Tiako P.F. (ed) *Automated Systems, Data, and Sustainable Computing*. Chronicle of Computing. OkIP.
 https://doi.org/10.55432/978-1-6692-0001-7_5

Initial GPU Optimization of Template Modeling Score (TM-score)

Hannah Johnson[1], Armon White[2], Yogesh Kale[3*], Elijah MacCarthy[1*]

[1, 2, 1*]Departments of Computer Science and Mathematics, Lane College, Jackson TN, 38301
[3*]Department of Computational Science and Engineering, NC A&T State University
1601 E Market St, Greensboro NC, 27411

ABSTRACT

Accuracy in the prediction of protein structures is key in understanding the biological functions of different proteins. Numerous measures of similarity tools for protein structures have been developed over the years, and these include Root Mean Square Deviation (RMSD), as well as Template Modeling Score (TM-score). While RMSD is influenced by the length of the protein and therefore the similarity between superimposed models can be affected by divergent loops in the models, TM-score is rather a robust and a more accurate method. TM-score, however, is much slower than RMSD in terms of calculations for the optimal superimposed model. Here, we present initial optimization work on GPU-TM-score, a GPU accelerated Template Modeling Score for fast and accurate measuring of similarity between protein structures. Our optimization is based on OpenACC parallelization and performance analysis of bottleneck regions and the KABSCH algorithm for the calculation of optimal superimposition within parallel architectures. Our initial results indicate an average 3.14× speedup compared to original TM-score on a benchmark set of 20 protein structures. This speedup is recorded on an Nvidia Volta V100 GPU compared to an AMD EPYC 7742 64-core processor.

Keywords: GPU, Optimization, OpenACC, Protein Structure, TM-score.

I. INTRODUCTION

The accurate comparison of predicted protein structures plays an important role not only in protein structure modeling, but also in branches of structural biology such as protein folding classifications (Yang & Jeffrey, 2005) and structure-based protein function annotation (Ling-Hong & Ram, 2012; Yang & Jeffrey, 2005; MacCarthy, Perry, & KC, 2019). These areas cover almost all branches of contemporary structural biology (Yang & Jeffrey, 2005) and this has become possible due to the increase in the number of solved protein structures in the Protein Data Bank (PDB). Though the usage of protein structure comparison tools has become popular, the speed and accuracy of these algorithms is very essential in keeping up with the ever-increasing gap between known protein structures and sequences in the UNIPROT protein library (MacCarthy, Perry, & KC, 2019; MacCarthy, 2020; KC, 2017)

Root Mean Square Deviation (RMSD) is one of the earliest, simplest and most commonly used metric for comparing protein structures (Kabsch, 1978; Yang & Jeffrey, 2005). When two protein structures are superimposed, the measure of the average distance between their atoms is referred to as Root Mean Square Deviation (RMSD) (MacCarthy, 2020). It measures the root-mean-square distance between corresponding residues after an optimal rotation of one structure to another (Yang & Jeffrey, 2005). This measure of similarity between two superimposed atomic coordinates is quantitative and represented in equation (1) below. Though RMSD is fast in terms of speed, it weights the distances between all residue pairs equally, thus, a small number of local structural deviations could result in a high RMSD, even when the global topologies of the compared structures are similar (Ling-Hong & Ram, 2012). Also, RMSD is not only dependent on the overall goodness of fit but also dependent on the length of compared structures, which leaves the absolute magnitude of RMSD meaningless (Yang & Jeffrey, 2005).

As a result of the flaws of RMSD, several other algorithms have been developed. These methods are based off the RMSD and compute transformations that superimpose corresponding atoms from one structure onto another structure by reducing the root-mean-squared deviation (RMSD) between the coordinates of superimposed structures (Ling-Hong & Ram, 2012). Some of these methods include Template Modeling Score (TM-score), Global Distance Test (GDT) and the Longest Continuous Segment (LCS) (MacCarthy, 2020).

TM-score (Yang & Jeffrey, 2005) which is represented in equation (2) below uses a variation of Levitt–Gerstein's (LG) metric, (Levitt & Gerstein, 1998) that provides a length independent measure and limits the impact of divergent pairs of atoms in superimposed structures. TM-score is sensitive to global topology because it weights small distances stronger than larger ones.

