GPU-Based Accelerations for Protein Sequence Alignment Using BLAST

Eric Arezza
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
ericarezza@cmail.carleton.ca

December 18, 2020

Abstract

This paper presents GPU parallel computing schemes for accelerating the NCBI BLAST tool for protein sequence alignment. An overview of BLASTP is described with five GPU-based methods named NCBI-GPU-BLAST, CUDA-BLASTP, GPU-BLASTP, CuBLASTP, and H-BLAST. Speedup analysis is presented as a major challenge in deciphering efficient parallelization while projecting speedup on modern hardware is estimated based on the presented methods.

1 Introduction

Parallel computing using a Graphics Processing Unit (GPU) is a commonly used hardware acceleration method to reduce the runtime of computations that would otherwise be performed serially. An advantage to only using a GPU instead of a computing cluster is that multiple machines are not required, so no message-passing for remote memory access is needed and the system can work standalone. A GPU is also cheaper, simpler, and provides a more general-purpose programmable option than employing FPGAs for parallel applications. Additionally, GPUs are ubiquitous, thus providing extensive support and functionality. For these reasons, using a GPU to speedup computations renders it a feasible and practical application for parallelization.

A useful application that can benefit from GPUs is computing sequence alignments of biological data such as protein's amino acid sequences. The Basic Local Alignment Search Tool (BLAST) is a standard tool in bioinformatics used by the National Center for Biotechnology Information (NCBI) that performs this function [1, 2]. Specifically, BLASTP is the program for protein sequence alignment and will herein be synonymously referred to as BLAST unless otherwise stated. In general, sequence alignment involves aligning a query sequence to subject sequence and calculating a similarity score between matching amino acids of the sequences. These alignments can help biologists identify functionally similar regions between sequences and understand evolutionary relatedness between organisms or cellular components [3]. A brief description of the BLAST process is explained in the following section to better understand its computational algorithm.

In many cases, multiple proteins may be desired for querying against entire databases of proteins, as done with BLAST, which requires each pairwise alignment to be performed and extends the time to obtain results. More significantly, as next-generation sequencing technologies improved sequencing throughput, protein databases have grown exponentially [4]. Due to this growth, parallel computing will be necessary to minimize the computing times for sequence alignment. Therefore, this paper addresses this topic by independently investigating GPU implementations of BLAST for accelerating protein sequence alignment. It begins with an overview of the BLAST algorithm, followed by descriptions of GPU methods, a problem statement, speedup analysis, and then concluding remarks.

2 Literature Review

2.1. BLAST Algorithm Overview

The BLAST algorithm can be divided into four sections with various opportunities for parallelization: seeding/hit detection, ungapped extension, gapped extension, and traceback.

1. To begin, proteins are parsed into a list of *k*-mer words of their amino acid sequence. For example, using the default *k* value of 3 for the sequence LMDKN would produce a word list of LMD, MDK, and DKN. The subject sequence from a protein database is then searched for matches to each word from the query, resulting in "hits". These words are put into a lookup table with position indices of hits as shown in Figure 1.

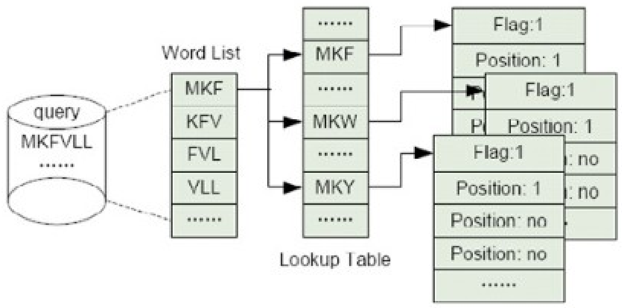


Figure 1: Word hit lookup table for seed extension [5].

2. Ungapped extension is performed whereby the seed is iteratively extended on each side and the resulting alignment between query and subject hit is scored based on a fixed scoring matrix and a given threshold. This can be visualized on an x-y plane with the query sequence on the y-axis and subject sequence on the x-axis. In this way, extension is performed along diagonals as each letter pair is additionally scanned from the seeds. If the seed extension increases the score then the seed continues extending, but if the score begins to drop off then the seed is no longer extended.

3. Gapped extension is then performed in a similar way allowing gaps in the seed extension and the resulting alignment (ungapped and/or gapped) becomes a local alignment hit between the query and subject sequences with an associated similarity score. If the score is above a given threshold, the alignment is saved as a High Scoring Pair (HSP). Figure 2 below illustrates an example of seed extension using the word IYP from a query against a subject sequence.

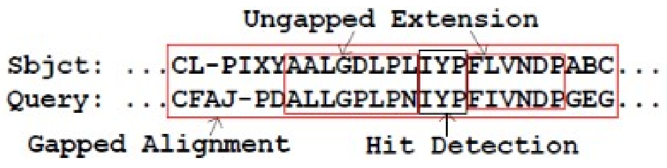

Ungapped Extension
Sbjct: ...CL-PIXYAALGDLPLIYPFLVNDPABC...
Query: ...CFAJ-PDALLGPLPNIYPFIVNDPGEG...
Gapped Alignment
Hit Detection

Figure 2: Illustration of BLAST seeding and extension for protein alignment [6].

4. Finally, these HSPs are traced back for sequence information, recalculated for alignment probability scoring, saved, and formatted for user output.

2.2. GPU-Accelerated Approaches

GPU-based BLAST algorithms have been published as early as 2010 [5] and have continuously been investigated. A summary of GPU strategies for accelerating certain stages of the BLASTP algorithm was presented by Said et al. [7] for CUDA-BLASTP [8], NCBI-GPU-BLASTP [9], GPU-BLASTP [6], and CuBLASTP [10] methods while proposing a strategy for parallelizing PSI-BLAST. Another earlier analysis of GPU accelerated methods for BLAST was conducted by Glasco [11]. It is apparent that this topic is recurring as new approaches are presented and the need for efficient BLAST acceleration grows.

