**SI2-SSE**
**STAMLA**

# Scalable Tree Algorithms for Machine Learning Applications

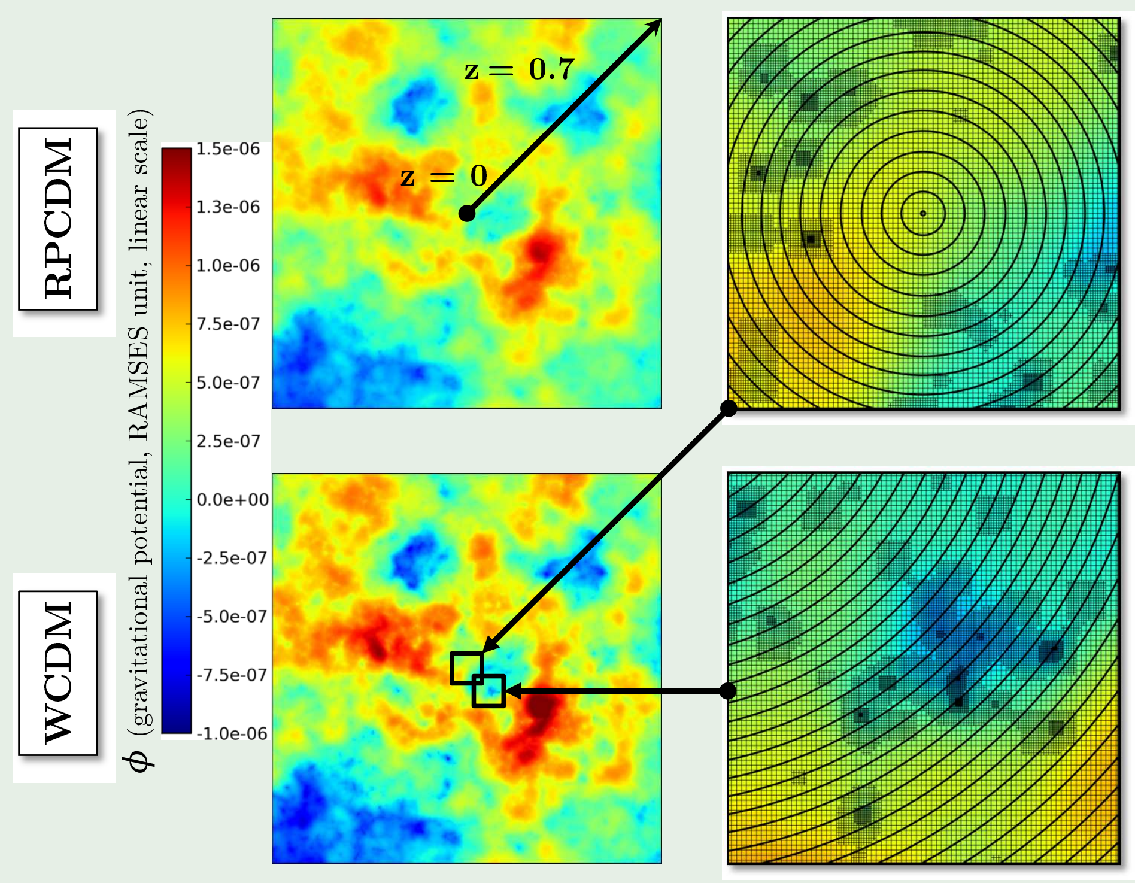
## Abstract

The STAMLA project aims at designing efficient data structures for high performance machine learning applications. It originates from research in numerical astrophysics where data structures have become one of the main bottleneck of present day simulation and analysis codes. The STAMLA project focuses on trees, since they are at the core of a wide range of applications, including machine learning, while still missing in standard libraries of most programming languages.

