Efficient Pairwise Statistical Significance Estimation for Local Sequence Alignment Using GPU

[1,2]Yuhong Zhang, [2]Sanchit Misra, [2]Daniel Honbo, [2]Ankit Agrawal,[2]Wei-keng Liao, [2]Alok Choudhary

[1]*School of Computer Science and Engineering*
University of Electronic Science and Technology of China
Chengdu, China

[2]*Department of Electrical Engineering and Computer Science*
Northwestern University
Evanston, USA

Email:{yuhong, smi539, dkh301, ankitag, wkliao, choudhar}@eecs.northwestern.edu

***Abstract*—Pairwise statistical significance has been found to be quite accurate in identifying related sequences (homologs), which is a key step in numerous bioinformatics applications. However, it is computational and data intensive, particularly for a large amount of sequence data. To prevent it from becoming a performance bottleneck, we resort to Graphics Processing Units (GPUs) for accelerating the computation. In this paper, we present a GPU memory-access optimized implementation for a pairwise statistical significance estimation algorithm. By exploring the algorithm's data access characteristics, we developed a tile-based scheme that can produce a contiguous memory accesses pattern to GPU global memory and sustain a large number of threads to achieve a high GPU occupancy. Our experimental results present both single- and multi-pair statistical significance estimations. The performance evaluation was carried out on an NVIDIA Telsa C2050 GPU. We observe more than 180× end-to-end speedup over the CPU implementation on an Intel© Core™ i7 processor. The proposed memory access optimizations and efficient framework are also applicable to many other sequence comparison based applications, such as DNA sequence mapping and database search.**

***Keywords*-pairwise sequence alignment; statistical significance; GPU;**

I. INTRODUCTION

Pairwise sequence alignment is an important cornerstone procedure in bioinformatics. Many applications have been developed for pairwise sequence alignment, such as BLAST [1], PSI-BLAST, FASTA [2], and SSEARCH, etc. Statistical significance represents the likelihood that the resulting similarity, in terms of alignment score, between two sequences cannot have arisen by chance alone. It is useful to determine whether the similarity is a functional or evolutionary link, or a chance event [3]. The recently proposed pairwise statistical significance estimation (PSSE) has been shown to be a promising alternative to the database statistical significance for the purpose of identifying homologs [4], [5], [6].

The past decades have witnessed dramatically increasing trends in the quantity and variety of publicly available genomic and proteomic sequence data [7], [8]. Although shown to be accurate, PSSE is computationally intensive. An unscalable implementation can become a bottleneck to be practically useful for many large-scale applications. However, the demonstrated significant improvement in retrieval accuracy using PSSE strongly motivates the use of high performance computing techniques to accelerate its processing speed [9], [10]. With General Purpose Graphics Processing Units (GPGPUs) becoming increasingly powerful, inexpensive, and relatively easy to program, it has become a very attractive hardware acceleration platform.

In this paper, we present a GPU implementation for PSSE that uses a tiled-based memory strategy to produce contiguous global memory access and increase the GPU occupancy. GPU occupancy is one of the key measures for GPU efficiency [11]. Through carefully analyzing the PSSE's data dependency and access patterns, the proposed tiled-based strategy reorganizes the data sequences by using aligned structure of arrays to coalesce the global memory access pattern. Such data access contiguity can significantly accelerate the global memory reads. Our design not only calculates an optimal tile size to be shipped to the GPU, but also issues a large enough number of threads to maximize the *occupancy*. With high occupancy, much of this global memory latency can be hidden by the thread scheduler while some threads are waiting for the global memory access to complete. This paper presents the results of both single- and multi-pair PSSE implementations on an NVIDIA Tesla C2050 GPU. We observe more than 180× speedups over the CPU implementation using an Intel© Core™ i7 processor.

The proposed tile-based memory access framework for PSSE can directly benefit homology detection based applications like database search and phylogenetic tree construction.

II. BACKGROUND AND RELATED WORK

Given a sequence pair s_1 and s_2 of lengths m and n respectively, the scoring scheme SC (substitution matrix, gap opening penalty, gap extension penalty), and the number of permutations N, the PSSE problem [4] is to calculate the following function:

$$PSSE(s_1, s_2, m, n, SC, N)$$

Through permuting s_2 N times randomly, the function generates N scores by aligning s_1 against each of the N permuted sequences and then fits these scores to an Extreme Value Distribution (EVD). The returned value is the pairwise statistical significance of s_1 and s_2.

