Presented at the DLSU Research Congress 2019
De La Salle University, Manila, Philippines
June 19 to 21, 2019

FPGA-Based Implementation of Bit-Vector Algorithm

Lorenzo Bautista[1,*], Roger Luis Uy[1], Kyle Chua[1], Janz Villamayor[1]
[1] *College of Computer Studies, De La Salle University, Manila, Philippines*
**Corresponding Author: lorenzo_bautista@dlsu.edu.ph*

Abstract: DNA pairwise sequence alignment involves matching two DNA sequences to identify and locate identical substrings which can be used in downstream analysis to discover biological relationship that leads to further scientific and medical advancements. One of the metrics to measure the similarity of the two sequences is the Damerau edit distance. In human genome, the length of one of the sequence n is 3 billion base pair long while the length of the other sequence m can be a thousand base pair long. Dynamic programming-based implementation of the Damerau edit distance has a runtime complexity of $\Theta(mn)$. Considering the length of the sequences, this process is computationally intensive. Thus, various algorithms have been proposed to improve its runtime. One such algorithm is the Bit-vector algorithm which has a runtime complexity of $\Theta(n)$. Several researchers have implemented this algorithm in various computing platform. In this study, our contribution is the implementation of the Bit-vector algorithm using Field-Programming Gate Array (FPGA) computing platform to take advantage of its computing capability, design flexibility, and power efficiency. Experiment results based on varying lengths of query and reference sequences show that our implementation is consistent with the algorithm's runtime complexity of $\Theta(N)$. The power consumption usage of the implementation is only 3.48W.

KEYWORDS: DNA sequence alignment; Bit-vector algorithm; SIMD computing capabilities; FPGA

1. INTRODUCTION

DNA sequence alignment is the process of comparing a query read or pattern p with length m against a reference sequence or text t with length n to determine regions of similarities that allows downstream analysis to assess the relationship between species and organisms. The similarity can be measured by the edit distance, which is defined as the number of required edit operations to make both sequences equal (Chang, Escobar, Valderrama, & Robert, 2014; Hyyrö, 2002; Myers, 1999). Valuable information can be obtained through accurate alignment and such information can be used in higher-level processes, such as phylogenetic trees, genetic structure prediction, and disease diagnosis (Hasan & Al-Ars, 2011).

Depending on the number of sequences it can concurrently process, sequence alignment algorithms can be classified in either pairwise or multiple sequence alignment; the former aligns exactly two sequences, while the latter aligns two or more sequences simultaneously (Hasan & Al-Ars, 2011). Researchers argue that multiple sequence alignment is more significant for scientific and research use. However, it is important to note that multiple sequence alignment is merely an extension of pairwise sequence alignment. Thus, multiple sequence

alignment benefits from enhancing pairwise sequence alignment (Chang, Escobar, Valderrama, & Robert, 2014). Sequence alignment algorithms can be further classified into two methods; Global or Local. Global method aligns the sequences from end-to-end and it is useful when identifying the total similarity of sequences. On the other hand, local method aligns fragments of the sequences and it is useful when identifying homologous regions (Langner, 2011). The sequence alignment involves two processes; filtration and verification. Genomic reference sequences are generally very long and in order to scope down the search size, it entails the need of filtration which locates candidate regions that are highly similar to the query sequence. A good filter is characterized by identifying candidate regions that have high probability of containing the query sequence (Varshavsky, Gottlieb, Linial, & Horn, 2006). Subsequently, the verification process confirms if the alignment of the filtered reference sequence and the query sequence are closely similar based on the edit distance metrics (Myers, 1999).

Edit distance metrics is based on the minimal number of insertions, deletions and substitutions to make two strings equal. It is also known as string matching with k differences. Damerau edit distance, which is a variant of the edit distance, involves four edit operations instead (i.e. insertion, deletion, substitution, transposition) to transform a query sequence into a reference sequence and vice-versa (Hyyrö, 2002). An insertion operation inserts a character in any position of a given string. For example, a string of characters is 'CTG' and we want to insert 'A' before 'C'. After an insertion was performed, the new string becomes 'ACTG' and is counted as one edit operation. As opposed to insertion, a deletion operation removes a character of a given string. For example, a string is 'ACTG' and we want to delete 'C'. After a deletion, the string would become 'ATG' and is counted as one edit operation. As for a substitution operation, a character of a given string would be replaced with another character. For example, a given string is 'CTATG' and we want to replace the character 'A' in to 'C'. After a substitution, the string would become 'CTCTG' and is counted as one edit operation. Lastly for a transposition operation, given a string, two characters' position are swapped. For example, a string is 'CATG' and we want to swap 'C' with 'A'. After a transposition, the string would become 'ACTG' and is counted as one edit operation. As an illustration, the Damerau edit distance between the string "CA" and "ABC" is 2.

