Biological Sequence Alignment on Graphics Processing Units

Gerrit Voss, Andre Schröder, Wolfgang Müller-Wittig, Bertil Schmidt

Centre for Advanced Media Technology, Nanyang Technological University, Singapore 639798, {vossg,habicht,mueller}@camtech.ntu.edu.sg, asbschmidt@ntu.edu.sg,

Abstract

Sequence alignment is a common and often repeated task in molecular biology. The need for speeding up this treatment comes from the rapid growth rate of biological sequence databases. In this paper we present a new approach to high performance biological sequence database scanning on graphics processing units. Using modern graphics processing units for high performance computing is facilitated by their enhanced programmability and motivated by their attractive price/performance ratio and incredible growth in speed. To derive an efficient mapping onto this type of architecture, we have reformulated the Smith-Waterman dynamic programming algorithm in terms of computer graphics primitives. This results in an implementation with significant runtime savings on two standard off-the-shelf computer graphics cards. To our knowledge this is the first reported mapping of biological sequence alignment onto a graphics processing unit.

1. Introduction

The fast increasing power of the GPU (*Graphics Processing Unit*) and its streaming architecture opens up a range of new possibilities for a variety of applications. With the enhanced programmability of commodity GPUs, these chips are now capable of performing more than the specific graphics computations they were originally designed for. Recent work shows the design and implementation of algorithms for non-graphics applications. Examples include scientific computing [7], image processing [17], and fast Fourier transform [10], just to name a few. The evolution of GPUs is driven by the computer game market. This leads to a relatively small price per unit and to very rapid developments of next generations.

Currently, the peak performance of state-of-the-art GPUs is approximately four to five times faster than that of comparable CPUs (see Table 1). Furthermore, the growth rate of the number of transistors used on GPUs is greater than for microprocessors (see Table 2). Consequently, the GPUs will become an even more attractive alternative for high performance computing in the near future.

Table 1: CPU-GPU performance comparison. (GFlops performance is based on multiply-add operations)

Processor	Type	GFlops	Throughput
Pentium4 3.4 GHz	CPU	13.6	5.96 GB/s
NVidia GeForce 6800 Ultra	GPU	51.2	32.7 GB/s
ATI Radeon X800 XT	GPU	66.6	33.4 GB/s

Table 2: Growth rate of number of transistors on CPUs and GPUs.

Processor	Annual growth	Decade growth
CPU	1.5×	60×
GPU	>2.0×	>1000×

However, there are still a number of challenges to be solved in order to enable scientists other than computer graphics specialists to facilitate efficient usage of these resources within their research area. The biggest challenge in order to solve a specific problem using GPUs is reformulating the proposed algorithms and data structures using computer graphics primitives (e.g. triangles, textures, vertices, fragments). Furthermore, restrictions of the underlying streaming architecture have to be taken into account, e.g. random access writes to memory is not supported and no cross fragment data or persistent state is possible (e.g. all the internal registers are flushed before a new fragment is processed). In this paper we show how biological sequence alignment based on the Smith-Waterman dynamic programming algorithm can benefit from this type of computing power.

The rest of this paper is organized as follows. In Section 2, we introduce the basic sequence alignment algorithm. Section 3 highlights previous work on parallelization of this algorithm on different parallel architectures. Important features of the GPU architectures are described in Section 4. Section 5 presents our mapping of the algorithm onto the GPU architecture. A performance evaluation for two different graphics cards is given in Section 6. Finally, Section 7 concludes the paper with an outlook to further research topics.

2. Smith-Waterman Algorithm

Surprising relationships have been discovered between protein sequences that have little overall similarity but in which similar subsequences can be found. In that sense, the identification of similar subsequences is probably the most useful and practical method for comparing two sequences. The Smith-Waterman algorithm [15] finds the most similar subsequences of two sequences (the local alignment) by dynamic programming (*DP*). The algorithm compares two sequences by computing a distance that represents the minimal cost of transforming one segment into another. Two elementary operations are used: substitution and insertion/deletion (also called a gap operation). Through series of such elementary operations, any segments can be transformed into any other segment. The smallest number of operations required to change one segment into another can be taken into as the measure of the distance between the segments.

