Whole genome alignment in High Performance Computing environments

Julio César García Vizcaíno   Directores: Antonio Espinosa, Juan Carlos Moure

Computer Architecture & Operating Systems Department
Universitat Autónoma de Barcelona

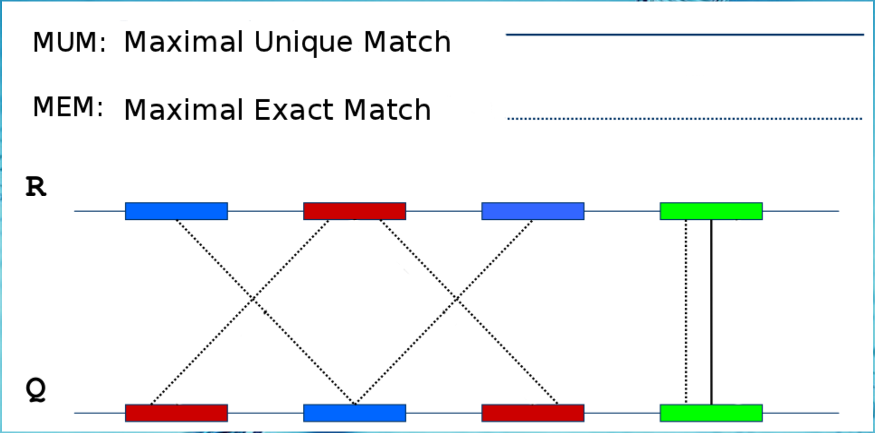
25 de mayo de 2012

Contents

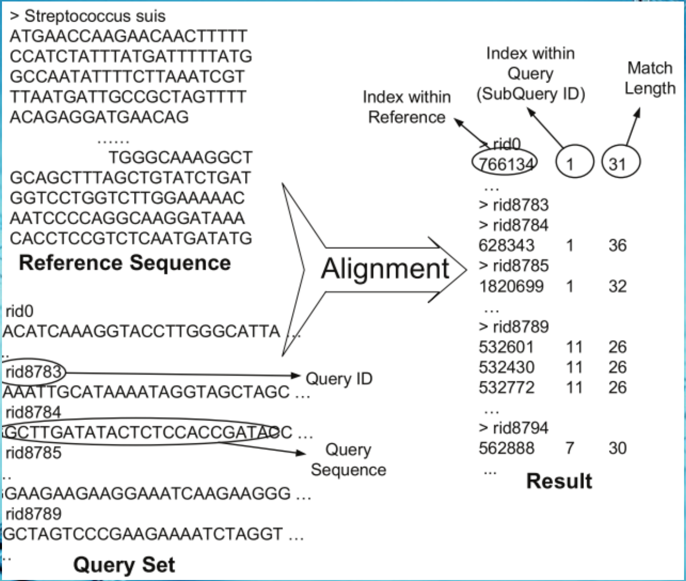
Search of Maximal Exact Matches

MUM: Maximal Unique Match

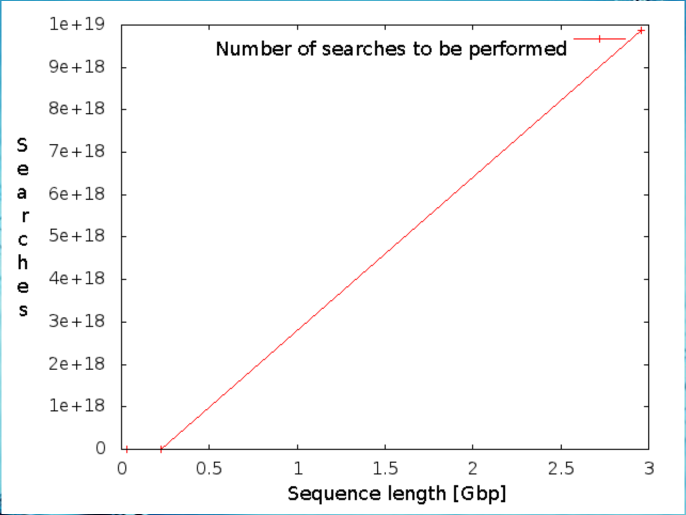
MEM: Maximal Exact Match



