Volume 3, Issue 8, August 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Codon Optimization using Map Reduce

Jyotiska Nath Khasnabish
International Institute of Information Technology, Bangalore
26/C Hosur Road, Electronics City, Bangalore – 560100, KA, India

Abstract— Codon Optimization helps increasing immuno-stimulatory nucleotide motifs or decreasing immune-suppresive motifs in DNA expression vectors. This enhances the efficiency of DNA vaccinations. In this paper, we investigate the existing Pattern Matching algorithm for Codon Optimization and try to rewrite the algorithm in MapReduce programming model for execution speed-up and faster computation.

Keywords— codon optimization, pattern matching, MapReduce.

I. INTRODUCTION

Codon Optimization [1] helps increasing the efficiency of DNA vaccines which has the ability to induce humoral and cellular immune responses in experimental animals and humans. But it has shown poor performance in case of vaccinations of larger animals. The immune enhancers which are also known as 'Adjuvants' are used to enhance antigen specific immune responses in animals. The lack of suitable adjuvants leads to inefficient performance for DNA vaccines.

A 'Codon' [1] is a triplet of DNA symbols (A, C, G, and T). The size of DNA alphabet being 4, there are 64 possible codons. The objective of this paper is to find out the best triplet codon for codon optimization given a DNA sequence. 'Genetic Code' is the mapping between DNA sequences to amino acid sequences. The pattern matching algorithm proposed by Ravi Vijaya Satya et al finds highest occurring codon which is also termed as 'Best Codon' given an amino acid sequence and a list of motifs. In the section 2 of this paper, we discuss the previous work done by various research organizations on Codon Optimization using various algorithms such as Pattern Matching, Dynamic Programming [2]. Also, we discuss several works on Bioinformatics which uses MapReduce programming model. CloudBLAST [3] and bCloudBLAST [4] are examples such cloud based approaches which uses MapReduce to parallelize tools and manage execution and machine virtualization to encapsulate the execution environment.

In section 3, we discuss the Pattern Matching algorithm proposed by Ravi Vijaya Satya et al to find out the 'Best Codon' given a DNA sequence. In section 4 and 5, we discuss about the MapReduce [5] programming paradigm and our proposed algorithm to the existing pattern matching algorithm in MapReduce.

In the next section, we show the results we received from our experiments with different datasets [6]. The datasets include DNA sequence of size 10000, 20000, 50000, 100000, 150000, 200000 with three iterations. We show the execution time of both the algorithms with respect to each dataset using a graph.

II. RELATED WORKS

This paper discusses the work by Ravi Vijaya Satya et al [1] where they design a pattern matching algorithm for Codon Optimization and CpG Motif Engineering. The objective of their algorithm was to (1) identify the best triplet codon for codon optimization and (2) engineer the nature and content of the CpG motifs of a gene expressed from a genetic vector.

Matsunaga et al [3] have combined MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. In their paper, they propose and evaluate an approach to the parallelization, deployment and management of bioinformatics applications that integrates several emerging technologies for distributed computing. They used MapReduce to parallelize tools and manage their execution. In the work of Meng et al [4] they present an improved MapReduce parallel implementation of BLAST program which are widely used tools for searching protein and DNA databases for sequence similarities. Their implementation, bCloudBLAST, shows very good scaling and speedup behaviour on large sequence databases.

III. BEST CODON ALGORITHM

In the work by Ravi Vijaya Satya et al [1] they discuss about a pattern matching algorithm for Codon Optimization. Here in this paper, we only concentrate on the part to find out the best triplet codon for codon optimization. Given a DNA sequence as input, we find all possible codons from the sequence. From the codons, we find the one which has the highest frequency or has maximum occurrence. We call that codons as the 'Best Codon'. The algorithm is as following:

Input: DNA sequence (S) $\sum N = \{A,C,G,T\}$
Dictionary D = {codon, freq}


```
Begin:
    for i in Sequence S:
        codon = S[i], S[i+1], S[i+2]
        if codon not in D:
         D[codon] = 1
        else:
         D[codon] = D[codon] + 1
        i = i + 1

    Sort dictionary D w.r.t freq
End
Output: Top element in dictionary D
```

In the next sections, we discuss how we can optimize this algorithm using MapReduce programming model for faster execution.