Although a GPU-enabled cluster can certainly improve speedup as proposed in CLUS_GPU-BLASTP [12], this will typically offer a roughly linear speedup with cluster size relative to the GPU implementation used. Consequently, this paper will only investigate the uses of a GPU on a single machine for BLAST. The most recent GPU implementation for BLAST protein alignment was developed by Ye et al. named H-BLAST [13] utilizing NCBI-BLAST base code and functions from GPU-BLAST and will be compared with previous methods.

	NCBI-GPU-BLAST	**CUDA-BLASTP**	**GPU-BLASTP**	**CuBLASTP**	**H-BLAST**
Database (proteins size)	env_nr (6,031,291)	GenBank nr (9,230,955)	Ncbi nr (9,874,397)	env_nr (6,000,000), swissprot (300,000)	Ncbi nr (14,324,397)
Query Proteins	51 mouse (UniProt)	P14144, P42018, Q52TG9, Q52KR2, P08678	4 proteins	3 proteins	250 (swissprot), 6 groups (100 to 600)
Query Lengths/ Amino Acids	Up to 4498	127, 254, 517, 1054, 2026	1000 to 4000	127, 517, 1054	100 to 5000, maximum 9000
FSA or NCBI Base Code	NCBI	NCBI	FSA ver.1.05	FSA	NCBI
NCBI BLAST Version	2.2.24	2.2.22	N/A	N/A	2.2.28
Reported Comparisons	3-4x BLASTP	Up to 10x BLASTP	6x BLASTP, 2x CUDA-BLASTP	6x BLASTP, 3.4x FSA-BLAST, 2.8x CUDA-BLASTP, 1.9x NCBI-GPU-BLASTP	4-10x BLASTP, 1.5-4x NCBI-GPU-BLASTP
GPU Used	Fermi C2050	Tesla GeForce GTX 280, Tesla GeForce GTX 295	Tesla C1060, Fermi C2050	Kepler K20c	Kepler K20x, Kepler K40m
CUDA Version	N/A	2.3	3.1	5.0	5.5
Available	http://archimedes.cheme.cmu.edu/?q=gpublast	https://sites.google.com/site/liuweiguohome/software	N/A	https://github.com/vtsynergy/cuBLASTP	https://github.com/Yeyke/H-BLAST

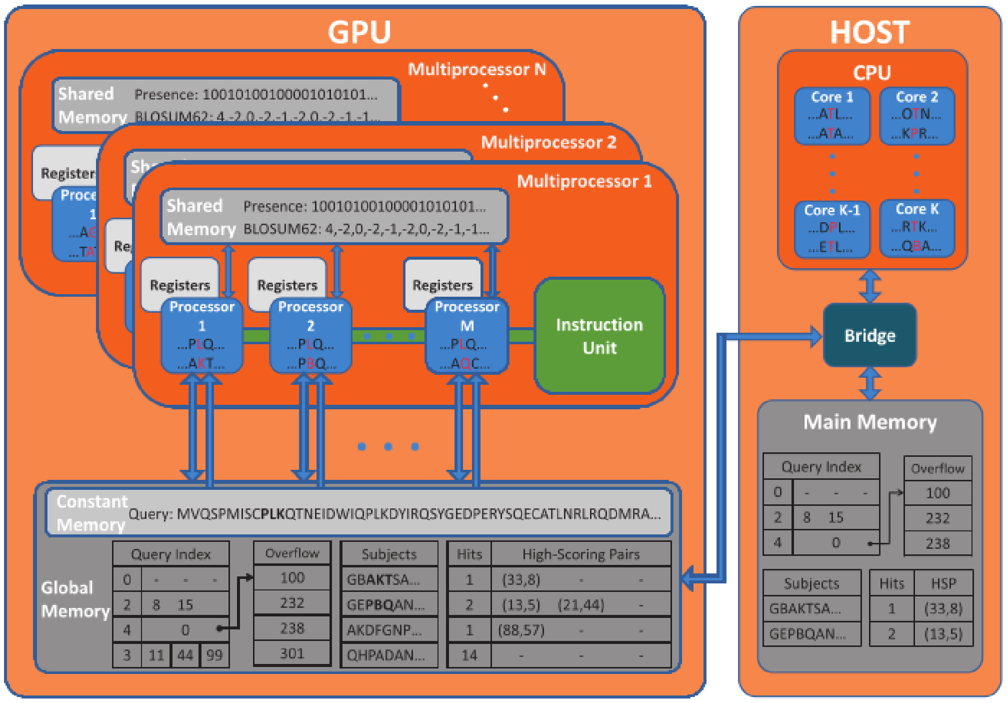
Table 1: Summary of GPU Methods for BLASTP.

BLAST's heuristic approach for sequence alignment was an initial advantage to faster processing times than the more accurate Smith-Waterman algorithm and contributed to its adoption as a standard tool. An enhancement of BLAST was further developed independently, termed Faster Search Algorithm (FSA-BLAST) to reduce computational complexity and time by about half per query while maintaining nearly identical functionality and output as BLAST [14]. Some GPU strategies implement the FSA-BLAST base code along with their parallel approach. For example, GPU-BLAST is built over FSA-BLAST while NCBI-GPU-BLAST is built over the original BLAST code. Thus, these methods should be compared appropriately.

Often, results are reported relative to each other which may use different NCBI-BLAST or FSA-BLAST base code, compared under different database and query searches, and more significantly using different CUDA-compatible GPUs and software as seen in Table 1 above. This investigation will re-evaluate previous results of such approaches providing an independent comparison of their performance relative to their implementations. A brief description of each parallelization scheme is discussed in the following subsections. Further analysis will be discussed in subsequent sections regarding BLAST algorithm speedup.