The EVD describes an approximate distribution of optimal scores of a gapless alignment [12], [3], [13]. Based on this distribution, the probability (i.e. *P-value*) of observing an sequence pair with a score S greater than x , can be given by

$$P(S > x) \approx 1 - exp(-Kmne^{-\lambda x}) = 1 - e^{-E(x)} \quad (1)$$

where K and λ are constants and $E(x)$, known as *E-value*, is the expected number of distinct local alignments with score values of at least x. Formula (1) can also be generalized to apply to multiple sequences, allowing the significance of specific sequence alignments to be evaluated [12].

Local sequence alignment methods are used to find the best-matching piecewise alignments with affine gap penalties. For these cases of gapped alignment, although no asymptotic score distribution has yet been established analytically, computational experiments strongly indicate these scores still roughly follow Gumbel law after pragmatic estimation of the λ and K parameters [14], [15], [2], [3], which can be obtained by using a censored maximum likelihood scheme [16].

In general, there are two primary methods to estimate the statistical significance of local sequence alignment. One is called database statistical significance adopted by many popular database search programs, such as BLAST [1], FASTA [2], and SSEARCH (using full implementation of Smith-Waterman algorithm [17]). This method depends on the size and composition of the database being searched. The other method is called the pairwise statistical significance estimation (PSSE), which is specific to the sequence-pair being aligned, and independent of any database.

PSSE is an attempt which makes the estimation process more specific to the sequence pair being compared. It can produce much more biologically relevant estimates of statistical significance than database statistical significance method [4], [5], [6].

Using high-performance computing (HPC) techniques to accelerate PSSE has become inevitable, as the data size grows larger demanding more computation power. Related work includes implementations using MPI [10] and FPGA [9]. However, the existing HPC approaches are limited to the single-pair estimation only.

III. DESIGN

The procedure of PSSE can be decomposed into three kernels: permutation, alignment, and fitting. As analyzed in [10], the permutation and alignment tasks comprise the overwhelming majority, a more than 99.8%, of the overall execution time in the CPU implementation. This provides a quick sanity check on where the speed-up should focus on. Permutation and alignment present high degrees of data independency that are naturally suited for SIMT (Single-Instruction, Multiple-Thread) architectures and hence can be mapped very well to task parallelism models of the GPU.

A. GPU Memory Access Optimization

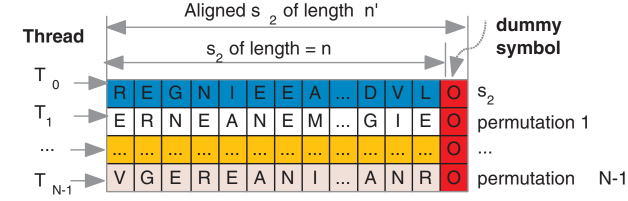
To obtain good performance, GPU programmers are expected to have a good understanding of memory hierarchy of GPU. Especially, global memory coalescing is the most critical optimization for GPU programming [18]. The *kernels* of PSSE usually work over large numbers of sequences which reside in the global memory. Therefore, the best performance of PSSE relies on hiding access latency to the global memory.

When a GPU kernel is accessing global memory, all threads in groups of 32 (i.e. *warp*) access a bank of memory at one time. A batch of memory accesses is considered coalesced when the data requested by a warp of threads are located in contiguous memory addresses. For example, if the data requested by threads within a warp are located in 32 consecutive memory addresses (i.e. the ith address is accessed by the ith thread), the memory can be read in a single access. If this is the case for all memory accesses, the performance can speed up by a factor of 32. Whether data is coalesced or not has a significant impact to an application's performance, as it determines the degree of memory access parallelism [11].

