Scalability Studies of the BLASTn Scan and Ungapped Extension Functions

Siddhartha Datta Ron Sass
Reconfigurable Computing System's Lab
Electrical & Computer Engineering Department
University of North Carolina at Charlotte
9201 University Blvd., Charlotte, NC 28223
{skdatta,rsass}@uncc.edu

Abstract

BLASTn is a ubiquitous and important tool used for large scale DNA analysis. As such, it is a good candidate for acceleration with FPGAs. The aim of this paper is two-fold. First, building upon our prior BLAST work we describe a design composed of multiple cores that can be scaled in two dimensions. The ungapped extension and a second dimension are new in this work. Second, we use this non-trivial example to explore spatially scalable designs. To provide the ability to move the design to a future generation chip, a mathematical model of performance that incorporates all of the system design parameters and the user's preference (high throughput vs low latency) is developed. We demonstrate here that the model correctly predicts the optimal ratio between the two dimensions on a Xilinx Virtex-4 and measures four to five times faster performance figures as compared to a state of the art general purpose processor.

1. Introduction

For the last two decades, applications have benefited from frequency scaling. Every new generation of ICs offer both more transistors and faster clock rates. Increasing the clock rate causes every sequential application to execute faster. Even though it is expected that the number of transistors will continue to increase at a dramatic rate, the system clock frequency in newer devices will grow much slower than in the past. Rather than relying on gains in frequency, future applications will need to be designed such that they are spatially scalable to utilize the additional transistors that each new technology generation provides. This is especially true of FPGA-based applications. While many designs consist of simple cores that can be easily replicated, this does not always result in a spatially scalable design whose performance grows linearly.

BLASTn is an excellent example of this phenomenon. Running on a single core, this critical bioinformatics application is compute-bound. However, the application quickly becomes I/O-bound by increasing the number of computational cores. This is because each instance needs a large look-up table that is stored in main memory and all the cores share the same I/O subsystem to read the databases. Thus, multiple cores are contending for space in the cache, a portion of the fixed bandwidth to memory, and bandwidth to the I/O subsystem. An FPGA core that accelerates BLASTn cannot be arbitrarily replicated because the off-chip bandwidth is fixed and, ultimately, limits performance.

This paper describes a parametrized implementation of NCBI BLASTn for FPGA devices. The goal is two-fold: first, prior work with a BLASTn core is extended and then this new core is used to explore spatially scalable designs. Specifically, the BLAST implementation described here adds another dimension to our prior work [1] that reduces latency (as well as improves throughput). This new core also introduces ungapped extensions to the basic hardware core — a function that was previously implemented in software. Using this new core, a spatially scalable design has been implemented. Based on the number of BLAST cores, their organization, and the off-chip memory bandwidth, this paper describes a mathematical formulation of the design's performance. This allowed us to develop a novel algorithm that — given the characteristics of an FPGA device, the board's memory bandwidth, and user preferences — calculates the optimal parameters for the design. This approach highlights a new level of portable scalability: as larger chips become available and I/O technology evolves, one simply enters the

This project was supported in part by the National Science Foundation under NSF Grants CNS 06-52468 (EHS) and CNS 04-10790 (CRI). The opinions expressed are those of the authors and not necessarily those of the Foundation.

978-0-7695-3917-1/09 $26.00 © 2009 IEEE
DOI 10.1109/ReConFig.2009.60

characteristics of a new system and the algorithm determines the design's generic parameters.

2. Background and Related Work

The BLAST application was originally developed at the National Center for Biotechnology Information (NCBI) [2], [3]. Development continued at Washington University, but the NCBI code has become the *de facto* standard. BLAST, which is actually a collection of programs, is used to compare an unknown genomic sequence, called a *query*, against an existing *subject* genomic database to identify high similarity regions. There are five "flavors" of BLAST that have been tuned to the nature of the subject database and query. These are: BLASTn, BLASTp, BLASTx, TBLASTn, TBLASTx. BLASTn is the program used to make local searches between a nucleotide based query and a nucleotide subject database. There are four different letters (*monomers*) in nucleotide sequences: A *(Adenine)*, C *(Cytosine)*, G *(Guanine)* and T *(Thymine)*. Each of these can be encoded in 2 bits.

