Comparative Analysis of Protein Alignment Algorithms in Parallel environment using CUDA

BLAST versus Smith-Waterman

Shadman Fahim  
Department of Computer Science and Engineering  
BRAC University  
Dhaka, Bangladesh  
shadmanbracu09@gmail.com

Shehabul Hossain  
Department of Computer Science and Engineering  
BRAC University  
Dhaka, Bangladesh  
rudrozzal@gmail.com

Gulshan Jubaed Prince  
Department of Computer Science and Engineering  
BRAC University  
Dhaka, Bangladesh  
jubaedprince@hotmail.com

Jia Uddin  
Department of Computer Science and Engineering  
BRAC University  
Dhaka, Bangladesh  
jia.uddin@bracu.ac.bd

Abstract—In bioinformatics to identify evolutionary relationships two sequences are matched to find similarities. Smith Waterman, a dynamic algorithm, is a common choice to carry out this alignment process. However, with the exponential growth of protein databases this algorithm’s time complexity increases. The demand for bioinformatics for their tasks to speed up is very high. Even a small seed up in computation would be very helpful in the field of bioinformatics. Thus, for a lot of the scientists this algorithm might not be the first choice. In today’s world the most popular and used bioinformatics tool is the BLAST (Basic Local Alignment Tool). BLAST, similar to Smith Waterman algorithm, is an alignment algorithm for scanning proteins from protein databases. This paper analyzes both the algorithms in a parallel environment with the help of NVIDIA GPU. For our experiments we utilized a GeForce GTX 660 NVIDIA GPU and env_nr dataset. Experimental results show that parallel implementation of BLAST algorithm mostly range 2-5 times faster than parallel Smith-Waterman.

Keywords—CUDA, GPU, BLAST, HSP, HSA, NCBI.

I. INTRODUCTION

Basic Local Alignment Search Tool (BLAST) is the most popular alignment algorithm in the world of science today. The algorithm uses dynamic programming which utilizes well defined mutation scores. This method is more than an order of magnitude faster than the existing heuristic algorithms [1]. It has been cited over 25,000 [2] and over 21,000 [3]. For such popularity US National Center for Biotechnology Information (NCBI) plays the most important role. NCBI provided a platform over the internet for everyone’s reach. Now any general person can go on the website of NCBI and get results for their queries. It is noted that hundreds and thousands of queries are being processed every now and then using the platform provided by NCBI. This increase BLAST’s usage by 2 to 3 times [4].

On the contrary, Smith Waterman is also a dynamic algorithm but isn’t used as much as BLAST. This algorithm generates more accurate results than what BLAST produces. However, its accuracy is maintained at the expense of computation time and computer power [5]. Computation speed is the burning topic in today’s world. How fast a task can be processed is the main challenge. One of such other challenges is searching through long detailed databases. With the exponential growth of protein databases demand to accelerate searching through such huge databases is very high. NCBI is having tremendous breakthroughs in this particular field. However, NCBI uses sequential search for the queries. With the availability of Graphics Processing Units (GPUs) it can be assumed that using its parallel techniques BLAST algorithm can have a faster processing time. An implementation of BLASTP algorithm is handled by GPU using Compute Unified Device Architecture (CUDA), CUDA-BLASTP. It is claimed that in CUDA architecture they have managed to achieve speedups of 10 times compared to sequential NCBI BLAST 2.2.22 on a GeForce GTX 295. It is also 3-4 times faster than multithreaded NCBI BLAST on an Intel Quad-Core processor [4]. Similarly, mpiBLAST is an open-source sequence tool that parallelizes the NCBI BLAST toolkit. It uses the database segmentation approach and the master-worker style. It achieves significant speedups in small or moderate number of processes [6]. On the other hand researchers started to implement Smith-Waterman in GPUs as well. CUDASW++ 2.0 has managed to achieve an average performance of 9.509 GCUPS on single-GPU version and an average performance of 14.484 GCUPS on dual-GPU version [5].
Hence there are scopes to speed-up the process and meet the demand of accelerating it. This paper demonstrates both the alignment algorithms. Also, it illustrates how this task is handled by a GPU using CUDA. CUDA by NVIDIA, a parallel computing architecture uses parallel compute engine in NVIDIA GPUs to solve many computationally intensive problems in a more efficient way than on Central Processing Unit (CPU) [7]. Using its parallel techniques we demonstrate how computation can speed-up. The database used in this research is taken from the NCBI website ftp://ftp.ncbi.nlm.nih.gov/blast/db/FASTA/env_nr.gz.

