Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2012) 1–5

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	9,87×10^{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=[redacted] ga ~~ctc~~ a gctac t ggtcagctatt acttaccgc$
Q=[redacted] 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 in the suffix tree represents a substring of the subject sequence which maximaly matches a prefix of `querysuffix`. 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 `loc` represent a substring of length at least `minmatchlength`?
2. does `loc` 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 a function `processmumcandidate` is called. It takes the necessary information about the MUM-candidate as its arguments.

2.1.2. *Complexity analysis*

- **Step 1:** Building a suffix tree can be done in $O(n)$ time using McGreight's algorithm [**?**].
- **Step 2:** Marking internal nodes takes $O(n)$ time.
- **Step 3:** Comparing R[i-1] and Q[i-1] for each marked nodes takes $O(n + m)$ time as the number of marked nodes is at most $n + m - 2$. By the same reasoning, traversing all internal nodes to extracting MUMs also takes $(n + m)$ time.
- In total,this algorithm takes $O(n + m)$ time to find all MUMs of the input sequences.
- The space complexity of this method is $O(n \log n)$ bits as we need to store the suffix tree of the input sequence.

Based on some experiments, it is found that MUMs can cover 100% of the known conserved gene pairs. Moreover, finding all MUMs can be done in linear time.

3. Parallelism technique

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 with MUMmer is explained, Xipe Totec. There are two resources to improve in this algorithm:

- Memory usage.
- Running time.

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 according to the number of available cores.
2. Parallel execution of as many instances of mummer as available cores.
3. Get the list of MUMs from the whole set of MEMs[2] found in the previous phase.

The following Figure 1 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 sequence with a fixed size and a fixed overlap. One issue arises when a genome is splitted because of the heuristic used in the algorithm is affected. So that, a longer sequence is more likely to have a better finding of MUMs while a smaller one can produce MUMs which are not effective MUMS.

Figure 1: Data-level parallelism technique for whole genome alignment.

Another consideration was the genome structure, because a genome is build from a finite alphabet $\Sigma = \{a, g, c, t, n\}$ but according to the MUMmer's algorithm each alignment is made using only the nucleotide base pairs, this means that letter "n" in a biological way can mean anything: (a,g,c,t). A complex structure of a genome can have a huge impact of our data level parallelism.
To get a MUM, it requires an important feature its uniqueness. Uniqueness can only be found when a whole genome is checked. 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 genome, because these MUMs are unique only in the chunk that has been read, the rest of the genome it is not known.

[2] Maximal Exact Match

One solution to solve this problem is to drop the MUMmer's heuristic in order to be able to find the correct MUMs when we apply our data-level parallelism. The new approach is to find a Maximal Exact Match (MEM), a MEM allows to drop the uniqueness of a match but with a high computational cost: a brute-force approach.
Nevertheless, the major problem with the use of MEM is that the number of occurrences increases exponentially when *shorter MEM* are used. Moreover, the high ratio of MEMs requires to save them in some place, memory or disk, that means a heavy use of resources in both CPU and Memory. This drawback is the main disadvantage because the hits of MEMs are increasing with the size of the genome. A new phase has to be designed to reduce the use of resources when a short MEM is searched.

4. Implementation

This section explains in detail how our proposal is implemented and the modifications in order to be executed in MUMmer. 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 explain how the split of genome data, search of MEMs and get MUMs are carried out in our proposal.

Figure 2: Xipe Totec: proposal for parallelization of whole genome alignment

4.1. Split genome data

As it was previously explained, the approach is to use a fixed size division of genome data in as many chunks as many available cores. Xipe Totec needs to know how many cores will be used in order to divide the genome data.
One key aspect of Xipe Totec is the way to split a genome. To align a genome requires a reference genome so that, there are two ways of using Xipe Totec:

- Splitting reference genome.
- Splitting query genome.

Both of them can reduce the computation time or memory usage.
To split genome data a perl script was coded to do this task, the script requires the following arguments:

- Genome data to be splitted.
- Number of chunks.
- Overlap size.
- Location to save the genome data.

4.2. Finding MEMs

This phase requires the mummer program, one of the several small programs in the MUMmer suite. mummer has several options. In order to get a correct alignment, our proposal requires to compute MEMs instead of MUMs, so that mummer is executed with:

- -n: to match only nucleotides.
- -maxmatch: option to get MEMs.
- -l *length*: to find MEMs of a some minimum length.

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, 4.3.

4.3. 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 unique in the list and order them using a modified version of LIS[3] algorithm. The algorithm is shown below:

Figure 3: Xipe Totec: Finding the real MUMs

```
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

[3] Longest Increasing Subsequence