FPGA based Architecture for DNA Sequence Comparison and Database Search

Euripides Sotiriades Christos Kozanitis Apostolos Dollas

Microprocessor and Hardware Laboratory
Technical University of Crete
Chania 73100 Greece
{esot,kozanit,dollas@mhl.tuc.gr}

Abstract

DNA sequence comparison is a computationally intensive problem, known widely since the competition for human DNA decryption. Database search for DNA sequence comparison is of great value to computational biologists. Several algorithms have been developed and implemented to solve this problem efficiently, but from a user base point of view the BLAST algorithm is the most widely used one. In this paper we present a new architecture for the BLAST algorithm. The new architecture was fully designed, placed and routed. The post place-and-route cycle-accurate simulation, accounting for the I/O, shows a better performance than a cluster of workstations running highly optimized code over identical datasets. The new architecture and detailed performance results are presented in this paper.

1. Introduction

When performing DNA Sequence Comparison, biologists are not interested in the exact match of the sequences but they are interested in the degree of similarity between them. For that reason all the algorithms that have been developed are based on similarity scoring. First efforts during the early 70s focused on finding the optimal matching between two DNA sequences [1]. Dynamic programming methods were selected due to the nature of the problem, which results in huge datasets. Needlamn and Wunsch [8] developed their algorithm which uses a dynamic programming approach in order to obtain an optimal global alignment of two sequences. Biologists know that when there is a need for a comparison between two long DNA sequences, global similarity will be very poor even though there are sub sequences with strong, meaningful similarities. Therefore, local alignment is more appropriate to locate these sections and global alignment misleads. Several dynamic programming algorithms for local alignment have been developed, such as Smith Waterman[7], FASTA[6] and BLAST[5]. The Smith Waterman algorithm provides optimal solutions and is able to recognize distantly related sequences and its complexity is O(mn), where m is the database size is and n is the query size. However, FASTA originally, and subsequently BLAST, which was developed as an improvement of FASTA, both use heuristic methods to provide near optimal solution based on their ability to discard sequences that are not related to the query sequence. It may be public wisdom, but there is no mathematical proof that BLAST is faster than FASTA but it's a fact that BLAST based programs are the most widely used sequence comparison codes in the area of computational biology. The success of BLAST is owed to it being open source software, with NCBI support and continuous development. For that reason, the acceleration of BLAST through custom hardware architectures seems an appropriate first step in the speedup of DNA sequence comparison, and this is the purpose of this work. Historically, DNA string matching is one of the first applications of reconfigurable computing [2, 3].

1.1 The BLAST Algorithm

BLAST is the acronym for Basic Local Alignment Search Tool and was first presented in [5]. A family of implementations has been developed, depending on the nature of data to be processed (nucleotides have a 4 letter alphabet, amino acids have a 20 letter alphabet, and there may be cases of both in a search.). BLAST is used for searching large genetic databases to find areas of high similarity (matches) between the data base and an input query. For example, it compares a nucleotide query against a nucleotide sequence database. Depending on query and database data each BLAST implementation is named BLASTp when the query is an amino acid and the database is a protein, BLASTn when both the query and the database are nucleotides, BLASTx when the query is nucleotide translated and the database is protein,

1-4244-0054-6/06/$20.00

tBLASTn when the query is an amino acid and the database is a nucleotide translated, and finally tBLASTx when the query and the database are nucleotides translated. The inputs of the algorithm are the genetic sequence database (or a part of it such as the human genome) and a query which tries to find areas of similarity in the database. Outputs of the algorithm are the positions of the areas of these two strings that have similarity, as well the score of these similarities. Each of these pairs, comprising of a database area and a query area, is called a High Score Pair (HSP). The score has significant value for biologists because it is used to compute several variables, the most important of which is the e-value.

The algorithm consists of three steps. In the *first step* the query is compiled to form a list of length w substrings. These substrings are called W-mers and are all the contiguous substrings of length w of the query sequence. Let ATGAACCTGAATACTGGGTTACCT be the query DNA sequence of length 24 and let w, the length of W-mers, be equal to 8. The word list will contain 17 W-mers.

ATGAACCT will be the first

TGAACCTG will be the second

GAACCTGA will be the third etc. and

GGTTACCT will be the last one.

The *Second step* is the search of the database for "hits". After the word list generation, the database sequences are searched for an **exact** match between any substring of the W-mers list and the database sequence. Every word of the word list found in the database is called hit and it is possible to be part of a High Score Pair (HSP).

