rupley SASA algorithm [22].

We take a two-stage approach to enable efficient GPU parallelization. First, we first conduct a round of CPU-based precomputation that captures pairwise proximity relationships between folding units and atoms and creates data structures holding precomputed SASA terms. Second, we pass these data structures to GPU cores so that the full SASA of *any* partially folded microstate can be computed quickly (without any geometric computation) and concurrently.

1) CPU Precomputation: For this step, we make use of a *K-D tree* [23], a binary tree structure computed over a k-dimensional point set to enable efficient nearest-neighbor searches. For example, if we construct a K-D tree on the atoms in an antigen structure then we can conduct rapid queries on the distances and *interaction* relationships between any pair of atoms. As is standard when computing SASA, we consider a pair of atoms to be interacting if their pairwise distance is close enough to exclude a water molecule.

For each of the w partition schemes, we use a K-D tree to identify and precompute the results of pairwise interactions between folding units in their folded and unfolded states. We refer to the precomputed quantity as an *interaction mask*; we will define several such masks and how they are used for GPU computation of SASA. First, let us define $I^{\mathrm{U}} \in \{0,1\}^{w\times n\times n}$, where n is the number of folding units in one partition scheme and w is the number of partition schemes. Our convention is that if $I^{\mathrm{U}}_{k,i,j} = 1$, it indicates that folding unit i interacts with unit j in partition scheme k. We note that we will define interaction masks as dense matrices for mathematical convenience, but all of our implementations make use of sparse representations for efficiency.

To construct I^{U}, the centers and radii of folding units are first determined by calculating the center of all atoms within the unit and measuring the distance from the center to the outermost atom, including the radius of the outermost atom. Then we use a K-D tree based on these centers and radii to determine interactions between different folding units for each partition scheme. This interaction information is precomputed and stored in I^{U}.

We also compute an interaction mask $I^{\mathrm{UA}} \in [0,1]^{w\times n\times n\times W\times p}$ between folding units and individual atoms, that identifies the surface point interaction of an atom in neighboring folding units. Here, p is the number of sampled surface points for one atom used for SASA calculation. Then for partition scheme k, $I^{\mathrm{UA}}_{k,i,j,a,b} = 1$ indicates that the b-th surface point of the a-th atom of folding unit i is interacts with any atom in folding unit j.

To build I^{UA} for each partition scheme, two K-D trees are constructed: one to identify interactions between atoms and folding units and another to identify interactions between atom surface points and other atoms. The first K-D tree is constructed using the centers and radii of the folding units, identifying potential folding units that contains atoms interacted with a given atom. This tree guides the construction of the second K-D tree, which analyzes the interactions between surface points and their potential interacted atoms. This K-D tree is based on the surface points of a single atom and is

used to identify all surface points that interact with another atom, excluding interactions with water molecules. Finally, this interaction information is stored in I^{UA}.

With these interaction masks defined, we have precomputed all the necessary pairwise promixity relationships and can associate SASA terms with all nonzero entries. We can build an auxiliary vector $\mathcal{A} \in \mathcal{R}^{n \cdot W}$ for this purpose. For atom i, the $\mathcal{A}_i$ can be defined as:

$$\mathcal{A}_i = \frac{4 \cdot \pi}{n} \cdot (r_i + r^p)^2 \tag{8}$$

where r_i is the radius of atom i and r^p is the radius of the probe (i.e., a water molecule).

2) GPU Computation: With folding unit interaction masks I^{U}, folding unit-atom interaction masks I^{UA} and $\mathcal{A}$, it is easy to calculate SASA for any given microstate C from a given partition scheme k. The number of atom surface points that do not interact with another atom can be calculated for the given microstate as

$$\overline{M_C} = \sum_{\text{atoms}} (\overline{I_k^{\text{U}} \wedge C \wedge C^T \wedge I_k^{\text{UA}}}) \wedge C \tag{9}$$

where the summation is across matching atoms in the C and $\wedge$ denotes the logical and operation. Then the SASA for a particular microstate C can be calculated as $\overline{M_C} \cdot \mathcal{A}$. These computations are dispatched to different GPU cores. Each core takes responsible for computation of just one unit of the mask operation in (9). Therefore, the full mask computation can be computed concurrently.

