GenCodex - A Novel Algorithm for Compressing DNA sequences on Multi-cores and GPUs

D Satyanvesh*, Kaliuday Balleda*, Ajith Padyana, P K Baruah
Sri Sathya Sai Institute of Higher Learning, Prasanthi Nilayam, India
{satyanvesh.d, kaliudayballeda}@gmail.com, {ajithpadyana, pkbaruah}@sssihl.edu.in

Abstract—**The DNA sequences are huge in size and the databases are growing at an exponential rate. For example, the human genome in raw format ranges from 2 to 30 Tera-bytes. The main reason for this is the invention of new species and increasing number of DNA profiles. The growth of the DNA affects the storage as well as bandwidth when these sequences need to be transferred. Applications such as DNA profiling, Real time DNA crime investigation require access to the DNA sequences in real time. The inherent property of DNA is that it contains many repeats which makes it highly compressible. However, the applications mentioned not only require good compression ratio but also needs faster compression. Multi-cores and GPUs can be used to perform the compression quickly. In this paper, we propose a new algorithm with a focus on the throughput along with the compression ratio. The algorithm scales well on GPUs and achieves a speedup of 11 on multi-cores and upto 23 on GPUs.**

Index Terms—**DNA Sequence, Bandwidth, Throughput, Compression Ratio.**

I. Introduction

In molecular biology, the genome consists of all the hereditary information for running and maintaining an organism. This biological information contained in genome is encoded in the form of DNA. DNA chain is made of four bases: Adenine (A), Guanine (G), Thymine (T), and Cytosine (C). When the cells divide to grow, every new cell needs a copy of the DNA to function properly. So, DNA replicates itself before the cell divides. Due to this, the genomic data increases constantly which leads to doubling of the DNA sequences. DNA also has many repeats. This property can be used to compress the data.

General purpose compression algorithms such as gzip, bzip2 do not work well for the DNA sequences since it consists of only 4 bases namely A, T, G and C. As a result, these algorithms expanded the DNA sequence instead of compressing it [1]. Other algorithms such as Biocompress-2 [2], GenCompress [3], DNACompress [4], DNABIT [5] and GENBIT[6] have been used in the recent years to compress the DNA sequences.

An important piece of information contained in DNA sequence is tandem repeats. But all the algorithms take quadratic or more amount of time for searching those tandem repeats in a huge DNA sequence [7]. Applications such as DNA profiling, Real time DNA crime investigation require access to the DNA sequences in real time. So, the compression must be very quick. The challenging problem is to achieve high throughput along with a better compression ratio. In this paper, we address these issues by obtaining a better compression ratio at a high throughput by using graphical processing units(GPUs) and multi-cores.

The rest of the paper is organized as follows: In section 2, we briefly discuss the related work. In section 3, we discuss the details of our proposed algorithm. Section 4 gives the implementation details of our algorithm. Section 5 describes the experimental setup. Section 6 discusses the results. Conclusion and future work are dealt in section 7.

II. Related Work

Grumbach and Tahi [2] proposed two lossless compression algorithms for DNA sequences, namely BioCompress and BioCompress-2, making use of the Ziv and Lempel data compression method [8]. BioCompress-2 finds both the exact and reverse repeats in the target sequence. It encodes them by repeat length and the position of a previous repeat occurrence. If there is no significant repetition then the arithmetic coding of order-2 is used to reduce the number of bits used. The only difference between BioCompress and BioCompress-2 is the use of arithmetic coding.

Gencompress [3] is a one-pass algorithm that searches for the approximate matches. This algorithm uses order-2 arithmetic encoding [1]. Gencompress detects the approximate complemented palindrome (A replaced by

*Student Author

T and C replaced by G) in DNA sequences. The average compression ratio is 1.7428 bits/bytes. Gencompress [3] achieves higher compression ratios compared to Biocompress or Biocompress-2.

DNACompress [4] uses Lempel-Ziv compression scheme. It finds all the approximate repeats including complemented palindromes and encodes approximate repeat regions and non-repeat regions. The average compression ratio is 1.7254 bits/bytes.