The remainder of the paper is organized as follows - Section II displays the architecture of a NVIDIA GPU, features of CUDA, Smith-Waterman algorithm and BLAST algorithm. Section III provides a detailed explanation of the implementation of the algorithms. The results of the experiments carried out and their analysis are included in Section IV. Finally Section V concludes the paper.

II. BACKGROUND STUDY

A. GPU Architecture and CUDA

GPU is considered to be efficient and have a better performance. Comparing to a CPU, a GPU provides a better performance because it offers a higher peak GFLOPS (Giga floating-point operations per second) [8]. The GPU that we used for the experimentations is GeForce GTX 660. Generally a GPU device has several multiprocessors with several processors inside each of them. Figure 1 enlightens it. There are mainly two types of memory in GPU. One is on-chip memory and the other is off-chip memory. The on-chip memory has low access latency but a relatively small size. On the other hand the off-chip memory has larger size and also higher access latency [9]. Moreover, these microprocessors contain the shared memory and caches, along with registers.

CUDA introduced by NVIDIA is a general purpose parallel computing platform and programming model [10]. The CUDA functions are called kernels. Unlike C functions that run only once this kernel runs N times in parallel by N different CUDA threads. Each thread that executes the kernel is given a unique thread ID. The threads are organized in a hierarchy consisting of blocks and grids. When calling a kernel function the size of the blocks and the number of threads per block are specified. An example of the function to call kernels is presented as:

```
kernel<<<numBlocks, numThreads>>>(parameter’s list).
```

B. BLAST Algorithm

Before BLAST, FASTA was developed by David J. Lipman and William R. Pearson in 1985 [11]. Besides fast algorithms like BLAST and FASTA, Smith-Waterman algorithm was used to search protein databases which guarantee the optimal alignments of the query and database sequences unlike BLAST and FASTA. However, the heuristic approach of BLAST algorithm is overall a lot faster. So, due to such highly populated protein databases Smith-Waterman search is both time consuming and computer power intensive.

So as mentioned earlier BLAST is the most popular heuristic search algorithm for protein scanning. Unlike Smith-Waterman algorithm where the entire sequence is compared BLAST locate high scoring short matches between the query sequence and the subject sequence [9]. Due to this the accuracy of BLAST decreases to some extent but then the processing speed increases exceptionally than Smith-Waterman. The Blast algorithm mainly has four stages [12]. The first stage is the hit detection where the query sequence is matched with the subject sequence in order to find matches. The query sequence is broken down to user defined size word lengths (W). Then the words are compared with subject sequences to detect similarities. The hits are then scored using the blossom62 scoring matrix. A sample snap shot of blossom62 is presented in Figure 2.

```
| A | R | N | D | C | Q | E | G | H | I | L | K | M | F | P | S | T | W | Y | V |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
```

Figure 2. The scoring matrix, Blossum 62.

The hits are not necessary to be exact matches, similar matches is also accepted as long as the score of that hit is greater than a certain threshold (T). These are then saved in an efficient data structure such as a lookup table.

In stage 2, the remaining hits are now sent to this stage for further processing. The hits are extended in both directions.
and as long as the accumulated score is increasing the extension is carried on. As soon as the score starts to decrease we stop. This is called un-gapped extension. The result is HSPs (highest scoring pairs). A sample of un-gapped extension is presented in Figure 3.

![Figure 3. Un-gapped extension.](image)

Consequently in stage 3, the HSPs sent to this stage are then extended further. However, unlike the last stage here gaps are allowed. With each gap there is a penalty. This is called the gapped extension. Finally in stage 4, scores all the alignments again from the previous stage. Once done scoring it produces the top scores. This is called the gapped alignment with trace-back. The entire BLASTP algorithm is enlightened in Figure 4.