B. Computation of Thermodynamic Quantities

Now that we have defined how to compute SASA efficiently, we describe how the stability constants (e.g. via (6)) can be computed. We first note that the summations of thermodynamic quantities for the protein backbone and non-backbone regions is distinct. In fact this distinction can be encoded in the masks themselves and thus be incorporated during GPU computations. Therefore, the properties of folding units, residues, and atoms are converted to various masks and vectors, which can be computed by GPUs in parallel. These masks can be computed at the CPU precomputation stage and reused between micro-states. For example, to compute ΔCp_{apol} and ΔCp_{pol} in (6), we incorporate the differences between the respective carbon atoms into the appropriate interaction mask.

C. Adaptive Monte Carlo Sampling

The previous subsections outlined how microstate contributions are computed and incorporated into the calculation of stability constants. We now focus on defining how we choose the microstate ensemble with a novel Monte Carlo sampling scheme. In the original COREX algorithm, (7) was used to select states. However, this simple approach depends critically on the Gibbs energy of native state ΔG_U. Ideally the actual fraction of accepted microstates should be close to P, the threshold given in (7). However, this depends heavily on the ΔG_U. In particular if ΔG_U is extremely small then the acceptance rate can be very high (100%), and vice versa. An acceptance rate that is too high will yield low quality microstates, while a low acceptance rate will require far more sampling time. The latter issue arises especially in proteins with extended conformations.

To address this difficulty, we make use of an adaptive threshold $\bar{P}$ that is adjusted while tracking the actual acceptance rate. To do this we simply check the difference between actual accept rate and expected accept rate and adjust the sampling threshold based on the difference, using:

$$\bar{P} = \gamma \cdot (P - \frac{|C^{\text{accept}}|}{|C^{\text{total}}|}) + \bar{P} \tag{10}$$

where C^{accept} and C^{total} denote the accepted states and all visited states separately. $\bar{P}$ represents the adaptive threshold utilized in sampling condition and P is the expected accept rate. γ is adaptive rate between $(0, 1]$ to control the threshold adjustment speed and avoid a sharp increase or decrease to the actual accept rate.

D. CPU/GPU Scheduling

Now that we have defined how to process each microstate and how to select the overall ensemble, we can define how to coordinate CPU and GPU computations. Above we described an adaptive Monte Carlo sampling scheme that iteratively selects microstates. However this limits parallelization, since we can dispatch a large number of microstates for concurrent ΔG_C calculations. Below we describe how CPU cores orchestrate and dispatch GPU processing.

The COREX computation for each partition scheme is maintained by an independent CPU core. First, each core conducts precomputation for SASA and thermodynamic quantities. Second, each core generates a batch of microstates and dispatches these for GPU processing. After ΔG_C values are computed concurrently, we accept/reject microstates based on the adaptive Monte Carlo sampling criterion. Once all batches are completed, a lead core aggreates thet results.

IV. Results

We implemented gpuCOREX in Python using PyTorch, an open-source machine learning and scientific computation library that supports GPU computing [24]. To evaluate the practical performance of gpuCOREX, we use three distinct benchmarks. First we consider the quality of COREX predictions over a benchmark of small proteins for which we are able to exhaustively enumerate the microstate ensemble, to determine the effectiveness of our adaptive Monte Carlo approach. We then evaluate gpuCOREX on two additional benchmarks: a collection of pathogenic antigens and a set of self antigens used in an experimental large-scale study of MHCII presentation. We compare gpuCOREX to pCOREX [18] as well as across several different hardware configurations. Overall, we find that gpuCOREX yields high quality COREX stability constants and thus enables APL computation at a speed analogous to current MHCII binding prediction tools.

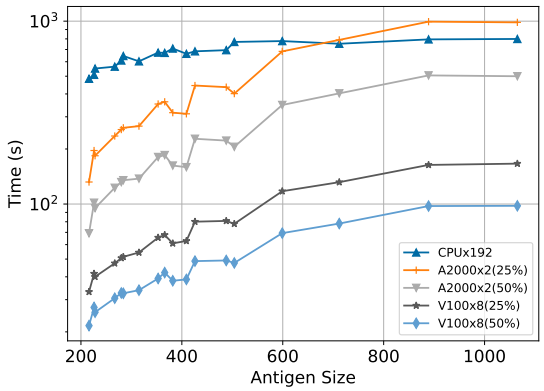

Fig. 2. **gpuCOREX performance for 18 antigens in the pathogen benchmark.** When compared to pCOREX, our gpuCOREX algorithm, the median runtime is significantly lower. A median runtime of 720s for pCOREX can be reduced 334.2s seconds and 171.56 seconds on 2 A2000 GPUs, and further to 64.2s and 39s on 8 V100 GPUs with adaptive Monte Carlo sampling, respectively.