GENBIT Compress algorithm [6] uses a little different method in which each input sequence is divided into fragments of 4 characters each. Hence each fragment can be encoded in 8 bits as each character is represented using 2 bits. If the consecutive fragments are same, then the specific 9^{th} bit is set to 1. If the consecutive fragments are different, then the specific 9^{th} bit is set to 0 for that 8 bit unique representation. The average compression ratio for this algorithm is 1.727 bits/bytes.

The literature shows that the existing algorithms concentrate mainly on compression ratio whereas the proposed algorithm and its parallel implementation not only achieves decent compression ratio but also has a better compress throughput.

III. Proposed Algorithm

Our method is efficient in compressing both repetitive and non-repetitive DNA sequences. The input sequence is divided into fragments of 4 characters each.

In the first phase, each character is represented using two bits namely, A=00, C=01, G=10, T=11. So each fragment is stored using 8 bits i.e., using just one byte. At the end of this phase, we get the compressed sequence where 4 characters of the original sequence are encoded into a single byte.

In the next phase, the fragments are represented using either one or two bytes. If a fragment is not appearing consecutively, then a single byte is allocated using its 8-bit unique representation. If a fragment is repeating two or more times, then the simple 8-bit representation is put in the first byte and the number of repetitions are represented in the second byte. For every eight bytes of the compressed data, we use an extra byte referred as code byte in which we set the corresponding bit to 1 if there is a repetition. So if a bit is 0 in the code byte, only 1 byte is considered in the compressed sequence and if a bit is set to 1 in the code byte, the next 2 bytes are considered in the compressed sequence. This is shown in the Fig. 1.

The proposed algorithm is named as GenCodex where x signifies the number of repetitions of a fragment occurring consecutively in a sequence. In this paper, we discuss about 256 repetitions occurring consecutively. The same can be extended if the repetitions occur 128, 64, 32, 16 times etc.

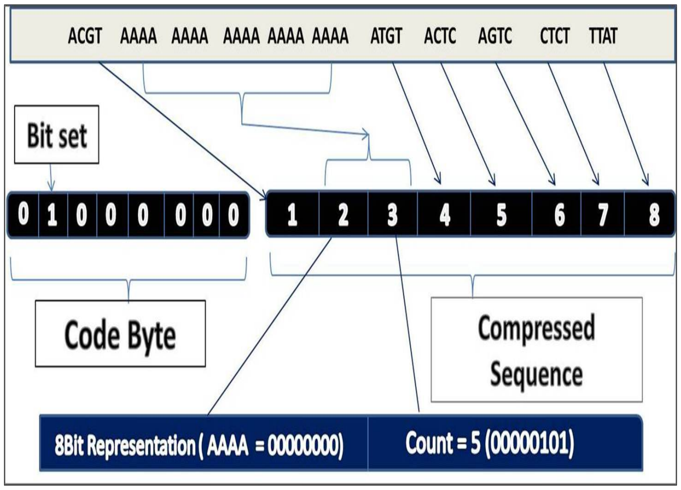
ACGT AAAA AAAA AAAA AAAA AAAA ATGT ACTC AGTC CTCT TTAT
Bit set
0 1 0 0 0 0 0 0
1 2 3 4 5 6 7 8
Code Byte
Compressed Sequence
8Bit Representation (AAAA = 00000000)
Count = 5 (00000101)

Fig. 1. Diagram to explain the algorithm

Algorithm 1 GenCodex Algorithm for Compression

Input: Original DNA sequence (Base pairs: A, C, G, T).