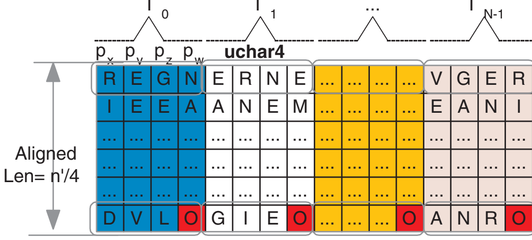
After permutation, if the sequence s_2 and its N permuted copies were stored contiguously one after another in the global memory. The intuitive memory layout is shown as in Figure 1 (a). Considering inter-task parallelism, where each thread works on the alignment of one of the permuted copies of s_2 to s_1, in this layout, for example, if thread T_0 accessed the first *uchar* (i.e. 'R'), and thread T_1 accessed the first *uchar* (i.e. 'E'), etc., the gap between positions accessed by the neighboring threads T_0 and T_1 is the sequence length n. This results in non-coalesced memory reads (i.e. serialized reads), which significantly deteriorates performance.

To achieve high parallelism of global memory access, the permuted sequences are stored in the CUDA built-in vector data type *uchar4*. In this case, the length of both s_1 and s_2 sequences must be a multiple of $t = sizeof(uchar4) = 4$. If the length is not a multiple of t, the sequence should be padded with dummy amino acids symbols to make the length a multiple of t. The align length $n' = 4 \times \lceil n/4 \rceil$. Further, to achieve coalesced access, the intuitive memory layout of sequences should be reorganized to a aligned structure of arrays, as shown in Figure 1 (b). In the optimized layout, the characters (in granularity of 4 bytes) that lie at the same index in different permuted sequences can stay at neighboring positions. Now, if the first *uchar4* of the first permuted sequence (i.e. 'REGN') is requested by thread T_0, the first *uchar4* of the second permuted sequence (i.e. 'ERNE') is requested by T_1, etc., the global memory access is coalesced, resulting in high performance.

After reorganization, sequences remain unchanged during the alignment. As a result, sequences can be thought of as read-only data, which can be bound to texture memory. For



(a) The intuitive layout that one sequence is appended after another



(b) The reorganized layout such that the uchar4 at the same indices in different sequences stay at neighboring positions

Figure 1: Two GPU memory layout strategies.

read patterns, texture memory fetches is a better alternative to global memory reads because of texture memory cache. It can further improve acceleration performance.

B. Maximize Occupancy

Hiding global memory latency is very important to achieve maximum performance on the GPU. This can be done by creating enough threads to keep the CUDA cores occupied while many threads are waiting on global memory accesses [18]. GPU *occupancy* is a metric defined below to determine how effectively the hardware is kept busy.

$$Occupancy = (B \times T_{num})/T_{max} \tag{2}$$

where T_{max} is maximum number of resident threads on a multiprocessor (which is a constant for a specific GPU), T_{num} is the number of active threads per block and B is the number of active blocks per Streaming Multiprocessor (SM).

B may depend upon the GPU physical limitations (e.g. the amount of registers, shared memory and threads supported in each model). It can be given in the following way:

$$B = \min(B_{user}, B_{reg}, B_{shr}, B_{hw}) \tag{3}$$

where B_{hw} is the hardware limit (only 8 blocks are allowed per SM), and B_{reg}, B_{shr}, are the potential blocks determined by the available registers, shared memory, respectively and B_{user} is blocks set by the user. Because

$$B \leq B_{hw} \tag{4}$$

We obtain

$$Occupancy \leq (B_{hw} \times T_{num})/T_{max} \tag{5}$$

As analyzed above, one can get higher occupancy according to the following rules:

- To avoid wasting computation on under-populated warps, the number of threads per block should be chosen as a multiple of the warp size (currently, 32).
- Total number of active threads per SM should be close to maximum number of resident threads per multiprocessor (T_{max}). In short, we should use all the available threads if possible.
- To achieve 100% occupancy in theory, $(B_{hw} \times T_{num})/T_{max} \geq 1$. Hence, setting T_{num} such that $T_{num} \geq T_{max}/B_{hw}$ is preferable.

IV. IMPLEMENTATIONS

To permute sequences, as required in single-pair and multi-pair implementations, a large number of random numbers are needed. Though the most widely used pseudorandom number generators, like linear congruential generator (LCG) can meet our requirements, CUDA 3.1 and prior versions do not include such function, Hence, we develop a random number generator similar to *lrand48()* on GPU.

