

Search of Maximal Unique Matches in Multi-core architectures

Julio César García
Vizcaíno
Computer Architecture and
Operating System
Campus UAB, Edifici Q, 08193
Bellaterra (Barcelona), Spain.
juliocesar.garcia@caos.uab.es

Juan Carlos Moure
Computer Architecture and
Operating System
Campus UAB, Edifici Q, 08193
Bellaterra (Barcelona), Spain.
juancarlos.moure@uab.es

Antonio Espinosa
Computer Architecture and
Operating System
Campus UAB, Edifici Q, 08193
Bellaterra (Barcelona), Spain.
antonio.espinosa@caos.uab.es

Porfidio Hernández
Computer Architecture and
Operating System
Campus UAB, Edifici Q, 08193
Bellaterra (Barcelona), Spain.
porfidio.hernandez@uab.es

ABSTRACT

Maximal Unique Matches are common substrings that match a reference and a query genome. They are exact, unique and maximal. The search of MUMs in large genomes is a heavy and repetitive task, so there is a fair chance of parallelize and execute this search in multi-core architectures. This research resembles an approach to find MUMs in genomic sequences and comparing the search in two data structures within multi-core architectures. The reference genome is indexed by using a suffix tree or enhanced suffix array in main memory and then parallelized algorithm finds the MUMs of query genome which is readed by several threads in chunks of fixed size. This approach is based on MUMmer, a genome alignment tool, which is able to find Maximal Unique Matches (MUMs).

Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous;
D.2.8 [Software Engineering]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

Indexed Search, Bioinformatics, Maximal Unique Match, Multi-core architectures, Parallelization

1. INTRODUCTION

Modern sequencing and computational technologies and advances in bioinformatics has made whole genome sequencing possible. One resulting challenge is the fast alignment of whole genomes. Dynamic programming is too slow for aligning two large genomes. One very successful approach to perform whole genome alignment is based on identifying “maximal unique matches”, which is based on the assumption that one expects to substrings occurring in two similar genomes. Maximal unique matches (MUMs) are almost surely part of a good alignment of the two sequences and so the whole genome alignment problem can be reduced to aligning the sequence in the gaps between the MUMs.

Definition 1. Assume we are given two sequences $R, Q \in \Sigma^*$, and a number $L > 0$. The maximal unique matches problem (MUM-problem) is to find all sequences $u \in \Sigma^*$ with: $|u| \geq L$, u occurs exactly once in R and once in Q , and for any character $a \in \Sigma^*$ neither ua nor au occurs both in R and Q .

The problem of searching maximal unique matches for a minimum length between a reference string and a query string has been identified in several applications, one of them is MUMmer [3]. MUMmer’s algorithm can perform searches of maximal unique matches (MUMs), although with a high use of main memory to store the reference string and a null use of multi-core architectures.

2. RELATED WORK

Search of Maximal Unique Matches to do Whole Genome Alignment was proposed in [2]. There have been some previous work in the parallelization of search of matches in genomic data, like [11, 9, 7], however these works are focused in fixed patterns and read alignment. On the other hand, there have been achievements in parallelization of Whole genome alignment like [8]. The parallelization of searching MUMs with a full-text index data structure is a research field not covered very deep, there is only one approach in [4]

but without access to source code to check their implementation and it is more focused with GPU and CPU hybrid architectures and suffix array. There are other implementations to search Maximal exact matches with threads in [12, 11, 6, 10]

3. PRELIMINARIES

Let's assume the reference string $R[0, \dots, n-1]$ of size n over an alphabet $\Sigma = \$, A, C, G, T$ which has a sentinel character $R[n] = \$$ that occurs nowhere else in the string and is lexicographically less than all the characters that occur in the alphabet. The suffixes of the reference string are zero indexed by their position in the original string by a full-text index data structure like a suffix tree or a suffix array.

Definition 2. A suffix tree, ST , for an n -character string R is a rooted directed tree with exactly n leaves numbered 0 to n . Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of R . No two edges out of a node can have edge-labels beginning with the same character. For any leaf i , the concatenation of the edge-labels on the path from the root to leaf i exactly spells out the suffix of R that starts at position i . That is, it spells out $R[i \dots n]$. [5].

Definition 3. Given an n -character string R , a suffix array, SA , for R is an array of the integers in the range 0 to n , specifying the lexicographic order of the n suffixes of string R . [5].

A search for a MUM between a Reference, R of length n , and Query, Q of length m , genome can be done in a suffix tree in $O(n + m)$ time and in a suffix array in $O(n \log m)$ time. However is possible to improve the search for MUMs algorithm in a suffix array by using: an inverse suffix array, a LCP-table and a child table. These improvements give a $O(m)$ time [1] to search for MUMs in a suffix array.

Definition 4. An inverse suffix array, ISA , is a table of size $n + 1$ such that $ISA[SA[q]] = q$ for any $0 \leq q \leq n$. [1].

