Search for Maximal Unique Matche Multi-core architectures

Julio Cesar Garcia Vizcaino, Antonio Espinosa, Juan Carlos M Porfidio Hernandez

Universitat Autónoma de Barcelona (UAB), Barcelona, Sp

{jcgarcia,aespinosa}@caos.uab.cat
{juancarlos.moure,porfidio.hernandez}@uab.es

Abstract. Maximal Unique Matches (MUMs) are common su that match a Reference and a Query genome. They are exact, uni maximal. The search for MUMs in large genomes is a heavy and re task, so there is a fair chance of parallelize and execute this s multi-core architectures. This research evaluates a parallelizatio search for MUMs in genomic sequences within multi-core archit The Reference genome is indexed by using a Suffix Tree or E Suffix Array in main memory and then a parallelized algorithm for MUMs between Reference genome and Query genome. Query is read by several threads in chunks of fixed size. This approach on MUMmer, a genome alignment tool, which is able to find M Unique Matches (MUMs). Results show a reduction in search t less usage of memory.

Keywords: Indexed Search, Bioinformatics, Maximal Unique Multi-core architectures, Parallelization

1 Introduction

Modern sequencing and computational technologies and advanc matics has made whole genome sequencing possible. One resultin the fast alignment of whole genomes. Dynamic programming is aligning two large genomes, hundreds of Mbp. One very successfu performing a whole genome alignment is based on identifying "ma matches". This heuristic is based on the assumption that one expe occurring in two similar genomes. Maximal unique matches (MUM surely part of a good alignment of the two sequences and so the

The problem of searching maximal unique matches for a minimu
tween a Reference genome and a Query genome has been identi
applications, one of them is MUMmer [3]. MUMmer's algorithm
searches for Maximal Unique Matches (MUMs), although with
main memory to store the Reference genome and a null use of mu
tectures.

We designed and tested a data-level parallelism to use in multi
tures with two different data structures: Suffix Tree and Enhanced

The rest of this paper is organized as follows. Section 2 discuss
work. Section 3 defines some definitions used in this paper. Sectio
implementation of parallelization of search for MUMs. Section 5
results of our parallelization. Section 6 concludes the paper.

2 Related work

Search for Maximal Unique Matches to do Whole Genome Alignm
posed in [2]. There have been some previous work in the paralleliza
of matches in genomic data, like [11, 9, 7], however these works a
fixed patterns and read alignment. On the other hand, there have
ments in parallelization of Whole Genome Alignment like [8]. The
of searching for MUMs with a full-text index data structure is a
not covered deeply, there is only one approach in [4] but without ac
code to check their implementation and it is more focused with G
hybrid architectures and Suffix Array. There are other implemen
11, 6, 10] but they search for Maximal Exact Matches with threads
imal Unique Matches . We propose the use of OpenMP to search
multi-core architectures with two full-text index data structures: S
Enhanced Suffix Array.

3 Preliminaries

Let's assume the Reference genome $R[0, \ldots, n-1]$ of size $|R|$
alphabet $\Sigma = \$, A, C, G, T$ which has a sentinel character $R[n-$
occurs nowhere else in the Reference genome and is lexicographical
the characters that occur in the alphabet. The suffixes of the Refe
are zero indexed by their position in the original Reference by a
data structure like a Suffix Tree or an Enhanced Suffix Array.

Definition 2. *A suffix Tree, ST, for an n-character string R is a*

Definition 3. *An enhanced suffix array consists of four arrays (common prefix (LCP), child and inverse suffix arrays) that hav information saved in a suffix tree. [1].*

We also define a suffix link, which is an important trick to achie plexity to search for MUMs in both Suffix Tree and Enhanced Su

Definition 4. *A suffix link is a pointer from string $\overline{aw}$ to substr*

For reasons of space we recommend the reader to check [1] for f of Enhanced Suffix Array. A search for a MUM between a Refe $|R| = n$, and a Query genome, $|Q| = m$, can be done in a Suffix steps and in an Enhanced Suffix Array in $O(m)$ steps. Once we ha resources used to index a Reference genome, we need to answer Where are the MUMs of R and Q of some minimum length L? algorithms used to answer this question using a Suffix Tree, see and a Enhanced Suffix Array, see Algorithm 2.