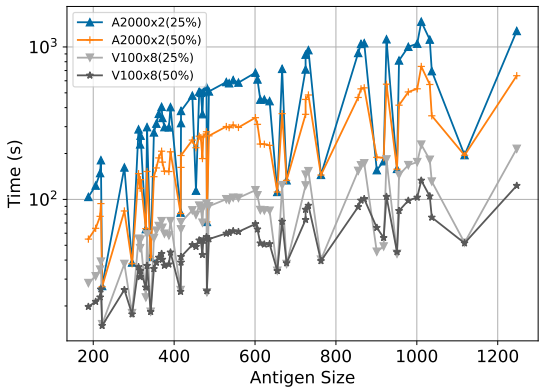

Fig. 3. **gpuCOREX performance for the 64 antigens in the Mellins benchmark.** Using 2 A2000 GPUs and 8 V100 GPUs, the median runtimes of gpuCOREX with adaptive Monte Carlo sampling are 391.6s and 206.1s with 25% and 50% thresholds, respectively, for the A2000 GPUs. For the V100 GPUs, the median runtimes are 72.03 seconds and 50.53 seconds, respectively.

A. Experimental Configuration

For all experiments, we follow the configuration from pCOREX experiments in [18]: for the small protein benchmark we use 50 samples per partition and for the remaining data sets we use 10,000 samples per partition. We consider two GPU environments for our experiments: a standard desktop configuration with two A2000 NVIDIA GPUs (24GB VRAM) and two Intel Xeon W-2245 3.9GHz CPUs (64GB RAM) and an Amazon EC2 P3.x16large instance with 8 V100 NVIDIA GPUs (128GB VRAM) and 96 virtual Intel E5 2628 3 GHz CPUs (488GB RAM).

B. Evaluation of Adaptive Monte Carlo Sampling

To evaluate our novel sampling approach, we compiled a benchmark of 13 small proteins, ranging in size from 63 to 117 residues. The benchmark was chosen so that we can compute COREX stability constants using the full microstate ensemble for these proteins to obtain "ground truth" output. We can then use these constants as a baseline to evaluate the quality of our adaptive Monte Carlo sampling scheme by measuring the percentage root-mean-squared deviation (RMSD%):

$$\text{RMSD}\%(y,\hat{y}) = \frac{100 \cdot \sqrt{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}}{\max(\hat{y} \cup y) - \min(\hat{y} \cup y)}\% \qquad (11)$$

where y and $\hat{y}$ denote ground truth and prediction values respectively.

We consider the output quality as well as the runtime of adaptive Monte Carlo sampling with 10%, 25%, 50%, 75%, and 90% thresholds and compare these to the fixed 75% threshold used in pCOREX [18] as a baseline. With a 50% threshold, the COREX output accuracy and computational time are similar to the baseline, with about a 10.0% deviation from the ground truth and requiring about 7.9 seconds. When the threshold is 25%, although slightly slower at approximately 8.2 seconds, the RMSD Percentage improves to about 7.5%, indicating better quality results than the baseline. Thresholds that are lower or higher do not seem to be justified given the time cost relative to RMSD improvements. Therefore, for the evaluations that follow, we consider just the 25% and 50% thresholds for adaptive Monte Carlo to simulate settings that practitioners may potentially consider.

C. Benchmark of Pathogenic Antigens

The pathogen benchmark comprises 19 proteins that induce immune responses in C57BL/6 mice and human subjects. These proteins range in size from 216 to 1,065 residues. This benchmark reflects a use case in which we may wish to conduct epitope prediction in the context of vaccine design. The runtimes for pCOREX with Monte Carlo sampling using a 75% threshold run on 192 processor cores represent our best results from prior work [18]. The runtimes of the proposed gpuCOREX with adaptive Monte Carlo sampling using different thresholds, including 25% and 50%, evaluated on 2 A2000 GPUs and 8 V100 GPUs, are compared to this baseline. The adaptive Monte Carlo gpuCOREX achieves runtimes of 334.15 seconds and 171.5 seconds on 2 V2000 GPUs with probability thresholds set to 25% and 50%, respectively, resulting in speedups of 2x and 3.8x compared to the baseline. When executed on 8 V100 GPUs, the runtimes decrease to 64.2s and 39s seconds, achieving remarkable speedups of 10 and 16 times over our prior best results.