IV. MAPREDUCE

MapReduce [5] is a programming model developed by Google for massive scale data processing on large clusters. It works on two functions: Map and Reduce. Map function processes a key/value pair to generate a set of intermediate key/value pairs. Reduce function merges all intermediate values associated with same intermediate key. The algorithms pertaining to a certain real world problem needs to be re-designed to be expressible in this model by writing Map and Reduce functions separately. MapReduce paradigm was designed to process huge amounts of data and the programs written in functional style are automatically parallelized and executed on a large clusters. In this way, the programmers does not need to know in details about distributed computing to write a program in this model. MapReduce programs are highly scalable and they are designed to run across several computers which makes it efficient to use.

The whole MapReduce computation takes a set of input key/value pairs and produces a set of output key/value pairs. In Map, the function takes an input pair and produces a set of intermediate key/value pairs. The library then groups all the pairs associated with some intermediate key I and passes them to the Reduce function. In Reduce, the function accepts an intermediate key I and a set of values for that key. Then the reducer merges together these values to a smaller set of values. The intermediate values are supplied to the user's reduce function via an iterator [5].

Many real life algorithms such as Distributed Grep, PageRank, Inverted Index and Sort are written in MapReduce for faster performance when deployed in a distributed environment [5].

V. PROPOSED ALGORITHM

In this paper, we propose an alternative pattern matching algorithm to find the 'Best Codon' given a DNA sequence using the MapReduce model. We have implemented the Map and Reduce function as following:

```
function map(codon):
    key = codon
    value = 1
    emit_intermediate(key, value)

function reduce(key, list_of_values):
    freq = length(list_of_values)
    emit(key, freq)

function execute(dna_seq):
    for i in dna_seq:
      codon = dna_seq[i], dna_seq[i+1],
        dna_seq[i+2]
      map(codon)
      i = i + 1

 for key in intermediate:
    reducer(key, intermediate[key])
```

VI. RESULTS

We have tested both the existing pattern matching algorithm and the proposed algorithm writing the program in Python (v2.7) and testing with different datasets.

Following is the configuration for testing:

TABLE 1
System Configuration

Processing Speed	2.3 GHz
Primary Memory (RAM)	6 GB
Disk Space	500 GB
Operating System	Ubuntu 12.04
Python Version	v2.7

We have tested both the programs with large datasets [6] to monitor and compare the program execution time in milliseconds. Here is the results we have received with the existing pattern matching algorithm to find out the 'Best Codon':

TABLE 2
Execution time using existing algorithm

Length of Sequence	Iteration 1 (ms)	Iteration 2 (ms)	Iteration 3 (ms)	Average (ms)
10,000	41.574	42.088	44.057	42.573
20,000	78.387	77.815	77.718	77.973
50,000	200.390	202.053	203.129	202.524
100,000	399.029	401.962	399.055	400.015
150,000	600.644	594.919	595.999	596.887
200,000	901.499	903.152	895.027	899.892

Here is the results we have found when we executed our MapReduce program –

TABLE 3
Execution time using MapReduce

Length of Sequence	Iteration 1 (ms)	Iteration 2 (ms)	Iteration 3 (ms)	Average (ms)
10,000	13.916	14.052	14.150	14.039
20,000	24.949	25.112	25.147	25.069
50,000	62.147	61.181	61.451	61.593
100,000	121.365	122.215	123.855	122.478
150,000	182.070	179.453	183.523	181.682
200,000	268.434	268.666	270.299	269.133

We have shown comparison of program execution time between both the implementations in the following graph:

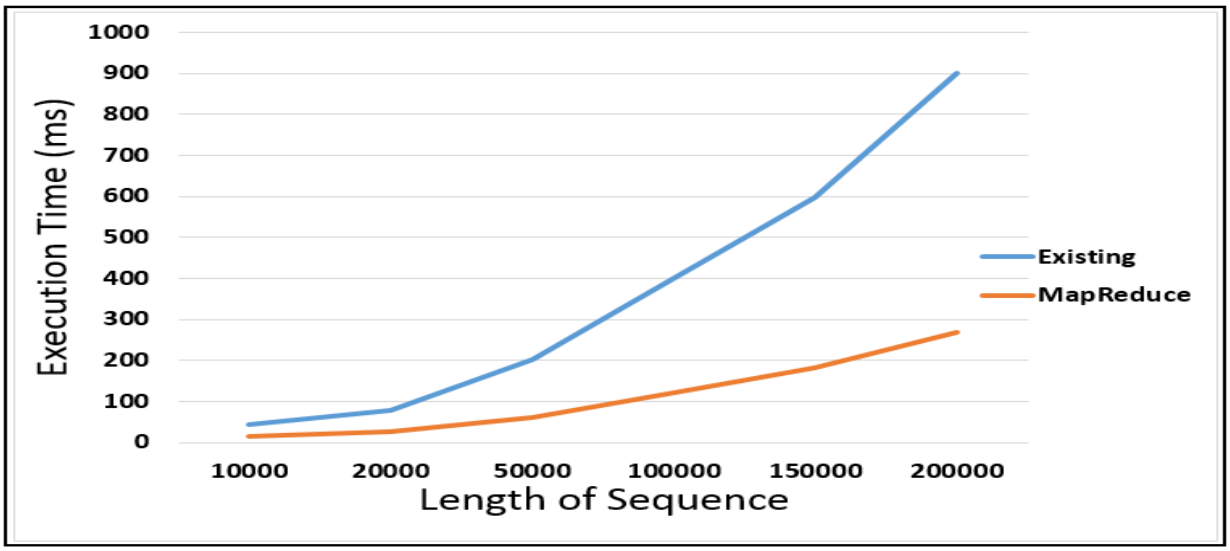

Fig. 1 Comparison between execution speeds of two algorithms

VII. CONCLUSION

We can see from the results that we gain significant performance improvement when we try to find out the 'Best Codon' using MapReduce program. When the length of sequence reaches 200,000 we gain more than 300% speedup from the existing pattern matching algorithm. The reason for this is the efficient use of Map and Reduce function to first calculate all the codon and then finding out the best codon from the list of codons. This ensures that we will be getting more speedup when we deploy our program with bigger sequences and use it in a distributed environment as MapReduce provides excellent scalability.

VIII. FUTURE WORK

We propose a number of future works to improve and scale this implementation –

1. Use of MapReduce on Hadoop and HDFS for more scalability, fault-tolerance and faster execution speed.
2. Use of longer DNA sequences to test the program.
3. Execute the program in a distributed environment to monitor scalability.

REFERENCES

[1] Ravi Vijaya Satya, Amar Mukherjee and Udaykumar Ranga. (2003): 'A Pattern Matching Algorithm for Codon Optimization and CpG Motif Engineering in DNA Expression Vectors', Proceedings of the Computational Systems Bioinformatics, 2003.

[2] Tuan D. Pham, James O' Connell and Denis I. Crane (2004): 'Constrained Codon Optimization by Dynamic Programming', Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing, 2004, Hong Kong.

[3] Andréa Matsunaga, Maurício Tsugawa and José Fortes (2008): 'CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications', Fourth IEEE International Conference on eScience, 2008.

[4] Zhen Meng, Jianhui Li, Yunchun Zhou, Qi Liu, Yong Liu, and Wei Cao (2011): 'bCloudBLAST: an Efficient MapReduce Program for Bioinformatics Applications', 4th International Conference on Biomedical Engineering and Informatics (BMEI), 2011.

[5] Jeffrey Dean and Sanjay Ghemawat (2004): 'MapReduce: Simplified Data Processing on Large Clusters', 6thSymposium on Operating System Design and Implementation (OSDI '04), 2004.

[6] Nucleotide Database, National Center for Biotechnology Information (http:// www.ncbi.nlm.nih.gov/nuccore)