The list of the generated "hits" is processed in the *third step*. Each substring which generated a match in the second step is extended locally in both directions as long as the score of this substring no longer gets improved following the scoring rules.

The scoring scheme of the algorithm is based on the PAM (Point Accepted Mutations) matrices which examine which amino-acid "substitutions" (i.e. mismatches) are evolutionary accepted. However, in this project we deal with the implementation of the algorithm in which both the query and the database consist only of nucleotide sequences and we use a simpler scheme where each match is scored with 5 and any mismatch is scored with -4 (almost all standard literature uses this scheme). With this scheme we may have answers slightly different than the use of the PAM matrices but biologically it does not matter.

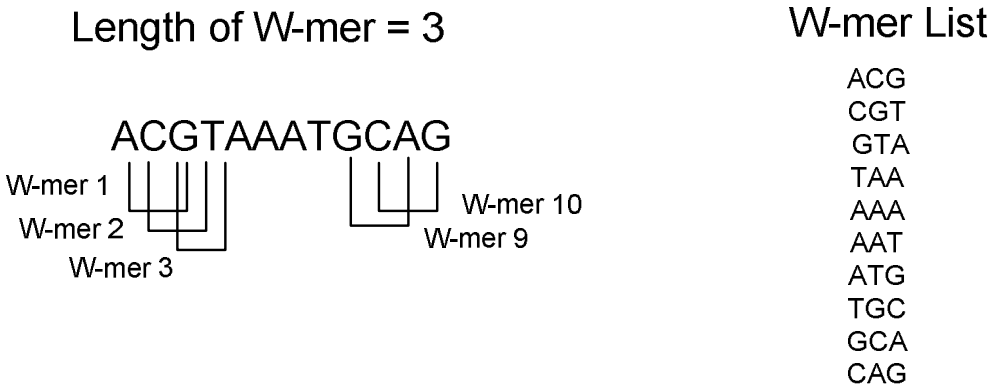

Figure 1 BLAST Algorithm Step 1

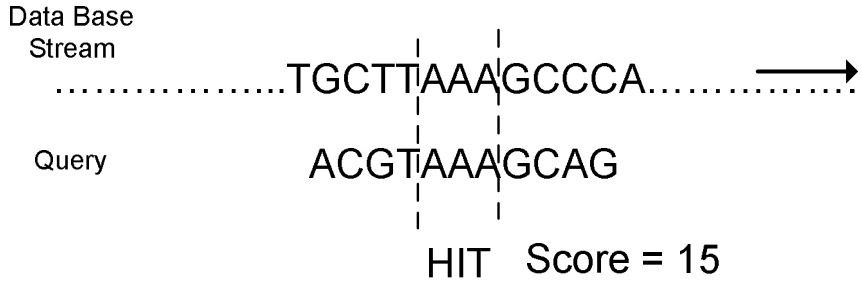

Figure 2 BLAST Algorithm Step 2

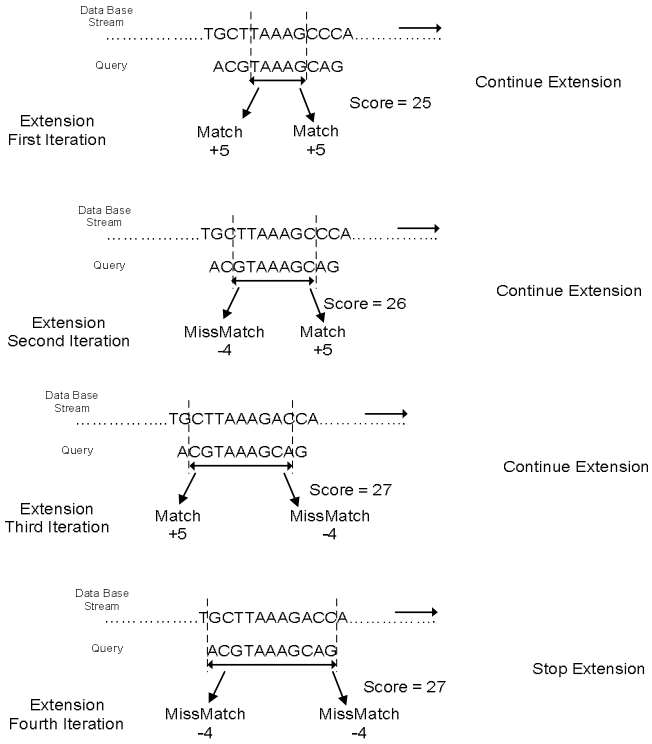

Figure 3 BLAST Algorithm Step 3

2. Sequence Comparison in Large Sequence Databanks: Implementations to Date

The genome of many organisms has been sequenced to date but the process is due to be done for many others. Large scale sequencing follows computer technology progress. One of the results of this procedure is the creation of large centralized databanks that store large quantities of genomic data. These databanks are categorized according to their contents (DNA or proteins). Major databanks are NCBI [18] which maintains GenBank that stores all DNA sequences that are made in public; EMBL [9] which is a large DNA archive in Europe. Important DNA archives are kept in DDBJ [10], and GSDB [11]. Regarding protein archives, PIR [12] in the USA and Swiss-Prot[13] in Europe are the most important databases.

Biologists consider BLAST based tools as one of the most important in computational molecular biology. For that reason several implementations and improvements of the original algorithm have been implemented and applied to all these huge datasets. NCBI is considered as the institute where BLAST development started and is mostly done to date. It maintains and updates GenBank with sequences of more than 100 billion bases (characters). The BLAST software that was produced at NCBI has been used as benchmark for computing systems such as IBM 375 MHz POWER3-II multiprocessor (SMP) and the 1.1 GHz POWER4 pSeries 690 Model 681, which according to published results [19] is the fastest system for BLAST.

2.1 Hardware Efforts for Sequence Comparison