Genome alignment: search of Maximal Exact Matches



Search of Maximal Exact Matches



Ways of finding exact matches

Brute Force (3 GB)

```
BANANA
BAN
 ANA
  NAN
   ANA
```

Naive

Slow & Easy

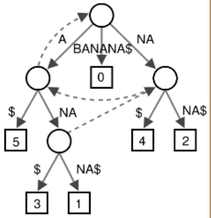
Suffix Array (>15 GB)

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

Vmatch, PacBio Aligner

Binary Search

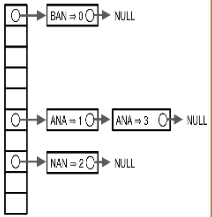
Suffix Tree (>51 GB)



MUMmer, MUMmerGPU

Tree Searching

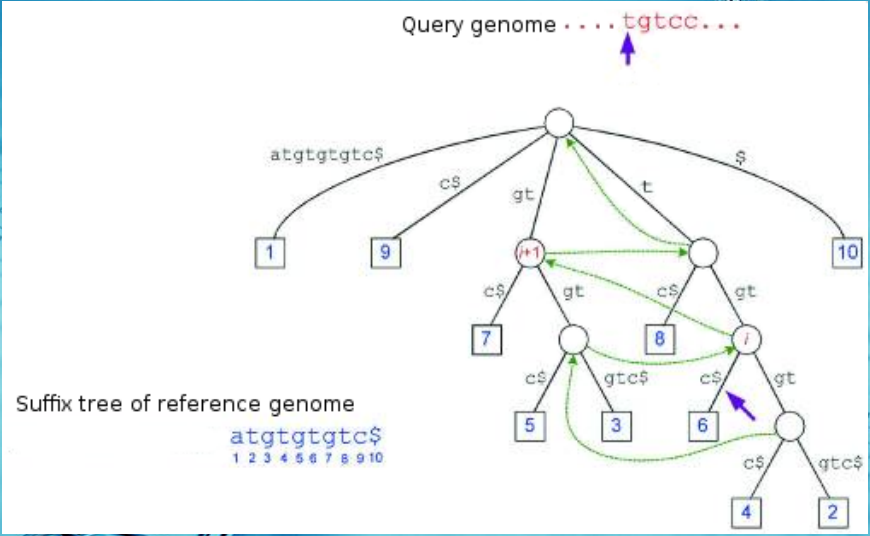
Hash Table (>15 GB)