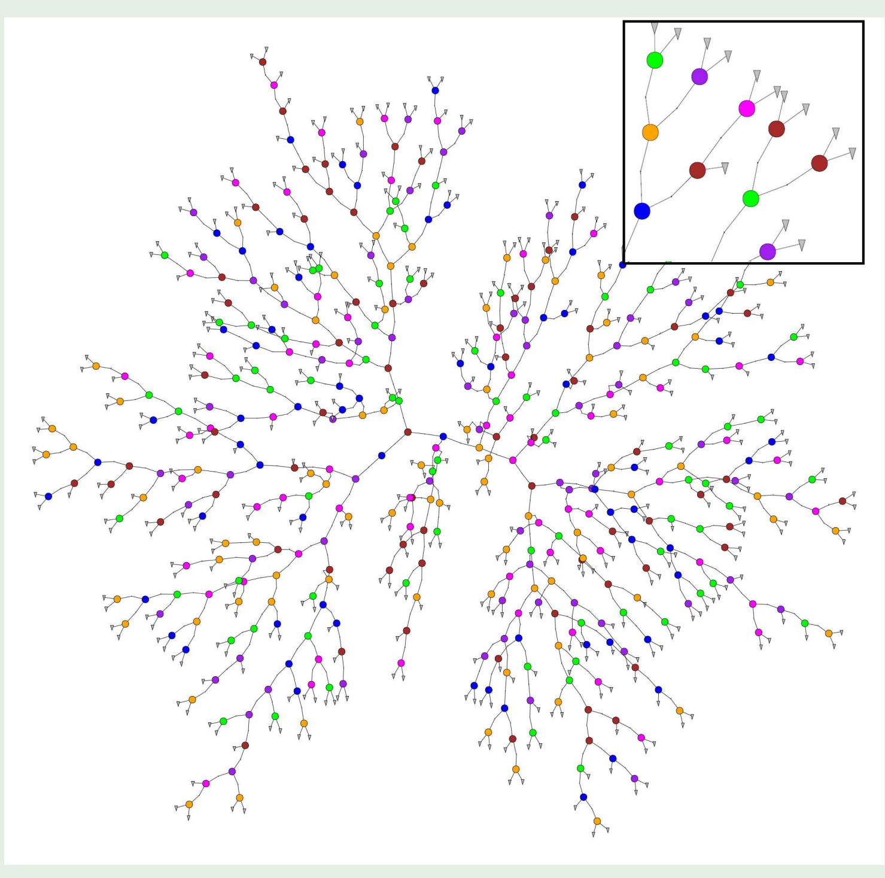
## Tree data structures in numerical astrophysics

The STAMLA project started from advanced optimizations on tree data structures in astrophysical codes. Below are illustrated two high performance applications relying on trees: on the left, an adaptive mesh refinement code for cosmology, displayed here for two cosmological models, and on the right, a machine learning algorithm to estimate photometric redshifts in large observational surveys.