The BLASTn algorithm is divided into four major steps. The first step involves creation of the query lookup table and overflow table. The query is examined once and all eight lettered words (called *w-mers*) used to construct a $2^{8\times2}$ entry lookup table. The offset(s) into the query for each of these words are stored in the lookup table. into a 16 bit hexadecimal number which is query lookup table where its offset is stored. If a certain word is repeated in the query, a pointer in the lookup table is updated and the repeated word's offset is put in an overflow table. Since BLASTn is a similarity search rather than exact matching, the software limits the maximum number of repetitions to eight. However, in practice the number of distinct 8-letter words appearing in a query is overwhelmingly zero or one. (We use this characteristic to define a very efficient application-specific cache in the next section.)

The second step (called the scan stage) uses the lookup table to identify exact matches of 8-letter words in the database and the query sequence. These exact matches are called *hits* and their score value is eight. The stride length is a command-line parameter that is typically set to four letters (eight bits). This defines the rate at which the database is shifted each lookup. (This part of the scan stage has been observed to take $\approx$ 78% of the total execution time and has been the focus of the previous hardware core.) Next, the offsets corresponding to hits in the previous step are sent to the ungapped extension step where each hit is extended to the left and right by the stride length number of letters. Their scores are incremented by one for every exact match found. All hits that score greater than a threshold score set by the user are sent to the gapped extension stage. Depending upon the user's selection, gapped extension is done using either the Needleman-Wunsch algorithm [4] or the Smith-Waterman algorithm [5]. (The ungapped extension typically consumes another 12% to 16% of the total execution time and is included in the hardware core described in this paper.) The complexity of the algorithm is:

$$O(N_{Seq} \times (N_H + N_G)) \quad (1)$$
$$\text{where } N_{Seq} \gg N_H \gg N_G$$

where N_{Seq} is the number of sequences in the database, N_H is the number of hits found during the scan stage and N_G is the number of hits sent for gapped extension.

Biological sequence searching using FPGAs and systolic arrays [6]–[8] has been an active area of research for over a decade. A lot of research has been done in implementing the Smith-Waterman [5], [9] and Needleman-Wunsch [4] algorithm over hardware platforms to enhance the performance of search algorithms. The bottleneck for BLASTn resides in the first stage where the hits are located. This issue has been addressed by Mercury BLAST [10]. More work has also been done in implementing the entire application on the FPGA like Tree-BLAST [11] and the TUC implementation [12]. Another interesting design was proposed by [13] to port the entire application on chip. RCBLAST [14] proposed an index based searching scheme for BLASTn. Commercial companies, like Time Logic [15], report high speedup numbers but provide little detail about their designs. Recently, the IBM Cell broadband Engine [16] has also been used to speedup BLASTn.

All the previous implementations have shown significant speedups compared to software; however, most do so by compromising fidelity. Mercury BLASTn [10] claims 98% to 99% accuracy in results. Tree-BLAST [11] reports extra alignments (although reporting the same *E-values*) and [17] considers two repetitions of any word in the query, instead of eight. While one could argue that BLAST is a heuristic and 100% compatibility with NCBI BLAST is unnecessary, it is difficult to convince biologists of the same. The typical user would have no idea whether the differences are statistically significant. Thus, in addition to "better performance", complete consistency with NCBI BLAST is important.

3. Design

The goal of this design is two-fold. First, we augment our previous high-throughput design [1] to execute ungapped extension. Second, we build a

bandwidth-aware scalable design that allows us to mathematically relate the bandwidth requirements to the size of various generics. Hardware implementations are based on the current 2.2.20 version of NCBI BLAST software.

3.1. Modified Lookup and Overflow Table

The inclusion of ungapped extension to the scan hardware core required modifications to be made to the query lookup and overflow tables. Four letters, each to the right, left and their count in the query are grouped into 32 bit words and written to the offset's next element in the query lookup table, called the Left/Right Data. If there is no offset written to a location, its value and the Left/Right Data's value is set to -1. The same procedure is repeated while writing to the overflow table. Hence, the size of each table is doubled to $2^{16} \times 64$ bits as shown in Figure 1(a). These modifications are made to eliminate accesses to main memory during ungapped extension of the hits. This process increases the latency of the software by one clock cycle per offset written to the tables, but is insignificant compared to the total execution time of the entire algorithm.