Algorithm 1: Search for MUMs in a Suffix Tree.

```
input  : R, Q, L
output: List of MUMs of length ≥ L, with start position in R and
begin
    ST ← buildST(R);
    foreach position i ∈ Q do
        length ← TraverseSuffixTree(Q[i],ST);
        if isLeafNode(Q[i]) then
            /* Leaf saves the position of a suffix in ST.
            ;
            if R[leaf − 1] ≠ Q[i − 1] and length ≥ L then
                MUMcands ← saveMUMcand(R_leaf,i,length);
    /* Get unique MUMs from list of MUM-candidates.
    ;
    MUMs ← cleanMUMcand(MUMcands)
```

4 Implementation

The search for MUMs between a Reference and Query genome has

Algorithm 2: Search for MUMs in an Enhanced Suffix Array.

```
input  : R, Q, L
output: List of MUMs of length ≥ L, with start position in R and
begin
    ESA ← buildESA;
    while Q [i ] < end do
        traverseESA(Q, Q [i], interval, Q [i ].length());
        if interval.depth ≤ 1 then
            interval.depth = 0; interval.start = 0; interval.end = n-1;
            continue;
        if interval.size() == 1 and interval.depth ≥ L then
            if leftMaximal(Q, Q [i ], SA [interval.start]) then
                saveMUMcand(SA [interval.start],i,interval.depth);
        repeat
            interval.depth = interval.depth-1;
            interval.start = ISA [SA [interval.start] + 1];
            interval.end = ISA [SA [interval.end] + 1]; i ++; if interv
            0 or suffixlink(inter) == false then
                interval.depth = 0; interval.start = 0; interval.end =
        until interval.depth > 0 and interval.size();
    MUMs ← cleanMUMcand(MUMcands)
```

MUM candidates which are not unique in both Reference and Q
Our approach, see Figure 1, is divided in three phases: Creation
or Enhanced Suffix Array; Splitting query genome data (chunks)
the number of available cores using 1 thread per core and para
of the search for MUMs for every chunk, then every thread has
MUM candidates; and Get MUMs from list of MUM candidates
A data-level parallelism to search for MUMs requires to split the
in chunks and stream each chunk against the full-text index d
This search is independent of each other chunk. The final stage t
must be performed with a full Query genome and with chunks of
is more obvious too. This approach is suitable to be performed
architectures because chunks can be assigned to each core and sea
in the full-text index data structure which is stored in main mem
good choice of parallelization involves a data partition technique.

Memory usage is improved, within this approach, with multipl
full-text index data structure, instead of one read at a time with