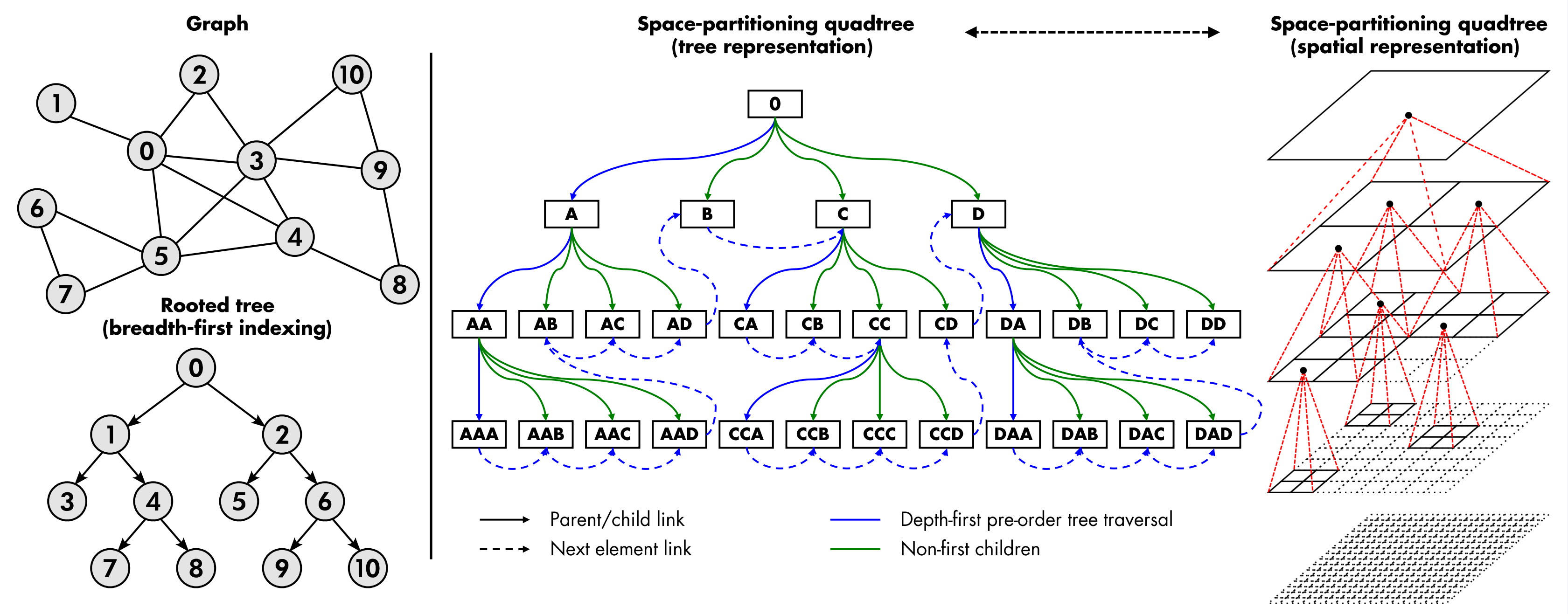
Gravitational potential in redshift shells



