Presented by: Jeffrey Zhang

June 3rd, 2026

Enhance and Develop GeneROOT Infrastructure

Project Introduction

Mentors: Martin Vassilev, Vassil Vassilev, Aaron Jomy

About Me - Jeffrey Zhang

Education:

- 3rd-year B.S. studying Physics at Nagoya University, Japan

Hobbies:

- Competitive Programming
- Participate Hackathons
- Develop Video Games

Technical Skills:

- Python, C++, C#
- React.js, Node.js,
- Numpy, Pandas,
- (new!) ROOT, RAMtools

Genome Data

- Composed of many sequences of ACTG.
- Very very long.
- Basic sequences can be ~10^2 gigabytes large.
- Dealing with data this massive makes every simple task such as
  - Storing
  - Transfering
  - Analysing

  a non-trivial problem

Introduction to RAMtools

- Genome sequences needs specialized storage formats
  - SAM (Sequence Alignment Map)
  - BAM (Binary Alignment Map)
  - CRAM (Compressed BAM)
  - TTree
  - (new!) RNTuple
- Borrowing ROOT framework from CERN to store genome sequences
  - Achieve self-contained reference-free compression
- SAMtools convert data among SAM, BAM, CRAM
- RAMtools convert data from SAM to TTree / RNTuple

Where the project stands today

During GSoC 2025, Aditya Pandey established the RNTuple data model for genome sequences. It supports SAM to RNTuple conversion, a benchmark against CRAM on the sample HG00154.

Still proof of concept, not yet a usable pipeline component. This project aims to close that gap.

1

Hard Coded Benchmarks

- A single test sample with hard-coded file paths.
- Benchmark not automated.
- Difficult for other researchers to reproduce the results.

2

Optimizations To Be Made

- `GetRowsInRange()` index lookup performs a linear scan instead log(N).
- Configurable index granularity when importing from SAM
- And More.

3

No RNTuple To SAM

- Cannot export to SAM
- Missing key analysis features such as Stats(), Sort(), Merge(), and others.

Improvements to RAMtools

- Benchmark on heavy bioinformatics datasets
  - Refactor existing benchmark suite
  - New benchmark metrics
- Benchmark against various file formats
  - Compare RNTuple against SAM, BAM, and CRAM
- Improve data compression algorithms
  - Crumble, QVZ, CALQ, P-block, etc
- Optimize indexing and search capabilities for large genomic datasets
  - Numerous fixes
- Add support for Stat(), View(), Split(), Merge(), and Sort()
  - Functionalities mirror SAMtools

1) Benchmark on Heavy Bioinformatics Datasets

Refactor existing benchmark suite

```
class RegionQueryFixture : public benchmark::Fixture {
public:
   void SetUp(const benchmark::State &state) override
   {
      region_idx_ = static_cast<int>(state.range(0));

      sam_file_ = "/media/aditya/213e0e46-6f86-4288-8b79-74851c34314f/
         output_big.sam";
      ttree_root_file_ = "/media/aditya/213e0e46-6f86-4288-8b79-74851
         c34314f/output_big_lzma.root";
      rntuple_root_file_ = "/media/aditya/213e0e46-6f86-4288-8b79-74851
         c34314f/output_root.root";
   }
}
```

Current `region_query_benchmark.cxx` hardcodes benchmark

- Introduce command-line based benchmarking.
- Allow users to import custom datasets from their local disk.
- Run only the selected tests from the benchmark.
  - Range Querying Performance
  - File Size Reduction
  - And more

1) Benchmark on Heavy Bioinformatics Datasets

Implement Google Benchmark Memory Manager

benchmark > generate_sam_benchmark.cxx

```
4
5   static void BM_GenerateSAM(benchmark::State &state)
6   {
7       int num_reads = state.range(0);
8       for (auto _ : state) {
9           GenerateSAMFile("benchmark_temp.sam", num_reads);
10          std::remove("benchmark_temp.sam");
11      }
12
13      state.counters["reads_per_second"] = benchmark::Count
14      state.counters["bytes_per_second"] = benchmark::Count
15  }
16
```