	∅	A	T	C	T	C	G	T	A	T	G	A	T	G
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	2	1	0	0	2	1	0	2
T	0	0	2	1	2	1	1	4	3	2	1	1	3	2
C	0	0	1	4	3	4	3	3	3	2	1	0	2	2
T	0	0	2	3	6	5	4	5	4	5	4	3	2	1
A	0	2	2	2	5	5	4	4	7	6	5	6	5	4
T	0	1	4	3	4	4	4	6	5	9	8	7	8	7
C	0	0	3	6	5	6	5	5	5	8	8	7	7	7
A	0	2	2	5	5	5	5	4	7	7	7	10	9	8
C	0	1	1	4	4	7	6	5	6	6	6	9	9	8

Figure 1. Example of the Smith-Waterman algorithm to compute the local alignment between two DNA sequences ATCTCGTATGATG and GTCTATCAC. The matrix *H(i,j)* is shown for the linear gap cost α = 1, and a substitution cost of +2 if the characters are identical and –1 otherwise. From the highest score (+10 in the example), a traceback procedure delivers the corresponding alignment (shaded cells), the two subsequences TCGTATGA and TCTATCA.

Consider two strings $S1$ and $S2$ of length $l1$ and $l2$. To identify common subsequences, the Smith-Waterman algorithm computes the similarity $H(i,j)$ of two sequences ending at position i and j of the two sequences $S1$ and $S2$. The computation of $H(i,j)$, for $1 \le i \le l1$, $1 \le j \le l2$, is given by the following recurrences:

$$H(i,j) = \max\{0, E(i,j), F(i,j), H(i-1,j-1) + sbt(S_1[i], S_2[j])\}$$
$$E(i,j) = \max\{H(i,j-1) - \alpha, E(i,j-1) - \beta\},$$
$$F(i,j) = \max\{H(i-1,j) - \alpha, F(i-1,j) - \beta\},$$

where *sbt* is a character substitution cost table. Initialization of these values are given by $H(i,0) = E(i,0) = H(0,j) = F(0,j) = 0$ for $0 \le i \le l_1$, $0 \le j \le l_2$. Multiple gap costs are taken into account as follows: α is the cost of the first gap; β is the cost of the following gaps. This type of gap cost is known as *affine gap penalty*. Some applications also use a *linear gap penalty*, i.e. α = β. For linear gap penalties the above recurrence relations can be simplified to:

$$\begin{aligned} H(i,j) = \max\{0, H(i,j-1) - \alpha, H(i-1,j) - \alpha, \\ H(i-1,j-1) + sbt(S_1[i], S_2[j])\}, \end{aligned}$$

Each position of the matrix H is a similarity value. The two segments of S_1 and S_2 producing this value can be determined by a traceback procedure. Figure 1 illustrates an example.

3. Related Work

A number of parallel architectures have been developed for biological sequence alignment. In addition to architectures specifically designed for sequence analysis, existing programmable sequential and parallel architectures have been used for solving sequence alignment problems.

Special-purpose architectures can provide the fastest means of running a particular algorithm with very high processing element (*PE*) density. Each PE is specifically designed for the computation of one cell in the DP matrix (see Figure 1). However, such architectures are limited to one single algorithm, and thus cannot supply the flexibility necessary to run a variety of algorithms required for analyzing DNA, RNA, and proteins. P-NAC was the first such machine and computed edit distance over a four-character alphabet [8]. More recent examples, better tuned to the needs of computational biology, include BioScan [14], BISP [2], and SAMBA [5].