Sequence comparison in Large Sequence databanks was one of the first applications for FPGAs. D. Hoang et. al. [2], [3] implemented the Needleman-Wunsch and dynamic programming algorithms using systolic array implementation on SPLASH 2 in order to achieve orders of magnitude higher performance than the conventional computers of that time. Using JBits S. Guccione *et. al.* [4] implement the Smith Waterman matching algorithm. The same algorithm was implemented at Virginia Tech [17] and the most recent implementation was at Nanyang Techological University [14]. Mapping dynamic programming algorithms on FPGAs seems to be suitable for the capabilities of FPGAs especially when systolic array architectures are used. On the other hand, mapping the BLAST algorithm on reconfigurable logic is not as suitable and only in [15] an implementation can be found. In this architecture Muriki et al. measured the time allocation for BLAST execution in software and according to their measurements replaced the code

segment that consumes almost 80% of execution time with a system call to their specific hardware.

Several impressive but not detailed results of DeCypher have been announced [23]. Unfortunately, lack of information about the architecture itself (number of chips, I/O, architecture type, etc.) as well as how the performance is calculated (types of queries, size of database, version of BLAST, etc.) do not allow for comparisons with our present work.

3. The TUC Architecture

The Technical University of Crete (TUC) architecture, described in this paper, was designed for BLASTn small query implementation (1000 letters) regardless of the data base size. Query sequences can be divided to three cases: small sequence which is between 100 to 2000 characters, medium which is between 2000 and 50000 characters, and large which is between 50000 and 200000 characters. Data base size can also be divided at three cases; small, medium, and large. Small consists of 4.7×10^6 characters, medium is between 5×10^6 and 200×10^6, and large is between 200×10^6 and 4×10^9 characters. NCBI codes consist of several hundreds of files calculating the BLAST algorithm and exporting several numbers which have biological meaning. All these numbers are calculated based on the score of HSP. These calculations produce substantial computing load but the most significant part of the computation power is consumed to find every HSP and extend it, calculating its score. Previous efforts for hardware implementation of BLAST using profiling show that almost 80% of CPU time is spent on these calculations [15].

The TUC architecture is divided into N identical computing machines, *each one of which implements all three steps of the algorithm.* Input data have a width of 2N bits, and come from N different channels. Every channel drives one of the N computing engines. Every machine has two major subsystems, one for step 2 of the algorithm and one for step 3. The first step of the algorithm (the W-mer calculation) is precalculated before algorithm is run. The precalculation results are the first inputs for the machine and they are stored in the memory, together with their position in the query. After this procedure the data stream of the database starts to be processed and if a match is found the second component of the architecture is activated and starts to extend the match, thus implementing the third step of the algorithm. The general design of the architecture is shown in Figure 4.