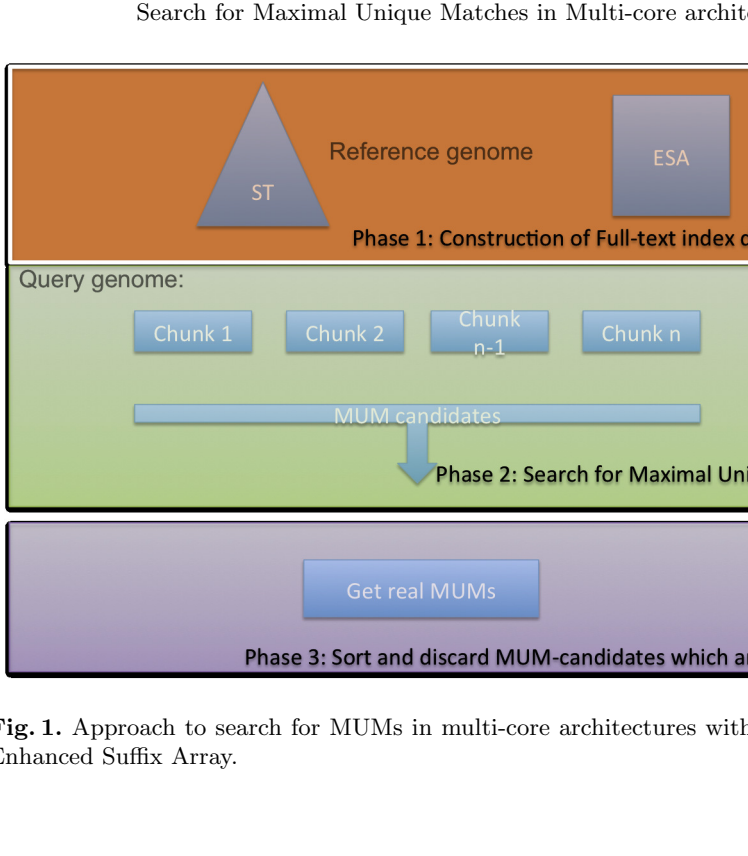


Fig. 1. Approach to search for MUMs in multi-core architectures with Enhanced Suffix Array.

The chunk size was computed with Query genome length divided available threads. Chunk size is a performance factor because we balanced workload among threads.

Phase 1 is the construction of the data structure: Suffix Tree Suffix Array; this phase is executed in a serial way. Phase 2 requir execution of the Algorithm (1 or 2) to search for MUMs, the first 2 is the split of Query genome in as many chunks as available split defines the start and end position in Query genome which is memory and shared by all threads. The chunk size is fixed acc number of threads. The parallel search for MUMs gets a list of MU which are ordered in each thread by its position in Query genom performed after all threads have finished its execution and it m of MUM-candidates and it is ordered by position in Query genom needed because the merge process may have unordered lists); this

5 Results

To verify that our approach, can have a better performance to sea
of a Reference and Query genome, in Table 1, we verify that outpu
and our approach have the same MUMs. Tests were carried out
ing node: 2 Processor Intel(R) Xeon(R) E5645 @ 2.4GHz of 6 c
32KB L1 cache, 256KB L2 and 12MB L3 shared cache per socket.
GCC 4.7.0 with OpenMP support + Linux. The main goal of thes

Table 1. List of Genomes used in experiments.

	1	2	3
Reference	4.64 [Mbp]	169 [Mbp]	1031 [Mbp]
Query	5.5 [Mbp]	167 [Mbp]	1357 [Mbp]

check the performance to search for MUMs in multi-core architec
OpenMP (threads). This performance involves the usage of Mem
(execution time to search for MUMs).
Times were collected during the execution of the experiments: con
of data structure and execution time to search for MUMs with
omp_get_wtime of OpenMP.

5.1 Construction of Full-text index data structure

This paper compares the performance of two data structures to sea
in multi-core architectures, two features for these structures are
time and memory footprint. Full-text indexes allow fast access to
any length, but they have a great memory and construction cos
affected by the type and implementation of the full-text index
used. We measure the construction time for the set of genomes in
2 shows that an ESA has a lower construction time for all the Refe
used. Since a full-text index data structure has to be build every
for MUMs is performed, a reduction in the construction phase in
allow us to improve the overall execution time. A future improve
to build the full-text index data structure in parallel or load from

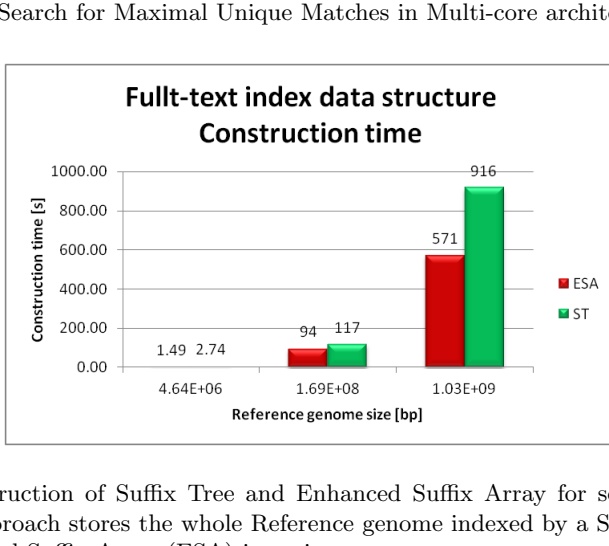


Fig. 2. Construction of Suffix Tree and Enhanced Suffix Array for s
sizes. Our approach stores the whole Reference genome indexed by a S
or an Enhanced Suffix Array (ESA) in main memory.