An approach presented in [13] is based on instruction systolic arrays (ISAs). ISAs combine the speed and simplicity of systolic arrays with flexible programmability. Several other approaches are based on the SIMD concept, e.g. MGAP [1], Kestrel [3], and Fuzion [13]. SIMD and ISA architectures are programmable and can be used for a wider range of applications, such as image processing and scientific computing. Since these architectures contain more general-purpose parallel processors, their PE density is less than the density of special-purpose ASICs. Nevertheless, SIMD solutions can still achieve significant runtime savings. However, the costs involved in designing and producing SIMD architectures are quite high. As a consequence, none of the above solutions has a successor generation, making upgrading impossible.

Reconfigurable systems are based on programmable logic such as field-programmable gate arrays (*FPGAs*). They are generally slower and have lower PE densities than special-purpose architectures, e.g. [11,17]. They are flexible, but the configuration must be changed for each algorithm, which is generally more complicated than writing new code for a programmable architecture. Solutions based on FPGAs have the additional advantage that they can be regularly upgraded to state-of-the-art-technology.

All these approaches can be seen as accelerators – an approach satisfying the demand for a low cost solution to compute-intensive problems. The main advantage of GPUs compared to the architectures mentioned above is that they are commodity components. In particular, most users have already access to PCs with modern graphics cards. For these users this direction provides a zero-cost solution. Even if a graphics card has to be bought, the installation of such a card is trivial (plug and play). Writing the software for such a card does still require specialist knowledge, but new high level languages such as *Cg* [9] offer a simplified programming environment.

4. GPU Architecture

Computation on a GPU follows a fixed order of processing stages, called the graphics pipeline (see Figure 2). The pipeline consists of three stages: vertex processing, rasterization stage and fragment processing. The vertex processing stage transforms 3-dimensional vertex world coordinates into 2-dimensional vertex screen coordinates. The rasterizer then converts the geometric vertex representation into an image fragment representation. Finally, the fragment processor forms a color for each pixel by reading texels (texture pixels) from the texture memory. Modern GPUs support programmability of the vertex and fragment processor. Fragment programs for instance can be used to implement any mathematical operation on one or more input vectors (textures or fragments) to compute the color of a pixel.

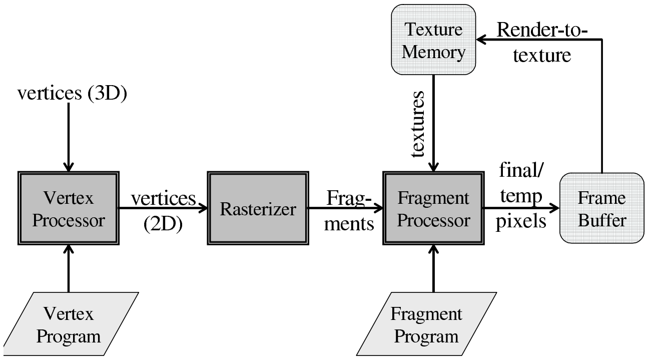

Figure 2: Illustration of the GPU graphics pipeline.

In order to meet the ever increasing performance requirements set by the gaming industry fourth generation GPUs use two types of parallelism. Firstly, multiple processors work on the vertex and fragment processing stage, i.e. they operate on different vertices and fragments in parallel. For example, a typical mid-range graphics card such as the nVidia GeForce 6800 GT has 6 vertex processors and 16 fragment processors. Secondly, operations on 4-dimensional vectors (the four channels Red/Green/Blue/Alpha (RGBA)) are natively supported without performance loss.

Several authors have described modern GPUs as *streaming processors*, e.g [12]. Streaming processors read an input stream, apply kernels (filters) to the stream and write the results into an output stream. In case of several kernels, the output stream of the leading kernel is the input stream for the following kernel (see Figure 3). The vast majority of general-purpose GPU applications use only fragment programs for their computation. In this case textures are considered as input streams and the texture buffers are output streams. Because fragment processors are SIMD architectures, only one program can be loaded at a time. Applying several kernels thus means to do several passes.