2.2.1. NCBI-GPU-BLAST

This method is originally named GPU-BLAST, but referred to here as NCBI-GPU-BLAST to differentiate it as being built using NCBI-BLAST code instead of FSA-BLAST. The authors identified seed and ungapped extension to consume up to 75% of compute time for BLAST searches. Thus, it focused its parallelizing efforts on these first two stages of BLAST. The intuition behind this method was to allocate BLAST data to GPU memory in which the most frequently accessed data resides in locations with the fastest read/write capability as shown in their schematic in Figure 3. However, the size of some data structures required them to be placed in global memory as further listed in Table 2.



Query Index			
0	-	-	-
2	8	15	
4	0		
3	11	44	99

Overflow
100
232
238
301

Subjects	Hits	High-Scoring Pairs		
GBAKTSA...	1	(33,8)	-	-
GEPBQAN...	2	(13,5)	(21,44)	-
AKDFGNP...	1	(88,57)	-	-
QHPADAN...	14	-	-	-

Query Index			
0	-	-	-
2	8	15	
4	0		

Overflow
100
232
238

Subjects	Hits	HSP
GBAKTSA...	1	(33,8)
GEPBQAN...	2	(13,5)

Figure 3: Memory allocation of BLAST data for NCBI-GPU-BLAST [9].

BLAST Data	**Data Structure**	**GPU Memory**	**Reasoning for Allocation**
Subsitution Matrix (BLOSUM62)	Table	Shared	Frequently used in alignment score calculations
Words present in query sequence	Vector	Shared	Small and provides fast access to check presence of word in query when matching to subject
Query sequence	Vector	Constant	Small and accessed by all threads
Query words index locations	Table	Global	Large data size
Query words index locations (overflow)	Table	Global	Large data size
Database subject sequences	Table	Global	Large data size
Indices of resulting ungapped alignments	Table	Global	Large data size
N/A	N/A	Local/Texture	Unused/desired when global memory is a bottleneck, not thread divergence

Table 2: BLAST data structures and memory organization in NCBI-GPU-BLAST.

In NCBI-GPU-BLAST, database subject sequences are first sorted by size and then given to seperate threads for parallel execution. Sorting the sequences prior to execution prevents thread diveregence as each thread in a warp can complete in similar times when comparing the query length to different subject lengths, unlike in Figure 4 where different hit detections between threads prevents parallel ungapped extension since hits are not predetermined. This enables the parallel executions to coalesce more efficiently. Additionally, NCBI-GPU-BLAST makes use of the host CPU to speedup BLAST. This is done by dividing the database sequences into a GPU workload and CPU workload such that both GPU and CPU complete their BLAST alignments at the same time and the CPU merges the results.

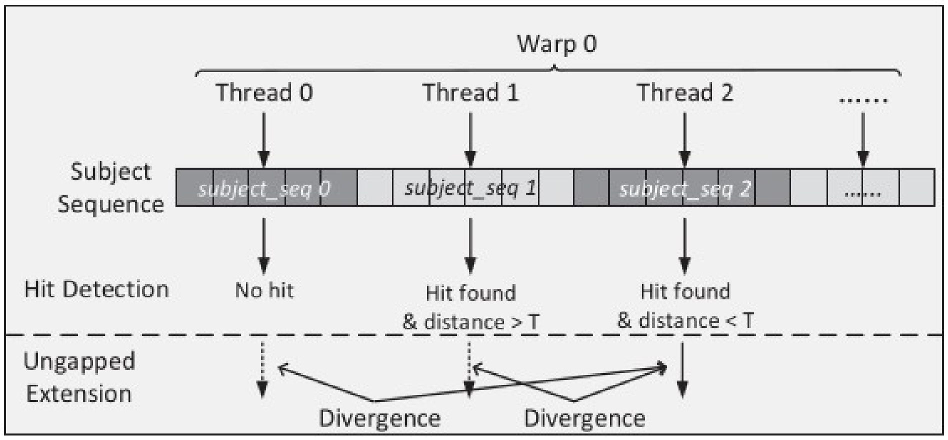


Figure 4: Branch divergence between GPU threads in hit detection preventing efficient parallel ungapped extension [10].

2.2.2. CUDA-BLASTP

This parallel approach is one method that attempts to speedup hit detection and ungapped extension as well as the gapped extension stage of BLAST. The first two stages are performed by dividing the database subject sequences into batches and providing different GPU threads with a batch of sequences to perform hit detection and ungapped extension. This is deemed a coarse-grained approach with results of each thread's BLAST batch fed back to the host CPU. Additionally, CUDA-BLASTP employs the use of a deterministic finite-state automation (DFA) structure as in Figure 5 for the query's word lookup table to speedup hit detection by bringing together frequently accessed query word positions with similar neighboring words. Each entry in the DFA points to an array of query positions and the DFA is mapped to a 2D texture in texture memory for efficient access by all threads.

AA	DFA[0].NextGroup	DFA[0].NextWord
...	...	...
AY	DFA[19].NextGroup	DFA[19].NextWord
CA	DFA[20].NextGroup	DFA[20].NextWord
...	...	...
CY	DFA[39].NextGroup	DFA[39].NextWord
⋮	DFA[*i*].NextGroup	DFA[*i*].NextWord
YA	DFA[300].NextGroup	DFA[300].NextWord
...	...	...
YY	DFA[399].NextGroup	DFA[399].NextWord

Figure 5: DFA structure used by CUDA-BLASTP [8].

For gapped extension, CUDA-BLASTP uses a banded alignment algorithm that calculates the alignments centered around the HSPs by a fine-grained parallel approach. This is performed by cycling through three shared-memory arrays of size Y to calculate along the X minor diagonals seen in Figure 6.