The Damerau edit distance is usually implemented as dynamic programming with a runtime complexity of $\Theta(mn)$ where m is the length of the query sequence, which is typically a thousand base pair long and n is the length of reference sequence, which is typically 3 billion base pair long in human genome. This type of implementation is computational-heavy and demands a significant amount of time, especially when it is implemented to run on Central Processing Unit (CPU) using high level programming language (Hasan & Al-Ars, 2011). Coupled this with the advancement on Next Generation Sequencing (NGS) technologies, scientists are able to generate DNA sequences at a much higher rate and lower cost, DNA sequence alignment cannot keep up with the rapid growth of sequence database (Chang, Escobar, Valderrama, & Robert, 2014). Thus, researchers are challenged to create various bioinformatics solutions which can be crucial in numerous scenarios, such as DNA forensics, early diagnosis of susceptibility to genetic diseases, and prevention of bacteria or virus evolution (Memeti & Pllana, 2015). This paved ways for implementing DNA sequence alignment on other computing platform such as Field Programmable Gate Arrays (FPGA) and High Performance Computing (HPC) technologies that can perform Single-Instruction-Multiple-Data (SIMD) unit on modern CPU, Graphical Processing Units (GPU) and Cell Broadband Engines (Cell BE) (Benkrid, et al., 2012).

The FPGA is a semiconductor device that allows developers and designers to create customable digital logic pre and postproduction. It is made up of two-dimensional arrays of logic gates (i.e. gate array) that when combined with other gate arrays, could implement simple calculations to more meaningful functions (Moore, 2017; Xilinx, n.d.). According to previous studies, FPGA provides faster, more power-efficient and cost-efficient solution compared to other technologies for sequence alignment because its architecture and pipelining suits the requirements of bit-vector algorithms (Benkrid, et al., 2012; Che, Li, Sheaffer, Skadron, & Lach, 2008). Using an FPGA, we implemented a DNA sequencing alignment on a query sequence and reference sequence using bit-vector algorithm. The implementation utilizes the flexibility and parallelism of the FPGA, specifically in the Zynq-based Artix-7 family line of FPGA chips. The system

was tested and verified for correctness through multiple test cases. Experiments were also perform using varying DNA sequences to measure the speed and power consumption of the FPGA implementation. The focus of this paper is the implementation of Hyyrö's (Hyyrö, 2002) bit-vector algorithm utilizing the FPGA architecture for pairwise sequence alignment. The system is capable of handling query sequences up to the length of 256 characters, through the use of 256-bit bit vector registers. Real-world DNA sequences obtained from the National Center for Biological Information (NCBI) online GenBank sequence database (Varshavsky, Gottlieb, Linial, & Horn, 2006) were utilized as dataset for experimentation.

2. METHODOLOGY

This paper provides a discussion on the implementation of bit-vector algorithm using ZYNQ Z-7000 FPGA development Board and evaluating our own implementation with real-world DNA sequences. The ZYNQ Z-7000 is a System-on-Chip (SOC) board composed of a Dual-core ARM Cortex-A9 processing system (PS) and an Artix-7 programmable logic (PL). The Damerau Edit distance implemented is based on Hyrrö's (2002) bit vector algorithm as illustrated in Figure 1. In his study, he did not present any performance evaluation since his study focused on the theories and framework of the algorithm. One can observe that bit-vector is the major data structure used in the algorithm. Bit vector *Eq* contains the DNA alphabet. Bit vectors *Ph* and *Mh* are used to record the changes in the value of the horizontal rows of the dynamic programming cell structure while *Pv* and *Mv* are used to record the changes in the value of the vertical columns of the dynamic programming cell structure. Bit vector *Xh* is used to record the relationship between bit vector *Eq* and *Mh* while bit vector *Xv* is used to record the relationship between bit vector *Eq* and *Mv*. The algorithm has only one loop as shown in line 6 and is dependent based on the length of reference sequence *n*. Thus, the runtime complexity of the algorithm is $\Theta(n)$. Note for each loop, most of the operations used are bitwise vector operations (i.e., AND, OR, XOR, NOT). For the purpose of this study, the algorithm was modified (See lines 14 and 15 of Fig. 1) such that the computation for the Damerau distance will continue regardless if the *k*-error threshold has been reached. This not only enables the evaluation of similarity between the two sequences, but also allows pinpointing highly similar regions. The pre-processing of the query sequence is also modified to obtain the reverse bitmask of each character. Readers can refer to Hyrrö's (2002) for further details of the algorithm.

```
1       <Preprocess B[σ] with P>
        ;Preprocess of bit-vectors for sequence P
2       Bit-vector Pv,Mv,Ph,Mh,Xv,Xh,Eq,Xp
;Setup vectors with 0^m
3       Score = m
4       Pv = 1^m
5       Mv = 0^m
;Initialize vertical delta values
6       for j = 1, 2, ..., n do
7       Eq = PEq[Σ[T[j]]]
        ;Bit-vectors for reference symbol j
8       Xv = Eq | Mv
9       Xh = (((~Xh) & Xv) << 1) & Xp
10      Xh = Xh | (((Xv & Pv) + Pv) ^ Pv) |
Xv|Mv   ;compute current delta vector
11      Ph = Mv | ~ (Xh | Pv)
12      Mh = Xh & Pv
        ;update horizontal delta values
13      Xp = Xv         ;store old pattern bit-
vector
14      if(Ph & 10^(m-1)) then score += 1
15      else if(Mh & 10^(m-1)) then score -= 1
        ;update score
16      Xv = (Ph << 1)
17      Pv = (Mh << 1) | ~ (Xh | Xv)
18      Mv = Xh & Xv
        ;update vertical delta values
```