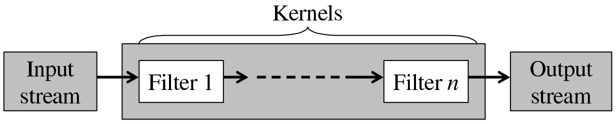

Figure 3: Streaming model that applies kernels to an input stream and writes to an output stream.

Hence, general-purpose GPU applications share the following three features [4]:

1. Textures are data input
2. Input is processed by a fragment program
3. Feedback is realized by using the output buffer of a completed pass as input texture for the following one (known as *render-to-texture* (RTT))

5. Mapping Sequence Alignment onto a GPU

In this section we describe how the Smith-Waterman algorithm can be efficiently mapped onto a GPU. We take advantage of the fact that all elements in the same anti-diagonal of the DP matrix can be computed independent of each other in parallel. Thus, the basic idea is to compute the DP matrix in anti-diagonal order. The anti-diagonals are stored as textures in the texture memory. Fragment programs are then used to implement the arithmetic operations specified by the recurrence relation.

Assuming, we are aligning two sequences of length M and K with affine gap penalties on a GPU. As a preprocessing step both sequences and the substitution matrix are loaded into the texture memory. We are then computing the DP matrix in $M+K-1$ rendering passes. In rendering pass k, $1 \le k \le M+K-1$, the values $H(i,j)$, $E(i,j)$, and $F(i,j)$ for all i, j with $1 \le i \le M$, $1 \le j \le K$ and $k = i+j-1$ are computed by the fragment processors. The new anti-diagonal is stored in the texture memory as a texture. The subsequent rendering pass then reads the two previous anti-diagonals from this memory.

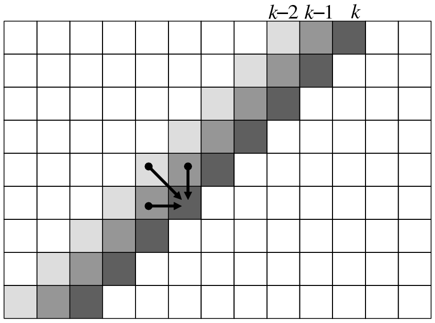

Figure 4: Data dependency relationship in the Smith-Waterman DP matrix: each cell (*i,j*) depends on its left neighbor (*i,j*–1), upper neighbor (*i*–1,*j*) and upper left neighbor (*i*–1,*j*–1). Therefore all cells along anti-diagonal *k* can be computed in parallel from the anti-diagonals *k*–1 and *k*–2.

Since diagonal *k* depends on the diagonals *k*–1 and *k*–2, we store these three diagonals as separate buffers. We are using a cyclic method to change the buffer function as follows: Diagonals *k*–1 and *k*–2 are in the form of texture input and diagonal *k* is the framebuffer. In the subsequent iteration, *k* becomes *k*–1, *k*–1 becomes *k*–2, and *k*–2 becomes *k*. This is further illustrated in Figure 5. An arrow pointing towards the fragment program means that the buffer is used as texture. An arrow pointing from the fragment program to a buffer means that the buffer is used as framebuffer.

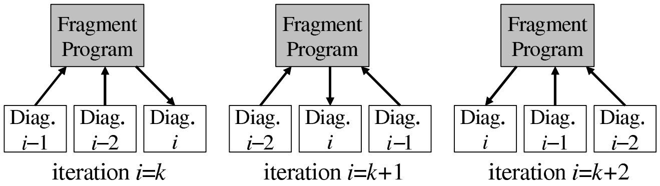

Figure 5: Cyclic change of the functions of buffers A, B, and C for computation of anti-diagonals in the DP matrix.

Note that the purpose of our Smith-Waterman GPU implementation is the acceleration of sequence database scanning. This application requires the alignment of a query sequences to all subject sequences of a given database. All subject sequences are ranked by the maximum score in the DP matrix. Reconstruction of the actual alignment (the traceback) is merely performed for the highest ranked sequences. Therefore, only the highest score of each pairwise alignment is computed on the GPU. Ranking the compared sequences and reconstructing the alignments are carried out by the front end PC. Because this last operation is only performed for very few subject sequences, its computation time is negligible.

