International Conference on Computational Science, ICCS 2010

Design and Implementation of a CUDA-Compatible GPU-based Core for Gapped BLAST Algorithm

Cheng Ling\textsuperscript{a}, Khaled Benkrid\textsuperscript{b}

\textsuperscript{a}Institute for Integrated Micro and Nano Systems, Joint Research Institute for Integrated Systems, The University of Edinburgh, The King’s Buildings, Mayfield Road, Edinburgh, EH9 3JL, UK
\textsuperscript{b}C.Ling@ed.ac.uk, K.Benkrid@ed.ac.uk

Abstract

This paper presents the first ever reported implementation of the Gapped Basic Local Alignment Search Tool (Gapped BLAST) for biological sequence alignment, with the Two-Hit method, on CUDA (compute unified device architecture)-compatible Graphic Processing Units (GPUs). The latter have recently emerged as relatively low cost and easy to program high performance platforms for general purpose computing. Our Gapped BLAST implementation on an NVIDIA GeForce 8800 GTX GPU is up to 2.7x quicker than the most optimized CPU-based implementation, namely NCBI BLAST, running on a Pentium4 3.4 GHz desktop computer with 2GB RAM.

Keywords: Gapped BLAST; Two-Hit method; CUDA-compatible GPU; Biological sequence alignment;

1. Introduction

Biological sequence alignment is a widely used and crucial operation in bioinformatics and genomic and proteomic research. The main purpose of it is to find the best possible alignment of a set of sequences with a plethora of real world applications including drug engineering, disease diagnosis, and evolutionary biology [1]. However, biological sequence alignment is also a computationally expensive application as its computing and memory requirements grow quadratically with the growth of biological sequence databases, in the case of pairwise sequence alignment, which often leads to long execution times on normal desktop computers which are inherently sequential in nature, as can be seen from the execution of SSEARCH program on desktop computers [2]. Given that biological sequence databases are growing exponentially year after year, there is a need for an equally fast hardware acceleration platform for biological sequence alignment [3].

Graphics Processing Units (GPUs) have been proposed recently as high performance and relatively low cost acceleration platforms among most commercial hardware for biological sequence alignment [4]. Early attempts to exploit the benefits of GPUs in biological sequence alignment consisted in the implementation of the Smith-Waterman algorithm (SW) algorithm [5] and have reported considerable speed-up compared to equivalent CPU-based implementations [6][7][8][9]. In [9], for instance, we have reported a CUDA-compatible [10] GPU implementation of the Smith-Waterman algorithm which was up to 17x quicker than the SSEARCH program...
running on a desktop computer. Since modern GPUs are increasingly powerful, inexpensive, and relatively easy to program by software programmers through standard APIs, an altogether new discipline looking at the exploitation of GPUs in general purpose computing [11][12][13] has emerged (the so-called GP-GPU).

BLAST [14] is a very popular biological pairwise sequence alignment algorithm, the main idea of which is to search heuristically for high-scoring segment pairs (HSPs) in the alignment matrix involved between a pair of sequences (usually a query sequence and a subject sequence from a biological database) before a local extension around these HSPs takes place. Since BLAST takes a heuristic approach to sequence alignment, it achieves less accurate results compared to the gold standard Smith-Waterman algorithm, but runs much faster in return (typically more than one order of magnitude). For instance, BLAST (blastp 2.2.21+) requires several seconds for a typical search of the Swiss-Prot protein sequence database (release 56.3) [15] on a modern desktop hardware, which is typically 50x faster than SSEARCH. Because of the popularity of BLAST in the Bioinformatics community, this paper presents the detailed design and implementation of the latest version of BLAST, namely Gapped BLAST with two-hit method [16], on CUDA-compatible GPUs. The remainder of this paper is organized as follows. First, essential background on BLAST and CUDA-compatible GPUs is presented. Then, the details of our multi-threaded parallel design and implementation of the Gapped BLAST with two-hit method algorithm on GPU are presented. After that, evaluation and comparison of our proposed implementation of the Gapped BLAST algorithm with the NCBI-BLAST (a widely used implementation of BLAST from the National Center for Biotechnology Information - NCBI) follows before conclusions are laid out.