Fig. 1. Hyrrö's (2002) Bit-vector algorithm for computing Damerau distance

The algorithm is implemented on Xilinx Vivado 2017.1 and Visual Studio 2017 and compiled using Xilinx SDK 2017.1. The system is composed of three elements: the Verilog HDL program, the C program, and the C# program. The Verilog HDL program handles the computation of the Damerau distance between the query sequence and reference sequence, and device setting specifications of the FPGA board. The C program handles the connection between the FPGA hardware and the host computer with the use of UART. The C# program acts as the GUI interface by which the users can give instructions to the FPGA board, such as sending data from the host computer to the FPGA board, processing the data, and performing the algorithm. It also acts as a console for the output

of the FPGA (i.e., execution time and Damerau edit distance score).

A text file for a query sequence and another for a reference sequence is read and sent to the FPGA board. The FPGA board converts the data to a 2-bit binary representation before storing into the DDR3 memory of the board. It would then send the converted 2-bit binary data from the DDR3 memory to the slave registers inside the FPGA chip. After receiving the 2-bit data, another conversion would take place. The data is reversed and converted into a *PEq* value that would be used as a map for the verification process. Each character from the reference sequence is taken one at a time to be verified and processed with the PEq. It goes through a 13-state process where each state takes up 1 clock cycle at a time based on the clock specification of the FPGA chip. At the end of the 13th state, a score is recorded and stored in one of the mapped slave registers that can be accessed by the C program of the FPGA and can be sent back to the host computer to be viewed by the C# program. Once the FPGA has processed the whole reference sequence, the C program would notify the host computer and send out the lowest score that was found, the approximate position/s where the sequence is located, and the time it took for the system to process it.

The total process time spend inside the FPGA is handled by an XTime timer routine of the C program. It first collects the current time of the system before running the whole program, and once the FPGA has reached the final reference sequence character, another XTime timer collects the current system time. It would then deduct the most current system time, then send out to the host computer in microseconds.

To obtain the power consumption of the FPGA board, a current sensor pin (J21) is being utilized. The J21 pin has a 10mΩ resistor across its two protruding pins where a multimeter, that is set to voltage reading, can obtain the voltage value flowing through the resistor. Once the voltage value is obtained, it would be divided by 0.01 to get the current flowing through the board. After the current value is computed, it is then multiplied by the working voltage of the board (12V) to get the total power consumption of the FPGA.

To evaluate the performance of the FPGA implementation, the DNA sequences of Homo sapiens (human), Mus Musculus (mouse), Solanum Pennellii (eudicots), Brachypodium Distachyon strain Bd21 (stiff brome), Ornithorhynchus Anatinus (platypus), Cajanus Cajan (pigeon pea), Pseudomonas Syringae (g-proteobacteria), Chthonomonas Calidirosea (bacteria), Prochlorococcus Marinus str. MIT 9211 (cyanobacteria), and Mycoplasma Conjuctivae (mycoplasmas) were selected for experimentation. The choice of selection is based from the DNA length of each genome which ranges from 47 million to 230 million base pair long. The dataset is composed of chromosome 1 sequences from the chosen species which can be obtained from the GenBank sequence

As for query sequences, randomly generated characters (i.e. 'A', 'C', 'G', 'T') were used as datasets based from the lengths of 64, 128, 192, and 256.

database of NCBI (Varshavsky, Gottlieb, Linial, & Horn, 2006). For the purpose of this study, the researchers have omitted the instances of the wildcard character 'N' for all sequences. Table 1 shows the reference sequence datasets and their corresponding length, excluding character 'N'.