![Figure 4. An overview of BLAST algorithm.](image)

In Figure 4, the query sequence and the subject sequence is compared. We can see there is an exact match in the sequence AAL. This is the stage one where the hit is detected. In the next stage the un-gapped extension is performed on pairs of high-scoring segment pairs (HSPs). Here, the initial match in extended in both directions until there the overall accumulated score starts to decrease. In stage 3, the un-gapped extension is extended using a gapped alignment [13]. To determine the level of the alignment a scoring matrix and a threshold value is used. Finally, in the last stage a trace-back algorithm is used to produce and score the alignments.

The time complexity of this particular algorithm is [14]:

\[ O(M) + O(MN) + O(1) = O(MN). \]

Where, M is the number of look ups in the hash table to finds seeds and N is the length of the query sequence. Also, because calculating the statistical significance of HSP is a constant time operation, these have a complexity of O(1).

C. Smith Waterman Algorithm

In 1981 before BLAST or FASTA were written Smith and Waterman suggested Smith-Waterman algorithm [15]. Later in 1982 Gotoh improved the algorithm [16]. This is a local alignment algorithm. Thus it matches the highest similarities between two proteins instead of the aligning the entire two proteins. Assuming two query sequences \( S_1 \) and \( S_2 \) having lengths m and n. The two sequences are arranged in a matrix form with \( m+1 \) rows and \( n+1 \) columns. Initially the first row and column are set to 0. Then the similarity matrix is computed for \( 1 \leq i \leq m, 1 \leq j \leq n \) using the formula as shown below. At last the trace back is performed to calculate the final overall score.

There are mainly three steps to run this algorithm, they are:

1. Initialization.

\[ M(0,j) = 0; \]
\[ M(i,0) = 0; \]
where M is the similarity score matrix.

2. Filling the matrix, M.

\[ M(i,j) = \max \left\{ \begin{array}{ll}
0, & 1 < i \leq m, 1 < j \leq n \\
M(i-1,j-1) + a, & M(i-1,j) - d, \\
M(i,j-1) - d & \end{array} \right. \]

Where,
\( a = \) match/mismatch value \\
\( d = \) gap penalty \\
\( m = \) length of a sequence \\
\( n = \) length of another sequence

3. Trace back the sequences for a suitable alignment.

\[ F = \max \{M(i,j)\}; \]
\[ \text{traceback}(F); \]

The time complexity of this algorithm is [14]:

\[ O(M+N) + O(MN) + O(MN) = O(MN). \]

Here M and N are the length of the sequences. O(M+N) is for initialization. For filling the matrix the time complexity is O(MN) whereas for trace back it is O(MN).

III. EXPERIMENTAL SETUP

For our experiments CUDA Toolkit 7.5 is used and NVIDIA GeForce GTX 660. All the experiments are conducted in a personal computer (PC) with the configuration Intel(R) Core i7-4470 CPU @3.4 GHz, 16GB RAM, running Ubuntu 14.04.
A. BLAST CUDA

Figure 5 demonstrates a detailed implementation of BLASTP algorithm. It brings light to what part of the code is sent to GPU for execution. Stages 1 and 2 of BLAST algorithm are processed in GPU namely hit detection stage and ungapped extension. First and foremost the CPU takes the query sequences. Then it sorts the database according to the number of subject sequences it contains. This helps in balancing the load among the threads. So, no threads in the same wrap (cluster of threads that can execute in parallel) work on subject sequences with large length difference. Later, the database is sent to kernel for calculating the HSP pairs. Once done, High Scoring Alignments (HSAs) are computed out in the CPU using gapped extension. At last final calculations are made and the results identical to NCBI-BLAST are displayed.

- **Input**

  The three main inputs of BLAST are the query sequence, database where the subject sequences are stored, and threshold value at which an alignment must score to avoid being cut off.