To illustrate in more detail, before each machine starts the database search, its setup mode asks for the precomputation of W-mers, with their position in the query and their loading to the corresponding memories. This procedure takes about

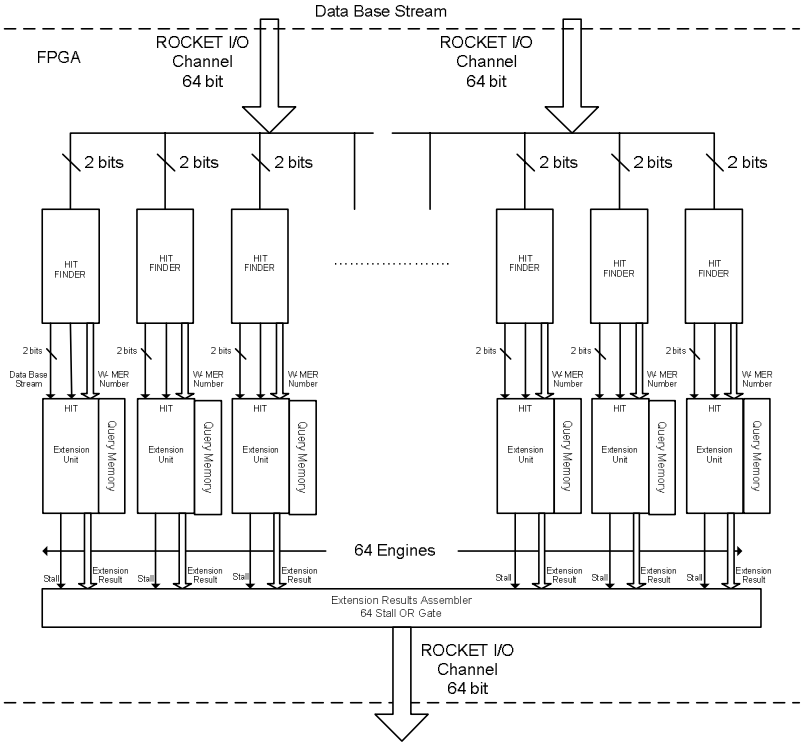

Figure 4 BLAST Machine

1000 cycles for 1000-character long queries.The input of the system in normal mode (database search) is the database stream, one character for each machine. Only the 10 MSBs of W-mers are stored in memory and at the address which corresponds to their 12 LSBs. The stored bits are called W-mer tags. The width of the memory is 23 bits, 10 for the W-mer tag, 1 for valid, and the remaining 12 to show the position of the corresponding W-mer in the input query.

The Hit Finder Unit except for the W-mer memory that was previously described has an input buffer which is 2 bits wide (1 character) and one thousand positions deep, called Future memory. The data stream from the input channel passes through this buffer. As long as there is no hit the buffer operates as a FIFO, getting 2 new bits from the stream in every cycle and driving one shift register (22 bits long) that shifts 2 bits (one letter) per cycle. That shift register has one eleven letter long substring, which is compared with all the W-mers. The 12 LSB of the shift register address the W-mer memory in order to read the W-mer tag. The W-mer tag is compared with the remaining 10 MSBs of the shift register. When a hit is found the Future memory continues to push its data to the shift register and starts to send them at the extension unit as well for the 3rd step of the algorithm. A new comparison is made during every cycle in which the shift register has new data. Conditions for a hit are to have two equal strings in the shift register and the W-mer memory, and the memory content to be valid. Figure 5 shows the Hit Finder Unit architecture. If a second hit comes when the previous is still extended the whole system goes to a stall mode. The system stops trying to find new hits and signals external devices to stop sending new data. In this case the extension unit operates in the normal mode. The Hit Finder unit stops normal operation but continues to pass the data stream to the extension unit.

The Extension Unit executes two comparisons in every cycle, according to the algorithm. It extents both sides and compares the two pairs of letters. The first pair comes from the query memory and the history memory and the remaining couple comes from the Query memory and the Future memory. The data from the input are buffered in the History and Future memories, as it can be seen in Figure 6. There are also counters and registers that keep several useful data, such as hit position for query and database, its length, and the score (which is the most important result to be calculated). Based on the score all the remaining useful data for biologists (e.g. e-value) can be calculated.