Prediction tree for photometric redshifts



## Trees and graphs

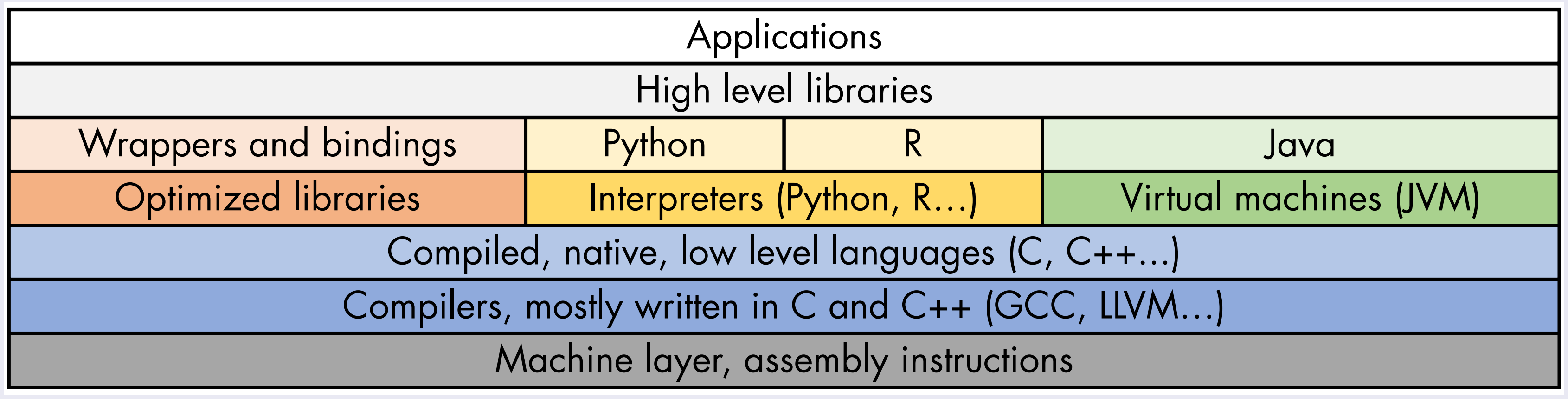


While graphs are generally stored as adjacency lists or matrices, rooted trees tend to be stored using explicit links between nodes. However, given a set of constraints at compile-time, like the arity of the tree, or its maximum depth, more efficient representations are available. In particular, implicit representations relying on indexing strategies demonstrate excellent cache-friendliness and vectorization properties. The STAMLA project aims at generating such representations at compile-time given a set of constraints depending on the application domain.

**Data movement**

| Operation | Approx. time | Remark |
|---|---|---|
| L1 cache reference | 0.5 ns | |
| One cycle on a 3 GHz processor | 1 ns | |
| Branch mispredict | 5 ns | |
| L2 cache reference | 7 ns | 14× L1 cache |
| Mutex lock or unlock | 25 ns | |
| Main memory reference | 100 ns | 200× L1 cache |
| Send 1 KB over a 1 Gbps network | 10 $\mu$s | |
| Read 1 MB sequentially from main memory | 250 $\mu$s | |
| Round trip within the same datacenter | 500 $\mu$s | |
| Read 1 MB sequentially from a SSD | 1 ms | 4× memory |
| Disk seek | 10 ms | 20× datacenter RT |
| Read 1 MB sequentially from disk | 20 ms | 80× memory |
| Send packet California→Netherlands→California | 150 ms | |