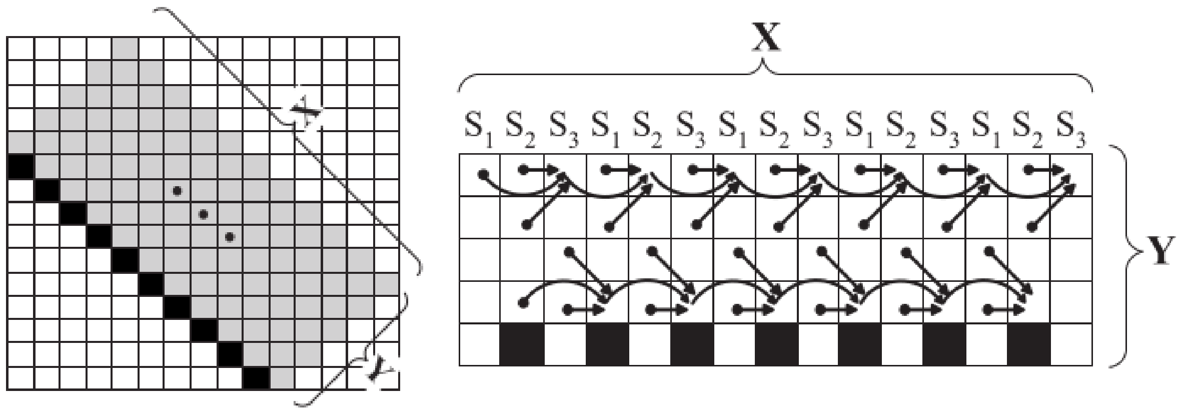

X
Y
X
S_1 S_2 S_3 S_1 S_2 S_3 S_1 S_2 S_3 S_1 S_2 S_3 S_1 S_2 S_3
Y

Figure 6: Banded alignment of Y threads over X minor diagonals for gapped extension in CUDA-BLASTP. Dots represent HSPs and black cells define the size of Y [8].

2.2.3. GPU-BLASTP

GPU-BLASTP was built over FSA-BLAST code and experimented with changing BLAST allocation of constant memory, texture memory, L1 memory, and GPU-CPU workload balancing for hit detection, ungapped, and gapped extension phases. The authors have suggested using constant memory for the query sequences and scoring matrix, caching the subject sequences and word lookup table in texture memory, and applying the GPU for hit detection and ungapped extension while multithreading gapped extension on the CPU. GPU-BLASTP is the only method mentioned here has not made their code available.

2.2.4. CuBLASTP

Also built over FSA-BLAST base code, CuBLASTP attempts to address the thread divergence issue for BLAST on GPUs while parallelizing hit detection and ungapped extension. It applies a fine-grained GPU approach for the first two BLAST stages and uses CPU multithreading for gapped extension and traceback. In CuBLASTP, ungapped extension can be performed using either a diagonal-based, hit-based, or window-based algorithm with window-based extension providing the greatest speedup improvement.

Thread divergence is first overcome by reorganizing memory access of hits for seed extension using binning along diagonals as seen in Figure 7. These bins are then sorted and hits that are farther apart than the given threshold are removed as depicted in Figure 8. The resulting hits in each bin are then in ascending order along diagonals and can then undergo ungapped extension.

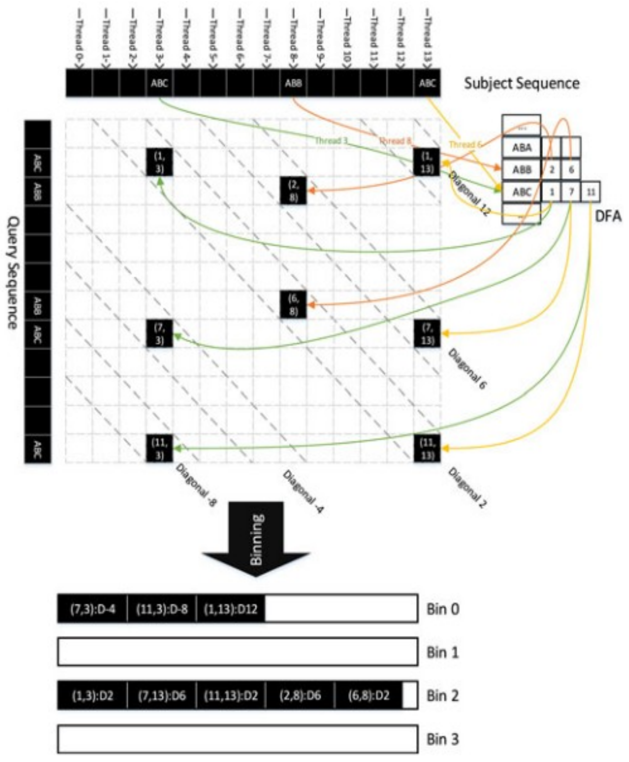


Figure 7: Binning of hits by multithreading on each subject sequence [10].

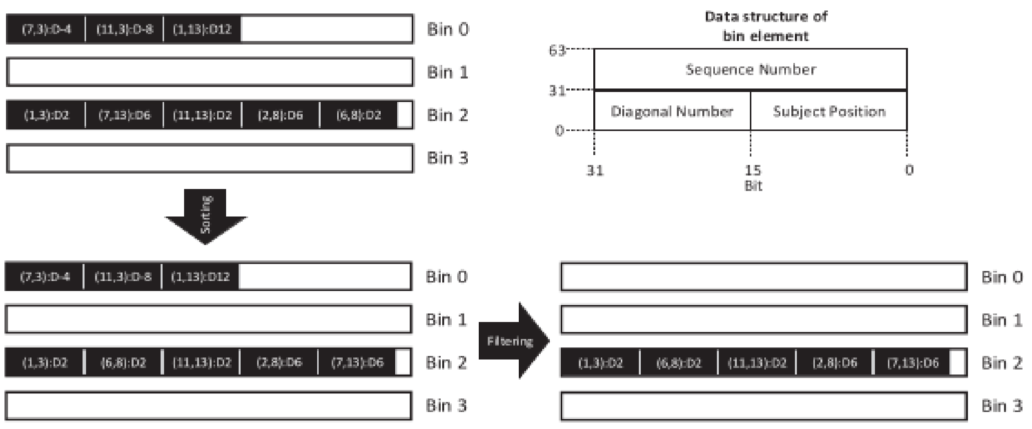


Figure 8: Sorting and filtering of hits prior to ungapped extension [10].