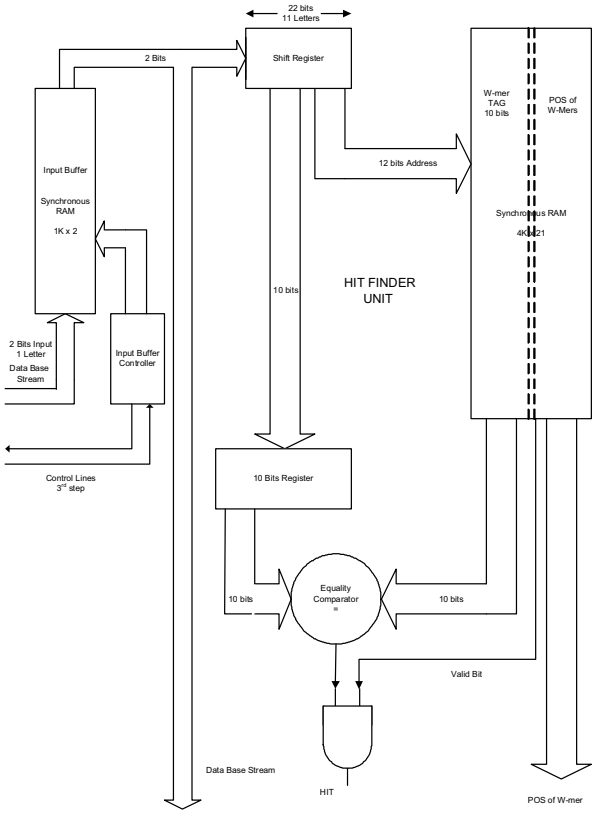

Figure 5 Hit Finder Unit Architecture

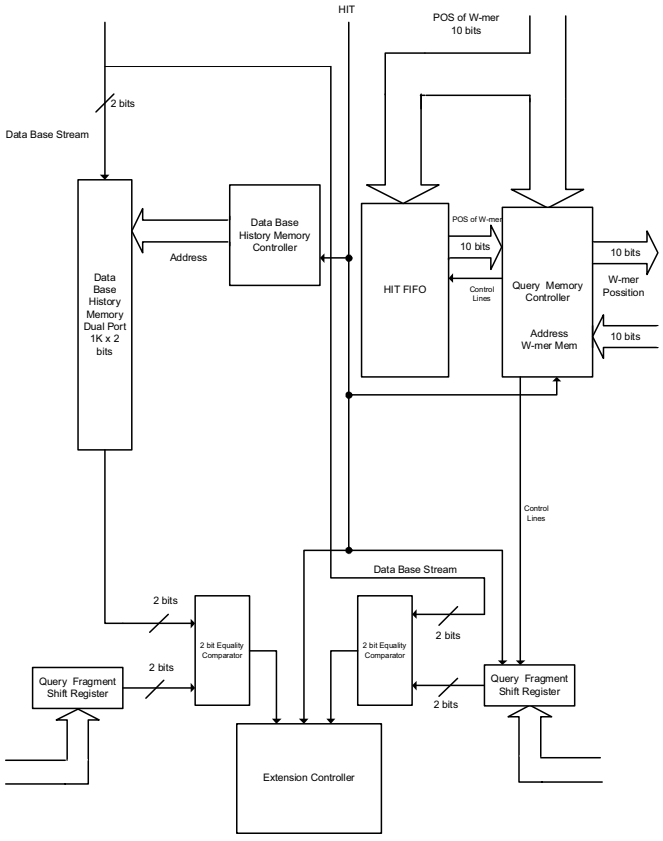

Figure 6 Step 3 Architecture

Number of parallel Machines	Number of FIFO16/RAMB16s (Total 552)		Number of 4 input LUTs (Total 126,336)	
1	8	1%	744	>1%
60	480	86%	46,522	36%
69	552	100%	53,836	42%

Table 1 Area Demands of TUC Architecture

4. TUC Implementation

The TUC Architecture has been coded in VHDL and exhaustively post place-and-route simulated for the VIRTEX 4 family using the 4VFX140FF1517-11 device. The first experiment was the measurement of a single machine (N=1) which run at 121.20 MHz and consumed less than 1% of logic recources and 8 BRAMs. More specifically every single machine needs 8 Blocks of BRAM, 5 of which are given to the memory of W-mer, 1 is used for query, 1 for History Memory and 1 for Future Memory. On the other hand it consumed 744 out of 126,336 LUTs. That shows that the critical resource for implementing many parallel machines is the BRAMs and this restricts parallelization to 69 for the specific device (it has 552 BRAMs divided by 8 BRAMs for each machine). The next implementation was for 60 parallel computing machines (N=60) where exactly 480 BRAMs (or 86%) where used but only 36% of the available LUTs were used. In the last experiment the critical resource BRAMs were exhausted - 552 are used to create 69 parallel computing machines running at 100.36 MHz. As in the previous experiments the percentage of LUT usage was low, only 42%.

