International Journal For Technological Research In Engineering
Volume 2, Issue 7, March-2015 *ISSN (Online): 2347 - 4718*

HIGH PERFORMANCE COMPUTING APPROACH FOR MOTIF SEARCH USING OPENMP AND OPENMPI

Prafulla K. Mendhe[1], Deepti Shrimankar[2]
Department of Computer Science & Engineering, VNIT, Nagpur, India

Abstract: Meaningful Patterns, or Motifs, often exist in large volumes of biological data. They are theorized to be highly significant elements in protein and DNA and are important for understanding the function of particular sequences. Identifying these motifs can help researchers to accurately target human disease with therapeutic drugs and understand gene function more precisely. There have been several identified problems, but this project looks primarily at parallel optimizations for the planted (l, d)-motif problem. In PMS we are given two integer l and d and n biological sequences. We want to find out all sequences of length l that appears in each of the input sequences with at most d mismatches. Being able to quickly identify these motifs is difficult due to the nature of the problem. The number of comparisons we need to make grows exponentially with d. It quickly becomes impractical to attempt sequential computation of the motif search as d grows. This project demonstrates the practical application of parallel techniques using openMP and openMPI with cluster computer which shows the degree to which performance can be improved through loop and block-level parallelization. It also discusses how these techniques can be applied to other similar problems in the field of bioinformatics.

Keywords: motif search, openMP, openMPI, high performance computing HPC, cluster, PMS algorithm.

I. INTRODUCTION

The detection of similarities between different DNA sequences can provide insight into the function and significance of those sequences. One category of such similarities is the motif, a pattern that recurs across many DNA sequences. Motifs can represent meaningful structures such as transcription factor binding sites [1]. Therefore, it is of interest to biologists and researchers to be able to quickly and accurately detect motifs in large collections of genetic data. The Planted Motif Search problem has been formalized to abstract the process of finding motifs. Planted-(l, d) Motif Search Problem Definition. The required inputs are: a list of n sequences, each of length m, to be examined for motifs; an integer l, which specifies the length of the desired motif; and an integer d specifying the Hamming distance permitted for any instance of the motif in an input sequence. Numerous algorithms have been proposed and implemented to detect motifs in DNA sequences using a variety of approaches [2]. Algorithms such as MEME [3] And Profile Branching [4] take a profile-based approach. They predict the positions at which the motif appears in each sequence. Other algorithms, like PairMotif [5], MITRA [6], and PROJECTION [7], predict the motif itself and are known as pattern based algorithms. This paper will present a number of proposed optimizations that can be applied in many approaches to the Planted Motif Search problem. While these modifications will not alter the asymptotic runtime of a given algorithm, they can provide considerable practical improvement over naïve approaches. The PMS algorithm and proposed improvements developed by developed by Rajasekaran et al. are the framework on which these implementations are based. The optimizations are applied incrementally, so the improvements can be analyzed separately. A large number of sequential algorithms have been proposed for PMP. These algorithms can be categorized into approximation algorithms, and exact algorithms. In approximation algorithms, the planted motif may not be extracted, whereas in exact algorithms, the planted motif can always be extracted. The approximation algorithms employ heuristic techniques such as graph [8], Expectation Maximization [9], [11], Gibbs Sampling [10], divide and conquer [12], combinatorial approaches [13], statistical methods [14], and profile-based search [15]. The exact algorithms employ exhaustive search techniques such as suffix tree [16], dynamic programming [17], hashing [19], and sorting [20]. As a result of exhaustive search, some of the earlier exact algorithms are impractical in terms of time or memory requirements for the challenging instances of PMP. The recent exact algorithms voting [19], PMSprune [21], PMS5 [22], HEPPMSprune [23], PMS6 [24], Modified Voting [26], CVoting [25] and qPMS7 [27] propose practical solutions for the challenging instances. In this paper, we present a simple method for parallelizing the recent efficient motif finding algorithm, PMS. We implement the parallel algorithm on a multi-core architecture using openMP and openMPI. The experimental results show that, the scalability of our parallel implementation is approximately optimal and independent of number of processors or the motif length. Also the scalability increases with the number of processors and number of cores.