Using the reorganized hits, a window-based ungapped extension is performed by dividing a warp into different windows and mapping each window to a diagonal of hits. Figure 9 illustrates this using two windows of eight threads each extending from the hit IYP. In this case, each thread can calculate the letter-pair extension scores simultaneously. Then the highest score is referenced to determine which pair results in a dropoff score larger than the given threshold (10 in this example) while an index flag is used for the position of the end of ungapped extension and where gapped extension and traceback can proceed on CPU threads.

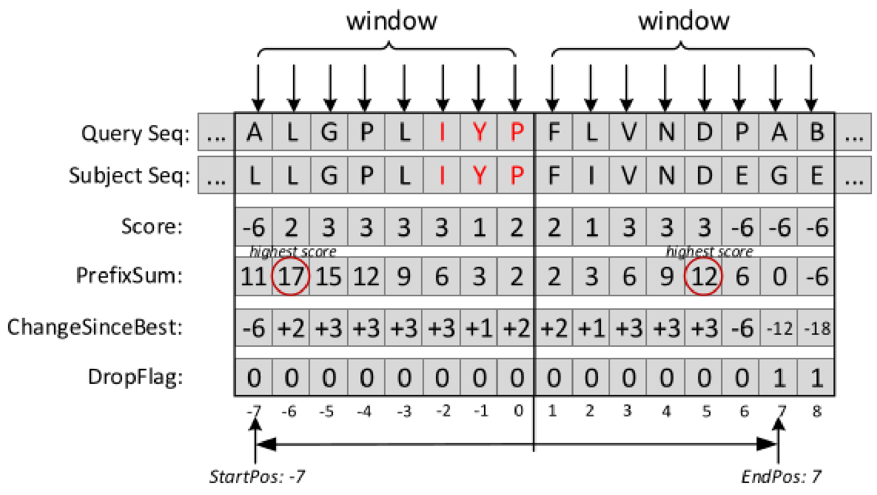

window
window
Query Seq: ... A L G P L I Y P F L V N D P A B ...
Subject Seq: ... L L G P L I Y P F I V N D E G E ...
Score: -6 2 3 3 3 3 1 2 2 1 3 3 3 -6 -6 -6
highest score
highest score
PrefixSum: 11 17 15 12 9 6 3 2 2 3 6 9 12 6 0 -6
ChangeSinceBest: -6 +2 +3 +3 +3 +3 +1 +2 +2 +1 +3 +3 +3 -6 -12 -18
DropFlag: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8
StartPos: -7
EndPos: 7

Figure 9: Fine-grained window-based ungapped extension in CuBLASTP [10].

2.2.5. H-BLAST

As with the previous parallel approaches, H-BLAST focuses its efforts on seeding and ungapped extension and uses code from NCBI-GPU-BLAST with a fine-grained approach. This method employs a batch queuing of execution on groups of subject sequences. A flexible GPU-CPU workload balancer is also a unique feature of H-BLAST to optimize BLAST searches depending on hardware and database sequences.

3 Problem Statement

All methods presented here have become practically obsolete due to the continuous and fast-paced updates in GPU hardware and software, namely NVIDIA architectures and CUDA developments. While BLAST has remained a useful tool for nearly two decades, programs using GPUs become quickly outdated and unusable without legacy hardware and driver installations. This limitation of code portability for GPU applications such as these parallel BLAST methods prevents direct analysis of their reported speedups. In this direction, using different GPU cards and CUDA versions also presents a flaw when comparing such implementations. As such, these methods do not account for differences in hardware as well as speedup testing parameters when reporting results for comparing to other methods. Thus, this paper attempts to present this issue and address this by providing a crude estimation of speedups for each method relative to how they were implemented and projects their performance if re-programmed for modern GPU systems.

4 Speedup Analysis

4.1. Hardware Comparisons

A major factor in determining speedup for parallel execution is the GPU hardware used. For example, a larger number of cores in a GPU will enable more threads to execute simultaneously which in practice would allow more subject protein sequences to be scanned at a time. Additionally, the amount of global memory on the GPU can determine the size of a protein database that can be loaded onto the GPU for running BLAST. This is one limitation to parallel BLAST algorithms as databases have become much larger than GPU device memories. This is handled by loading subject sequences in batches. Additionally, the clock rate and subsequent bandwidth of GPU processors contributes to the overall speed of computations and memory access. Finally, local and cache memory can be taken advantage of for parallel BLAST despite their small sizes as seen in the previous section.

The GPU specifications for each parallel method are shown in Table 3 below. It can be seen that there are many hardware differences between implementations which complicate the interpretation of their reported speedups. Most significantly, the GPU hardware plays a critical role for each method in parallelizing hit detection and ungapped extension for BLAST. Therefore, these stages of BLAST are greatly affected by the GPUs used.

Likewise, the CPU specifications are shown in Table 4 below and also contribute to the parallelized speedup of BLAST. Especially for the implementations that rely on concurrent GPU-CPU execution and for the gapped extension stage, the CPU performance and connection with the GPU will affect these runtimes. Thus, the hardware used for these implementations can make it difficult to decipher which method performs best in theory and how much the BLAST speedup can be attributed to the computer systems they use.

