ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2012) 1–6

Procedia Computer Science

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 lengthbetween a reference string and a query string has been identified in several applications, one of them is MUMmer. Altough 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 matches. One approach of parallelism is to take advantage of multicore architectures nowadays.

Data structure	L [bp[1]]	Search operations	Search [s]	Memory usage [MB]
Suffix tree	20	$9{,}87\times10^{18}$	169189,4	48665,12

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

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 ~~ctc~~ a gctac t ggtcagctatt acttaccgc$
Q=~~ac~~ tt ~~ctc~~ 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 maximaly 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

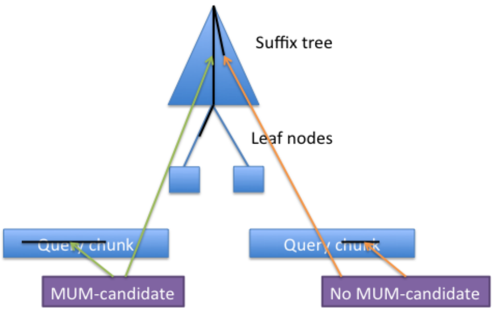

Figure 1: Finding MUMs in a suffix tree.

If all conditions 1-3 are true, then this MUM is stored in a list of MUM-candidates, see Figure 1. 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 architecures 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 2 shows the process of our data-level parallelism technique. The division of genome data was 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

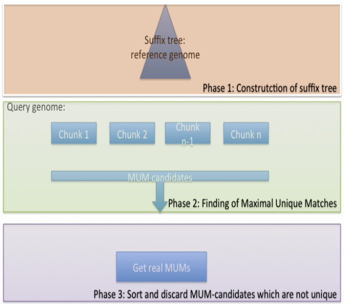
Suffix tree:
reference genome
Phase 1: Construtction of suffix tree
Query genome:
Chunk 1
Chunk 2
Chunk n-1
Chunk n
MUM candidates
Phase 2: Finding of Maximal Unique Matches
Get real MUMs
Phase 3: Sort and discard MUM-candidates which are not unique

Figure 2: Data-level parallelism in multicore architectures for whole genome alignment.

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, see Figure 3. 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, see Figure 3.

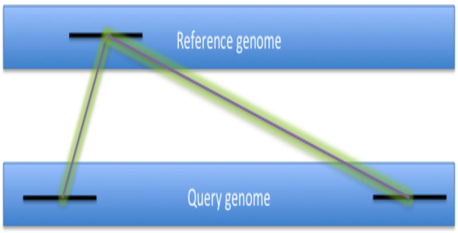

Figure 3: MUM in reference genome but not in query genome.

4. Implementation

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. The algorithm to find MUMs is a process which can be executed without any data dependency. However, when we split a query sequence the following problems arise:

- Chunk size is a performance factor.
- Different MUM-candidate in query sequence:
  - Additional MUMs.
  - Lost MUMs.

OpenMP defines the schedule for the loop iterations among the total number of threads. The total number of iterations is the number of chunks created. Every iteration means the whole search of MUMs within a chunk, if the right end of chunk is not the end of the query sequence and there are still nucleotides to match then traversal of suffix tree until it occurs a mismatch or a MUM-candidate is found.
The key factors in this phase are:

- Number of chunks.
- Size of chunks.
- Number of threads.
- OpenMP schedule and its own chunk size.

Get real MUMs

List of MUM-candidates are ordered with quicksort according to position in query sequence. Every thread has found a set of MUM-candidates from previous phase but the all threads don't produce the MUM-candidates in order. That's why a quicksort is required.
After quicksort we need to get the real MUMs. A real MUM is unique in the whole reference and query genome. Those MUM-candidates which are overlapped by bigger MUMs are discarded, see Figure 4.
The final output has the following format:

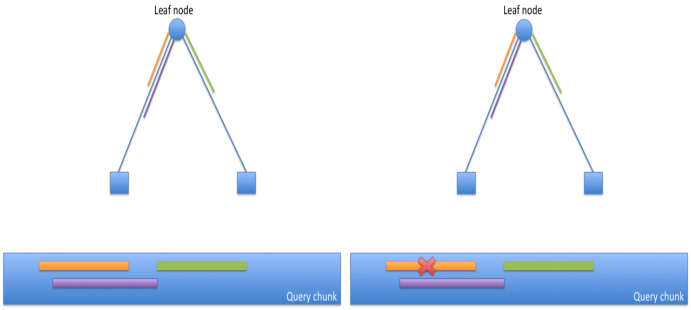

Figure 4: Getting Real MUMs.

```
>Information about the query sequence
Position_in_R Position_in_Q Length_of_MUM
```

5. Experiments and results

To verify that our approach, can have a better performance to align a whole genome a set of tests were deployed. These tests were carried out in the following node:

- Hardware:
  - 2 Processor Intel(R) Xeon(R) E5645 @ 2.4GHz of 6 cores each one, 32KB L1 cache, 256KB L2 and 12MB L3 shared cache per socket.
  - RAM: 96 GB
- Software:
  - Linux Kernel 2.6.32-220.el6.x86_64 #1 SMP
  - gcc 4.7.0 with OpenMP support

  - Likwid 2.3.0
- Genomes:
  - Reference: Human chromosome 21 single fasta file
  - Query: Mouse chromosome 16 single fasta file

The main objective of the tests was check the performance of finding MUMs in multicore architectures by using OpenMP (threads). The variables to control were:

1. Number of chunks.
2. Number of threads.

References