II. PMS ALGORITHM

In this section, we briefly describe PMS algorithm, which has great properties in terms of space and running time complexities. For any l-mer u we define its d-neighborhood as the set of l-mers v for which $Hd(u, v) \leq d$. For any set of l-mers T we define the common d-neighborhood of T as the intersection of the d-neighborhoods of all l-mers in T. To compute common neighborhoods, a natural approach is to traverse the tree of all possible l-mers and identify the common neighbors. A node at depth k, which represents a k-

mer, is not explored deeper if certain pruning conditions are met. Thus, the better the pruning conditions are, the faster will be the algorithm.

```
Algorithm 1. GenerateNeighborhood(T, d)
for (i = 1..|T|) do ri:= d;
GenerateNeighborhood(T, r, 1)
GenerateNeighborhood(T, r, p)
 if (p ≤ l) then
  if (not prune(T, r)) then
   for α ϵ ∑ do
    xp := α
     for (i = 1..|T|) do
       T'i := Ti[2..|S|]
        r'i := ri;
        if (Ti[0] ≠ α) then r'i:=r'i-1;
     end for
    GenerateNeighborhood(T ', r', p + 1)
   end for
  end if
else
   report l-mer x
end if
Algorithm 2. PMS(T, d)
   for (i = 1..n) do Ri= {u|u ϵ Si}
   stack = {}
  GenerateMotifs(1, stack, R)
GenerateMotifs(p, stack, R)
   for (u ϵ Rp) do
      stack.push(u)
      R' :=filter(R, stack)
      if (R'.size > 0) then
        if (ThresholdCondition) then
N :=GenerateNeighborhood(stack, d)
for (m ϵ N) do
  if (isMotif(m, R')) then output m;
       else
GenerateMotifs(p + 1, R')
    stack.pop()
end for
```

Fig. 1. PMS Algorithm

III. HPC APPROACH

We have created the HPC cluster of 5 computers; each computer is having the same configuration. Each computer has Intel core i3-2130 CPU @3.40 GHz, 3072 KB cache and running Ubuntu 12.04 OS and connected over Ethernet LAN.

A. Method to create openMPI Cluster

To be able to run a parallel computation on a network of computers via openMPI, one has to be able to log in to any of the machines without having to enter a password. This can be achieved easily using secure shell key authentication. We assume that the user want to run parallel jobs on a network of machines called master, slave1, slave2 etc. Master is the logging node, i.e. master is likely to be open to the Internet or to other machines.

- Giving each node a static ip will make things simpler. To make configuration easier, change every node's /etc/hosts file.
  127.0.0.1localhost
  192.168.1.100master
  192.168.1.101slave1
  192.168.1.102 slave2
- Installing and configuring a shared file system (master)
  - install nfs server on master node
    sudo apt-get install -y nfs-kernel-server
  - set up a shared directory on master
    sudo mkdir /share
    echo /share *(rw,sync) | sudo tee –a /etc/exports
    sudo /etc/init.d/nfs-kernel-server restart
- Installing and configuring a shared file system (slave)
  a. install nfs on slave nodes
     sudo apt-get install -y nfs-common
  b. mount the shared directory from slave nodes.
     sudo mount master:/share /share
- Create a user (all nodes)
  Add a user that will run MPI programs. This user has to have the same name on all the nodes, the same UID, and has to have his home diretory as /share.
  sudo adduser --home /share/mpiuser --UID 999 mpiuser
  sudo chown mpiuser /share
- Install SSH server(all nodes)
  sudo aptitude install -y openssh-server
- Set up public key authentication(master)
  su - mpiuser
  ssh-keygen -t rsa
  cd .ssh
  cat id_rsa.pub >> authorized_keys
  chmod 700 ~/.ssh
  chmod 600 ~/.ssh/authorized_keys
- Installing software(all nodes)
  sudo apt-get install -y gcc g++ build-essential
  sudo apt-get install libopenmpi-dev openmpi-bin openmpi-common openmpi-doc

B. Parallel Implementation

To parallelize PMS using openMP and openMPI we create m - l + 1 sub problems, one for each l-mer in the first string. The first string in each sub problem is an l-mer of the original first string and the rest of the strings are the same as in the original input. The processor with rank 0 is a master and the others are slaves. The master spawns a separate slave thread to avoid using one processor just for scheduling. The master reads the input and broadcasts it to all slaves. Then each slave requests a sub problem from the master, solves it and repeats. The master loops until all jobs have been requested and all slaves have been notified that no more jobs are available. At the end, all processors send their motifs to

the master which outputs them. As each machine having 2 cores we set OMP_NUM_THREADS = 2 and using loop level and block level parallelism in slave nodes we increases the performance. The algorithm for master slave is shown below.

