

International Conference on Computational Science, ICCS 2012

# Towards speed up search of maximal unique matches in multicore architectures

---

## Abstract

Maximal Unique Matches are common substrings that are found between a reference and a query sequence. They are exact, unique and maximal; that is, they cannot be extended in left or right direction without incurring a mismatch. The computation of MUMs in large sequences is a heavy and repetitive task because the genomes are closely related, so there is a fair chance of parallelize and execute this search in multicore architectures. This research resembles a first novel approach to find MUMs in genomic sequences in parallel way. The reference genome is indexed by using a suffix tree in main memory and then the parallelized algorithm finds the MUMs against a query genome which is readed by several threads. This approach is based on MUMmer, a genome alignment tool, which is able to find Maximal Unique Matches (MUMs).

*Keywords:*

---

## 1. Problem

The problem of searching maximal unique matching for a minimum length between a reference string and a query string has been identified in several applications, one of them is MUMmer. Although MUMmer's algorithm can perform searches of maximal unique matches (MUMs) the use of resources are not well used:

- High use of main memory to store the reference string.
- A null use of multicore architectures.

If the length of reference and query are very huge, the amount of operations to perform in the search of MUMs increases, see table 1. The use of parallelism could help reduce the execution time for the search of maximal unique

Data structure	L [bp <sup>1</sup> ]	Search operations	Search [s]	Memory usage [MB]
Suffix tree	20	$9,87 \times 10^{18}$	169189,4	48665,12

Table 1: Search of Maximal Unique Matches between a reference sequence (2960,21Mbp) and query sequence (2716,96Mbp)

matches. One approach of parallelism is to take advantage of multicore architectures nowadays.

This problem has a time complexity of  $O(m + k)$  where  $m$  is the length of the query sequence and  $k$  is the number of maximal unique matches of some minimum length. This problem is a very high intensive computing task, for every substring in the query sequence the search for a maximum unique match has to be performed.

## 2. The MUM: an heuristic approach

### 2.1. Definition MUM

Although a pair of conserved genes rarely contain the same entire sequence, they share a lot of short common substrings and some of them are indeed unique to this pair of genes. For example the following two sequences, R and Q:

R=ac ga ctc a gctac t ggtcagctatt acttaccgc\$  
 Q=ac tt ctc t gctac ggtcagctatt c acttaccgc\$

It is clear that sequences R and Q have many common substrings, they are:

- ac
- ctc
- gctac
- ggtcagctatt
- acttaccgc

Among those five common substrings, ac is the only substring that is not unique. It occurs more than once in both sequences. You can also observe that actually a, c, t, and g are common substrings of R and Q. However, they are not maximal, i.e. they are contained in at least one longer common substrings. We are only interested in those that are of maximal length.

Our aim is to search for all these short common substrings. Given genomes R and Q, we need to find all common substrings which are unique and of maximal length. Each of such common substrings is known as Maximum Unique Match (MUM). For almost every conserved gene pairs, there exist at least one MUM which is unique to them.

For example, assuming  $d = 3$ , sequences R and Q in the previous example has four MUMs: ctc, gctac, ggtcagctatt, acttaccgc. Substring ac is not an MUM because its length is smaller than the value of  $d$  and it is not unique to both sequences.

R=~~ac~~ ga ete a gctac t ggtcagctatt acttaccgc\$  
 Q=~~ac~~ tt ete t gctac ggtcagctatt c acttaccgc\$

The concept of MUM is important in whole genome alignment because a significantly long MUM is very likely to be part of the global alignment.

#### 2.1.1. Finding MUMs in a suffix tree

The key idea in this method is to build a suffix tree for genome R, a data structure which allows finding, extremely efficiently, all distinct subsequences in a given sequence.

By construction, the location of a match in the suffix tree represents a substring of the subject sequence which maximally matches a prefix of query suffix. Thus it is only necessary to verify that, the substring of the subject sequence is long enough, that it is unique in the subject sequence and that the match is also left maximal. This is done as follows:

1. Does location represent a substring of length at least minimum match length?
2. Does location correspond to a leaf edge? Then then the string represented by the location is unique in the subject sequence.
3. Is the substring left maximal? This is true if one of the following conditions hold:
  - The suffix of the query currently considered is the first suffix, or
  - The string represented by loc is a prefix of the subject string, or
  - The characters immediately to the left of the matching strings in the subject sequence and the query sequence are different

If all conditions 1-3 are true, then this MUM is stored in a list of MUM-candidates. It takes the necessary information about the MUM-candidate:

- Position in reference sequence.
- Position in query genome.
- Length of match.

MUMs can cover 100% of the known conserved gene pairs. Moreover, finding all MUMs can be done in almost linear time.

### 3. Parallelism technique

General-purpose, commodity CPUs currently have SIMD (Single Instruction Multiple Data) functional units and corresponding SIMD instructions. This kind of CPUs allow to be used in several ways of parallel techniques, such as data-level parallelism.

In addition to using SIMD technology, the availability of multicore architecture makes possible to execute the same task with a different kind of data.

The sequential version of the MUMmer's algorithm trades extra computation for memory and high computation time when executed in a single machine.