2. Background

2.1. BLAST Algorithm

BLAST is a heuristic algorithm which filters dissimilar regions in sequence pairs, as it consists of three heuristic layers of seeding, extension and evaluation. Unlike the Smith-Waterman algorithm, BLAST does not explore the entire search space between two sequences. Minimizing the search space is the key to its speed, at the cost of a loss in sensitivity however [17]. This section presents the first three stages of the Gapped BLAST with the two-hit method. The first stage consists in rapidly comparing sequences from a database with a given query sequence to identify hits with a score over a particular threshold value ($T$). The second stage performs ungapped extension between two hits in the same diagonal line to obtain high scoring pairs (HSPs). The third stage performs gapped extension. The following presents each of these stages in details.

Stage1. The first stage begins by extracting fixed length overlapping sub-sequences (words), from a subject sequence, the default length is $w=3$ for amino acid sequences and $w=11$ for nucleic acid sequences. For instance, suppose $w=3$ for the protein fraction MKFVLL. The words extracted from it are MKF, KFV, FVL and VLL respectively. In general, for a sequence of length $m$ ($m>2$), the number of words extracted is $(m-w)+1$. Afterwards, the extracted words are compared with words of the same length from the query sequence one by one using a mutation scoring matrix. A match between a pair of words is considered as high scoring if the words are identical or the comparison score is over a particular threshold value ($T$). A lookup table (see Fig.1) is created for this purpose to facilitate the matching of subject sequence words with all the possible matching query sequence words.

![Fig. 1. The structure of lookup table](image)
This lookup table contains all possible words, the number of which depends on $w$ and the number of alphabet used. Since there are 25 continuous alphabets from $A$ to $Y$ in the BLOSUM62 mutation matrix, with five unused alphabets ($B, J, O, U, X$), each amino acid can thus be coded as a five-bit binary value. For $w=3$, the address of the lookup table is from 0 to 25368 in decimal, of which the slots of address composed by unused alphabets are not used. Overlapping words extracted from the query sequence are compared with all addresses in the lookup table one by one. Three letters in each address can be restored from the address number through the following operations: $\text{address} >> 10$, $(\text{address} >> 5) & 31$, $\text{address} & 31$ respectively. Addresses with a score over the specified threshold value are labeled. After creating the lookup table, overlapping words from subject sequences are read and transformed into 15-bit binary codes, named table offsets. These are used to access the exact positions in the lookup table, checking if there exist identical or similar words and their positions in the query sequence. For example, let $DCT$ denote a word from the subject sequence, the offset value is computed as $\left\langle ('D'-'A')<<10 \mid ('C'-'A')<<5 \mid ('T'-'A') \right\rangle$. Suppose the following letter is $M$, the next word is then updated to $CTM$. To achieve this, the bits standing for $C$ and $T$ in the previous word are kept through a bit AND operation with 1023, resulting in a 10-bit binary representation of $CT$, then left shifted by 5 bits, and $\text{ORed}$ with $\left\langle ('M'-'A') \right\rangle$ to insert the $'M'$. Once a hit is found, the position in the subject sequence (X-index) and the position in the query sequence (Y-index) are passed into stage2, namely ungapped extension.

Stage2. The original BLAST algorithm extends alignments between the query sequence and subject sequence in both left and right directions for all hits. These extensions stop when the accumulated score of the HSPs fall a certain cut-off depth $X1$ below the best score obtained so far (see Fig.2. left). After the alignment terminates, it is trimmed back to the maximum score. In order to save more time, but also keep the same sensitivity as the original BLAST, a new generation of BLAST, namely Gapped BLAST, has been proposed. It takes a relatively lower threshold value ($T$) than the original BLAST to guarantee more potential hits. However, only few hits will be paired with a second hit on the same diagonal to be extended. Therefore, the frequency to trigger ungapped extension in Gapped BLAST is less than the original BLAST algorithm. When there are two non-overlapping hits $h1[d1,q1]$ and $h2[d2, q2]$ on diagonal $d$, where $d1, d2, q1, q2$ stand for the indices of the hits in the subject and query sequences respectively, within distance $A=q2-q1$ such that $2<A<40$, the inward ungapped extension will be triggered. The two hits are combined as a new region by computing the score of residues between them without deletion and insertion i.e. without gaps (see Fig.2. centre). Afterwards, the outward ungapped extension starts from the edge of the left and right hand sides of the two hits and terminates when the accumulated total score falls a certain cut-off depth below the best score obtained so far. Then, the extension is trimmed back to its state with the best score obtained. Finally, if the alignment of ungapped extension has a sufficiently high score $S$, it is then passed into stage3, namely gapped extension.