Master:

```
spawn(Slave0)
read(input)
broadcast(input)
while (job < nJobs + N)
recv(w, JobRequest)
send(w, job++)
join(slave0)
for i = 1 to N-1
recv(i, motifs)
```

Slave:

```
broadcast(input)
do
send(0, JobRequest)
recv(0, job)
        if (job < nJobs)
process(job)
while (job < nJobs)
send(0, motifs)
```

Fig. 2. Parallel PMS algorithm

IV. RESULTS AND DISCUSSION

In this section, we present our experiments for motif search on different sets of databases. The PMS is implemented on a HPC cluster of 5 computers. Each computer has Intel core i3-2130 CPU @3.40 GHz and 3072 KB cache and running 12.04 Ubuntu OS. The program is coded in C++ language with openMP and openMPI directives.

A. Experimental result on Simulated Data Set

The input sequences are generated by using simulated data sets with parameters t =20sequences and m = 600 characters, where the characters are A, C, G, T. Each (l, d) input instance dataset is generated as follows: We generate random strings with length (m - l) each, where the characters appear randomly with equal probability. Then we generate randomly an l length string M and plant a copy of it in each sequence at random position after mutating it with at most d random mutations.

C / Instances	1	2	3	4	5
(15,5)	33.48	16.32	15.5	13.82	12.4
(21,7)	20.6	13	12	11.32	10.53
(23,8)	92.6	57.5	51.6	47.4	45
(26,9)	171	94	89	80	75
(28,10)	402	280	250	230	198
(30,11)	1811	1300	1200	1030	880

TABLE I: Running time in seconds for different challenging instances. C is the number of computers in HPC.

B. Experimental result on Real Data Set

We test PMS on a set of real biological data which are used in the literature [26]. The data for this set contains the upstream DNA regions of a set of genes from different species.

Gene Family	Instance (*l*, *d*)	Motif detected
Insulin	(11,1)	CCTCAGCCCCT CTAGCCATCTG CTTCAGCCCCC GCCATCTGCCG GCCTCAGCCCC GGCCATCTGCC TAGCCATCTGC
Growth Hormone	(10,2)	ATGTATAAAA TATAAAAAGA
c-fos	(11,1)	ACAGGATGTAC CCATATTAGGA
PDR3	(10,1)	TTCCGCGGAA
Histone H1	(11,1)	AAACAAAAGTG AGACAAAAGTT TAAACAAAAGT
ECB	(10,1)	AAAAATTATT AGTAAAAAAA
PHO4	(13,2)	AAAAATAGAAGTG AAAATAAATGTGA AAAGAAATTTATC AATAAAGCAAGAA AATGCAATAAAAA AATGCGATAAAAA ATAAAGCAAGAAA
c-myc	(10,1)	AAGAAAAAAA AGAAAAAAAA GAAAAAAAAA
MCB	(9,1)	AAGAATAAA ATAAAAGAA TTCTTTCTT TTCTTTGTT

TABLE II: Motifs Detected in real biological dataset.

CONCLUSION

In this paper, we have presented a simple scalable and efficient parallel openMP and openMPI implementation for PMS algorithm using cluster computer. Also we have presented the method for creating HPC cluster. The efficiency of the algorithm is validated by testing it on both simulated as well as real biological databases. The result proved that the speed up increases with the increase of cluster computers.

REFERENCES

[1] Zambelli, F., Pesole, G., Pavesi, G.: Motif Discovery and Transcription Factor Binding Sites Before and After the Next-Generation Sequencing Era. Briefings in Bioinformatics Vol. 14 (2013) 225-237

[2] Das, M., Dai, H.-K.: A Survey of DNA Motif Finding Algorithms. BMC Bioinformatics, Vol. 8 Suppl. 7. (2007) S21

[3] Bailey, T.L., Elkan, C.: Fitting a Mixture Model by Expectation Maximization to Discover Motifs in

Biopolymers. Proceedings of the 2nd International Conference on Intelligent Systems for Molecular Biology (1994) 28-36

[4] Price, A., Ramabhadran, S., Pevzner, P.A.: Finding Subtle Motifs by Branching from Sample Strings. Bioinformatics Vol. 1 (2003) 1-7

[5] Yu, Q., Huo, H., Zhang, Y., Guo, H.: PairMotif: A New Pattern-Driven Algorithm for Planted (l,d) DNA Motif Search. Public Library of Science Vol. 7 Issue 10 (2012) Special Section 1-10

[6] Eskin, E., Pevzner, P.: Finding Composite Regulatory Patterns in DNA Sequences. Bioinformatics S1 (2002) 354-363

[7] Buhler, J., Tompa, M.: Finding Motifs Using Random Projections. Proceedings of the 5th Annual Conference on Computational Molecular Biology (RECOMB) (2001)

[8] P. A. Pevzner, S.-H. Sze et al., "Combinatorial approaches to finding subtle signals in dna sequences." in ISMB, vol. 8, 2000, pp. 269–278.