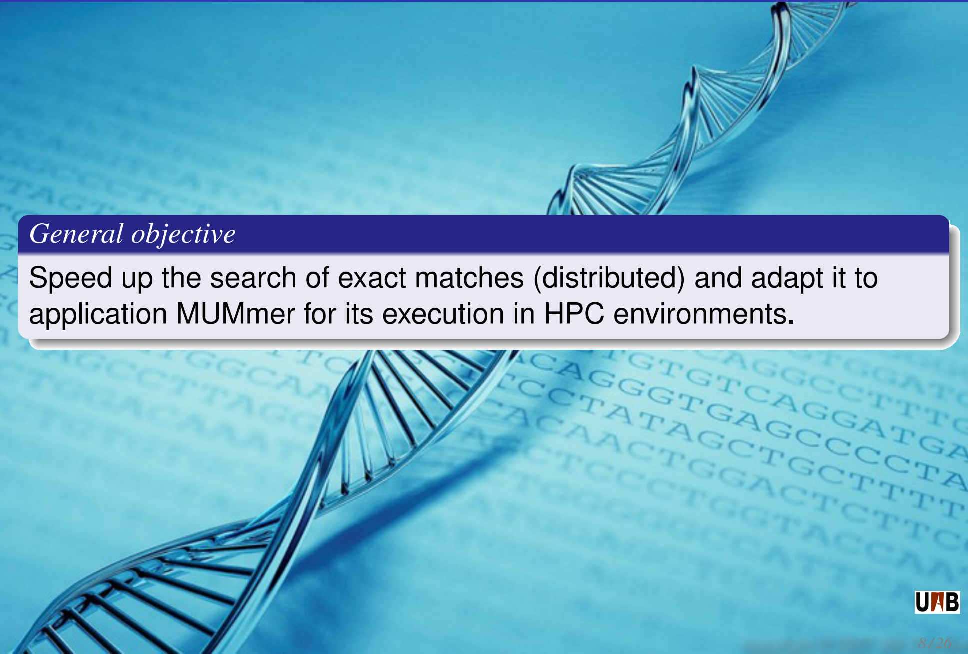
BLAST, MAQ, ZOOM, RMAP, CloudBurst

Seed-and-extend

Traversal of suffix tree



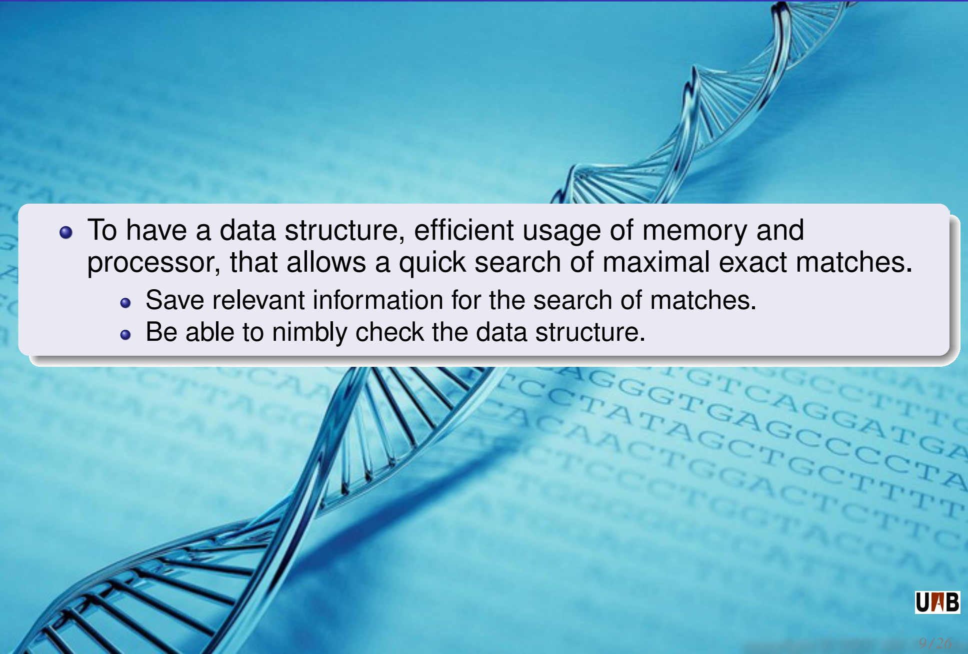
General objective



General objective

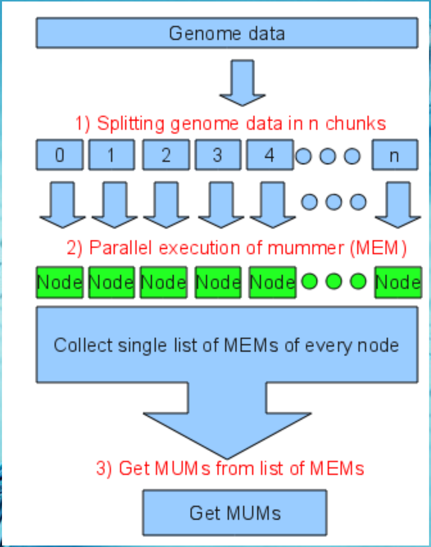
Speed up the search of exact matches (distributed) and adapt it to application MUMmer for its execution in HPC environments.