In the above experiments it was assumed that there will be an input data stream of up to 69 characters, 2 bits each in parallel at a speed of 100.39 MHz. For that data stream a 138 bit wide bus is needed, with a speed of 13.86 Gbps. The device that was selected for the experiments supports up to 20 ROCKET I/O serial transceivers with 3.125 Gbps each [16]. That gives an upper bound of 62.50 Gbps aggregate bandwidth, and with a realistic and measurable 2.5 Gbps per link we have a total bandwidth for the system of 50Mbps, i.e. an upper bound of 248 parallel computing machines. This amount exceeds any expected number, so the architecture is not input-starved. Output is not a problem and can be accomplished at normal speeds through other pins.

5. TUC performance – Results

The TUC architecture performance is calculated according to post place and route timing information of Xilinx software 7.1.03 which includes *Device speed data version: "ADVANCED 1.54 2005-05-25"* for the certain device. Table 2 has speed measurements for the three experiments.

Actual runs of NCBI software blast-2.2.12 were performed on a 2GHz Xeon with 2GB main memory and the CPU usage was profiled running SUSE 9.1 Linux. For a small query (987 letters) in a large NCBI data base (igSeqNt.ftptemp) with 44,419,359 total letters the 2GHz Xeon measured at 1.380 sec CPU time which is an actual throughput of **32.19 10^6** characters/sec. The CPU time as a fraction of the total time indicates that the database was all stored in memory and the application was not thrashing.

Muriki *et. al.* [15] is the only group that did actual runs of BLAST algorithm in FPGAs to date but their results proved worse than software implementations due to I/O problems, PCI bottleneck and the old technology (XC 4085) which they used (the performance was five times slower than the purely software version). For that reason their results are not used in this comparison.

Number of parallel Machines	Speed (MHz)	Width of Data Stream (characters)	Actual Throughput (characters/sec)
1	121	1	121.20 10^6
60	103	60	6,192.58 10^6
69	100	69	6,924.84 10^6

Table 2 Speed and throughput of TUC Architecture

Number of Processors	Type of Processors	Time (sec)	Database Size (characters)	Actual System Throughput (characters/sec)	Actual Throughput per Chip (characters/sec)
1	POWER3	43.63	$4\ 10^9$	$91.68\ 10^6$	$91.68\ 10^6$
	Model 681 1.1	21.32	$4\ 10^9$	$187.62\ 10^6$	$187.62\ 10^6$
2	POWER3	24.09	$4\ 10^9$	$166.04\ 10^6$	$83.02\ 10^6$
	Model 681 1.1	11.39	$4\ 10^9$	$351.18\ 10^6$	$175.59\ 10^6$
4	POWER3	14.23	$4\ 10^9$	$281.10\ 10^6$	$70.27\ 10^6$
	Model 681 1.1	6.53	$4\ 10^9$	$612.56\ 10^6$	$153.14\ 10^6$
8	POWER3	9.25	$4\ 10^9$	$432.43\ 10^6$	$54.05\ 10^6$
	Model 681 1.1	4.33	$4\ 10^9$	$923,79\ 10^6$	$115.47\ 10^6$
16	POWER3	7.56	$4\ 10^9$	$529,10\ 10^6$	$33,07\ 10^6$
	Model 681 1.1	3.33	$4\ 10^9$	$1201,20\ 10^6$	$75,07\ 10^6$

Table 3 blastn Benchmarks with a Small Single Query and Large Database

System	Actual Throughput (10^6 characters/sec)
2GHz Xeon	32.19
TUC Architecture N=1	121.20
TUC Architecture N=60	6,192,58
TUC Architecture N=69	6,924.84
IBM single chip	187.62
IBM System	1,201.20

Table 4 Systems Throughput

	SpeedUp of TUC Architecture N=1	SpeedUp of TUC Architecture N=60	SpeedUp of TUC Architecture N=69
2GHz Xeon	3.76	192.37	215.12
IBM single chip	0.65	33.00	36.90
IBM System (16 chips)	0.10	5.15	5.76

Table 5 TUC Architecture SpeedUp

From Table 3 it can be shown that the fastest system throughput is achieved with the 16 processors Model 681 1.1 system, which has a throughput of $1,201.20\ 10^6$ characters/sec. However, the fastest single chip system is IBM Model 681 1.1 with $187.62\ 10^6$ characters/sec.