Output: Compressed Sequence along with Code-bytes.

```
  Phase 1:
  I. Divide the sequence into fragments consisting of 4 characters each.
  II. Assign unique two bit number to each of the character (A=00, C=01, G=10, T=11).
  Phase 2:
  III. If consecutive fragments are not same, then we represent the fragment with its 8-bit representation (1 byte) itself.
  IV. If the fragment is repeating two or more times consecutively, then we represent that fragment using 2 bytes and set the corresponding bit in the Code byte.
  V. Repeat the same process until the end of the sequence.
  VI. Output the compressed sequence along with the code bytes that are used to represent the compressed sequence.
```

Best Case: Consider an input sequence consisting of 4080 characters where each fragment is repeating 255 times consecutively. Since each fragment in the compressed sequence requires 2 bytes, we need a total of 8 bytes i.e., 64 bits for the compressed data and one

Algorithm	GENBIT	DNABIT	GenCodex
Best Case	1.125	1.04	0.017
Average Case	1.727	1.53	1.42
Worst Case	2.238	1.58	2.25

TABLE I
COMPRESSION RATIOS FOR DIFFERENT ALGORITHMS (IN BITS/BYTES)

byte (8 bits) for the code byte used for this compressed data. So a total of 72 bits are required to represent the compressed data.

Compression ratio = Number of bits / total number of bytes = 72 / 4080 = 0.017 bits/bytes.

Average Case: The repetitions of each fragment range from 0 to 255. A detailed analysis on DNA sequences reveals that a fragment repeating for 2 or 4 times is more common than a fragment repeating for 255 times. So, for the average case analysis, probabilities are assigned suitably for each pattern. The compression ratio is 1.42 bits/bytes.

Worst Case: Consider an input sequence of length 32 bytes where no fragment is repeating. A total of 72 bits are needed for representing the compressed data.

Compression ratio = Number of bits / total number of bytes = 72 / 32 = 2.25.

The compression ratios with different algorithms for the best, average and worst cases are shown in the Table I.

IV. Implementation

Serial Implementation: In the first phase, each character is read from the file and is allocated two bits. By using the bit-wise shift operations, four characters are encoded into a single byte instead of four bytes. In the second phase, the fragments are allocated either a single byte or two bytes according to the number of repetitions in the input sequence. But, there is a special case wherein we always set the 8^{th} bit in an code byte to 0. As described earlier, there is one code byte for every 8 bytes of compressed data. Setting the 8^{th} bit in the code byte to 1 implies that there is a repetition in the 8^{th} and 9^{th} bytes of the sequence and we need to allocate two bytes (occupying 8^{th} and 9^{th} bytes) in the compressed sequence, but this 9^{th} byte corresponds to second code byte which is already allocated. In this case, the 8^{th} bit in the code byte is set to zero and the 8^{th} byte in the compressed sequence represents just the 8-bit representation of the fragment. The process repeats from the 9^{th} byte onwards.

Since the input sequences are huge in size and the chunks of fragments are independent of each other, there is a scope for parallelization.

Parallel implementation:
The proposed algorithm has been implemented on multi-cores as well as on GPUs.

A. Multi-Core:

The algorithm is implemented using OpenMP on multi-core. The input sequence is distributed among the cores available and each core finds the repeated fragments and compresses the data allocating the bytes accordingly. The compressed sequence is stored in a buffer and is written into an output file.

B. GPUs:

The algorithm is implemented on GPUs using CUDA. The steps involved are as follows:

1) The resultant output array from the phase 1 is copied into the global memory of GPU from the CPU host memory.
2) The kernel is launched with the number of threads and the blocks varying according to the size of the given input sequence.
3) Each thread finds the repetitions and stores the result in a buffer in the global memory.
4) After all the threads finish their job, this buffer is copied from global memory to host memory.
5) From this, the compressed data is finally written to an output file which is done sequentially.

V. Experimental Setup

The serial code was run on Intel(R) Pentium(R) Dual core 2.20 GHz processor with 4GB RAM on Ubuntu 10.04 LTS. The parallel code was run on Lonestar supercomputer(TACC) which has over 22,000 cores with QDR InfiniBand networking (40Mb/s, sub-10us latency). Each core runs at 3.3 GHz (Intel Xeon, 12 MB L3 cache) and has 24 GB, 1333 MHz RAM per 12-core node. NVIDIA Tesla M2070 card with 448 cores and 6 GB global/device memory was used for GPU runs.