Specific objective

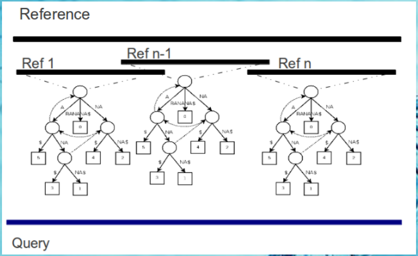


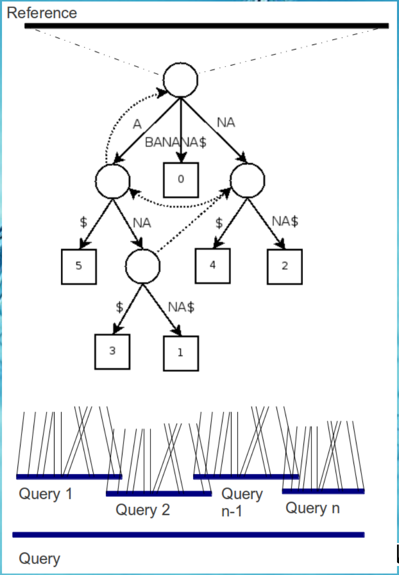
- To have a data structure, efficient usage of memory and processor, that allows a quick search of maximal exact matches.
  - Save relevant information for the search of matches.
  - Be able to nimbly check the data structure.

Naive solution

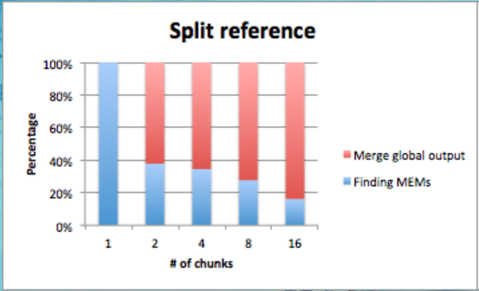

Genome data
1) Splitting genome data in n chunks
0
1
2
3
4
n
2) Parallel execution of mummer (MEM)
Node
Node
Node
Node
Node
Node
Collect single list of MEMs of every node
3) Get MUMs from list of MEMs
Get MUMs