Implementation	GPU	Cores	Clock Rate (effective) [GHz]	Bandwidth [Gb/s]	Shared/ Constant Memory [KB]	L1/L2 Cache [KB]	Global Memory [GB]
NCBI-GPU-BLAST	Fermi C2050	448	1.15 (3.0)	144	64/64	64/768	3.072
CUDA-BLASTP	Tesla GeForce GTX 295	480	1.242 (1.998)	111.89 x2		/448	1.792
	Tesla GeForce GTX 280	240	1.296 (2.214)	141.7		/256	1.024
GPU-BLASTP	Tesla C1060	240	1.3 (1.6)	102.4		/256	4
	Fermi C2050	448	1.15 (3.0)	144	64/64	64/768	3.072
CuBLASTP	Kepler K20c	2496	1.3 (5.2)	208		16/1280	5.12
H-BLAST	Kepler K20x	2688	1.3 (5.2)	249.6		16/1536	6.144
	Kepler K20m	2496	1.3 (5.2)	208		16/1280	5.12

Table 3: GPU Specifications for Implementations. *Note: Specifications listed were retrieved from respective published papers, [15], or [16].*

Implementation	CPU	Cores	Clock Rate [GHz]	Memory [GB]
NCBI-GPU-BLAST	Intel Xeon	6	2.67	12
CUDA-BLASTP	Intel Quad-Core i7-920	4	2.66	12
	Intel Quad-Core i7-921	4	2.66	12
GPU-BLASTP	Intel Core 2 Duo	2	2.2	4
	Intel Core 2 Duo	3	3.2	5
CuBLASTP	Intel Core i5-2400	4		8
H-BLAST	Intel Xeon E5-2670 x2	8	2.6 (boosted 3.3 on single core)	
	Intel Xeon E5-2670 x3	9	3.6 (boosted 3.3 on single core)	

Table 4: CPU Specifications for Implementations.

4.2. Testing Parameters Comparisons

Another variable that can affect the reported speedups between GPU methods is the framework that tests them. Some of these parameters can be seen in Table 1. Although each method considers speedup as a factor of query protein length, they often do not use the same query sizes which prevents a one-to-one comparison with other methods. To add, subject sequences and databases differ between implemented methods which compounds the direct comparisons. Besides the data used, other variables are inconsistent between methods when testing them. For example, the number of CPU threads for running BLAST sequentially or in parallel differs. From reported results, this change does not correlate linearly with speedup causing even more difficulty in analyzing methods independently. As another example, not all methods report speedup performance for each BLAST stage explicitly or may compare runtimes with FSA-BLAST or other GPU methods. Finally, speedup results are not always presented numerically, but are instead only shown graphically and may show as an average or without an average speedup for query lengths.

4.3. Estimating Relative Speedups

Mentioned in the problem statement above, lacking the specific computing resources prevents these GPU programs from being able to run locally. Thus, directly implementing each of them independently is impractical. Also, GPU-BLASTP has no available source code to be able to run regardless of resource availability. Therefore, a rough estimation for comparing each method is necessary.

However, as explained in sections 4.1 and 4.2, determining how to compare these parallel schemes is extremely complicated. Consequently, this paper's estimations can be considered naive and crude irrespective of the inherent difficulty in consolidating information from each method. On that premise, Table 5 below presents the speedup comparisons reported by each GPU-based implementation not accounting for hardware differences. Note that some values have been averaged over query length differences, but are generally recorded for lengths of 500 to 1000 amino acids. Even without a direct relation to GPU-CPU specifications, it is still a challenge to compare speedups provided with the information in Table 5. Although, one can speculate that greater speedups shown by H-BLAST may correlate with better hardware presented in Table 3 as can be seen with the difference between GPU-BLASTP using a Fermi GPU versus a Tesla GPU. Otherwise, there is no obvious distinction as to which method works best for parallelizing BLAST.

BLAST Stages	**CPU Threads**	**NCBI-GPU-BLAST**	**CUDA-BLASTP**	**GPU-BLASTP on Tesla**	**GPU-BLASTP on Fermi**	**CuBLASTP**	**H-BLAST**
Hit Detection and Ungapped Extension	1	3.2x	3.18x				8x
	2	2x		4x	5x		
	4	1.5x				3.1x	10x
Gapped Extension	1	2.5x	3.18x				3x
	2	2x		1x	2.5x		
	4	1.5x					7.5x
Overall	1		10.1x				
	2			4.5x	5.5x		
	4		5x			3.4x	

Table 5: Estimated Implementation Speedups Relative to Sequential BLASTP.

4.4. Projected Parallel Performance

Despite the aformentioned challenges, determining the potential of GPU methods on modern computing systems can provide some insight to current and future considerations for parallelizing BLAST. One example of the most current NVIDIA and CUDA releases as of the time of this paper is the GeForece RTX 3090 based on the Ampere architecture and CUDA 11.2 [17]. This is significantly more recent in architecture platform and CUDA version than the GPUs used in the presented parallel methods with greater specifications shown in Table 6.

GPU	**Cores**	**Clock Rate (effective) [GHz]**	**Bandwidth [Gb/s]**	**L1/L2 Cache [KB]**	**Global Memory [GB]**
Ampere GeForce RTX 3090	10496	1.219 (19.5)	936.2	128/6000	24

Table 6: Specifications for the NVIDIA GeForce RTX 3090 GPU.

Considering most methods benefit from GPUs for accelerating hit detection and ungapped extension, this projection will assume that the RTX 3090 would apply to these stages and subsequently the overall speedup of BLAST. Noticably, modern GPU hardware has about 4.2 times as many cores and 3.75 times faster effective clock rate and bandwidth than the best GPU specifications used in any implemented GPU-based BLAST method. This would enhance parallel execution significantly. Moreover, device memory is also larger by 3.9 times and cache memory is increased two-fold for L1 and 3.9 times for L2 cache. These improvements would also enable faster GPU execution by increasing subject sequence batch sizes and memory access speed for BLAST data structures. Thus, if there was a linear correlation to each parallel scheme, these methods would benefit from newer GPU systems with an expected 4-fold improvement from their reported speedups.

Having newer features enabled with the latest CUDA revision, fine-grained paralellization of BLAST can be implemented on a GPU with greater speedup compared to previous methods simply due to recent features. For example, L2 cache memory can be set aside for persistent data access to reduce latency from global memory access. While none of the methods strategize using L2 cache, this is just one example to show that there is more opportunity for programming enhancements and potential algorithms using newer hardware.