Also, TM-score is normalized in a way that the magnitude of scores of random structures is not dependent on the size of the protein. Thus, an average pair of randomly related structures have the value of 0.17 (Yang & Jeffrey, 2005). The TM-score metric works by sampling different subsets of

* Corresponding author E-mail: emaccarthy@lanecollege.edu

atoms and using the Kabsch algorithm in guiding their superposition while evaluating the LG score over the entire protein. The optimal superposition can then be obtained from the several sampled atom subsets. Because of the numerous local superpositions that must be sampled, the TM-score algorithm is much slower than the calculations from the RMSD algorithm.

$$\mathrm{RMSD} = \sqrt{\frac{1}{L}\sum_{i=1}^{L} d_i^2} \qquad (1)$$

d_i is the distance between the two atoms in the i-th pair an L is the number of pairs of equivalent atoms.

$$\text{TM-score} = \mathrm{Max}\left[\frac{1}{L_{\text{Target}}}\sum_{i}^{L_{\text{ali}}}\frac{1}{1+\left(\frac{d_i}{d_0\left(L_{\text{Target}}\right)}\right)^2}\right]. \qquad (2)$$

L is the number of residues of the query sequence, L_{ali} is the number of aligned residues in a threading alignment. When L and L_{ali} are identical, then it implies the model is full length. d_{i} is the distance of the ith C_α pair between model and native after superposition, and $d_0 = 1.24\sqrt[3]{N-15} - 1.8$.

II. Proposed Innovation

Our proposed innovation to TM-score is to port bottleneck regions of the algorithm to the GPU using OpenACC. Though some optimization work on TM-score has been done in the past (Hung, 2012), this targeted the porting from an OpenCL approach. There is little or no work done with regards to porting TM-score to the GPU using OpenACC. Thus, our proposed innovation optimizes TM-score by using OpenACC and ensures portability of GPU TM-score. This portability of GPU TM-score refers to providing users the flexibility of using GPU TM-score with any kind of accelerators, unlike other GPU optimizations that tie ported applications to specific architectures (Ford, 2009; Preis, Virnau, & Paul, 2009; Stantchev & Dorland, 2008; Gross & Janke, 2011).

From our performance analysis of the sequential algorithm, we noticed that the KABSCH method for calculating optimal superimposed models in the TM-score algorithm is the most expensive. This accounts for close to 75% of the total computational time. Our focus therefore is porting this region of the application to the GPU and optimizing it for performance gain. We propose the usage of OpenACC parallel and kernel regions to port hotspots within this section of code to the GPU. Since the device cannot access the host memory, our main task in this parallelization is to optimize data transfers so we can begin seeing the effect of our OpenACC acceleration on the speed.

III. Methods

A. GPU and OpenACC Application Programming Interface

Recent generations of GPUs use a unified architecture that is much suitable for scientific computing (Ford, 2009) and these are referred to as General Purpose GPUs (GPGPU). The general-purpose GPUs are used extensively in scientific computing to optimize applications that are suitable for parallel architectures. These GPUs are designed for compute-intensive, highly parallel computations. By virtue of this, the TM-score algorithm is a good candidate for porting to the GPU since its calculations are compute-intensive.

The general-purpose GPU operations follow the Single Instruction Multiple Data (SIMD) paradigm, and it comprises several streaming multiprocessors. The GPU uses these streaming multiprocessors to achieve high throughput which is obtained through the local caches and on-chip memory, thus, reducing bandwidth to external memory. There are numerous transistors on the GPU compared to CPU and these are for the purpose of data processing rather than caching and flow control. This therefore results in the compute-intensive and highly parallel computations of the GPU (Block, Virnau, & Preis, 2010). Since the GPU operation follows the SIMD paradigm, it is more suitable for tasks that can be expressed as data-parallel computations. By so doing, data elements are mapped to the numerous parallel processing threads. There are thousands of threads on the GPU and these are put into several batches/groups. In CUDA, these group of threads are called thread block and in OpenACC, they are referred to as a gang.

Open Accelerators (OpenACC) are a directive-based Application Programming Interface (API) that optimizes and accelerates work on GPUs. OpenACC enables developers to write applications that offload codes with their associated runtime data to GPUs from a host CPU. This is accomplished by using preprocessor directives that work like the directives in OpenMP. This prevents writing low level functions that virtually change the original code.