Table 1. Summary of datasets used

	Genome Reference	Length (No. of Characters)
Human	GRCh38.p12	230481014
Mouse	GRCm38.p4 C57BL/6J	195471971
Eudicots	SPENNV200	109333515
Stiff Brome	Bd21	75071545
Platypus	Ornithorhynchus _anatinus-5.0.1	47594283
Pigeon Pea	C.cajan_V1.0,	17676265
G-proteobacteria	DC3000	6397126

Bacteria	T49	3437861
Cyanobacteria	MIT 9211	1688963
Mycoplasmas	HRC/581T	846214

3. RESULTS AND DISCUSSION

We aimed to investigate the effect of varying the sequence lengths on the computation time and power consumption of the FPGA implementation. The procedure was performed on a ZedBoard Zynq Evaluation and Development Kit with 512 MB DDR3 RAM using a Zynq-7000 family of FPGA combined with a Cortex-A9 Processing System (PS) and Artix-7 Programmable Logic (PL). It is interfaced with a GUI terminal program in C# that is being run on an ASUS Zenbook Laptop equipped with Intel Core i7-4500U 1.8 GHz 64-bit processor and 8GB of RAM. For the GUI terminal software, the communication settings are as follows: baud rate of 115200, no parity bits, 8 bits of data, 1 stop bit, and no hardware flow control.

To perform the sequence length test, we simulated the length of a given reference sequence and query sequence of varying lengths (i.e. 64, 128, 192, 256) and inputted the values inside the code of the PS, then program the ZedBoard with a serial monitor open to receive the outputs of the device. An Xtimer used inside the PS code is activated before the start of the verification process and deactivate when the ZedBoard reaches the threshold of the reference sequence. Table 2 shows the summary of the experiment results on the ZedBoard.

The computation time consists of the pre-processing of the query sequence and the actual computation time. It shows that the FPGA is consistent with the theoretical time complexity of $\Theta(N)$ given that the reference sequence of length N *is* the same regardless of the size of the query sequence.

The power consumption in all test cases were consistently outputting at 3.48W regardless of the length of the query and reference sequences as shown in Table 2. The power consumption was obtained by setting up two male-to-female jumper cables with the male connected directly in the probe ports of a multimeter while the female heads were connected directly to the current sense (J21) of the ZedBoard. To compute for the power, see Equation 1 and 2. It can be observed that the power consumption doesn't change regardless of the query size and the sequence length.

$$I = V_{J21} / 0.01 \quad \text{(Eq. 1)}$$

$$P = V * I \quad \text{(Eq.2)}$$

where:

- I = current flowing through the 10mΩ resistor
- V_{J21} = Voltage across pin (J21)
- 0.01 = 10mΩ resistor
- V = Voltage input of the device (from power supply)
- P = Power

Table 2. Summary of experimentation results for FPGA-based implementation

Reference	Query Size	Average Computation Time (Seconds)	Average Power Consumption
human	64	41.54	3.48W
	128	41.531	
	192	41.55	
	256	41.541	
mouse	64	34.604	3.48W
	128	34.589	
	192	34.589	
	256	34.588	
eudicots	64	18.334	3.48W
	128	18.334	
	192	18.326	
	256	18.334	
stiff brome	64	13.515	3.48W
	128	13.512	
	192	13.509	
	256	13.51	
platypus	64	8.023046	3.48W
	128	8.023047	
	192	8.017616	
	256	8.023048	

pigeon pea	64	2.464212	3.48W
	128	2.46253	
	192	2.462779	
	256	2.462556	
g-proteobacteria	64	1.15268	3.48W
	128	1.153484	
	192	1.153485	
	256	1.15269	
Bacteria	64	0.736787	3.48W
	128	0.736789	
	192	0.73636	
	256	0.736402	
Cyanobacteria	64	0.373852	3.48W
	128	0.373876	
	192	0.373879	
	256	0.373871	
Mycoplasmas	64	0.152695	3.48W
	128	0.152702	
	192	0.152686	
	256	0.152327	