Table 4 has the actual throughput for systems implementing BLAST algorithm and in Table 5 the Speedup of TUC architecture against the other systems.

6. Conclusions and Future Work

Significant improvements on this architecture and its implementation can be achieved. More specifically this implementation contains several memories which have been all implemented using BRAMs. The number of BRAMs in the specific device is 552. As 8 BRAMs are used for each machine, this gives a parallelisation of 69 machines. Improving the W-mer addressing with a hash function implementation can save up to 3 or 4 BRAMs for each machine. Balancing between using of BRAMs and distributed memory can save 2 more BRAMs for each machine because half of logic cell are not used. That will give a new design which uses 2 or 3 BRAMs for each machine and it will increase parallelisation to 180 computing machines. The use of Power PC cores that are embedded to VIRTEX 4 must be examined as alternative solution to extension unit. Finally it would be very interesting to study architectures for bigger queries.

Acknowledgements

We wish to thank several people that contributed to this project. Georgia Adamopolou introduced the problem to us giving us important details about BLAST algorithm implementation. Support for the FPGA

hardware in the lab came from donations from Xilinx, Inc. This work was funded from the Greek Ministry of Education Grand for the project GSRT "IRAKLEITOS: Research Scholarships for TUC", Program. TUC, sub-program #10, "Structures for Reconfigurable Computing".

7. References

[1] J. Meidanis and J.C. SetUbal, Introduction to Computational Molecular Biology, PWS Publishing Company, 1997.

[2] D. Hoang et. al. "FPGA Implementation of Systolic Sequence Alignment". Proceedings of the 2nd International Workshop on Field-Programmable Logic and Applications, Lecture Notes in Computer Science 705, pp 183-191, 1992.

[3] D. Hoang "Searching Genetic Databases on Splash 2", Proceedings IEEE Workshop on FPGAs for Custom Computing Machines (FCCM), pp 185-191, 1993.

[4] S. Guccione and Eric Keller "Gene Matching Using JBits". Proceedings of the 12th International Conference on Field-Programmable Logic and Applications, Lecture Notes In Computer Science; Vol. 2438, pp 1168-1171, 2002.

[5] S. Altschul et. al. "Basic Local Alignment Search Tool", J. Mol. Biol., vol. 215, pp 403-410, 1990

[6] Pearson, W. and Lipman, D. "Improved tools for biological sequence analysis". In *Proceedings of National Academic Science*, 85, pages 2444–2448, 1988

[7] T.F. Smith, M.S. Waterman "Identification Of Common Molecular Subsequences" J. Mol. Biol., vol. 147, pp 195-197, 1981

[8] S. B. Needleman and C. D. Wunsch "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins", ", J. Mol. Biol., vol. 48, pp 443-453, 1970.

[9] www.ebi.ac.uk

[10] www.ddbj.nig.ac.jp

[11] scop.wehi.edu.au/gsdb/gsdb.html

[12] pir.georgetown.edu

[13] www.isb.sib.ch

[14] T. Oliver et. al. "Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs", Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays (FPGA), pp 229 – 237.

[15] K. Muriki et. al. "RC-BLAST: Towards a Portable, Cost-Effective Open Source Hardware Implementation", Proceedings 19th IEEE International Symposium Parallel and Distributed Processing (IPDPS), pp 196b-196b 2005.

[16] http://www.xilinx.com/products/silicon_solutions/fpgas/product_tables.htm#V4FX

[17] K. Puttegowda et. al. "A Run-Time Reconfigurable System for Gene-Sequence Searching", Proceedings. 16th International Conference on VLSI Design pp 561 – 566, New Delhi 2003

[18] www.ncbi.nlm.nih.gov

[19] C. P. Sosa et. al. "Some Practical Suggestions for Performing NCBI BLAST Benchmarks on a pSeries ™ 690 System", http://www.redbooks.ibm.com/abstracts/redp0437.html?Open.

[20] Sidhu, R.; Prasanna, V.K. "Fast Regular Expression Matching Using FPGAs", Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp 227 – 238, 2001

[21] Hutchings, B.L.; Franklin, R.; Carver, D." Assisting network intrusion detection with reconfigurable hardware", Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp 111-120, 2002

[22]http://www.timelogic.com/benchmark_blast.html