The data-sets used for the experiments are taken from GenBank (ftp://ftp.ncbi.nlm.nih.gov/genbank). This is a publicly available repository for DNA sequences. These sequences used in the experiments size upto 1.43 GB.

SIMD (Single Instruction Multiple Data) was used while parallelizing the code. The optimization flag O3 was used while compiling the code. The algorithm was implemented on multi-cores using 12 threads.

DNA Sequence	Input size (in Bytes)	Gen-Compress	DNA Compress	Genbit	**GenCodex**
HSCOMT2	1700	436	416	392	**377**
HUMCYC1A	2206	560	540	516	**496**
HSU37106	2256	573	561	546	**528**
HSGTRH	3938	995	967	918	**889**
HUMGALK1A	7086	1703	1708	1691	**1629**
HSU01102	4280	1035	1052	986	**950**
HSC1INHIB	16309	3789	3960	3575	**3465**
HSCST4	3489	869	842	832	**807**
HUMA1ATP	4786	1200	1171	1110	**1065**
HSTNT2	8657	2052	2049	2038	**1973**
HUMRBPA	8682	2143	2116	2070	**2000**
HUMHSKPQZ	2334	619	591	564	**544**
HUMRETBLAS	175019	40183	41688	39059	**37770**
HUMTBGA	6275	1594	1541	1486	**1441**
HSAT3	13347	3189	3250	2987	**2892**
D87675	285457	66649	68519	64537	**62384**

TABLE II
SIZE OF THE COMPRESSED SEQUENCE FOR DIFFERENT ALGORITHMS (IN BYTES)

DNA Sequence	Input Size (KB)	Gen-Compress	DNA Compress	Genbit	**GenCodex**
HSCOMT2	1.7	2.05	1.95	1.84	**1.77**
HUMCYC1A	2.2	2.03	1.95	1.87	**1.79**
HSU37106	2.25	2.03	1.98	1.93	**1.87**
HSGTRH	3.93	2.02	1.96	1.86	**1.8**
HUMGALK1A	7	1.92	1.92	1.90	**1.83**
HSU01102	4.2	1.93	1.96	1.84	**1.77**
HSC1INHIB	16.309	1.85	1.94	1.75	**1.69**
HSCST4	3.4	2	1.93	1.9	**1.85**
HUMA1ATP	4.786	2	1.95	1.85	**1.78**
HSTNT2	8.6	1.9	1.89	1.88	**1.82**
HUMRBPA	8.682	1.97	1.94	1.90	**1.84**
HUMHSKPQZ	2.3	2.12	2.02	1.93	**1.86**
HUMRETBLAS	175.019	1.83	1.90	1.78	**1.7**
HUMTBGA	6.275	2.03	1.96	1.89	**1.83**
HSAT3	13.34	1.91	1.94	1.79	**1.73**
D87675	285.457	1.86	1.92	1.81	**1.74**

TABLE III
COMPRESSION RATIOS FOR DIFFERENT DATA-SETS(IN BITS/BYTES)

VI. Results

The serial and parallel implementations of the algorithm were evaluated on data-sets of different sizes. We noticed that our algorithm performs better if the consecutive repetitions are more (upto 255 repetitions). Table II shows the compressed size in bytes for all the algorithms using different data-sets.

The compression ratio remains same for both the serial and parallel versions of our algorithm. We observed that the compression ratio of our algorithm is good when there are more repetitions. Table III shows the compression ratios in terms of bits/byte for different algorithms.

The parallel implementation outperformed the serial implementation in terms of the throughput (time taken to compute the data) for all the data-sets. The parallel version achieves a speedup of upto 11 on a 12-core M2070 card and upto 23 on M2070 GPUs for the data-sets used in our experiment. The results are shown in the Fig. 2. The experiments show that the algorithm scales well on GPUs and works better even for the huge sequences.

We observe that as the data size increases, GPUs perform better. This can be observed from the Table IV. This scalability is achieved because the work-load on the threads increases as the data-size increases on the multi-core.