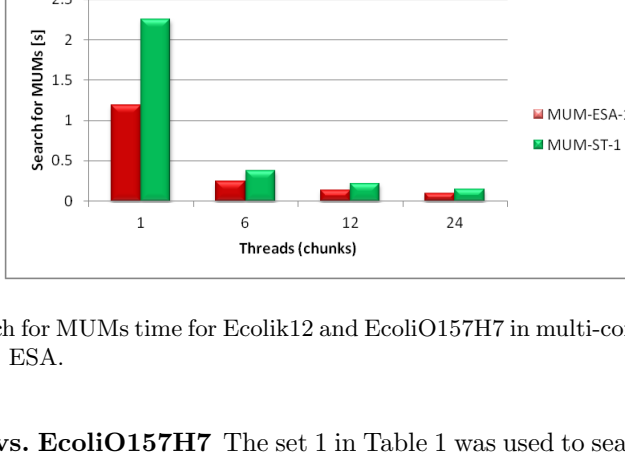


Fig. 3. Search for MUMs time for Ecolik12 and EcoliO157H7 in multi-co
with ST and ESA.

EcoliK12 vs. EcoliO157H7 The set 1 in Table 1 was used to sea
in a ST and ESA. In Figure 3, we show that ESA has a better se
Figure 3. However, the space consumption shows that ESA has a b
footprint, a reduction of 44% of memory space compare to Su
MB). This first experiment shows that it is possible to search fo
two different full-text index data structures with thea reduction i

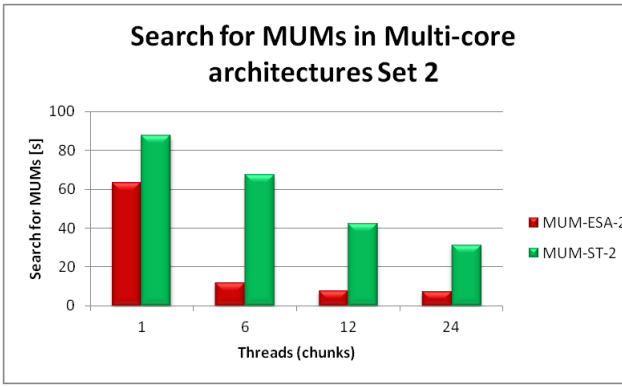
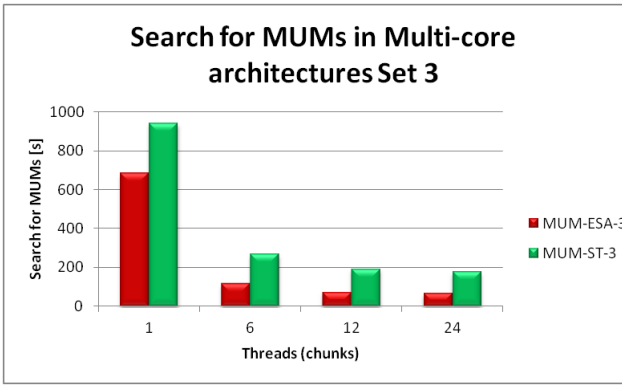


Fig. 4. Search for MUMs time for D.melanogaster and D.pseudoobscu architectures with ST and ESA.

need to reduce the search time. On the other hand, the space c almost constant in both data structures. However, again ESA has of memory. The reduction in memory footprint for ESA is near the ST.