These experiments clearly demonstrate that gpuCOREX significantly improves upon CPU-based parallelization. On a high-performance platform like a cloud server with 8 V100 GPUs, tasks can be completed in less than 100 seconds in the worst case and on average 38s. In a desktop environment with 2 A2000 GPUs, tasks can be solved in under 505s and more typically in less than a few minutes. This demonstrates that the proposed method is applicable to both large-scale datasets and personal tasks, across high-performance platforms and desktop environments.

D. Benchmark of proteins from MHCII immunopeptidomics studies

For our last test set we compiled a set of self-proteins that were evaluated in a large-scale MHCII presentation study. Olsson *et al.* [25] conducted a large scale immunopeptidomics study in which the effect of immunomodulatory molecules DM and DO were tuned in a mouse model to study the result on MHCII peptide presentation. This study considered over 10,000 peptides from hundreds of antigens. We collected a subset of 64 proteins ranging in length from 188 to 1,247 residues from this study for our benchmark. For this study we utilized gpuCOREX with adaptive Monte Carlo sampling using a 25% threshold on 2 A2000 GPUs serves as the performance baseline as a substitute for pCOREX performance. The performance of gpuCOREX with adaptive Monte Carlo sampling using a 50% threshold on 2 A2000 GPUs, and using both 25% and 50% thresholds on 8 V100 GPUs, is compared to this baseline. gpuCOREX takes median times of 391.6s and 206.1s 2 A2000 GPUs with 50% threshold and achieves approximately 1.74x average speedup over the baseline. On 8 V100 GPUs, gpuCOREX takes a median time of 72s and 50.5s with 25% and 50% thresholds, respectively. It achieves significant average speedups of about 5x and 7.7x compared to the baseline for these thresholds on 8 V100 GPUs.

Overall these results demonstrate the efficiency and scalability of gpuCOREX with adaptive Monte Carlo sampling across different GPU configurations, making it well-suited for handling large-scale computations in benchmarks from -omics studies, which typically involve hundreds of proteins.

ACKNOWLEDGMENT

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. We also thank James O. Wrabl and the Hilser Lab for helpful discussions.

REFERENCES

[1] J. S. Marshall, R. Warrington, W. Watson, and H. L. Kim, "An introduction to immunology and immunopathology," *Allergy, Asthma & Clinical Immunology*, vol. 14, no. 2, pp. 1–10, 2018.

[2] B. V. Kumar, T. J. Connors, and D. L. Farber, "Human t cell development, localization, and function throughout life," *Immunity*, vol. 48, no. 2, pp. 202–213, 2018.

[3] K. Shah, A. Al-Haidari, J. Sun, and J. U. Kazi, "T cell receptor (tcr) signaling in health and disease," *Signal transduction and targeted therapy*, vol. 6, no. 1, p. 412, 2021.

[4] J. Neefjes, M. L. Jongsma, P. Paul, and O. Bakke, "Towards a systems understanding of mhc class i and mhc class ii antigen presentation," *Nature reviews immunology*, vol. 11, no. 12, pp. 823–836, 2011.

[5] Y. Kim, J. Sidney, C. Pinilla, A. Sette, and B. Peters, "Derivation of an amino acid similarity matrix for peptide: Mhc binding and its application as a bayesian prior," *BMC bioinformatics*, vol. 10, pp. 1–11, 2009.

[6] B. Peters and A. Sette, "Generating quantitative models describing the sequence specificity of biological processes with the stabilized matrix method," *BMC bioinformatics*, vol. 6, no. 1, pp. 1–9, 2005.

[7] M. Nielsen, C. Lundegaard, P. Worning, S. L. Lauemøller, K. Lamberth, S. Buus, S. Brunak, and O. Lund, "Reliable prediction of t-cell epitopes using neural networks with novel sequence representations," *Protein Science*, vol. 12, no. 5, pp. 1007–1017, 2003.