A. SINGLE-PAIR PSSE IMPLEMENTATION

The single-pair PSSE processes only one pair of query and subject sequences. Its pseudocode is shown in Algorithm 1. Step 2-3 complete the permutation kernel in GPU. Step 4 executes the memory layout reorganization of the s_2 and its N permuted copies. In step 5, Smith-Waterman algorithm is used to perform the alignment task in our work. This dynamic programming method identifies the optimal local alignment between a pair of sequences. In [19], two different methods are proposed to parallelize this alignment task. The first method is called inter-task parallelism. Each thread performs one alignment. Hence, multiple alignments are performed in parallel in a thread block; The second method is called intra-task parallelism. All threads in a thread block perform a single alignment. Threads cooperate with each other to exploit the parallel characteristics of cells in the minor diagonals of the local alignment matrix. We use a similar alignment kernel as proposed in [19] with some modifications to further improve performance. Steps 6-8 describe the fitting kernel which is implemented on the CPU. The final statistical significance is returned in Step 9.

B. MULTI-PAIR PSSE IMPLEMENTATION

The multi-pair PSSE processes multiple queries and subject sequences. Through analyzing our experimental results of the single-pair PSSE, we find that inter-task parallelism performs better than intra-task parallelism (results shown later). Hence, we use the inter-task parallelism in our multi-pair implementation. Based on the guidelines for optimizing memory and occupancy described earlier, we compare three implementation strategies.

Algorithm 1: Pseudocode of single-pair PSSE

Input: (s_1, s_2): Sequence-pair; M: Substitution matrix; G: Gap opening penalty; GE: Gap extension penalty; N: Number of permutes;
Output: pss: Pairwise statistical significance

```
1 Transfer Sequence-pair (s1, s2) from CPU to GPU;
2 Generate random numbers (RNs) in GPU;
3 Permute sequence s2 N times using the RNs;
  /* permutation kernel in GPU */
4 Reorganize sequence s2 and its N times permuted
  copies to new memory layout shown as Figure 1 (b);
5 Align sequence s2 and its N times permuted copies
  against s1 using inter- (or intra-) task parallel
  Smith-Waterman();
  /* alignment kernel in GPU */
6 Transfer N alignment scores into CPU for fitting;
7 (K, λ) ← EVD_CensoredMLFit (scores) ;
8 pss ← 1 − exp(−Kmne^{−λx});
  /* fitting kernel in CPU */
9 return pss
```

(1) Intuitive strategy
Given Q query sequences and S subject sequences, the intuitive strategy is to simply perform the same 'single-pair' procedure $Q \times S$ times. In other words, in each iteration, we send a single pair of query and subject sequences to the GPU. The GPU processes that pair and returns the result to the CPU. The same procedure is repeated for all query and subject sequence pairs. Similar to the single-pair PSSE, this strategy suffers from low occupancy. The cause will be analyzed together with its performance results in the next section.

(2) Data reuse strategy
In the first strategy, the subject sequence from the database is permuted only for one query sequence. When another query sequence is processed, the same subject sequence needs to be permuted again. In fact, the N permuted sequences can be reused by all the query sequences. "One permutation, all queries" is the thought of our second strategy. Because of the reuse of permuted sequences, higher performance is expected than the first strategy. However, the occupancy, to be shown in the next section, is still not elevated.

(3) Tile-based strategy
The low occupancy of the first two strategies is due to the underutilized computing power of the GPU. In addition, these two strategies will not work well when the size of subject sequence database becomes too big to be fitted into the GPU global memory. For instance, even if the original subject sequence database is just of size 5 MB, it becomes $5\times1000 = 5000$ MB after each sequence is permuted, say, $N = 1000$ times.

Herein we develop a memory tiling technique that is self-tuning based on the hardware configuration and can achieve a close-to-optimal performance. Not only can this tiling approach calculate an optimal size to fit in GPU memory, but launch a large enough number of threads to produce a high occupancy during the execution. The tile size (namely, the number of subject sequences to be transferred to the GPU) T can be calculated by

$$T = \lfloor \frac{SM_{num} \times T_{max}}{N} \rfloor \quad (6)$$

where SM_{num} is the total number of SMs in GPU, T_{max} is the maximum number of resident threads per SM, and N is the number of permutations.