3.2. Scan Ungapped Extension Core

The scan ungapped extension core is the hardware implementation of the `blast_nascan` and the `nt_word_finder` functions. The core has six main components: (1) a subject data FIFO, (2) a scan finite state machine, (3) an ungapped extension and bus master finite state machine, (4) two Hit Index Tables, (5) four lookup queues and (6) two output FIFOs (offset and score). The general setup of the core is shown in Figure 1(b). The subject database is buffered into the subject data FIFO from the DDR2 through a direct channel (Native Port Interface) of the Multi Port Memory Controller (MPMC). The size of this FIFO is set to 512×64 bits. The Hit Index Tables are dual port BRAMs of size 2048×32 bits. The first bit of each offset element in the query lookup table is grouped into 32 bit words and written by the software running on the PowerPC-405 to the Hit Index Tables via the Processor Local Bus (PLB). This table facilitates the hit check process locally as opposed to multiple reads from DDR2 for every word in the subject database, which is expensive in terms of latency. A detailed study of the benefits of the Hit Index Table is listed in [1]. The scan finite state machine's operations are (1) pop four bytes of data from the subject FIFO (2) increment the subject offset counter (3) calculate addresses in the Hit Index Tables and (4) make hit checks every clock cycle. The pipelined architecture of the scan finite state machine ensures all these steps are completed in one clock cycle. Lookup queues are created to store the matched subject database word, matched database word's offset and the subject data's Left/Right Data. Each of the four lookup queues are dedicated to each hit check made every clock cycle, in case more than one hit is identified simultaneously. The size of each lookup queue is set to 16×64 bits.

The ungapped extension and bus master finite state machine arbitrates across the lookup queues in a round robin technique. If a lookup queue is not empty, the state machine pops its data element and issues a read request across the PLB to fetch the query offset and Left/Right Data from the query lookup table stored in the DDR2. If the data fetched is a pointer to the overflow table, sixteen elements (eight query offsets and eight corresponding Left/Right Datum) are fetched. Calculation of the ungapped extension score takes one clock cycle to execute, after the Left/Right Data is read by the core. This score is calculated as the subject and query offsets are written to the offset FIFO. On the subsequent clock cycle, the ungapped extension score is written to the score FIFO. The sizes of the output and score FIFOs are set to 256×32 bits.

The total time taken to execute the scan function and the ungapped extension function in hardware is denoted by T_{HW}. T_{SHW} denotes the time taken by the core to execute the scan function for a sequence of the database of length S_{Len}. Due to the pipeline in the scan finite state machine, during the first clock cycle (T_F), the first subject data element is popped. During the second clock cycle (T_{HIT}), the Hit Index Table addresses are calculated. The third clock cycle onwards, all processes execute every clock cycle.

$$\max(T_{SHW}) = T_F + T_{HIT} + (S_{Len}/4) \tag{2}$$

T_I is the time taken to issue a read request to the PLB and wait for an acknowledge, measured to be seven clock cycles. T_{FLT} is the time taken to read query offset and Left/Right Data from the query lookup table. T_{FOT} is the time taken to read eight overflow query offsets and their corresponding Left/Right Data. T_{UHW} is the time taken to fetch the query offsets from the DDR2 and compute the ungapped extension score.

$$\begin{aligned} \max(T_{UHW}) &= (2T_I) + T_{FLT} + T_{FOT} = 48 \\ \therefore \max(T_{HW}) &= (N_{Seq}((S_{Len}/4) + 48)) \end{aligned} \tag{3}$$

Since the ungapped extension is run during the scan in hardware, the complexity of the algorithm is reduced to $O(N_{Seq} \times (N_G))$.

The performance of the software execution of NCBI `blast_nascan` and `nt_word_finder` functions

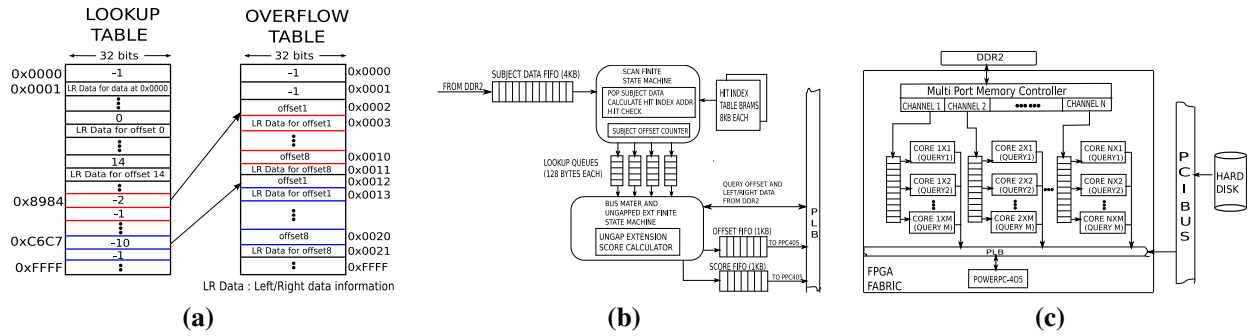


Figure 1: (a) Modified Lookup and Overflow tables (b) Scan and Ungapped Extension core (c) High-Level Design