Previously the algorithm for sequence alignment was described in detail. Now our own proposal of a parallelization of WGA within multicore architecture is explained. There are two resources to improve in this algorithm:

- Memory usage.
- Running time.

The former was generally improved because it allows being executed in architectures where there is no restriction memory. To improve the performance of the algorithm a data-level parallelism technique is deployed in advance to genome alignment.

Our technique is divided in three phases following:

1. Splitting genome data (chunks) according to the number of available cores using 1 thread per core.
2. Parallel execution of the task of finding MUMs for every chunk where every thread has its own list of MUM-candidate.
3. Get the final list of MUMs from every MUM-candidate list of all threads.

The following Figure 1 shows the process of our data-level parallelism technique. The division of genome data was

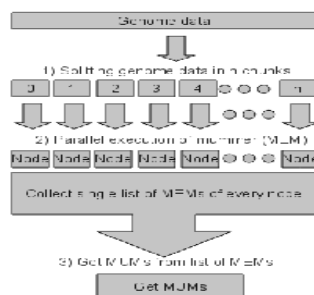


Figure 1: Data-level parallelism in multicore architectures for whole genome alignment.

used using the paradigm of data-level parallelism which consists of a generation of chunks of a query sequence with a fixed size. The main idea behind using a Maximal Unique Match (MUM): it is possible to cover a huge region of a

genome when reference and query genomes are very closely related. However To get a MUM, it requires an important feature its uniqueness. Uniqueness can only be found when a whole genome is checked. If some part of it is only evaluated we could miss the rest of the genome. In other words, after finding MUMs within a chunk it is not possible to determine if the MUM found is or not a "unique" MUM, globally in the query genome, because these MUMs are unique only in the chunk that has been read, the rest of the genome it is not known until all query genome has been read.

#### 4. Implementation

This section explains in detail how our novel implementation works. The following diagram, see Figure 2, shows the add-ons of our approach, one is executed before MUMmer and the another after MUMmer. The following sections

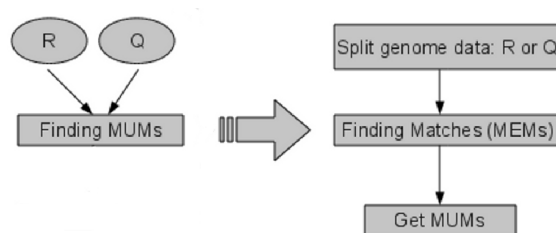


Figure 2: Finding MUMs in multicore architectures.

explain how to split query genome, search of MUMs and get real MUMs in our proposal.

##### *Split query genome*

As it was previously explained, the approach is to use a fixed size division of query genome in as many chunks as many available cores. To split query genome the algorithm needs to know in advance how many chunks will be used. Then every chunk is computed with two pointers (left and right end) which points out query genome in main memory.

##### *Finding MUMs*

The parallelization is carried out with OpenMP List of MEMs is saved to a file which has the following format:

```
>Information about the sequence
Position_in_R Position_in_Q Length_of_MEM
```

This list has to be joined with the output of every chunk computed and then the whole list is manipulated in the following phase, 4.1.

##### *4.1. Getting MUMs*

This is the most important phase in our approach because it outputs the final list of MUMs those that are the same to the serial execution of mummer.

This phase needs the list of MEMs to process them and find those matches that are unique. The following diagram shows the basic idea behind this phase, see Figure 3. To get the MUMs we filter those MEMs that are

Figure 3: Xipe Totec: Finding the real MUMs

unique in the list and order them using a modified version of LIS<sup>2</sup> algorithm. The algorithm is shown below:

<sup>2</sup>Longest Increasing Subsequence

Input: List of MEMs: Position in R, position in Q, length of MEM

Sort MEMs by increasing position and decreasing length in R

**for**  $i := 0$  to  $n$  in Total\_MEMs **do**

**if** MEM[ $i$ ] is not unique **then**

        Sort this subset by increasing position in Q and pick up the first MEM and drop the rest

**else**

        MEM[ $i$ ] is a MUM

**end if**

**end for**

Sort MEMs by increasing position and decreasing length in Q

**for**  $i := 0$  to  $n$  in Total\_MEMs **do**

**if** MEM[ $i$ ] is not unique **then**

        Sort this subset by increasing position in R and pick up the first MEM and drop the rest

**else**

        MEM[ $i$ ] is a MUM

**end if**

**end for**

The output of this algorithm gives the MUMs.

## 5. Experiments and results

To verify that our approach, Xipe Totec, can align a whole genome a set of tests were analyzed. These tests were carried out in a cluster with the following features:

- Hardware:
  - Processor Dual-Core Intel(R) Xeon(R) CPU 5160 @ 3.00GHz 4MB L2 (2x2)
  - Number of processors: 2
  - RAM: 12 GB Fully Buffered DIMM 667 MHz
- Software:
  - Linux Kernel 2.6.16.46-0.12-smp x86\_64 GNU/Linux
  - gcc 4.3.2
  - MUMmer 3.22
  - Perl 5.8.8

## References