In the Tesla C2050 used in our implementation, there are 14 SMs and the maximum resident threads per multiprocessor is 1024. Let $N = 1000$, hence, $T = \lfloor 14\times1024/1000 \rfloor = 14$, which means there are 14 distinct subject sequences to be transferred to the GPU's global memory at a time. Based on the second strategy (i.e. data reuse), the 14 subject sequences and their permuted copies are aligned against one query sequence at a time, until all the query sequences are processed. As a result, 14×1000 alignment scores in total are obtained in each round, which are subsequently transferred into CPU for fitting.

V. EXPERIMENTAL RESULTS

All our implementations are carried out on a 2.67GHz Intel© Core™ i7, quad-core 2.67 GHz processor, running an x86_64 GNU Linux with 4 GB of RAM and a Tesla C2050 GPU. Our programs are developed using the CUDA 3.1 driver compiler. The scoring matrix BLOSUM50 is used for alignment with affine gap penalty of $10+2k$ for a gap of length k. We make use of the same number of permutations, N=1000, as several previous studies do [4], [9], [10]. We use a non-redundant subset of the CATH 2.3 database provided by Sierk and Pearson [20]. This dataset consists of 2771 domain sequences as our subject library, and includes 86 CATH queries as our query set.

A. SINGLE-PAIR PSSE RESULT ANALYSIS

In our single-pair implementation using both the intra- and inter-task parallelism methods, four pairs of sequences of length 200, 400, 800, and 1600 chosen from CATH 2.3, are performed by 64, 128, 256, and 512 threads per block, respectively. The results are shown in Figure 2(a).

For intra-task parallelism, among the sequence lengths of 200, 400, and 800, the best achieved speedups over the CPU implementation are 18.78×, 31.19× and 39.72×, respectively, when the number of threads per block is equal to 128. However, for the sequence length of 1600, the best speedup is 46.34×, when there are 256 threads per block. From these observations, we can see no general principle that can parameterize the optimal settings like the number of threads per block and blocks per grid, because many possible combinations of optimizations and configurations, such as

the number of blocks and the available registers and shared memory, may contradict with each other.

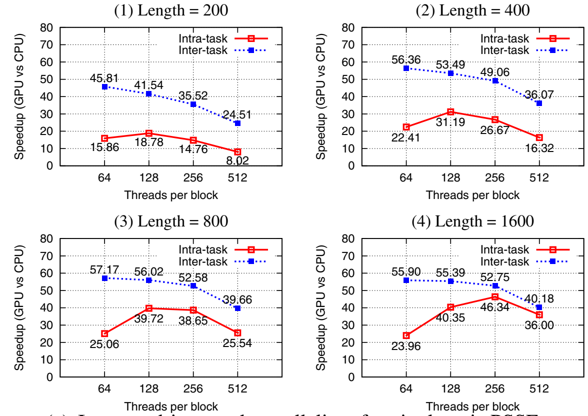
Seemingly, intra-task parallelism can create enough thread blocks (one block for each alignment task) to keep the occupancy high. However, the alignment matrix of PSSE must be serially computed from the first to the last antidiagonal. Only the cells of the alignment matrix belonging to the same antidiagonal can be computed in parallel. The intra-task parallelism is, therefore, limited. In this case, most threads in a block have no work to do, which cause the worse performance than the inter-task parallelism method.

In contrast, inter-task parallelism would have a total of 1000 threads (one for each alignment task). If each block contains 64 threads, then the total number of blocks is $B = \lceil 1000/64 \rceil = 16$. The assignment of 16 blocks to 14 available SMs will result in 12 SMs with one active block and two SMs with two blocks. Therefore, the occupancy for the two SMs with two blocks is $(2 \times 64)/1024 = 12.5\%$. For the 12 SMs with only one block its occupancy is $(1 \times 64)/1024 = 6.25\%$. Because of the low occupancy, there are not sufficient threads to keep CUDA cores busy when the global memory is accessed. As a result, the latency-hiding capabilities of this method are limited. As the number of threads per block T_{num} increases, the total active blocks B decreases subsequently and some SMs even have no blocks assigned. As a result, we observe decreasing speedups as the number of threads grows. When the number of threads per block is 64, for the sequence length of 200, 400, 800, and 1600, the best speedups are 45.81×, 56.36×, 57.17×, and 55.90×, respectively.