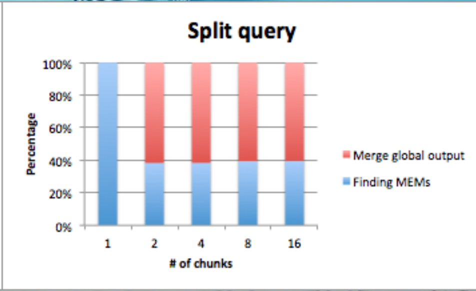
Split sequence



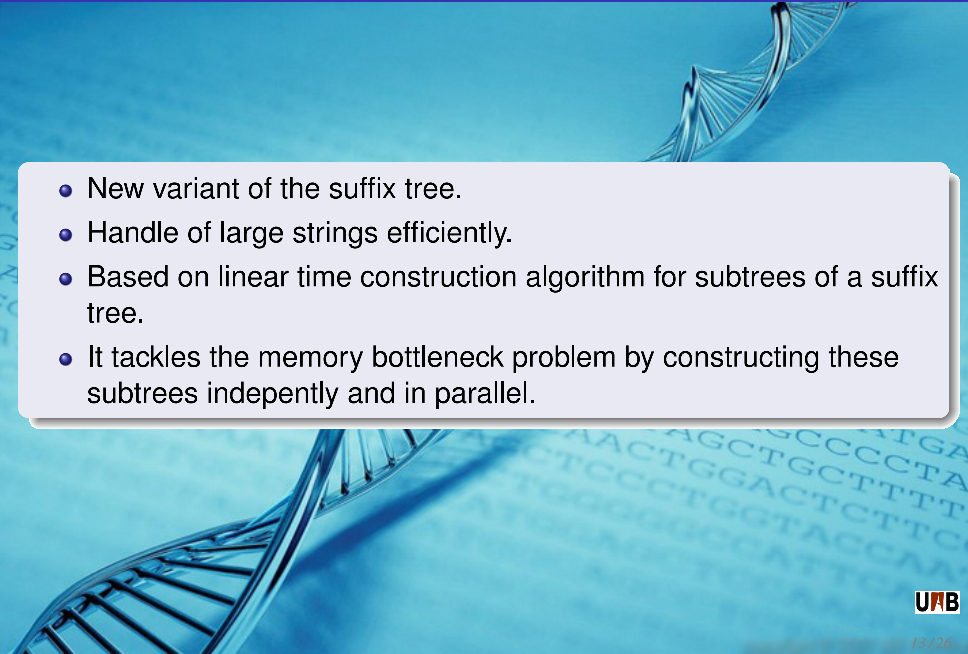


Naive solution cont.



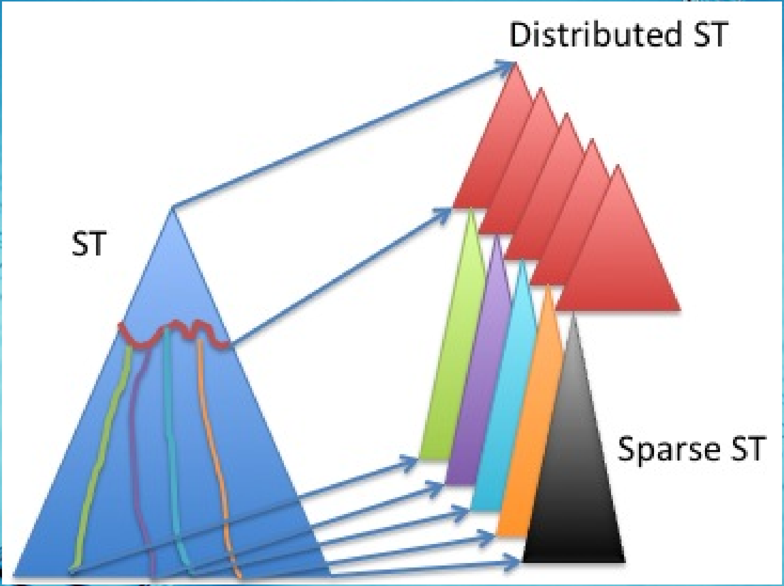


Distributed suffix tree



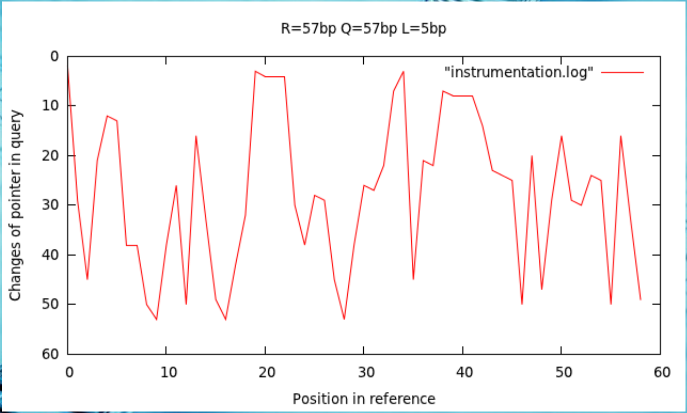
- New variant of the suffix tree.
- Handle of large strings efficiently.
- Based on linear time construction algorithm for subtrees of a suffix tree.
- It tackles the memory bottleneck problem by constructing these subtrees indepently and in parallel.

