Improvements to ramtools: Final Presentation

Georgi Haralanov

Goals

The main purpose for the project was to increase speed and memory efficiency of ramtools querying. This was to be done by including a command-line option for a multithreaded mode. The old functionality of single threaded runs is kept

Opening of the indexing data

When opening the data file 3 separate rntuples are open: metadata, indexing and main record. The metadata and main record show expected opening time as opposed to indexing which took up to 4 seconds to open each run. The storing and retrieval of non standard data types is unoptimized in ROOT. This was fixed by creating a separate indexing table which is comprised of the constituent elements of the RAMNTupleIndex class.

Index data class

```
 9 /**
 8  * \class RAMNTupleIndex
 7  * \brief Sparse genomic index for fast region queries on RNTuple files.
 6  *
 5  * The index is stored as a plain `std::vector<IndexEntry>` for efficient
 4  * serialisation and complemented by a lazily-initialised `std::map` that
 3  * supports O(log n) look-ups by (refid,pos).
 2  */
 1 class RAMNTupleIndex {
73 public:
 1     struct IndexEntry {
 2         int32_t refid;
 3         int32_t pos;
 4         int64_t entry;
 5     };
```

Speed up of opening of the file

```
std::pair<Long64_t, Long64_t> range;
try {

   auto index = ROOT::RDF::FromRNTuple("INDEX_FAST", file);
   range = FindIndex(index, refid, start, end);
} catch (...) {

   std::cerr << "[-]Fast index wasn't found\n[*]Creating fast index ...\n";
   auto index_old = ROOT::RDF::FromRNTuple("INDEX", file);
   ROOT::RDF::RSnapshotOptions opts;
   opts.fOutputFormat = ROOT::RDF::ESnapshotOutputFormat::kRNTuple;
   opts.fMode = "UPDATE";
   index_old.Snapshot("INDEX_FAST", file, {"index_entries.pos", "index_entries.refid"
   , "index_entries.entry"}, opts);
   auto index = ROOT::RDF::FromRNTuple("INDEX_FAST", file);
   range = FindIndex(index, refid, start, end);
   std::cerr << "[+]Index created!\n";
}
```

Addition of the multithreaded option

Multithreading was added as a separate function mirroring the old ones API with the addition of the number of threads that should be used. The ROOT implementation of Implicit MulTithreading (IMT) is used. The functionality of restricting the entry range upon which the RDataFrame operates with IMT enabled was added to ROOT in order to enable efficient use of the indexing data.