Stage3. Gapped BLAST is an advancement of BLAST with the two-hit method. It allows for the insertion of gaps in the subject or query sequence. To achieve this, the alignment of ungapped extension is extended both backwards and forwards from the centre of the ungapped alignment (see Fig.2. right). The algorithm for gapped alignment
utilized in Gapped BLAST is a modified version of the Needleman-Wunsch algorithm where the alignment is pruned when alignment scores (in both directions) fall below a certain cut-off value.

2.2. The Needleman-Wunsch Algorithm

The Needleman-Wunsch algorithm is a dynamic programming algorithm which finds the optimal global alignment between two biological sequences. The optimal global alignment obtained by the algorithm is achieved in two stages. Firstly, an alignment matrix is calculated based on a mutation correlation between the two sequence characters (e.g. protein amino acids, DNA base pairs). Then, the optimal global alignment is found by tracing from the element in the bottom right corner of the alignment matrix to the upper left corner. For instance, let $D$ denote a database sequence of length $m$:

$$D: 0123 1 \ldots d$$

and let $Q$ denote a query sequence of length $n$:

$$Q: 0123 1 \ldots q$$

The mutation scoring matrix which gives a score describing the likelihood of substitution between characters $id$ and $jq$, and finally $G_{init}$ & $G_{ext}$ denote penalties for opening a new gap and continuing an existing gap respectively. With the above, the alignment matrix of the Needleman-Wunsch algorithm is described by the following three equations:

$$H_{i,j} = \max\{E_{i,j}, F_{i,j}, H_{i-1,j-1} + W_{di,qj}\}$$ (1)  

$$E_{i,j} = \max\{H_{i,j-1} - G_{init}, E_{i,j-1} - G_{ext}\}$$ (2)  

$$F_{i,j} = \max\{H_{i-1,j} - G_{init}, F_{i-1,j} - G_{ext}\}$$ (3)  

Where, $i$ and $j$ are the alignment matrix cell indices. The values of $E_{i,j}$ and $F_{i,j}$ are defined as 0 if $i<1$ or $j<1$. The gap penalty is called linear if $G_{init} = G_{ext} = G$ (otherwise it is called affine). In the case of a linear gap model, Eqs.1 can be re-written more succinctly as:

$$H_{i,j} = \max\{H_{i-1,j} - G, H_{i,j-1} - G, H_{i-1,j-1} + W_{di,qj}\}$$ (4)  

The cost for a linear gap of length $L$ is:

$$S = -GL$$ (5)  

Where the cost of an affine gap of length $L$ is:

$$S = -G_{init} - (L-1)G_{ext}$$ (6)  

Usually, the penalty of opening a gap ($G_{init}$) is bigger than the penalty of extending a gap ($G_{ext}$).

From Eqs.1, it is clear that the score of $H_{i,j}$ depends on the values of its upper neighbour $H_{i,j-1}$, left neighbour $H_{i-1,j}$ and left-upper neighbour $H_{i-1,j-1}$. Therefore, to find the best alignment, there are three choices:

- An alignment between $d_i$ and $q_j$: In this case, the new score $H_{i,j}$ is $H_{i-1,j-1} + W$.
- An alignment between $d_i$ and a gap in the query sequence: In this case, the new score $H_{i,j}$ is the maximum between $H_{i,j-1} - G_{init}$ and $E_{i,j-1} - G_{ext}$, matrix $E$ is used to distinguish the gap type in the query sequence. If $H_{i,j-1}$ is high enough to open a gap, $E_{i,j}$ is updated to the $H_{i,j-1} - G_{init}$, which is the new score of $H_{i,j}$, otherwise, $E_{i,j}$ is updated to $E_{i,j-1} - G_{ext}$, so does $H_{i,j}$.
- An alignment between a gap in the subject sequence and $q_j$: in this case, the new score $H_{i,j}$ is the maximum between $H_{i-1,j} - G_{init}$ and $F_{i-1,j} - G_{ext}$, matrix $F$ is used to distinguish the gap type in the subject sequence. If
the $H_{i-1,j}$ is high enough to open a gap, $F_{i,j}$ is updated to the $H_{i-1,j} - G_{init}$, which is the new score of $H_{i,j}$, otherwise, $F_{i,j}$ is updated to $F_{i-1,j} - G_{ext}$, so does $H_{i,j}$.