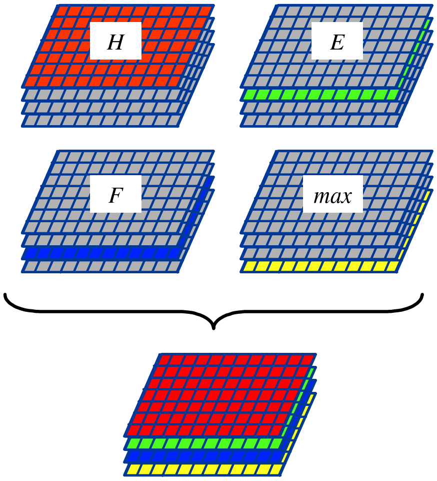

Figure 6: Using the RGBA channels of two-dimensional texture buffers for the computation of H, E, F and max.

In our GPU algorithm the maximum computation of the matrix *H* is incorporated as follows: The value $max\{H(i,j),H(i-1,j),H(i,j-1)\}$ is calculated for each cell and stored in the A-channel of an RGBA-color pixel. The R-, G-, and B-channel are used for the computation of $H(i,j)$, $E(i,j)$ and $F(i,j)$ respectively (see Figure 6).

Furthermore, it is possible to perform *N* pairwise comparisons at the same time by using two-dimensional buffers instead of one-dimensional buffers. This is shown in Figure 7 in which the buffer is filled from bottom up. Each buffer contains *N* diagonals of length $L(k)$, where $L(k)$ denotes the length of diagonal *k*. The computation is invoked by drawing an $N \times L(k)$ quad in each iteration.

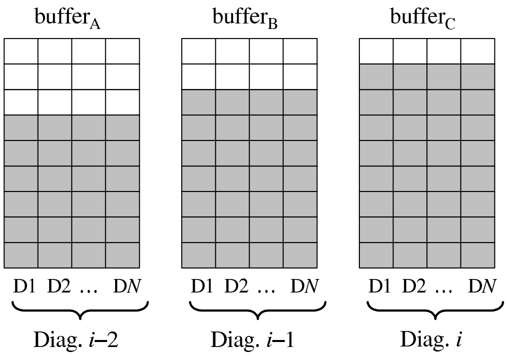

Figure 7: Buffers at iteration step *i*=8 after rendering *N* diagonals of length 8 each in parallel (in different fragment processors).

The 1:1 mapped texture coordinates can be chosen in such a manner that no computation of texture coordinates is necessary in the shader program (see Figure 8). Due to this fact, the sampled values can be prefetched. The adapted texture mapping is solved by a cell count offset to the drawn quad whereby the quad size is identical. The offset depends on the lengths of the two sequences to be aligned (M and K). If $M<K$, the diagonals with equal lengths are on the same height after the matrix shifting, otherwise they are one cell apart each. After all passes have been rendered, the results can be found in row $min\{M,K\}$ of the buffer.

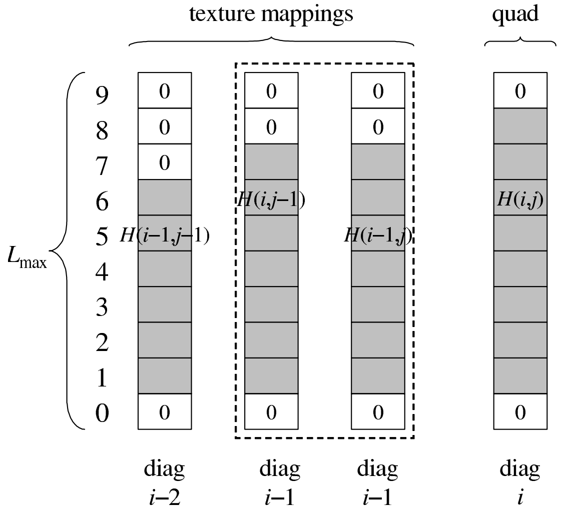

Figure 8: Texture mappings at iteration step i=8. Diagonal i–1 is mapped with two texture coordinates.