Table 1: Time taken to execute the Scan and Ungapped Extension functions for sequences of env_nt.nt (3GB) Database

Database Sequence Length	Hardware Core on FPGA	AMD Opteron (2GHz) One Core	Number of hits
638 Bytes	112μs	121μs	41
6378 Bytes	573μs	750μs	399
9138 Bytes	613μs	1639μs	743
13359 Bytes	583μs	1149μs	590
14480 Bytes	631μs	1987μs	779

run on a 2GHz AMD Opteron (single core) with 128KB L1 cache, 1MB L2 cache and 8GB DDR2 are as follows: T_{SSW} is the time taken to run scan function in software. T_L is the time taken to update the `for` loop and incrementing the subject offset counter. T_S is the time taken to shift every byte of the subject database to the index register which is compared to the query lookup table for locating hits. Let T_{HC} be the time taken to make a hit check and update the offset data structures in case of a hit. These instructions are measured to take four clock cycles. All cases assume 100% L1 cache hits and the entire database sequence and query is pre-buffered into the L2 cache.

$$T_{SSW} = (S_{Len}(T_L + T_S + T_{HC})) = 4S_{Len} \quad (4)$$

T_U is the time taken to calculate ungapped extension scores. T_{UL} is the time to make loop comparisons over the number of hits found. T_{SA} and T_{QA} is the time taken to access four letters to the left and right of the hit in the subject database and query respectively. Each of these instructions take one clock cycle to execute. T_C is the time taken to make eight comparisons and update the score. NH is the total number of hits found in the scan function.

$$T_U = (NH(T_{UL} + T_{SA} + T_{QA} + T_C)) = 13NH \quad (5)$$

$$\therefore \min(T_{SW}) = N_{Seq}(13NH + 4S_{Len}) \quad (6)$$

Since, the AMD Opteron is clocked at 2GHz and the FPGA core at 100MHz, from equations 3 and 6:

$$\therefore \min(Speedup) \propto (2.6NH + 0.8S_{Len}) \quad (7)$$

According to 7, the implementation of the scan and ungapped extension of the FPGA is theoretically about $1.2 - 1.5$ times faster and the rate increases with the increase in number of hits. Table 1 lists the experiments' results which validates this point.

3.3. Scalable Design Model

In [1], various queries' hit index tables were loaded into multiple cores connected across a single direct channel of the Multi Port Memory Controller. This resulted in incomplete utilization of all its ports, but high throughput due to hits located across several queries in parallel. The latency of the system is inversely proportional to its throughput. Sequences of the database streamed over multiple channels from the DDR2 in parallel results in low latency but the number of cores loaded with different queries' Hit Index Tables is reduced, reducing the throughput. For a XC4VFX60 FPGA, the channel's bandwidth (ch_bw)is measured to be 1.6GB/s when data is read continuously over one channel (continuous mode). However, when reads occur simultaneously over multiple channels, the burst mode bandwidth is measured to be 0.98GB/s. With two (n) Hit Index Tables in the core, data is consumed at a rate of 0.4 GB/s. The mathematical model for designing an optimal ($N \times M$) grid of cores with N channels for streaming in the sequences of the database over M queries for any platform FPGA is as follows:

$$N = \begin{cases} 1 & - \text{ max throughput} \\ num\_ch & - \text{ min latency} \\ \left\lfloor \frac{ch\_bw}{0.2n} \right\rfloor & - \text{ optimal performance} \end{cases} \quad (8)$$

where num_ch is the total number of channels available. Each Hit Index Table utilizes four 18Kb BRAM blocks. Each lookup queue and output FIFO utilizes one 18Kb BRAM block each. The subject