As mentioned in the introduction, protein databases are growing, so despite the hardware improvements that can be delivered using modern GPUs there are limitations to the potenial modern speedups. For example, CuBLASTP performed BLAST on the Swisssprot database which had around 300,000 sequences, however, there are now 474,714 database sequences. This 50% increase would reduce the time for BLAST searches by at least half. Therefore, naively considering a linear relationship between hardware specification and database size for a current GPU-based BLAST method, one may expect to see an overall speedup improvement of twice the reported speedups of each method.

Figure 10 below shows execution time of sequential BLAST for protein sequence lengths ranging from 40 up to 35213. So realistically, speedups can continue to be expected to behave non-linearly dependent on query length and database with all other variables considered. Finally, Figure 11 shows a rough estimation of potential speedups if each method were translated to modern systems based on details mentioned above and assuming linear improvements.

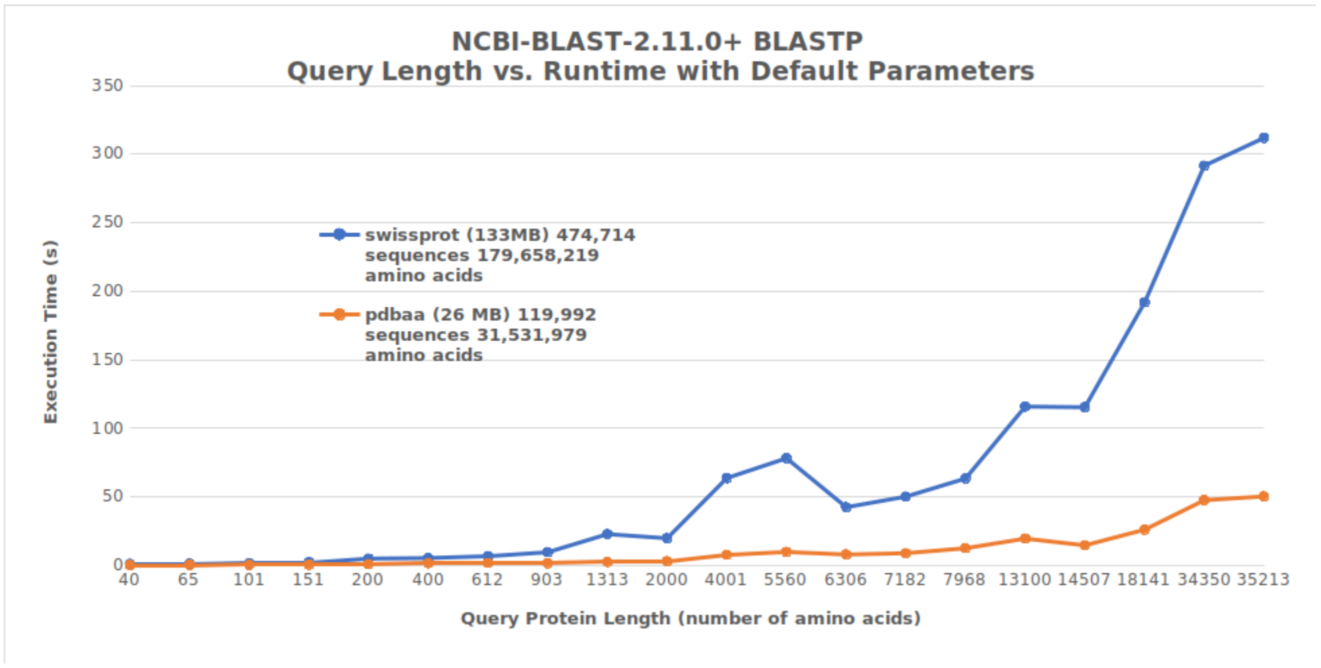


Figure 10: Serial execution times of BLAST over a large range of query lengths.