Distributed suffix tree



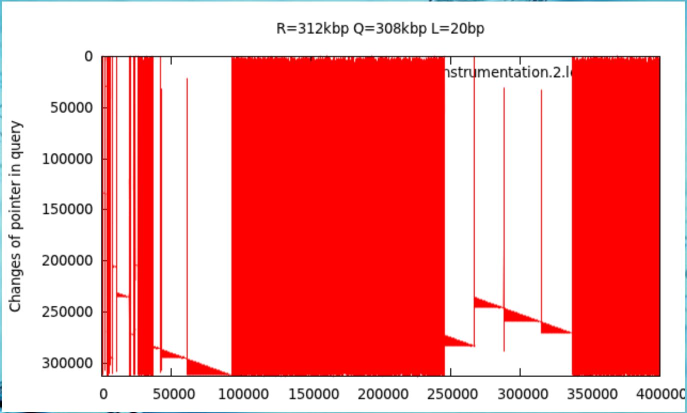
Traversal of a suffix tree

Every suffix of query (pointer) is searched in suffix tree. By using suffix links we jump to other depth of suffix tree and we avoid to check *x* characters.



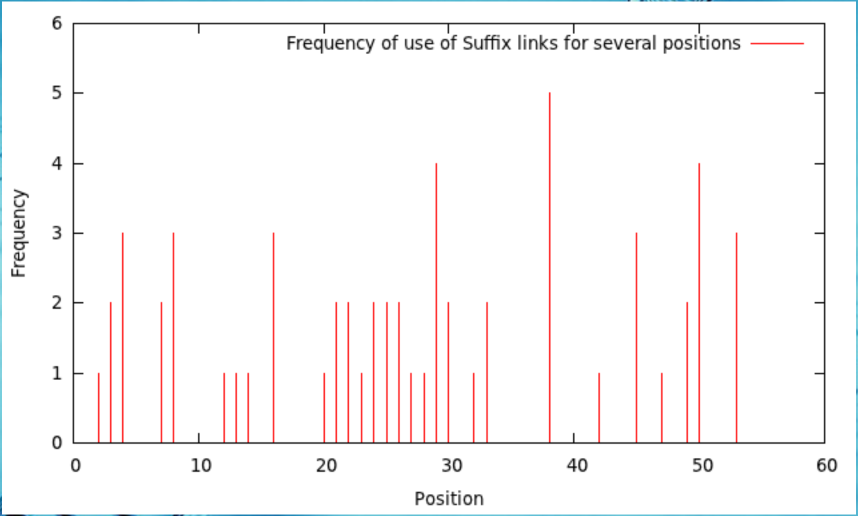
Traversal of a suffix tree

The jumps in suffix tree are done while checking the parent node after finishing the last match.



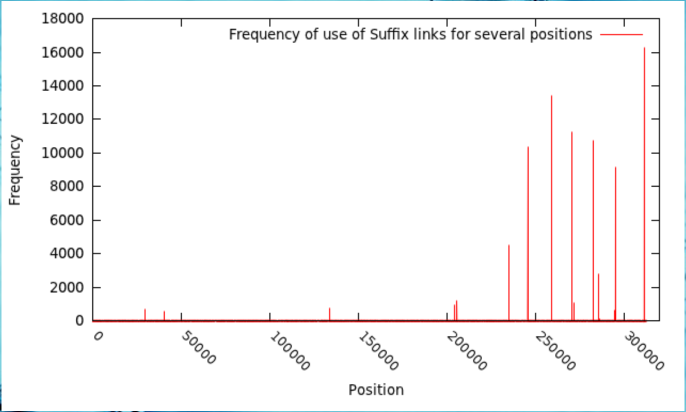
Suffix links

The location of suffix links are made during suffix tree construction.