FIFO utilizes two 18Kb BRAM blocks. For adding every query, four lookup queues, two output FIFOs and n number of Hit Index Tables are added to the system. For every channel, one subject FIFO is added to the system. The DDR2 wrapper consumes twenty-one 18Kb BRAM blocks and eight 18Kb BRAM blocks for every channel added to the system. A 32KB and 64KB cache for the PPC consumes sixteen and thirty-two 18Kb BRAM blocks respectively. The trimode macc FIFO and pci bridge consume two 18Kb BRAM blocks each. Let B be the total number of 18Kb BRAM blocks available on the chip.

$$M = \begin{cases} \left\lfloor \dfrac{B-(10N+41)}{(4n+6)N} \right\rfloor - \text{32KB Cache} \\ \\ \left\lfloor \dfrac{B-(10N+57)}{(4n+6)N} \right\rfloor - \text{64KB Cache} \end{cases} \quad (9)$$

Hence, using equations 8 and 9, depending upon any system's specifications, its optimal $N \times M$ grid size can be determined. Figure 1(c) demonstrates the layout of the system.

4. Experimental Setup and Evaluation

To evaluate the effectiveness of the proposed technique, a set of experiments was designed to test the absolute rate of computation and scalability of the FPGA accelerated implementation. A Xilinx ml410 evaluation board embedded with the XC4VFX60 FPGA is used for the tests. Two Powerpc-405 processors are fabricated on the FPGA. Linux 2.6.26 is run on one of the processors at 300MHz. The BLASTn executables, formatted databases and formatted queries are stored in a WD1600AAJS 160GB hard disk. The interface to the hard disk is through PCI and the PCI-to-PLB bridge (Figure 1(c)). Four configurations of cores (1×12 , 2×6 , 3×4 and 5×2) on the Virtex-4 were successfully synthesized and tested with the BLASTn algorithm running on Linux. All the four configurations' synthesis reports show 99% on-chip memory and 98% slice utilization. 12 queries of various sizes, ranging from 0.77KBases to 136KBases, were searched over the `env.nt` and `env.nr` databases, downloaded from the NCBI website. For performance comparison purposes, these tests were run on a server with 2 dual core AMD Opterons (2GHz) with 128KB L1 cache, 1MB L2 cache, 8GB DDR2, 1TB HDD (SATA interface) and two blades of TimeLogic's Decypher machine.

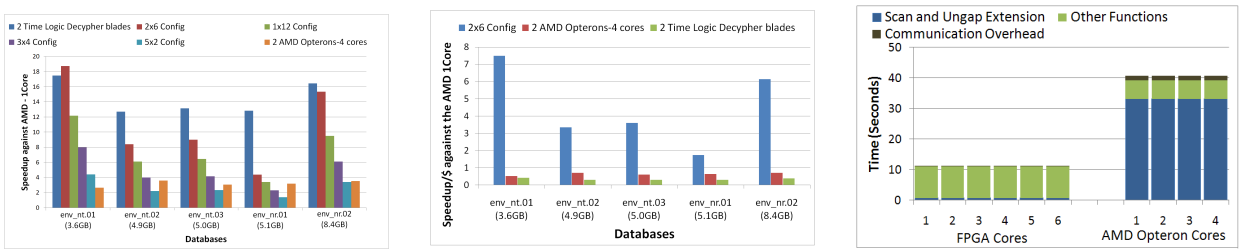
The results of the tests are shown in Figure 2(a). The bar chart indicates speedups measured with different configurations on the FPGA, the AMD Opteron server and the Decypher machine over a single core's performance of the AMD Opteron server. The best performance on the FPGA was achieved for the 2×6 configuration of cores, validating our mathematical model (equations 8, 9) described in section 3. The Decypher machine though is measured to be about twice as fast as the 2×6 configuration of cores on the FPGA. The cost of the Decypher machine is quoted at $42000, the AMD Opteron server is quoted at $3000 and the Xilinx ml410 board is quoted at $2500. The performance/$ results shown in Figure 2(b) plot the ratio of speedups attained against a single core of the AMD Opteron for the same tests described in Figure 2(a). The FPGA's best implementation priced 16 times lower than the cost of the Decypher machine proves to be an inexpensive yet one of the fastest existing solutions to the BLASTn problem having complete consistency with the NCBI results.

Figure 2(c) shows the measured relations between spatial computing and their execution time taken. The scan and ungapped extension functions are measured to utilize on an average 3% and the communication overhead takes 1% of the total execution time when run on the FPGA, as opposed to an average 87% and 5% respectively on the AMD Opteron server. This indicates the compute-bound problem in software is converted to be I/O bound by increasing the number of cores and introducing parallelism in the hardware implementation. The majority of the total execution time of the algorithm is taken to read data from the hard drive to the RAM. Hence, with an increase in the number of cores, more physical area on the chip is utilized but it provides lower execution latencies.