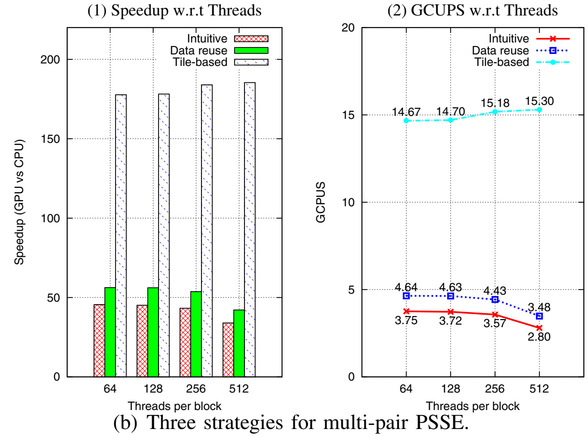
In principle, getting performance out of a GPU is to keep the CUDA cores busy all the time. Given a small sized dataset, both inter- and intra-task parallelism methods fail to achieve this goal, although each is caused by a different reason.

B. *MULTI-PAIR PSSE RESULT ANALYSIS*

In the intuitive and data reuse strategies, most of the SMs suffer from the same low occupancy as the one-pair PSSE implementation using the inter-task parallelism method. As expected, we observe poor performance for both strategies, as shown in Figure 2(b). The data reuse strategy produces higher performance than the intuitive strategy, because the number of permutations is reduced by a factor of Q, the number of query sequences. To alleviate the low occupancy problem, our proposed tile-based strategy uses a carefully tuned tile size that can effectively improve the occupancy. For instance, when the number of threads per block is 64, the number of blocks per SM is $min(1024/64, 8) = 8$, resulting in an occupancy of $(8 \times 64)/1024 = 50\%$, based on Equations (3) and (2). Such occupancy is a much better improved number than the cases using the simple and data-reuse strategies. The maximum occupancy of different strategies for an SM are given in Table I.



(a) Intra- and inter-task parallelism for single-pair PSSE.



(b) Three strategies for multi-pair PSSE.

Figure 2: Performance results of single- and multi-pair PSSE implementations.

Table I: The maximum occupancy for three strategies

Threads/blocks	64	128	256	512
Intuitive Occu.	12.5%	6.25%	6.25%	6.25%
Data-reuse Occu.	12.5%	6.25%	6.25%	6.25%
Tile-based Occu.	50%	100%	100%	100%

Due to a high occupancy, the tile-based strategy achieves significant speedups of 177.7×, 178.1×, 183.9×, and 185.4× for 64, 128, 256, and 512 threads per block, respectively, as shown in Figure 2(b). With the increasing number of threads per block, the maximum number of active blocks per SM decreases accordingly. As a result, slightly higher performance can be obtained by reducing the block-switch time in an SM. The GCUPS (billion cell updates per second) value is another commonly used performance measure in bioinformatics [19]. The tile-based strategy achieves performance results from 14.67 to 15.30 GCUPS, as shown in the right chart of Figure 2(b). Although a direct comparison across different GPU implementations in not fair, just for the sake of completeness, we cited below the GCUPS performance reported by some existing

implementations of the Smith-Waterman alignment task. [21] has a peak performance of 4.65 to 8.99 GCUPS for various query lengths on an NVIDIA 9800. [22] has a peak performance of 3.5 GCUPS on a workstation running two GeForce 8800 GTX.

In summary, low occupancy is known to interfere with the ability to hide latency on memory-bound kernels, causing performance degradation [11]. However, increasing occupancy does not necessarily increase performance. In general, once a 50% occupancy is achieved, further optimization to gain additional occupancy takes little effect.

VI. CONCLUSIONS

We present a comparative performance analysis on single- and multi-pair GPU implementations. Through a careful design by considering memory optimization and occupancy maximization, our proposed tile-based strategy results in high end-to-end speedups. The performance improvement for PSSE can be very useful for a wide variety of sequence comparison based applications, such as sequence mapping and database searches. As the size of biological sequence databases grows rapidly, even more powerful high performance computing approaches (such as GPU clusters, FPGA and GPU heterogeneous platforms) will become highly demanded in the near future, for which this work can serve as a stepping stone.