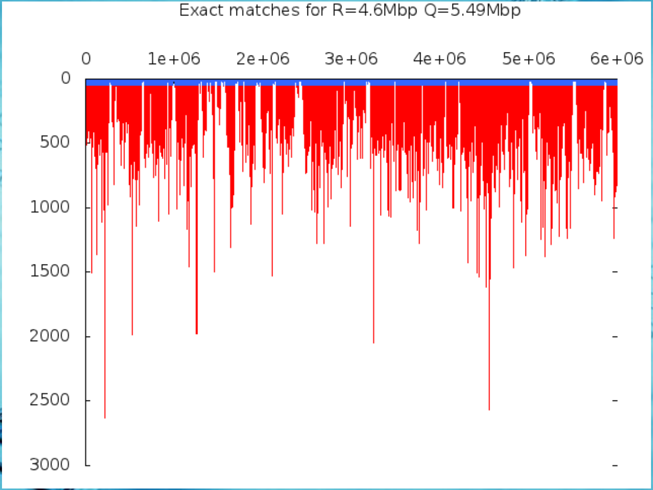
Suffix links

The location of suffix links are more likely to be in deeper regions of suffix tree.



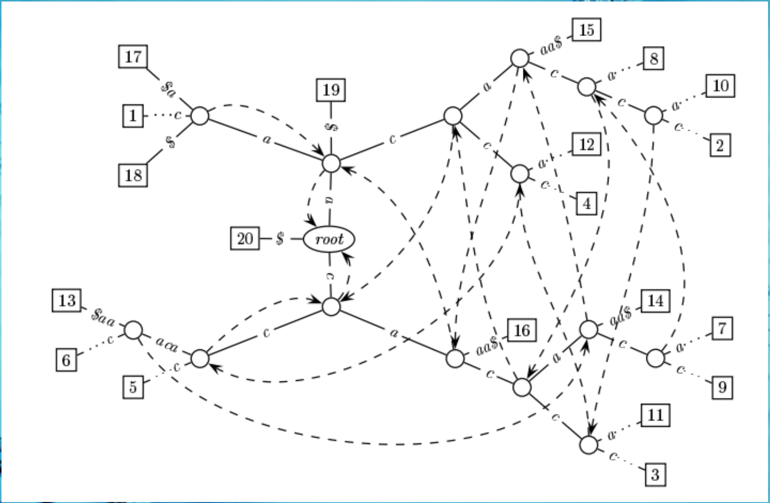
Access to suffix tree

Finding of maximal matches (path from root) are marked in suffix tree.
Improved detection of maximal matches.



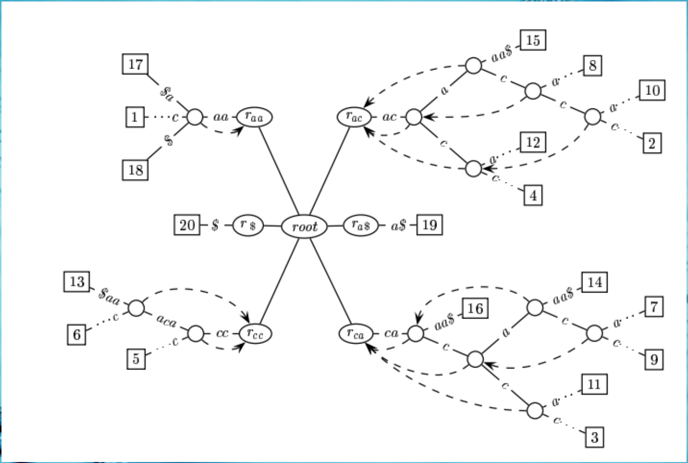
Suffix tree: example

Standard suffix tree of aacacccacacaccacaaa$ with standard suffix links.