- **Kernel Call**

  Stage 1 and stage 2 are mainly performed in this kernel. At first the query sequence inputted is broken down to several words of length 3. The protein database is stored in the global memory of the GPU. Kernel is then called and the 3 letter words are sent to corresponding threads with a batch of the database. There each word is matched with the subject sequence. Whenever a match is confirmed un-gapped extension is carried out. This generates HSPs. Finally, the HSPs are read back to the host (CPU) for further processing. An overview of the first kernel is enclosed in red dotted box in figure 5.

  - **CPU Readback**

    The HSPs are read backed to the CPU. Here the later part which is the gapped extension is processed. The results of gapped extension are HSAs. HSAs are filtered if they fail to overcome the threshold value. Finally, after trace backing the final results are outputted on the display.

B. Smith-Waterman CUDA

This algorithm is very time consuming as matrices are generated against every subject sequence from the database. Thus running this algorithm in GPUs is preferable as GPUs are designed to compute matrices. The implementation of smith-waterman algorithm is illustrated in Figure 6.

The operations in dotted blocks are carried out by GPU and the rest of the blocks are performed by CPU. Similar to the BLAST CUDA implementation the database is sorted so no threads on the same cluster work on subject sequences with large length difference. Each thread in the GPU is assigned to fill in the similarity matrix against one sequence from the database. Once done matching the similarity between the sequences the matrix is saved in the local memory. Then the third step of the algorithm is performed which is trace back. First the thread figures out the maximum value in the matrix

![Figure 5. The experimental setup of BLAST algorithm.](image)

![Figure 6. The experimental setup of Smith-Waterman Algorithm.](image)
and starts tracing back till it reaches zero. At last along the line of the trace back the alignment found is scored. Then the alignments with scores are read back to CPU where it organizes the results and displays it.

IV. EXPERIMENTAL RESULTS ANALYSIS

The env_nr database we used is of 1.5 GB. Our input query sequences are of yeast and that too retrieved from the NCBI website. The database has a total of 6,891,928 sequences; 1,364,236,057 letters. We varied the query length sequence from 26 to 1002.

At first we carried out a test to figure out the optimal block and thread size to carry out our experiments. Firstly, we kept block size constant and varied the thread size. Then we changed the block size and completed the task again.

![Figure 7. Comparison of execution time varying block and thread sizes.](image)

To find the optimal thread and block sizes we used the Ubp6p protein sequence which is of length 499. The graph generated is shown in Figure 7. For the last value which is marked as X, the GPU we used runs out of global memory to finish the task. GeForce GTX 660 has a memory space of 2 GB. Thus a GPU with a higher global memory will give the result. Similarly when we tried the same job with a larger sequence of length 1002 any thread size greater than 256 shows the unavailability of memory space. Finally, we conclude to complete the experiments with keeping the block size constant at 256 while changing the thread size twice 128 and 256.

According to our results BLAST performs much faster than smith-waterman algorithm. When using 256 block size and 128 numbers of thread on each block BLAST is around 2-5 times faster than smith-waterman shown in figure 8. On the other hand in figure 9 when the block and thread sizes are changed to 256 and 256 respectively BLAST perform with a speed of 1.5-4.5 times faster.

![Figure 8. Comparison of runtimes of both the algorithm using Table 2.](image)

<table>
<thead>
<tr>
<th>Thread size</th>
<th>Time for block size 256</th>
<th>Time for block size 512</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>48.709</td>
<td>43.422</td>
</tr>
<tr>
<td>64</td>
<td>16.131</td>
<td>13.495</td>
</tr>
<tr>
<td>128</td>
<td>9.538</td>
<td>9.141</td>
</tr>
<tr>
<td>256</td>
<td>9.235</td>
<td>9.154</td>
</tr>
<tr>
<td>512</td>
<td>9.201</td>
<td>9.133</td>
</tr>
<tr>
<td>1024</td>
<td>5.505</td>
<td>X</td>
</tr>
</tbody>
</table>

![Table 1. Execution time of UBP6P protein in BLASTP. Thread size is varied with block size.](image)