[8] C. Lundegaard, K. Lamberth, M. Harndahl, S. Buus, O. Lund, and M. Nielsen, "Netmhc-3.0: accurate web accessible predictions of human, mouse and monkey mhc class i affinities for peptides of length 8–11," *Nucleic acids research*, vol. 36, no. suppl_2, pp. W509–W512, 2008.

[9] M. Nielsen, C. Lundegaard, T. Blicher, K. Lamberth, M. Harndahl, S. Justesen, G. Røder, B. Peters, A. Sette, O. Lund *et al.*, "Netmhcpan, a method for quantitative predictions of peptide binding to any hla-a and-b locus protein of known sequence," *PloS one*, vol. 2, no. 8, p. e796, 2007.

[10] I. Hoof, B. Peters, J. Sidney, L. E. Pedersen, A. Sette, O. Lund, S. Buus, and M. Nielsen, "Netmhcpan, a method for mhc class i binding prediction beyond humans," *Immunogenetics*, vol. 61, pp. 1–13, 2009.

[11] E. Karosiene, C. Lundegaard, O. Lund, and M. Nielsen, "Netmhccons: a consensus method for the major histocompatibility complex class i predictions," *Immunogenetics*, vol. 64, pp. 177–186, 2012.

[12] R. R. Mettu, T. Charles, and S. J. Landry, "Cd4+ t-cell epitope prediction using antigen processing constraints," *Journal of immunological methods*, vol. 432, pp. 72–81, 2016.

[13] S. J. Melton and S. J. Landry, "Three dimensional structure directs T-cell epitope dominance associated with allergy," *Clinical and Molecular Allergy*, vol. 6, pp. 1–12, 2008.

[14] H.-N. P. Nguyen, N. K. Steede, J. E. Robinson, and S. J. Landry, "Conformational instability governed by disulfide bonds partitions the dominant from subdominant helper T-cell responses specific for HIV-1 envelope glycoprotein gp120," *Vaccine*, vol. 33, no. 25, pp. 2887–2896, 2015.

[15] S. Carmicle, G. Dai, N. K. Steede, and S. J. Landry, "Proteolytic sensitivity and helper T-cell epitope immunodominance associated with the mobile loop in hsp10s," *Journal of Biological Chemistry*, vol. 277, no. 1, pp. 155–160, 2002.

[16] T. Charles, D. L. Moss, P. Bhat, P. W. Moore, N. A. Kummer, A. Bhattacharya, S. J. Landry, and R. R. Mettu, "Cd4+ t-cell epitope prediction by combined analysis of antigen conformational flexibility and peptide-mhcii binding affinity," *Biochemistry*, vol. 61, no. 15, pp. 1585–1599, 2022.

[17] J. Vertrees, P. Barritt, S. Whitten, and V. J. Hilser, "Corex/best server: a web browser-based program that calculates regional stability variations within protein structures," *Bioinformatics*, vol. 21, no. 15, pp. 3318–3319, 2005.

[18] Avik, J. Bhattacharya, S. Wrabl, R. Landry, and Mettu, "Parallel computation of conformational stability for cd4+ t-cell epitope prediction," in *2023 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2023.

[19] A. Bhattacharya, M. C. Lyons, S. J. Landry, and R. R. Mettu, "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*, 2022, pp. 1–10.

[20] V. J. Hilser and E. Freire, "Structure-based calculation of the equilibrium folding pathway of proteins. Correlation with hydrogen exchange protection factors," *Journal of molecular biology*, vol. 262, no. 5, pp. 756–772, 1996.

[21] V. J. Hilser and S. T. Whitten, "Using the corex/best server to model the native-state ensemble," *Protein Dynamics: Methods and Protocols*, pp. 255–269, 2014.

[22] J. Ribeiro, C. Ríos-Vera, F. Melo, and A. Schüller, "Calculation of accurate interatomic contact surface areas for the quantitative analysis of non-bonded molecular interactions," *Bioinformatics*, vol. 35, no. 18, pp. 3499–3501, 2019.

[23] N. Sample, M. Haines, M. Arnold, and T. Purcell, "Optimizing search strategies in kd trees," in *Fifth WSES/IEEE world multiconference on circuits, systems, communications & computers (CSCC 2001)*, 2001.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[25] N. Olsson, W. Jiang, L. N. Adler, E. D. Mellins, and J. E. Elias, "Tuning DO: DM ratios modulates MHC class II immunopeptidomes," *Molecular & Cellular Proteomics*, vol. 21, no. 3, 2022.