Chicken vs. Zebrafish The set 3 in Table 1 have genomes o [Gbp] and we show in Figure 5 that an execution with one threa performance in search time in ESA over ST. In order to understa



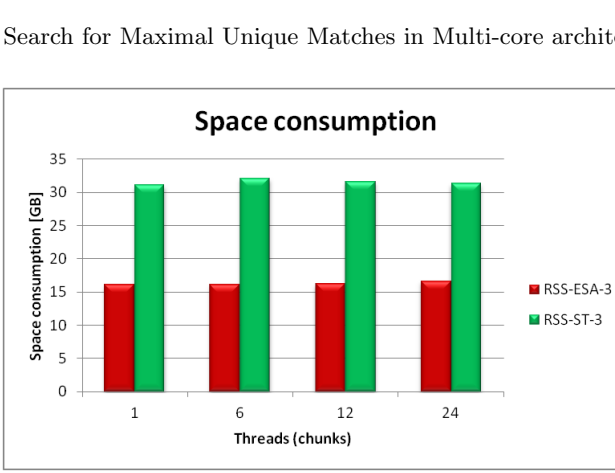


Fig. 6. RAM consumption to search for MUMs for Chicken and Zebrafis architectures with ST and ESA.

6 Conclusions

This paper presents an evaluation of performance to search for M Reference genome and Query genome in multi-core architectures in a Suffix Tree and an Enhanced Suffix Array. The search for formed in a Suffix Tree or an Enhanced Suffix Array and the can help in the process of Whole Genome Alignment. The resu MUM-problem can be solved with a ST or ESA and with the use architecture we reduce the execution time. There is a reduction in with an Enhanced Suffix Array over a Suffix Tree. We have evalua data structure which provides the same functionality of a Suffix for MUMs, with less use of memory in multi-core architectures.

Improvements involve a better use of CPU when we have n thread per core. From the results obtained we conclude that an En Array is suitable to solve the MUM-problem. To improve the p CPU usage of our approach, the following proposals may be used this may work because we know in advance what intervals to loo and Query genome; Data cache management: we may need to sto of ESA in some cache-level of processor. Since we need to get t search by checking the first character of the suffix within the int use some SSE instructions to check more than one suffix at the sa

References

1. M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix hanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, M
2. A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. Wh Salzberg. Alignment of whole genomes MUMMER. *Nucl Acids Re* 1999.
3. A. L. Delcher, S. L. Salzberg, and A. M. Phillippy. Using MUM similar regions in large sequence sets. *Current protocols in bioinfor board Andreas D Baxevanis et al*, Chapter 10(1934-340X (Electroni - Journal Article SB - IM):Unit 10.3, 2003.
4. G. Encarnac and N. Roma. Advantages and GPU Implement Performance Indexed DNA Search based on Suffix Arrays. *Arch* 49–55, 2011.
5. D. Gusfield. *Algorithms on strings, trees, and sequences: comput computational biology*. Cambridge University Press, 1997.
6. Z. Khan, J. S. Bloom, L. Kruglyak, and M. Singh. A practical finding maximal exact matches in large sequence datasets using spar *Bioinformatics*, 25(13):1609–1616, July 2009.
7. C. S. Kouzinopoulos, P. D. Michailidis, and K. G. Margaritis. Par of multiple pattern matching algorithms for biological sequences performance results. *Systems and Computational Biology - Bioi Computational Modeling*, 2005.
8. X. Meng and V. Chaudhary. Exploiting multi-level parallelism for h using general purpose processors. *October*, 2005.
9. H. Mongelli. Efficient Two-Dimensional Parallel Pattern Matching
10. E. Ohlebusch, S. Gog, and A. Kügel. Computing matching statistic exact matches on compressed full-text indexes. In *SPIRE*, pages 3
11. M. Oğuzhan Külekci, W.-K. Hon, R. Shah, J. Scott Vitter, and B parallel sparse index for genomic read alignment. *BMC genomics*, Jan. 2011.
12. M. Vyverman, B. De Baets, V. Fack, and P. Dawyndt. essaMEM imal exact matches using enhanced sparse suffix arrays. *Bioinfor England)*, pages 2–4, Feb. 2013.