5. Conclusion

This paper describes a BLASTn accelerator core that performs the scan and ungapped extension functions. The core, used in a spatially scalable design, uses a mathematical model of performance and a novel algorithm to determine the optimal parameters for a given FPGA system. A variety of configurations were tested and the model correctly predicted their performance of the configuration. The resulting FPGA design has been shown to be very competitive; it outperforms a typical AMD Opteron compute server in terms of throughput and performance/$.

The design described here runs on single FPGA node. However, this node is just one of 64 nodes in a high-performance computing cluster [18]. The layout of the single node system was designed so that the replicated cores could, in the future, be extended to span all of the FPGAs in this cluster. This, combined with parallel disk access and the high speed network, will change the bandwidth numbers and introduce new variables into the mathematical formulation. The next



(a) **(b)** **(c)**

Figure 2: (a) Speedup Numbers against one core of the AMD Opteron (b) Performance with respect to cost of equipment (c) Spacial Computing vs Time

step will be to extend the spatial scalability study to include this multi-FPGA platform.

References

[1] S. Datta, P. Beeraka, and R. Sass, "RC-BLASTn: Implementation and Evaluation of the BLASTn Scan Function," in *FCCM '09*, April 2009.

[2] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and psi-blast: a new generation of protein database search programs." *Nucleic Acids Res*, vol. 25, no. 17, pp. 3389–3402, September 1997.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool." *J Mol Biol*, vol. 215, pp. 403–410, October 1990.

[4] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino-acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, pp. 443–453, 1972.

[5] T. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[6] R. Hughey, "Parallel sequence comparison and alignment," *Application-Specific Systems, Architectures and Processors, IEEE International Conference on*, vol. 0, p. 137, 1995.

[7] D. Hoang, "Searching genetic databases on splash 2," in *FPGAs for Custom Computing Machines, 1993. Proceedings. IEEE Workshop*, Apr 1993, pp. 185–191.

[8] D. H. Department, D. T. Hoang, and D. P. Lopresti, "Fpga implementation of systolic sequence alignment," in *FPL*, 1992.

[9] C. W. Yu, K. H. Kwong, K. H. Lee, and P. H. W. Leong, "A smith-waterman systolic cell," in *FPL 2003*. Springer, 2003, pp. 375–384.

[10] P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, A. Jacob, and J. Lancaster, "Biosequence similarity search on the mercury system," *J. VLSI Signal Process. Syst.*, vol. 49, no. 1, pp. 101–121, 2007.

[11] M. C. Herbordt, J. Model, B. Sukhwani, Y. Gu, and T. VanCourt, "Single pass streaming blast on fpgas," *Parallel Computing*, vol. 33, no. 10-11, pp. 741–756, 2007.

[12] E. Sotiriades and A. Dollas, "A general reconfigurable architecture for the blast algorithm," *J. VLSI Signal Process. Syst.*, vol. 48, no. 3, pp. 189–208, 2007.

[13] F. Xia, Y. Dou, and J. Xu, "Families of fpga-based accelerators for blast algorithm with multi-seeds detection and parallel extension." in *Communications in Computer and Information Science*, vol. 13. Springer, 2008, pp. 43–57.

[14] K. Muriki, K. D. Underwood, and R. Sass, "Rc-blast: Towards a portable, cost-effective open source hardware implementation," in *IPDPS '05- Workshop 7*, 2005, p. 196.2.

[15] www.timelogic.com, "Time logic biocomputing solutions."

[16] A. Wirawan, K. C. Keong, and B. Schmidt, "Parallel dna sequence alignment on the cell broadband engine." in *Lecture Notes in Computer Science*, 2007, pp. 1249–1256.

[17] S. Kasap, K. Benkrid, and Y. Liu, "Design and implementation of an fpga-based core for gapped blast sequence alignmentn for the two-hit method," *Engineering Letters*, vol. 16, pp. 443–452, 2008.

[18] R. Sass, W. V. Kritikos, A. G. Schmidt, S. Beeravolu, and P. Beeraka, "Reconfigurable computing cluster (rcc) project: Investigating the feasibility of fpga-based petascale computing." in *FCCM*, 2007, pp. 127–140.