2.3. The CUDA Programming Model

CUDA is a parallel computing architecture developed by NVIDIA Corporation, which makes the computing engines of graphics processor units accessible to general purpose software developers through a standard programming language e.g. C. Unlike the traditional way of having to rely on graphics API (OpenGL and Direct3D) interface to access GPUs, general purpose API functions are available to exploit the architecture parallelism in CUDA GPUs. When programming in CUDA, GPUs can be seen as multi-threaded parallel computing devices, operating as a coprocessor of the main CPU. Fig.3 shows the architecture of CUDA-compatible GPU used in our implementation, namely the NVIDIA GeForce 8800 GTX. The latter has 16 Stream Multiprocessors (SMs), with each SM having eight Stream Processors (SPs) used as Arithmetic Logic Units (ALUs) with 8KB constant cache, 8KB texture cache and 16KB shared memory. The shared memory can be read and written by any thread in a block assigned to a SM. In addition, each SP has its own registers (1024) and operates the same kernel code as the other SPs, but with different data sets. Access speed to shared memory is as fast as accessing SP registers as long as there are no bank conflicts [10]. In addition, a device memory offers global access to a larger (768 MB) but slower storage. Any threads in any SPs can read from or write to any locations in the device memory. Since computational results can be transferred back to CPU memory through it, device memory can be thought as a bridge which achieves communication between GPU and CPU.

In CUDA, local memory is allocated automatically if the size of variable required is bigger than the register size, and there is no keyword specified for it. It is not cached and cannot be accessed in a coalesced manner like device memory. Texture memory within each SM can be filled with data from the device memory. It acts like a cache, and so does constant memory, which means that they can speed up the fetch time of data. However, threads running in the SMs are restricted to read only access to these memories. The host CPU, on the other hand, does have write access to these memories.

![Fig. 3. Block Architecture of NVIDIA's GeForce 8800 GTX;](image-url)
3. Design and Implementation of Gapped BLAST with Two-Hit Method on GPU

Since Gapped BLAST consists of a series of independent steps, two kernel functions are designed to fit its implementation on GPU. Firstly, subject sequences are read by blocks of unique id. Threads within each block search the words from subject sequences using the lookup table as explained section 2.1 above. As hit searching is an independent process for each thread in the blocks, there is no need for synchronization. Thread iteration is applied when the length of the subject sequence is larger than the number of threads defined in the blocks. It is generally a good idea to define a small number of threads in blocks, which reduces the probability of idle state threads i.e. threads with no operation in hand, since threads in the same block run under the Single Instruction Multi Data (SIMD) model. Fig.4 shows a simplified multi-threaded architecture of the first kernel. Each three amino acids are packed up as a word, which is transformed to a 15-bit offset value that is used to access the lookup table for hit searching in the query sequence. The lookup table is copied from the host memory to the device memory of the GPU. HIT1 FINDER stands for the process of searching the first hit in two-hit method, and all processes within the block of dotted lines in Fig. 4 below are executed in parallel. Afterwards, HIT2 FINDER starts to search for another hit along the same diagonal of the first hit within a certain distance. Note here that this step is not parallel for all threads as, in general, there could be no matched hits found in the previous step or the search time for the second hits is not the same for the threads which have already obtained hit1. Therefore, this heuristic layer (seeding) influences the efficiency of GPU, with the worst situation being that only one thread enters the extension stage and the rest of threads enter the idle state in a thread batch, which leads to 7 idle Stream processors (SPs) in a Multi-stream Processor (MP). Then, UNGAPPED EXTENDER processes the sequences with two-hit condition triggered and passes the start and the end points of the high-scoring pairs to the host. The gapped extension part is done by GAPPED EXTENDER in the second kernel after the evaluation of ungapped alignments passed from the first kernel.