B. OpenACC Implementation

Since the TM-score algorithm contains the KABSCH method which is compute intensive, we port this region and other bottleneck regions of code to the GPU using OpenACC. Within this algorithm is the calculation of the optimal rotation matrix as well as the covariance matrix. We split these computations within the Kabsch algorithm on the GPU by assigning each to a parallel region. By a parallel region, we mean a gang of thread blocks.

We therefore insert OpenACC parallel directives that map these regions of code to a gang of thread blocks executed in parallel. This ensures a simultaneous execution of the different regions. We also optimize the number of thread blocks launched based on the amount of work to be completed in each region. This way, not too many or too few threads are launched, rather, an optimized number to ensure a speedup. Also, the device (GPU) cannot access the host memory, thus, we move data associated with these regions to the GPU to ensure calculations are completed accurately.

The data movements are accomplished by using OpenACC data regions in moving required data unto and from the device. These data transfers introduce an enormous overhead challenge within the optimization. From our initial optimization, the enormous data transfers account for 73% of the total GPU time. It is necessary therefore to optimize these data transfers so we can begin seeing an influence on the computational time.

We accomplish this by merging several small data copies into single copies. Many small data copies happening at different times take away from time that is supposed to be used for computations. Thus, merging many small copies into larger ones enables the device to be devoted to compute-intensive operations. Also, results that are not immediately required on the host after calculations on the device are kept on the device without being transferred to the CPU. This way, data movement time is split in half.

Also, regions that are not necessarily bottlenecks but contain data that is needed on the GPU are moved to the device to reduce data transfers. After optimizing data transfers, we are able to reduce the data movement time to 27% of total GPU time.

IV. RESULTS

We present the results from our initial optimization efforts in this section. An NVIDIA V100 GPU is used for the GPU implementation whereas an AMD EPYC 7742 64-core processor is used for sequential runs.

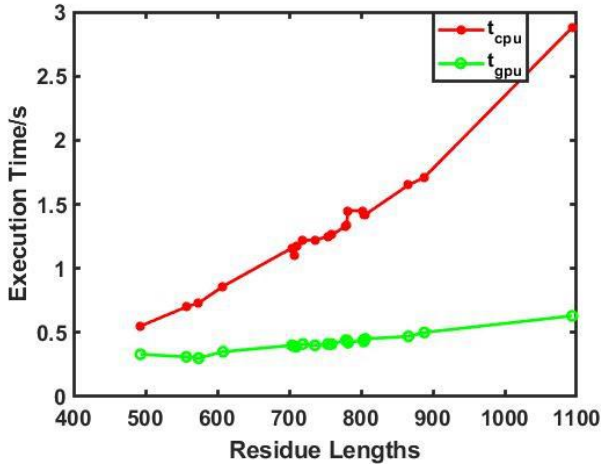

Fig. 1. GPU and CPU execution time of proteins of varying residue length.

On a benchmark dataset of 20 proteins obtained from the PDB, we record an initial average speedup of 3.14× as presented in Tab. 1. Also, from Tab. I, the peak recorded speed-up is 4.57× for residues within the range, 1001 - 2000. In Fig. 1, we present the GPU and CPU execution times against the residue lengths of protein structures being compared. It can be observed that the speedup is more pronounced as the residue length becomes larger. This is presented vividly in Fig. 2 where the residue ranges are plotted against the average execution times.

This in part is due to the fact that as residue lengths become larger, it takes considerably longer for the computations to be completed. Thus, the effect of the GPU optimization is much realized at this point. This means the efficiency of the optimization is clearly seen as the problem size becomes larger.