## The software stack

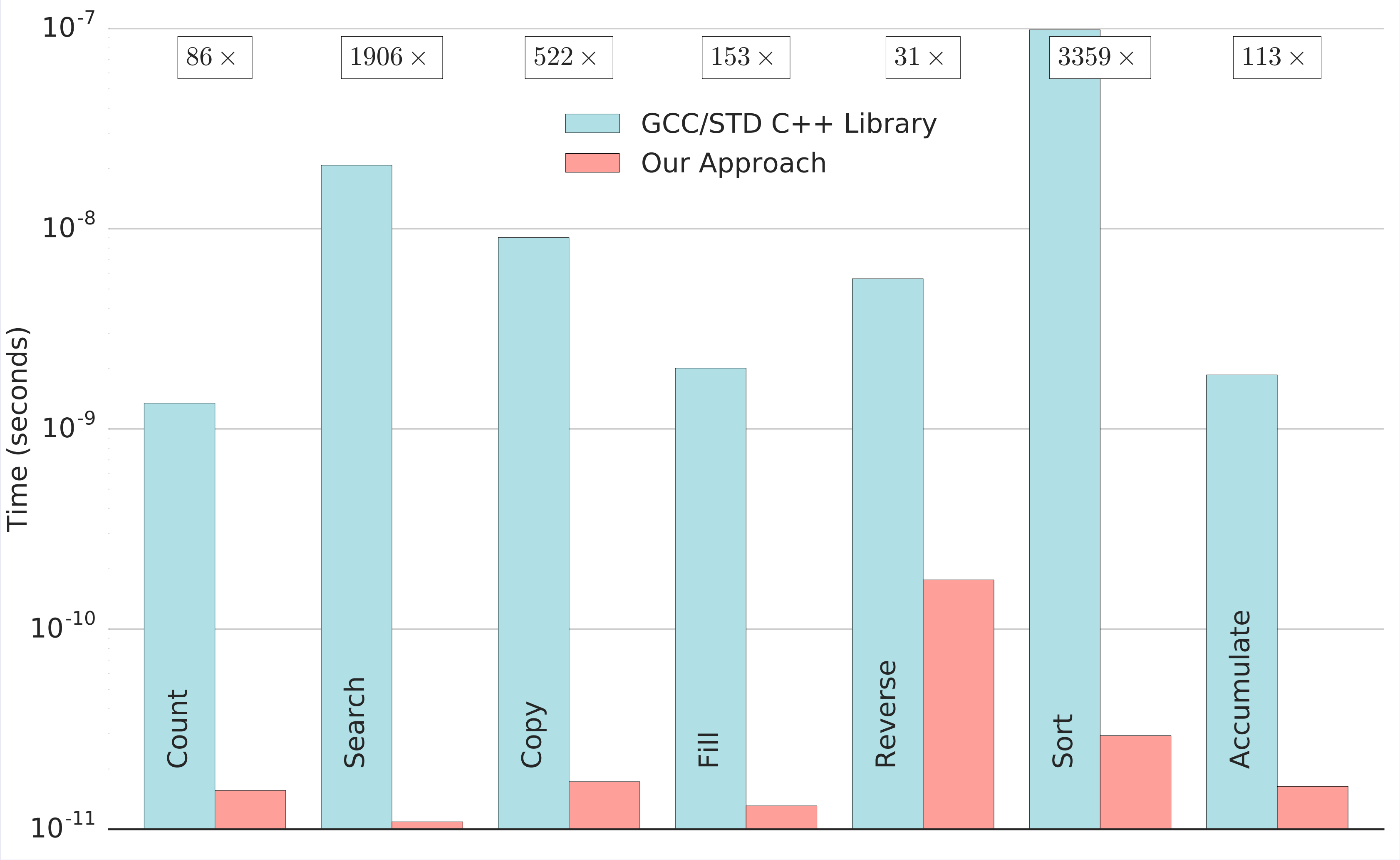


<table>
<tr><td colspan="4">Applications</td></tr>
<tr><td colspan="4">High level libraries</td></tr>
<tr><td>Wrappers and bindings</td><td>Python</td><td>R</td><td>Java</td></tr>
<tr><td>Optimized libraries</td><td colspan="2">Interpreters (Python, R...)</td><td>Virtual machines (JVM)</td></tr>
<tr><td colspan="4">Compiled, native, low level languages (C, C++...)</td></tr>
<tr><td colspan="4">Compilers, mostly written in C and C++ (GCC, LLVM...)</td></tr>
<tr><td colspan="4">Machine layer, assembly instructions</td></tr>
</table>

Because softwares are built as a stacks, low-level improvements and optimizations can be easily propagated to higher levels. Here, we are using C++ with template-based generative programming approaches to generate efficient code. PYTHON wrappers are automatically generated to interface high level machine learning libraries, including SCIKIT-LEARN, with native code.

## Bit manipulation and C++ standardization

Low-level optimization includes the improvement of bit manipulation algorithms involved in indexing strategies for implicit trees. The approach we developed allows to map efficiently complex bit manipulation patterns to specific assembly instruction sets. It outperforms the standard present-day C++ approach provided by compilers by two to three order of magnitudes, depending on the algorithm. Our bit library is currently discussed in the C++ standards committee to become a part of the next revision of the language.



## Current work and future directions

Current work involves optimization of kd-trees and nearest neighbor algorithms. Advanced memory allocation strategies with customized paging system already led to an order of magnitude improvement on tree construction. Because kd-tree construction also rely heavily on sorting, we developed a sorting network generator at compile-time that outperforms recursive algorithms for small sets of elements. Future directions of research include improvements on approximate nearest neighbor (ANN) algorithms for high dimensional datasets.

Achieving genericity, performance and ease of use, is one of the most challenging aspect of the STAMLA project. A significant effort is currently put on the software architecture and abstraction side, the goal being to be able to generate the whole complexity of tree data structures from a minimal set of abstractions, including explicit, implicit and serialized tree representations.

**PI: Vincent Reverdy & Robert J. Brunner - SI2 PI meeting - February 2017 - License: CC-BY**