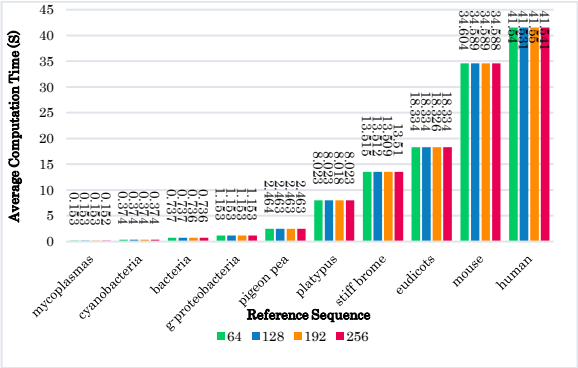

Fig. 2 Average computation time for FPGA-based implementation

4. CONLUSIONS

Based from the result, our FPGA implementation is consistent with the theoretical runtime complexity of $\Theta(N)$ regardless of the query size. It also shows that regardless of the length of the reference and query sequence, the power consumption was consistently at 3.48W. This shows that the FPGA has a low power consumption. Though we are able to note that there is a large latency time when sending a large dataset from host PC to the FPGA board. Also, there are some instances that the PS side failed to response properly when sending large datasets.

Future experiments may use PCI Express or Ethernet as a way to communicate with the FPGA directly to store large sets of data. The FPGA board has a built-in 10/100/1G Ethernet port peripheral that can be set using a Linux-based PS setting. Making use of the FPGA as a standalone computer/device may solve the issues of sending large dataset between host computer and FPGA, and at the same time, speedup the process of reading large data sets. The board has its own graphic ports such as HDMI or VGA that can be used to output the result. It also has a SD Card slot that can be used as a storage device to store raw data and can be access directly by the system thereby bypassing the data transfer step.

5. REFERENCES

Benkrid, K., Akoglu, A., Ling, C., Song, Y., Liu, Y., & Tian1, X. (2012). High performance biological pairwise sequence alignment: FPGA versus GPU versus cell BE versus GPP. (K. Sano, Ed.) *International Journal of Reconfigurable Computing, 2012*, 1-15.

Chang, X., Escobar, F. A., Valderrama, C., & Robert, V. (2014). Exploring Sequence Alignment Algorithms on FPGA-based Heterogeneous Architectures. *International Work-Conference on Bioinformatics and Biomedical Engineering 2014*. Retrieved from https://pdfs.semanticscholar.org/ae2e/7e73fecbbd5f4e9bb56699d9043d909579fc.pdf

Che, S., Li, J., Sheaffer, J. W., Skadron, K., & Lach, J. (2008). Accelerating Compute-Intensive Applications with GPUs and FPGAs. *IEEE*, 101-107.

Hasan, L., & Al-Ars, Z. (2011). An Overview of Hardware-Based Acceleration of Biological Sequence Alignment. 187-202. Retrieved from https://ce-publications.et.tudelft.nl/publications/7_com

putational_biology_and_applied_bioinformatics.pdf

Hyyrö, H. (2002). A Bit-Vector Algorithm for Computing Levenshtein and Damerau Edit Distances. *Nordic Journal of Computing - Special issue: Selected papers of the Prague Stringology conference*, 29-39. Retrieved from https://pdfs.semanticscholar.org/813e/26d8920d17c2afac6bf5a15c537b067a128a.pdf

Langner, L. (2011). Parallelization of Myers Fast Bit-Vector Algorithm using GPGPU. Retrieved from https://www.mi.fu-berlin.de/en/inf/groups/abi/teaching/theses/master_dipl/langner_bitvector/dipl_thesis_langner.pdf

Memeti, S., & Pllana, S. (2015). Accelerating DNA Sequence Analysis Using Intel(R) Xeon Phi(TM). *2015 IEEE Trustcom/BigDataSE/ISPA, 3*, 222-227. doi:10.1109/Trustcom.2015.636

Moore, A. (2017). *FPGAs For Dummies 2nd Edition.* New Jersey: John Wiley & Sons Inc.

Myers, G. (1999, May). A Fast Bit-vector Algorithm for Approximate String Matching Based on Dynamic Programming. *Journal of the ACM, 46*(3), 395-415. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.9395&rep=rep1&type=pdf

Varshavsky, R., Gottlieb, A., Linial, M., & Horn, D. (2006). Novel Unsupervised Feature Filtering of Biological Data. *22*(14), 507-513. doi:https://doi.org/10.1093/bioinformatics/btl214

Xilinx. (n.d.). *What is an FPGA?* Retrieved November 20, 2017, from Xilinx: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html