6. Performance Evaluation

We have implemented the proposed algorithm using the high-level GPU programming language *GLSL* (OpenGL Shading Language) [6] and evaluated it on the following two graphics cards:

- *nVidia GeForce 6800 GT*: 350 MHz engine clock speed, 500MHz memory clock speed, 6 vertex processors, 16 fragment processors, 256 MB memory
- *ATI Radeon X850XT*: 520 MHz engine clock speed, 540 memory clock speed, 6 vertex processors, 16 fragment processors, 256 MB memory

Tests have been conducted with these cards installed in a PC with an Intel Pentiun4 2.6GHz, 2GB RAM running Windows XP.

A performance measure commonly used in computational biology is *cell updates per second* (*CUPS*). A CUPS represents the time for a complete computation of one entry of the matrix H, including all comparisons, additions and maxima computations. Table 3 show the CUPS performance of the Radeon X850XT and the GeForce 6800 GT for various sequence lengths using affine gap penalties. This hardware allows handling query sequences up to a length of 2047 and aligning them to a set of 2048 subject sequences each of length up to 2048. This restriction is imposed by the maximum texture buffer size of these graphics cards. However, this limitation is not severe since 99.5% of the sequences in the Swiss-Prot database are of length <2048. Furthermore, it is expected that the texture buffer sizes will increase in next-generation graphics hardware.

From Table 3 it can be observed that the performance improves when the ratio of query length and subject length is close to one. Such a ratio provides the best balance of data transfer (framebuffer to texture memory) and computation for the used GPUs. Consequently, the best performance values for each column are exactly on (or close to) the main diagonal in Table 3 (bold numbers).

Since CUPS does not consider data transfer time, varying subject sequence lengths and initialization time, it is often a weak measure that does not reflect the behavior of the complete system. Therefore, we will use database scans for different query lengths in our evaluation. Table 4 reports the performance for scanning the Swiss-Prot protein databank (release 46.3, which contains 176'469 sequences with an average length of 361) for query sequences of various lengths using our implementation on a Radeon X850XT and GeForce 6800 GT.

Table 3: Performance (in Mega CUPS) for various query/subject sequence lengths combinations of our implementation (for affine gap penalties) on an Radeon X850XT (left) and GeForce 6800 GT (right).

		Subject lengths											
		63		127		255		511		1023		2047	
Query lengths	63	176	131	176	151	141	125	154	135	106	98	66	62
	127	212	176	212	177	178	166	196	184	157	144	110	100
	255	192	165	238	204	209	199	233	212	204	187	166	144
	511	153	143	199	182	225	214	246	238	231	219	198	184
	1023	110	100	159	145	202	188	232	221	257	242	234	215
	2047	69	64	113	103	160	149	199	192	232	223	250	235

For the same application an optimized C-program on a Pentium IV 2.6 GHz has a performance of 50 MCUPS for. Hence, our GPU implementation achieves a speedup between 2.1 and 4.4 depending on the query length.

Table 4: Performance (in MCUPS) for scanning the Swiss-Prot database (release 46.3, March 2005) on different graphics cards. Speedup compared to a Pentium4 2.6GHz is also reported.

Query length	NVidia GeForce 6800 GT	ATI Radeon X850XT
63	104 (2.1)	113 (2.3)
127	145 (2.9)	160 (3.2)
255	180 (3.6)	196 (3.9)
361	189 (3.8)	196 (3.9)
511	202 (4.0)	219 (4.4)
1023	196 (3.9)	208 (4.2)
2047	166 (3.3)	173 (3.5)

7. Conclusions and Future Work

In this paper we have demonstrated that the streaming architecture of GPUs can be efficiently used for biological sequence analysis. To derive an efficient mapping onto this type of architecture, we have reformulated the Smith-Waterman algorithm in terms of computer graphics primitives. The evaluation of our implementation on two off-the-shelf graphics cards shows a speedup of approximately four compared to a Pentium IV 2.6GHz. To our knowledge this is the first reported implementation of biological sequence alignment on graphics hardware.