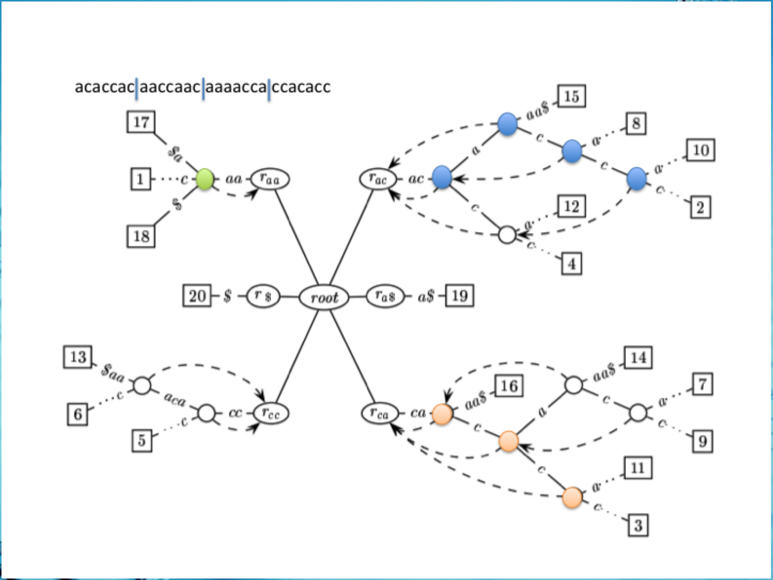


Distributed suffix tree

The SSTs for aacacccacacaccacaaa$ with their respective root nodes labelled $r_{aa}, r_{ac}, r_{ca}, r_{cc}, r_{a\$}$ and $r_{\$}$.



Distributed and parallel search of maximal matches

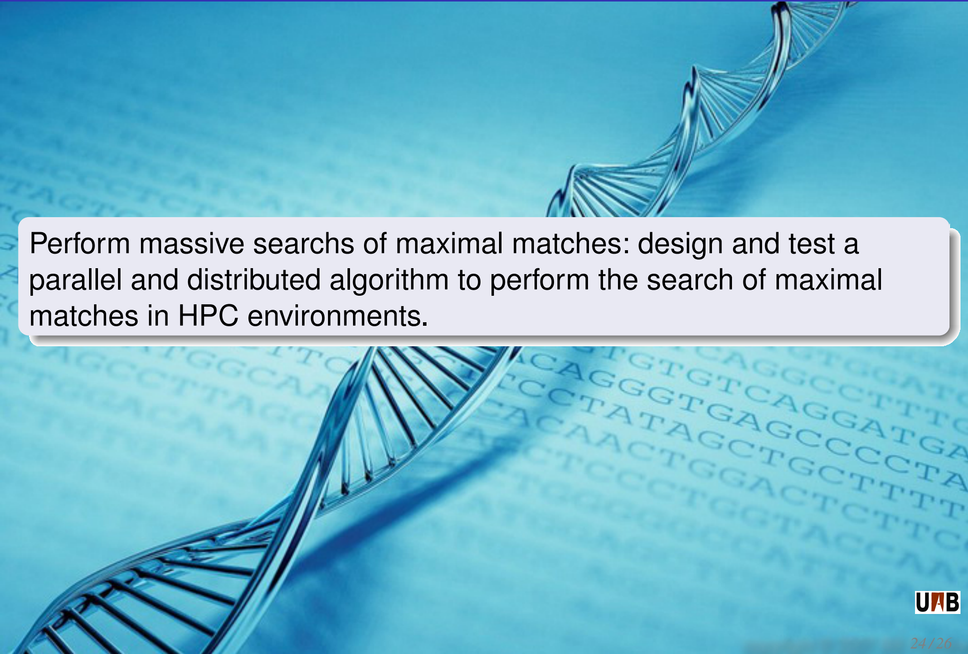


Conclusions

First year

It has been adapted a data structure which can be deployed in HPC environments.
It may be used to implement parallel and distributed techniques for search of maximal matches.
This data structure is able of handling large input sequences to search maximal exact matches.

Future work



Perform massive searchs of maximal matches: design and test a parallel and distributed algorithm to perform the search of maximal matches in HPC environments.

Thanks!

Whole genome alignment in High Performance Computing environments

Julio César García Vizcaíno   Directores: Antonio Espinosa, Juan Carlos Moure

Computer Architecture & Operating Systems Department
Universitat Autónoma de Barcelona

25 de mayo de 2012