VII. Conclusions and Future Work

A new compression algorithm is proposed to compress the DNA sequences. The main focus was on the throughput along with the compression ratio. As the number of consecutive repeats increases, the algorithm achieves the best compression. If the fragments are repeating only twice or there are no repetitions then the algorithm may not perform better. The compression ratio remains same for both the serial and parallel versions. We noticed a very good improvement in the throughput when the algorithm was implemented on multi-cores and GPUs.

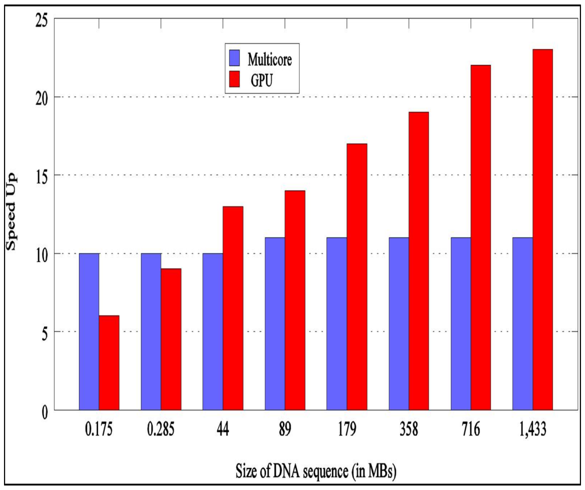

Fig. 2. Speedup on multi-cores and GPUs

Size of the Data	Sequential	Multi-core	GPU
175019	0.276	0.028	0.059
285457	0.456	0.047	0.056
44804864	8.682	1.97	1.94
89609728	141.401	13.7	10.573
179219456	283.237	27.44	17.355
358438912	567.532	57.225	30.268
716877824	1130.988	110.88	52.633
1433755648	2272.355	219.911	102.601

TABLE IV
TIMINGS(IN MILLISECONDS) ON MULTI-CORE AND GPUS

We observed a speedup of 11 on multi-cores and 23 on GPUs. Experiments showed us a good scalability on GPUs for the standard data-sets. The results show that our method achieves a good compression ratio along with better throughput compared to other existing methods.

The proposed algorithm can be extended to RNA sequences for compression. It also helps to calculate phylogeny. The GPU implementation can be used to solve the multiple-sequence alignment problem. A half-byte can be used instead of a full code-byte in order to save the space consumed by the extra code-byte when there are no repetitions.

Acknowledgment

We would like to dedicate this work to founder Chancellor of SSSIHL, Bhagawan Sri Sathya Sai Baba. Without His grace, this work would have remained a dream for us. This work was partially supported by a NVIDIA grant under professor partnership program and the Extreme Science and Engineering Discovery Environment(XSEDE), which is supported by National Science foundation grant number OCI-1053575.

References

[1] TC Bell et al. Prentice Hall, 1990.

[2] A Grumbach & F Tahi. Compression of dna sequences. In *Proceedings of the IEEE Data Compression Conference, Snowbird, UT, USA.*, March 30April 2. 1993.

[3] Ming Li Xin Chen, Sam Kwong. A compression algorithm for dna sequences. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, Tokyo, Japan*, April 8-11, 2000.

[4] Bin Ma Xin Chen 1, Ming Li and John Tromp. Dnacompress: fast and effective dna sequence compression. volume 18, pages 1696–1698, 2002.

[5] P Raja Rajeswari & Dr Allam AppaRao. Dnabit compress genome compression algorithm. In *Bioinformation*, volume 5, pages 350–360, 2011.

[6] P Raja Rajeswari & Dr Allam AppaRao. Genbit compress - algorithm for repetitive and non repetitive dna sequences. In *International Journal of Computer Science and Information Technology*, volume 2, pages 25–29, 2010.

[7] Sadakane K. Matsumoto, T. and H Imai. Biological sequence compression algorithms. In *Genome Informatics Workshop*, pages 43–52. Universal Academy Press, 2002.

[8] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. In *IEEE Trans. Inform. Theory*, volume 23, pages 337–343, 1977.