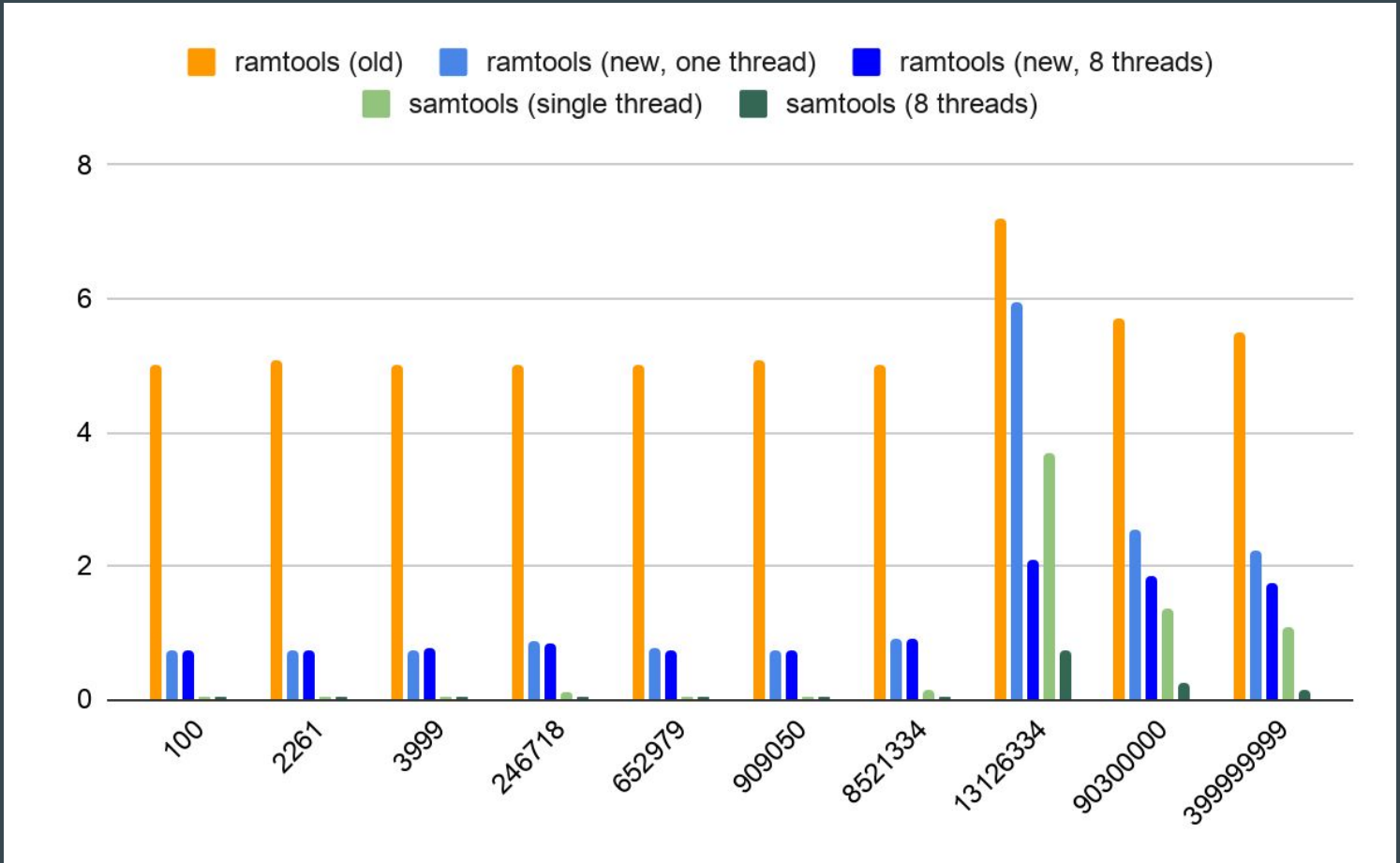
Code responsible for multithreading

```
8     st.Print();
7     st.Start();
6     std::cout << range.first << ' ' << range.second << '\n';
5     ROOT::EnableImplicitMT(numthreads);
4     std::vector<std::string> files = {file};
3     auto ram = ROOT::Internal::RDF::FromRNTuple("RAM", files, range);
2     st.Print();
1     st.Start();
13    auto filterfunc = [refid, start, end](int32_t refidentry, int32_t pos, uint16_t flag) {
1        return !(flag & FLAG_FILTER) && (refid == refidentry) && (pos >= start) && (pos <= end);
2     };
3     auto filtered = ram.Filter(filterfunc, {"record.refid", "record.pos", "record.flag"});
4     auto count = filtered.Count();
5     auto res = *count;
6     st.Print();
7     return res;
8  }
```

Final results

When compared to the previous implementation, the RDataFrame one out competes it even with a single thread. With small amounts of reads the query seems to be fast enough for a smooth workflow but still slower than samtools which could be attributable to some sort of optimization of the serialization and deserialization of files in their C library - htslib. This phenomenon is seen across the whole range of conducted tests. If any attempt of further improvements is to be made than htslib should be studied for more insight on the matter.

Benchmark results



Moving forward

For the near future the current speed of querying should be more than sufficient for use/debugging of the tool. More attention could be directed to the compression used and file support. An interesting approach that could be explored might be something similar to the difference based compression used in CRAM. Storing a reference sequence in a unique rntuple alongside another rntuple which contains the target sequence differences. This could allow for the compression of multiple different genome sequences into one file while saving space by sharing the same reference sequence.

Any Questions?