ACKNOWLEDGEMENT

This work was supported in part by NSF award numbers: CCF-0621443, SDCI OCI-0724599, CNS-0551639, IIS-0536994, and HECURA-0938000. This work was also partially supported by DOE grants DE-FC02-07ER25808 and DE-FG02-08ER25848 and China Scholarship Council.

REFERENCES

[1] S. F. Altschul *et al.*, “Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs.” *Nucl. Acids Res.*, vol. 25, no. 17, pp. 3389–3402, 1997.

[2] W. R. Pearson, “Empirical Statistical Estimates for Sequence Similarity Searches,” *J. of Molecular Biology*, vol. 276, pp. 71–84, 1998.

[3] R. Mott, “Accurate Formula for P-values of Gapped Local Sequence and Profile Alignments.” *J. of Molecular Biology*, vol. 300, pp. 649–659, 2000.

[4] A. Agrawal *et al.*, “Pairwise statistical significance versus database statistical significance for local alignment of protein sequences,” in *Bioinfo. Res. and App.*, vol. 4983, 2008, pp. 50–61.

[5] A. Agrawal, “Pairwise statistical significance of local sequence alignment using sequence-specific and position-specific substitution matrices,” *IEEE/ACM TCBB*, vol. 99, no. 1, 2009.

[6] A. Agrawal *et al.*, “PSIBLAST_PairwiseStatSig reordering PSI-BLAST hits using pairwise statistical significance,” *Bioinformatics*, vol. 25, no. 8, pp. 1082–1083, 2009.

[7] R. David, “COMPUTATIONAL BIOLOGY: Bioinformatics–Trying to Swim in a Sea of Data,” *Science*, vol. 291, no. 5507, pp. 1260–1261, 2001.

[8] S. Yooseph *et al.*, “The sorcerer II global ocean sampling expedition: Expanding the universe of protein families,” *PLoS Biol*, vol. 5, no. 3, p. e16, 2007.

[9] D. Honbo *et al.*, “Efficient Pairwise Statistical Significance Estimation using FPGAs,” in *BIOCOMP*, vol. 4983, 2010, pp. 571–577.

[10] A. Agrawal *et al.*, “MPIPairwiseStatSig: Parallel pairwise statistical significance estimation of local sequence alignment,” *HPDC*, pp. 470–476, 2010.

[11] NVIDIA, “NVIDIA CUDA C: Best Practices Guide 3.1,” 2010.

[12] S. Karlin *et al.*, “Methods for Assessing the Statistical Significance of Molecular Sequence Features by Using General Scoring Schemes,” *PNAS, USA*, vol. 87, no. 6, pp. 2264–2268, 1990.

[13] A. Dembo *et al.*, “Limit distribution of maximal non-aligned two-sequence segmental score,” *Ann. Prob.*, vol. 22, pp. 2022–2039, 1994.

[14] M. S. Waterman *et al.*, “Rapid and Accurate Estimates of Statistical Significance for Sequence Database Searches,” *PNAS, USA*, vol. 91, no. 11, pp. 4625–4628, 1994.

[15] S. F. Altschul *et al.*, “Local Alignment Statistics.” *Methods in Enzymology*, vol. 266, pp. 460–80, 1996.

[16] S. R. Eddy, “Maximum likelihood fitting of extreme value distributions,” 1997, unpublished work.

[17] T. F. Smith *et al.*, “Identification of Common Molecular Subsequences.” *J. of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[18] S. Ryoo *et al.*, “Optimization principles and application performance evaluation of a multithreaded GPU using CUDA,” in *PPOPP*, 2008, pp. 73–82.

[19] Y. Liu *et al.*, “CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units,” *BMCRN*, vol. 2, no. 1, p. 73, 2009.

[20] M. L. Sierk *et al.*, “Sensitivity and selectivity in protein structure comparison,” *Protein Science*, vol. 13, no. 3, pp. 773–785, 2004.

[21] L. Ligowski *et al.*, “An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases,” in *IPDPS*, 2009, pp. 1–8.

[22] S. Manavski *et al.*, “CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment,” *BMCB*, vol. 9, no. S-2, 2008.