Fig. 4. Design Architecture for the Gapped BLAST Algorithm with Two-Hit Method (kernel 1);
3.1. High Level Application Software

Fig.5(left) shows the organization of the proposed GPU-based core for Gapped BLAST algorithm with two-hit method, which is also typical in many hardware accelerated platforms. In a host-based high level application, characters in the subject sequence database e.g. Swiss-Prot and query sequence are read into memory. In order to make threads fetch their proper data, the length of the subject sequence needs to be copied into the device memory for the kernel application to access. Another important duty of the host application is to create the lookup table, the structure and function of which were described in section 2.1 above. CUDA-compatible GPUs do not support structure pointer, despite the fact that it is feasible to dynamically allocate the required memory size for the words in a structure on the host. However, it is impossible to copy them on to GPUs and use them as such. There are two ways to solve this problem, the first is to separate the structure members into basic data types; in this case, more pointers are needed and it is tedious to manage them. Another method is to predefine a fixed large enough value for the required memory address which is very simple to manage, hence we chose this method.

3.2. Two-Hit Finder

The part in dotted line in Fig.5(right) shows the simplified design of the Two-Hit Finder process. For each thread, words extracted from the subject sequence are read into three registers defined as symbol1, symbol2 and symbol3 through three read operations. These operations can be trimmed to one read operation through changing the structure of the subject sequence. Then, the word is transformed to a 15-bit offset value, with the first symbol stored in the most significant 5-bit. The transformation can be applied by operations bit-shift (<<), and bit-or (|). The offset value is used to access the lookup table and find where identical or similar words occur. If the flag bit of lookup_table[offset] is labelled as 1, there is at least one similar word in the query sequence, and its indices are read and stored in the coordinate registers hit1_x and hit1_y. After that, the process searches the second hit along the same diagonal within a certain distance A until a hit occurs, or the distance exceeds the size of scanning window. The inward score is accumulated simultaneously. Sometimes, the first hit occurs by chance with no other hit found in the same diagonal, which is not uncommon when searching large sequences. Therefore, though a fixed search distance helps to find potential hits, it does not mean that the finally obtained HSPs are really of high score. To save more time, we define a cut-off value for the inward score (the latter one is always bigger than 11 at the start – the threshold value used in NCBI-BLAST) and the search for the second hits is terminated when the accumulated total score decreases to 5. It is possible that potential hits may be lost by this thread, but as the whole search space is separated and searched by a batch of threads, the lost hit can still be searched by other threads. If the Two-Hit condition is satisfied, outward ungapped extension is then trigged and carried in both right and left directions. This extension is terminated when the maximum accumulated score falls a cut-off depth $X1$ or the index of the hits reach the boundary of sequences. Otherwise, threads will search other hits through an offset value based on the index of the last word.

![Fig. 5. (left) Organization of the proposed method system; (right) Program flow chart of the inner kernel;](image-url)
Finally, the score of ungapped extension is obtained by adding inward and outward scores, which is passed into the host with the coordinates of the hits and sequence id number. Note here that as each pair of sequences is searched by one block, for each thread in the block, the alignment of thread with the highest score is passed into the stage of gapped extension; the alignments with lower scores are not [18].

3.3. High-Scoring Pairs Evaluation

As illustrated in Fig.4, the Two-Hit condition in Gapped BLAST may not occur in some threads. It is possible that most threads in a block return no hits and have been in idle state until the end of the first kernel. In this case, large amount of computing resources are held and they cannot enter the stage of gapped extension if both of the functions are included in one kernel. To avoid this, the first kernel sends the alignments of ungapped extension back to the host first, then those blocks with the Two-Hit condition triggered and whose score are greater than the empirically determined cut-off score $S$ are kept and pushed into the second kernel for gapped extension. However, this approach does not eliminate the idle time of GPU completely, especially for long query sequences with many hits. The expected number of HSPs with a score at least equal to $S$ is given by Eqs.7. [19]:

$$E = kmne^{-\lambda k}$$  \hspace{1cm} (7)

where $E$ is the statistical significance threshold for reporting matches against database sequences, with a default value of 10 for large sequences and 1000 for shorter sequences, and $mn$ is the search space [19] which is the product of query sequence length and subject sequence length. If a protein is compared with a whole database rather than a single sequence, $n$ is the number of residues in the database. The $\lambda$ and $k$ used in the proposed method are the typical values for ungapped local alignment using BLOSUM62, which are 0.318 and 0.13 respectively.