The very rapid growth of genomic databases demands even more powerful parallel solutions in the future. Because comparison and alignment algorithms that are favored by biologists are not fixed, programmable parallel solutions are required to speed up these tasks. As an alternative to inflexible special-purpose systems, hard-to-upgrade SIMD systems, and expensive supercomputers, we advocate the use of graphics hardware platforms. The main advantage of graphics hardware compared to previously used architectures for biological sequence analysis is that they are commodity components. This facilitates easy upgrading to next-generation GPUs at a very reasonable price.

Our future work includes extending our described mapping to database scanning with hidden Markov Models using the Viterbi algorithm and making our implementation available as a special resource in a computational grid.

References

[1] Borah, M., Bajwa, R.S., Hannenhalli, S., Irwin, M.J.: A SIMD solution to the sequence comparison problem on the MGAP, *in Proc. ASAP'94*, IEEE CS (1994) 144-160.

[2] Chow, E., Hunkapiller, T., Peterson, J., Waterman, M.S.: Biological Information Signal Processor, *Proc. ASAP'91*, IEEE CS (1991) 144-160.

[3] Di Blas, A. et. al: The UCSC Kestrel Parallel Processor, *IEEE Transactions on Parallel and Distributed Systems* 16 (1) (2005) 80-92.

[4] Fernando R: EuroGraphics 2004 Tutorial 5: Programming Graphics Hardware – High-Level Shading Languages, http://download.nvidia.com/developer/presentations/2004/Eurographics/EG_04_ProgrammingLanguages.pdf (2004)

[5] Guerdoux-Jamet, P., Lavenier, D.: SAMBA: hardware accelerator for biological sequence comparison, *CABIOS* 12 (6) (1997) 609-615.

[6] Kessenich J., Baldwin D., Rost R.: The OpenGL Shading Language, *Document Revision 59*, http://www.opengl.org/documentation/oglsl.html (2005)

[7] Krüger J, Westermann R: Linear algebra operators for GPU implementation of numerical algorithms, *ACM Trans. Graph*. 22, 908-916, (2003).

[8] Lopresti, D.P.: P-NAC: A systolic array for comparing nucleic acid sequences, *Computer* 20 (7) (1987) 98-99.

[9] Mark W.R., Glanville R.S., Akeley K., Kilgard M.J.: Cg: A system for programming graphics hardware in a C-like language. *ACM Trans Graph* 22, 896-907, (2003).

[10] Moreland K, Angel E: The FFT on a GPU, Proceedings SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003, 112–119 (2003).

[11] Oliver, T., Schmidt, B., Maskell, D.: Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs, 13th ACM International Symposium on Field-Programmable Gate Arrays, (FPGA 2005), Monterrey, CA

[12] Purcell T.J., Buck I., Mark W.R., Hanrahan P.: Ray tracing on programmable graphics hardware, *ACM Trans Graph*, 703-712, (2002).

[13] Schmidt, B., Schröder, H., Schimmler, M: Massively Parallel Solutions for Molecular Sequence Analysis, *Proc. 1st IEEE Int. Workshop on High Performance Computational Biology,* Ft. Lauderdale, FL (2002).

[14] Singh, R.K. et al.: BIOSCAN: a network sharable computational resource for searching biosequence databases, *CABIOS*, 12 (3) (1996) 191-196.

[15] Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences*, Journal of Molecular Biology* 147 (1981) 195-197

[16] Xu, F., Mueller, K.: Ultra-fast 3D filtered backprojection on commodity graphics hardware," *IEEE International Symposium on Biomedical Imaging'04*, Washington D.C., (2004)

[17] Yamaguchi, Y., Maruyama, T., Konagaya, A.: High Speed Homology Search with FPGAs, *Proc.PSB'02* (2002) 271-282.