Definition 5. A LCP-table, LCP , is an array of integers from 0 to n . Where $LCP[0] = 0$ and $LCP[i]$ is the length of the longest common prefix of $SA[i - 1]$ and $SA[i]$ for $1 \leq i \leq n$. [1].

Definition 6. A child table, $CHILD$, is a table of size $n + 1$ indexed from 0 to n and each entry contains three values: up, down, and nextIndex. These values are defined as follows:

$$CHILD[i].up = \min\{q \in [0 \dots i-1] | LCP[q] > LCP[i] \text{ and } \forall k \in [q + 1 \dots i - 1] : LCP[k] \geq LCP[q]\},$$

$$CHILD[i].down = \max\{q \in [i+1 \dots n] | LCP[q] > LCP[i] \text{ and } \forall k \in [i + 1 \dots q - 1] : LCP[k] \geq LCP[q]\},$$

$$CHILD[i].nextIndex = \min\{q \in [i+1 \dots n] | LCP[q] = LCP[i] \text{ and } \forall k \in [i + 1 \dots q - 1] : LCP[k] > LCP[i]\}.$$

To simulate a traversal of a suffix tree on an enhanced suffix array we require the child table. In order to perform the traversal we get an interval l -interval $[i..j]$ and a character $a \in \Sigma$ as input and returns the child interval $[l..r]$ of $[i..j]$ whose suffixes have the character a at position l . Note that all the suffixes in $[l..r]$ share the same l -character prefix because $[l..r]$ is a subinterval of $[i..j]$. This process is performed in $O(|\Sigma|)$ time.

Once we have defined the resources used to index a Reference genome, we need to answer the question: Where are the MUMs of R and Q of some minimum length L ? We show the algorithms used to answer this question using a Suffix Tree and a Enhanced Suffix Array.

Algorithm 1: Search for MUMs in a Suffix Tree.

```

input : R, Q, L
output: List of MUMs of length  $\geq L$ , with start position in R and Q and length

1 begin
2   Build suffix tree for R genome
3   foreach position  $i \in Q$  do
4     length  $\leftarrow$  TraverseSuffixTree( $Q[i]$ )
5     if isLeafNodeQ[i] then
6       if  $R[\text{leaf} - 1] \neq Q[i - 1]$  and length  $\geq L$  then
7         saveMUMcand( $R[\text{leaf}], i, \text{length}$ )
8   MUMs  $\leftarrow$  cleanMUMcand
9 end

```

4. CONCLUSIONS

5. ACKNOWLEDGMENTS

This work was supported by grant from Ejecución eficiente de aplicaciones multidisciplinares: nuevos desafíos en la era multi-/many core, with reference TIN2011-28689-C02-01.

6. REFERENCES

- [1] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, Mar. 2004.
- [2] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg. Alignment of whole genomes MUMMER. *Nucl Acids Res*, 27(11):2369, 1999.
- [3] A. L. Delcher, S. L. Salzberg, and A. M. Phillippy. Using MUMmer to identify similar regions in large sequence sets. *Current protocols in bioinformatics editorial board Andreas D Baxevanis et al*, Chapter 10(1934-340X (Electronic) LA - eng PT - Journal Article SB - IM):Unit 10.3, 2003.
- [4] G. Encarnac and N. Roma. Advantages and GPU Implementation of High-Performance Indexed DNA Search based on Suffix Arrays. *Architecture*, pages 49–55, 2011.
- [5] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*, volume 24. Cambridge University Press, 1997.
- [6] Z. Khan, J. S. Bloom, L. Kruglyak, and M. Singh. A practical algorithm for finding maximal exact matches in large sequence datasets using sparse suffix arrays. *Bioinformatics*, 25(13):1609–1616, July 2009.

- [7] C. S. Kouzinopoulos, P. D. Michailidis, and K. G. Margaritis. Parallel Processing of Multiple Pattern Matching Algorithms for Biological Sequences : Methods and Performance Results. 2005.
- [8] X. Meng and V. Chaudhary. Exploiting multi-level parallelism for homology search using general purpose processors. *October*, 2005.
- [9] H. Mongelli. Efficient Two-Dimensional Parallel Pattern Matching with Scaling. *North*.
- [10] E. Ohlebusch, S. Gog, and A. Kügel. Computing matching statistics and maximal exact matches on compressed full-text indexes. In *SPIRE*, pages 347–358, 2010.
- [11] M. Oğuzhan Külekci, W.-K. Hon, R. Shah, J. Scott Vitter, and B. Xu. Ψ -RA: a parallel sparse index for genomic read alignment. *BMC genomics*, 12 Suppl 2:S7, Jan. 2011.
- [12] M. Vyverman, B. De Baets, V. Fack, and P. Dawyndt. essaMEM: finding maximal exact matches using enhanced sparse suffix arrays. *Bioinformatics (Oxford, England)*, pages 2–4, Feb. 2013.