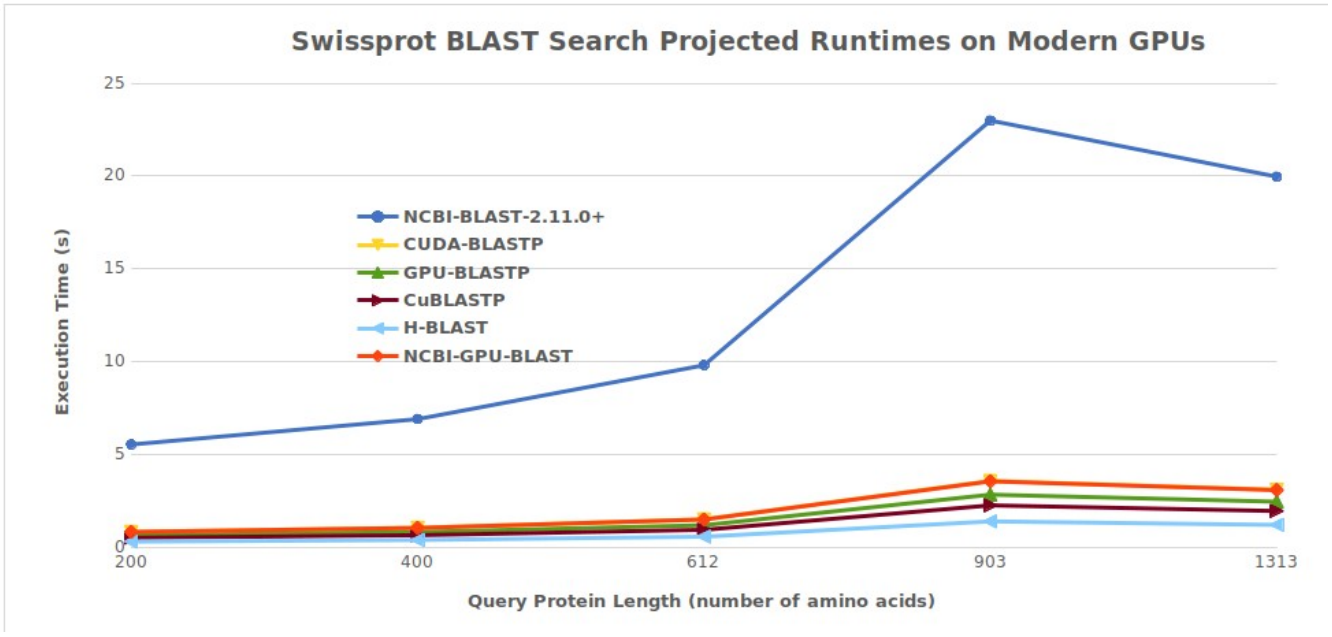


Figure 11: Projected execution times based on modern GPU enhancements.

5 Conclusion

The performance of GPU enabled computing systems has improved significantly in recent years. Performing these GPU-based BLAST methods using today's systems would continue to reduce runtimes for protein sequence alignment by at least two-fold despite increases in protein database sizes. Newer GPU features could also enable further speedup improvement strategies. These improvements would likely benefit hit detection and ungapped extension stages of BLAST the most considering that these are the most computationally intensive and parallelizable stages of BLAST. Coupling GPUs with

CPUs to run in a staggered and overlapped scheme has also been shown to be crucial for minimizing execution times.

The publications of GPU-based methods for BLAST protein alignment proves difficult to compare. Without running GPU-based algorithms directly, one can only speculate on relative speedups on current data under the same computing conditions. This drawback to the longevity of these NVIDIA and CUDA implementations provides an opportunity for a newer GPU BLAST algorithm to be designed with a conscious effort to build a standard testing framework for evenly comparing any future implementations.

References

[1] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990 Oct 5;215(3):403-10. doi: 10.1016/S0022-2836(05)80360-2. PMID: 2231712.

[2] Altschul SF, Madden TL, Schäffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. 1997 Sep 1;25(17):3389-402. doi: 10.1093/nar/25.17.3389. PMID: 9254694; PMCID: PMC146917.

[3] Xiong, J. 2006. SEQUENCE ALIGNMENT. In Essential Bioinformatics. Pages 29-94. Cambridge: Cambridge University Press.

[4] The UniProt Consortium, UniProt: a worldwide hub of protein knowledge, Nucleic Acids Research, Volume 47, Issue D1, 08 January 2019, Pages D506–D515, https://doi.org/10.1093/nar/gky1049

[5] Ling, Cheng, and Khaled Benkrid. Design and Implementation of a CUDA-Compatible GPU-Based Core for Gapped BLAST Algorithm. Procedia computer science 1.1. Pages 495–504. 2010. https://doi.org/10.1016/j.procs.2010.04.053

[6] S. Xiao, H. Lin and W. Feng. Accelerating Protein Sequence Search in a Heterogeneous Computing System. 2011 IEEE International Parallel & Distributed Processing Symposium, Anchorage, AK. Pages 1212-1222. 2011. doi: 10.1109/IPDPS.2011.115.

[7] M. Said, M. Safar, M. Taher and A. Wahba. Accelerating iterative protein sequence alignment on a heterogeneous GPU-CPU platform. 2016 International Conference on High Performance Computing & Simulation (HPCS), Innsbruck. Pages 403-410. 2016. doi: 10.1109/HPCSim.2016.7568363.

[8] Liu W, Schmidt B, Müller-Wittig W. CUDA-BLASTP: accelerating BLASTP on CUDA-enabled graphics hardware. IEEE/ACM Trans Comput Biol Bioinform. 2011 Nov-Dec;8(6):1678-84. doi: 10.1109/TCBB.2011.33. PMID: 21339531.

[9] Vouzis, P. D., & Sahinidis, N. V. GPU-BLAST: using graphics processors to accelerate protein sequence alignment. Bioinformatics. Oxford, England. 27(2), 182–188. 2011. https://doi.org/10.1093/bioinformatics/btq644

[10] Zhang J, Wang H, Feng WC. cuBLASTP: Fine-Grained Parallelization of Protein Sequence Search on CPU+GPU. IEEE/ACM Trans Comput Biol Bioinform. 2017 Jul-Aug;14(4):830-843. doi: 10.1109/TCBB.2015.2489662. Epub 2015 Oct 12. PMID: 26469393.

[11] Glasco, D. An Analysis of BLASTP Implementation on NVIDIA GPUs. 2012. [Online] Available: https://www.semanticscholar.org/paper/An-Analysis-of-BLASTP-Implementation-on-NVIDIA-GPUs-Glasco/42bb3ca76542c08566547de2d828b2e3e61af4f3

[12] Rani, S., Gupta, O.P. CLUS_GPU-BLASTP: accelerated protein sequence alignment using GPU-enabled cluster. J Supercomput. 73, 4580–4595. 2017. https://doi.org/10.1007/s11227-017-2036-4

[13] Weicai Ye, Ying Chen, Yongdong Zhang, Yuesheng Xu, H-BLAST: a fast protein sequence alignment toolkit on heterogeneous computers with GPUs, Bioinformatics, Volume 33, Issue 8, 15 April 2017, Pages 1130–1138, https://doi.org/10.1093/bioinformatics/btw769

[14] M. Cameron, H.E. Williams, and A. Cannane. Improved Gapped Alignment in BLAST. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 1(3), 116-129, 2004.

[15] GPUZoo. List of NVIDIA video cards. Web. Retrieved from https://www.gpuzoo.com/GPU-NVIDIA/index.html

[16] TechPowerUp. GPU Specs Database. Web. Retrieved from https://www.techpowerup.com/gpu-specs/?mfgr=NVIDIA&sort=name

[17] NVIDIA. GeForce. Products, Graphics Cards. Retrieved from https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090/