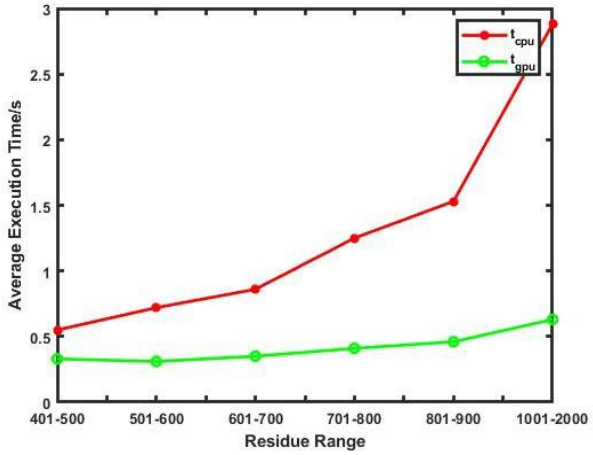

Fig. 2. GPU and CPU Execution time of proteins of varying residue length.

We have implemented an initial OpenACC optimized GPU-TM-score algorithm which is currently 3.14× faster than original TM-score. We hope to improve upon these results after further targeted porting and optimization. This work when completed, should help in comparing protein structures a lot faster by using any kind of accelerators as OpenACC works with all kinds of GPUs.

TABLE 1. Average Execution Time for Benchmark Data

Residue Range	t_{cpu}/s	t_{gpu}/s	SpeedUp
401 - 500	0.55 (1)	0.33	1.67
501 - 600	0.715 (2)	0.305	2.34
601 - 700	0.86 (1)	0.35	2.46
701 - 800	1.25 (10)	0.411	3.04
801 - 900	1.53 (5)	0.458	3.34
1001 - 2000	2.88(1)	0.63	4.57
Average	**1.298**	**0.414**	**3.14**

V. FUTURE WORK

We aim to optimize the GPU-TM-score algorithm further by targeting both the computations and the data transfers that have been reduced currently to 27%. Though this optimization is host-device memory bound, we are hoping to have the average speedup improved.

ACKNOWLEDGMENT

The authors would like to thank Dr. Melanie Van Stry and Dr. Aminah Gooch of Lane College for their input. This work

was supported in part by the NSF awards, HRD 2011938 and DUE 1833960.

REFERENCES

Betancourt, M., & Skolnick, J. (2001). Universal similarity measure for comparing protein structures. *Biopolymers*, 305–309.

Block, B., Virnau, P., & Preis, T. (2010). Multi-GPU accelerated multi-spin Monte Carlo simulations of the 2D Ising model. *Computer Physics Communications*, 1549-1556.

Ford, E. (2009). Parallel algorithm for solving Kepler's equation on Graphics Processing Units: Application to analysis of Doppler exoplanet searches. *New Astronomy*, 406-412.

Gross, J., & Janke, W. &. (2011). Massively parallelized replica-exchange simulations of polymers on GPUs. . *Computer physics communications.*, 1638-1644.

Hung, L. H. (2012). Accelerated protein structure comparison using TM-score-GPU. *Bioinformatics*, 2191-2192.

Kabsch, W. (1978). A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 827 –828.

KC, D. (2017). Recent advances in sequence-based protein structure prediction. *Briefings in bioinformatics*, 1021-1032.

Levitt, M., & Gerstein, M. (1998, May 26). A unified statistical framework for sequence comparison and structure comparison. *National Academy of sciences*, pp. 5913-5920.

Ling-Hong, H., & Ram, S. (2012). Accelerated protein structure comparison using TM-score-GPU. *Bioinformatics*, 2191–2192.

MacCarthy, E. (2020, May). Gpu Parallelization of Replica Exchange Monte Carlo Simulation and Application to Protein Structure Prediction. In *Doctoral dissertation, North Carolina Agricultural and Technical State University.* Greensboro: Proquest.

MacCarthy, E., Perry, D., & KC, D. (2019). Advances in protein super-secondary structure prediction and application to protein structure prediction. In K. A, *Protein supersecondary structures* (pp. 15-45). New York: Humana Press.

Murzin, A. G., Brenner, S. E., Hubbard, T., & & Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. . *Journal of Molecular Biology*, 536 –540.

Preis, T., Virnau, P., & Paul, W. &. (2009). GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model. *Journal of Computational Physics.*, 4468-4477.

Stantchev, G., & Dorland, W. &. (2008). Fast parallel particle-to-grid interpolation for plasma PIC simulations on the GPU. . *Journal of Parallel and Distributed Computing.*, 1339-1349.

Yang, Z., & Jeffrey, S. (2005). TM-align: a protein structure alignment algorithm. *Nucleic Acids Research*, 2302–2309.