<table>
<thead>
<tr>
<th>Query Sequence Length</th>
<th>GPU blocks</th>
<th>GPU threads</th>
<th>BLAST CUDA</th>
<th>SW CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>26 (SCY_4187)</td>
<td>256</td>
<td>128</td>
<td>2.432</td>
<td>11.615</td>
</tr>
<tr>
<td>499 (Ubp6p)</td>
<td>256</td>
<td>128</td>
<td>9.699</td>
<td>28.482</td>
</tr>
<tr>
<td>752 (Gcn20p)</td>
<td>256</td>
<td>128</td>
<td>22.370</td>
<td>36.799</td>
</tr>
<tr>
<td>1002 (SAP155)</td>
<td>256</td>
<td>128</td>
<td>24.585</td>
<td>45.749</td>
</tr>
</tbody>
</table>

![Table 2. Runtime in seconds for both the algorithms using 256 GPU blocks and 128 threads.](image)
Figure 9. Comparison of runtimes of both the algorithm using Table 3.

**TABLE 3. RUNTIME IN SECONDS FOR BOTH THE ALGORITHMS USING 256 GPU BLOCKS AND 256 THREADS.**

<table>
<thead>
<tr>
<th>Query Sequence Length</th>
<th>GPU blocks</th>
<th>GPU threads</th>
<th>BLAST CUDA</th>
<th>SW CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>26 (SCY_4187)</td>
<td>256</td>
<td>256</td>
<td>2.446</td>
<td>11.531</td>
</tr>
<tr>
<td>499 (Ubp6p)</td>
<td>256</td>
<td>256</td>
<td>9.185</td>
<td>28.420</td>
</tr>
<tr>
<td>752 (Gcn20p)</td>
<td>256</td>
<td>256</td>
<td>21.963</td>
<td>25.764</td>
</tr>
<tr>
<td>1002 (SAP155)</td>
<td>256</td>
<td>256</td>
<td>23.773</td>
<td>34.821</td>
</tr>
</tbody>
</table>

Figure 10. Comparison of runtimes achieved by CPU and GPU using Table 4.

**TABLE 4. RUNTIME OF BLASTP IN SECONDS FOR CPU AND GPU.**

<table>
<thead>
<tr>
<th>Query Sequence Length</th>
<th>BLAST 128</th>
<th>BLAST 256</th>
<th>BLAST CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>26 (SCY_4187)</td>
<td>2.432</td>
<td>2.446</td>
<td>4.189</td>
</tr>
<tr>
<td>499 (Ubp6p)</td>
<td>9.699</td>
<td>9.185</td>
<td>27.474</td>
</tr>
<tr>
<td>752 (Gcn20p)</td>
<td>22.370</td>
<td>21.963</td>
<td>45.494</td>
</tr>
<tr>
<td>1002 (SAP155)</td>
<td>24.585</td>
<td>23.773</td>
<td>54.652</td>
</tr>
</tbody>
</table>

Since BLAST is the most used algorithm we did more experimenting with it. We ran the entire algorithm in CPU and compared the results with that obtained using GPU. The results are portrayed in figure 10. BLAST 128 means block size 256 and thread size 128. While BLAST 256 means block size 256 and thread size 256.

V. CONCLUSION

This paper examined how much BLASTP algorithm and Smith-Waterman varies from each other in computation time using the parallel techniques of CUDA. We have collected the database of protein from NCBI and CUDA Tool kit 7.5 is used. It is 2-5 times faster than the BLAST. Smith-Waterman being a very exhaustive algorithm while performing in CPU is being handled pretty well in GPU. It still gives good results compared to BLAST CUDA. However, falls short in execution time. On the bright side, smith-waterman gives more accurate results than BLAST. Thus, while choosing which algorithm for their alignment task one has to decide based on accuracy or execution time. One of the limitations of our research is that we failed to record the time of Smith Waterman in CPU. The CPU we used were not powerful enough to process matrices of all the sequences in the database. Nevertheless, we hope our results will motivate others to work on GPUs because in today’s world being fast is important.

REFERENCES