3.4. Gapped Extender

The second kernel, illustrated in Fig.6, implements the gapped alignment using the modified Needleman-Wunsch algorithm with a linear gap model. Each thread reads the coordinates of the centre computed by the edge points. The gapped alignment of the query sequence and related subject sequence is launched in both directions from this point sequentially. As each thread computes a complete HSP, massive threads are harnessed to compute the alignment matrices of different HSPs in parallel. For each alignment matrix, cells are computed column by column. Gapped BLAST requires some modifications to the original Needleman-Wunsch algorithm as explained in section 2.2 above. Firstly, the extension is terminated when the maximum value of the current column falls a certain cut-off depth $X2$ below the highest value obtained so far. Second, the trace-back procedure starts from the cell with the highest value, rather than bottom rightmost cell.

Fig. 6. Design architecture for gapped extension;
4. Results and Evaluation

The proposed method was implemented on an NVIDIA GeForce 8800 GTX GPU hosted on a Mac Pro computer running Ubuntu 8.10 Linux operation system. The length of query sequences searched ranges from 64 to 4095, all of them can be found in UNIPROT database. All query sequences run against the Swiss-Prot protein sequence database [15]. Table 1. left shows the timing performance of the proposed method, the cut-off depths for the ungapped extension $X_1$ and gapped extension $X_2$ are 11 and 48 respectively.

Table 1. (left) Timing performance of the proposed method, (right) Performance comparison between the NCBI BLAST and the proposed method.

<table>
<thead>
<tr>
<th>sequence ID</th>
<th>Length</th>
<th>$E$</th>
<th>HSPs</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A4T9V0</td>
<td>64</td>
<td>1000</td>
<td>621</td>
<td>1.2</td>
</tr>
<tr>
<td>2. Q2IJ63</td>
<td>128</td>
<td>1000</td>
<td>955</td>
<td>1.7</td>
</tr>
<tr>
<td>3. P28484</td>
<td>256</td>
<td>1000</td>
<td>569</td>
<td>2.7</td>
</tr>
<tr>
<td>4. Q1JLB7</td>
<td>512</td>
<td>10</td>
<td>405</td>
<td>4.6</td>
</tr>
<tr>
<td>5. P08715</td>
<td>1024</td>
<td>10</td>
<td>104</td>
<td>9.7</td>
</tr>
<tr>
<td>6. Q8IYD8</td>
<td>2048</td>
<td>10</td>
<td>738</td>
<td>18.1</td>
</tr>
<tr>
<td>7. Q06277</td>
<td>4095</td>
<td>10</td>
<td>572</td>
<td>36.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sequence ID</th>
<th>Length</th>
<th>$E$</th>
<th>HSPs</th>
<th>GPU (s)</th>
<th>CPU (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A4T9V0</td>
<td>64</td>
<td>1000</td>
<td>621</td>
<td>1.2</td>
<td>3.2</td>
<td>2.7</td>
</tr>
<tr>
<td>2. Q2IJ63</td>
<td>128</td>
<td>1000</td>
<td>955</td>
<td>1.7</td>
<td>4.6</td>
<td>2.7</td>
</tr>
<tr>
<td>3. P28484</td>
<td>256</td>
<td>1000</td>
<td>569</td>
<td>2.7</td>
<td>6.4</td>
<td>2.4</td>
</tr>
<tr>
<td>4. Q1JLB7</td>
<td>512</td>
<td>10</td>
<td>405</td>
<td>4.6</td>
<td>10.4</td>
<td>2.3</td>
</tr>
<tr>
<td>5. P08715</td>
<td>1024</td>
<td>10</td>
<td>104</td>
<td>9.7</td>
<td>16.5</td>
<td>1.7</td>
</tr>
<tr>
<td>6. Q8IYD8</td>
<td>2048</td>
<td>10</td>
<td>738</td>
<td>18.1</td>
<td>30</td>
<td>1.7</td>
</tr>
<tr>
<td>7. Q06277</td>
<td>4095</td>
<td>10</td>
<td>572</td>
<td>36.7</td>
<td>70</td>
<td>1.9</td>
</tr>
</tbody>
</table>