[9] C. E. Lawrence and A. A. Reilly, "An expectation maximization (em) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences," Proteins: Structure, Function, and Bioinformatics, vol. 7, no. 1, pp. 41–51, 1990.

[10] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, "Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment," science, vol. 262, no. 5131, pp. 208–214, 1993.

[11] T. L. Bailey and C. Elkan, "Unsupervised learning of multiple motifs in biopolymers using expectation maximization," Machine learning, vol. 21, no. 1-2, pp. 51–80, 1995.

[12] Y. M. Fraenkel, Y. Mandel, D. Friedberg, and H. Margalit, "Identification of common motifs in unaligned dna sequences: application to Escherichia coli lrp regulon," Computer applications in the biosciences: CABIOS, vol. 11, no. 4, pp. 379–387, 1995.

[13] I. Rigoutsos and A. Floratos, "Combinatorial pattern discovery in biological sequences: The teiresias algorithm." Bioinformatics, vol. 14, no. 1, pp. 55–67, 1998.

[14] M. Gelfand, E. Koonin, and A. Mironov, "Prediction of transcription regulatory sites in archaea by a comparative genomic approach," Nucleic Acids Research, vol. 28, no. 3, pp. 695–705, 2000.

[15] C.-W. Huang, W.-S. Lee, and S.-Y. Hsieh, "An improved heuristic algorithm for finding motif signals in dna sequences," IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), vol. 8, no. 4, pp. 959–975, 2011.

[16] M.-F. Sagot, "Spelling approximate repeated or common motifs using a suffix tree," in LATIN'98: Theoretical Informatics. Springer, 1998, pp. 374–390.

[17] M. Blanchette, B. Schwikowski, and M. Tompa, "Algorithms for phylogenetic footprinting," Journal of Computational Biology, vol. 9, no. 2, pp. 211–223, 2002.

[18] A. M. Carvalho, A. T. Freitas, A. L. Oliveira, M.-F. Sagot et al.,"A highly scalable algorithm for the extraction of cis-regulatory regions." in APBC, 2005, pp. 273–282.

[19] F. Y. Chin and H. C. Leung, "Voting algorithms for discovering long motifs." in APBC, 2005, pp. 261–271.

[20] S. Rajasekaran, S. Balla, and C.-H. Huang, "Exact algorithms for planted motif problems," Journal of Computational Biology, vol. 12, no. 8, pp. 1117–1128, 2005.

[21] J. Davila, S. Balla, and S. Rajasekaran, "Fast and practical algorithms for planted (l, d) motif search," Computational Biology and Bioinformatics, IEEE/ACM Transactions on, vol. 4, no. 4, pp. 544–552, 2007.

[22] H. Dinh, S. Rajasekaran, and V. K. Kundeti, "Pms5: an efficient exact algorithm for the (, d)-motif finding problem," BMC bioinformatics, vol. 12, no. 1, p. 410, 2011.

[23] M. M. Abbas, M. Abouelhoda, and H. M. Bahig, "A hybrid method for the exact planted (l, d) motif finding problem and its parallelization," BMC bioinformatics, vol. 13, no. Suppl 17, p. S10, 2012.

[24] S. Bandyopadhyay, S. Sahni, and S. Rajasekaran, "Pms6: A fast algorithm for motif discovery," in Computational Advances in Bio and Medical Sciences (ICCABS), 2012 IEEE 2nd International Conference on. IEEE, 2012, pp. 1–6.

[25] Y. Xu, J. Yang, Y. Zhao, and Y. Shang, "An improved voting algorithm for planted (l, d) motif search," Information Sciences, vol. 237, pp. 305–312, 2013.

[26] M. M. Abbass and H. M. Bahig, "An efficient algorithm to identify dna motifs," Mathematics in Computer Science, vol. 7, no. 4, pp. 387–399, 2013.

[27] H. Dinh, S. Rajasekaran, and J. Davila, "qpms7: A fast algorithm for finding (l, d)-motifs in dna and protein sequences," PloS one, vol. 7, no. 7, p. e41425, 2012.

[28] Marius Nicolae and Sanguthevar Rajasekaran, "Efficient sequential and parallel algorithms for planted motif search" BMC Bioinformatics 2014.

[29] Mostafa M.Abbas, Qutaibah M. Malluhi, P. Balakrishnan, "Scalable Multi-Core Implementation for Motif Finding Problems" 13th International Symposium on Parallel and Distributed Computing 2014.

[30] H. Faheem, "Accelerating motif finding problem using grid computing with enhanced brute force," in Advanced Communication Technology (ICACT), 2010 The 12th International Conference on, vol. 1. IEEE, 2010, pp. 197–202.