In addition, Table 1. right shows the performance comparison between the proposed method and the widely used NCBI-BLAST running on CPU. The latter is executed on a Pentium4 3.4 GHz desktop computer. The speed-up figures show a 1.7x to 2.7x speed improvement achieved by our GPU implementation. The search speed of GPU mainly depends on the number of hits in each search space, which rests with the homological ratio between the query sequence and the database sequences. We thus list the exact sequence IDs of the query sequences in Table 1. (left) so that researchers can compare their implementations rigorously against ours. We also note that since the running results of NCBI-BLAST do not give the exact E value for different lengths of query sequences, our implementation adopts a general E value [19], which means that the comparison is not guaranteed to be completely fair. Nonetheless, the discrepancies are expected to be cancelled out with the number of experiments. It is worth noting that unlike previous GPU implementations of the Smith-Waterman algorithm which resulted in 15x-30x speed-ups compared to optimised CPU-based method, the performance benefit for BLAST on GPU are relatively smaller, although not negligible. The reason for this is due to the fact that BLAST is a heuristic algorithm, where the operations-to-transfer ratio of BLAST is much smaller than it in the Smith-Waterman algorithm. Indeed, performance benefits can be more readily achieved when the ratio of operations to elements transferred is higher [20]. More precisely, let $N$ denote the length of the query and subject sequences, the size of alignment matrix in the Smith-Waterman algorithm is $N^2$ and $2N$ elements are transferred into the device from host. Assuming the computation time of one matrix element equals to 1, the operations-to-transfer ratio is thus $N:2$, in which case the larger the matrix, the greater the performance benefits as long as there is large enough bandwidth for data communication and enough threads running simultaneously. However, this parameter is small in BLAST, as BLAST does not explore the entire search space. In addition, BLAST has many conditional processing which leads to larger kernel sizes and threads divergence, which in turn reduces the overall performance on CUDA-compatible GPU. Moreover, the number of hits in each alignment matrix is not fixed and is irregular, so thread operations cannot be predicted and it is impossible for threads to perform the same operations at the same time especially on large amounts of data. Therefore, the execution time for individual blocks depends on the completion time of the last executing thread, which decreases the overall performance. This situation would have been even worse if we had merged kernel1 and kernel2 in our proposed method, as data transferred from the host is used by a small number of threads entering the stage of gapped extension. Nonetheless, even with this, our proposed method results in a 1.7x-2.7x speed-up compared to the most optimized CPU implementation of Gapped BLAST using CPU and GPU platforms from the same technology (90nm).
5. Conclusion

In this paper, the detailed design implementation of Gapped BLAST algorithm has been presented. Through analyzing the execution of different steps of the algorithm, bottlenecks and the approach minimizing their effects have been laid out. The final GPU implementation of our design resulted in a 1.7x-2.7x speed-up compared to the most optimized CPU-based implementation, namely NCBI-BLAST. While this speed-up is smaller compared to the speed-up achieved by GPU implementations of the Smith-Waterman algorithm, it is not negligible especially given the complexity of the algorithm.

To our knowledge, this is the first GPU-based implementation of the Gapped BLAST algorithm ever reported in the literature. The work presented in this paper is part of a bigger project which seeks to harness the computational performance of CUDA-compatible GPUs in the field of Bioinformatics and Computational Biology (BCB). Future work includes the GPU implementation of phylogenetic tree construction algorithms and molecular dynamics simulation.

References

15. A. Bairoch, R. Apweiler. The SWISS-PROT protein knowledgebase and its supplement TrEMBL. Nucleic Acid Research, release 56.3, 14-Oct-08
18. M. Cameron, H E. Williams and A Cannane. A Deterministic Finite Automaton for Faster Protein Hit Detection in BLAST, journal of computational biology, 2006