Currently only uses the Speed metrics from Google Benchmark (`generate_sam_benchmark.cxx`).

```
ADD_CASES(TC_ConsoleOut, {{"^BM_empty %console_report$"}});
ADD_CASES(TC_JSONOut, {{"\"name\": \"BM_empty\",$"},
                       {"\"family_index\": 0,$", MR_Next},
                       {"\"per_family_instance_index\": 0,$", MR_Next},
                       {"\"run_name\": \"BM_empty\",$", MR_Next},
                       {"\"run_type\": \"iteration\",$", MR_Next},
                       {"\"repetitions\": 1,$", MR_Next},
                       {"\"repetition_index\": 0,$", MR_Next},
                       {"\"threads\": 1,$", MR_Next},
                       {"\"iterations\": %int,$", MR_Next},
                       {"\"real_time\": %float,$", MR_Next},
                       {"\"cpu_time\": %float,$", MR_Next},
                       {"\"time_unit\": \"ns\",$", MR_Next},
                       {"\"allocs_per_iter\": %float,$", MR_Next},
                       {"\"max_bytes_used\": 42000$", MR_Next},
                       {"}", MR_Next}});
```

Snippet of the usage for Memory Manager from documentation here.

2) Benchmark Against Various File Formats

Compare RNTuple against SAM, BAM, and CRAM

Only `chromosome_split_benchmark.cxx` currently shells out to SAMtools. I'll extend that system()-call pattern across every benchmark script.

- Measure SAM, BAM, and CRAM against RAMtools / RNTuple on identical datasets.
- One baseline harness, portable to a standalone benchmark repository later.

cross_format_benchmark.cxx — PATTERN

```
for (auto _ : state) {
  // drive the reference toolchain via system()
  system("samtools view -bS in.sam -o out.bam");
  system("samtools sort  out.bam   -o sorted.bam");
  system("samtools index sorted.bam");
}
// same datasets, same metrics, head-to-head:
//   SAM · BAM · CRAM  vs  RAMtools / RNTuple
```

```
for (auto _ : state) {
  std::string bam_file = "bench_st_tmp.bam";
  std::string sorted_bam = "bench_st_sorted.bam";

  system(("samtools view -bS " + sam_file + " -o " + bam_file + " 2>/dev/
     null").c_str());

  system(("samtools sort " + bam_file + " -o " + sorted_bam + " 2>/dev/
     null").c_str());

  system(("samtools index " + sorted_bam + " 2>/dev/null").c_str());
```

Current `chromosome_split_benchmark.cxx` calls SAM through shell command

3) Improve data compression algorithms

Do literature review on compression algorithms specialized for genome datasets

Examples:

- Crumble
- QVZ
- CALQ
- P-block
- And more

RAMNTupleRecord.h — EXTEND THE ENUM

```
enum EQualCompressionBits {
  kPhred33        = 1 << 14,
  kIlluminaBinning = 1 << 15,
  kDrop           = 1 << 16,
  kCrumble        = 1 << 17,  // new

  kCALQ           = 1 << 18,  // new

  …
};
```

4) Optimize Indexing and Search Capabilities for Large Genomic Datasets

Numerous Improvements

- Existing `GetRowsInRange()` in `RAMNTupleRecord.cxx` performs a linear scan over (time complexity O(N)) the entire index vector for every query. Can fix it with binary search with O(Log(N)).
- Configurable index granularity for SAM to RNTuple file conversion.

RAMNTupleRecord.cxx — SORT ONCE, SEARCH FAST

```cpp
// sort fIndex by (refid, pos) — then binary-search
std::sort(fIndex.begin(), fIndex.end(),
  [](const IndexEntry &a, const IndexEntry &b){
    if (a.refid != b.refid) return a.refid < b.refid;
    return a.pos < b.pos;
  });

// GetRow now folds into the general case:
GetRow(refid, pos) = GetRowsInRange(refid, pos, pos).front();
```

SamToNTuple.cxx

src > ramcore > SamToNTuple.cxx

```cpp
25  namespace {
26
27  constexpr uint16_t kUnmapped = 0x4;
28  constexpr int32_t kPositionInterval = 10000;
29  constexpr int64_t kMappedInterval = 100;
30
31  } // namespace
```

4) Optimize Indexing and Search Capabilities for Large Genomic Datasets

(Cont.)

Remove `fIndexMap`, `fIndex` alone is sufficient for binary search.

```
// RAMNTupleIndex Implementation
void RAMNTupleIndex::AddItem(int32_t refid, int32_t pos, int64_t row)
{
   fIndex.push_back({refid, pos, row});
   auto key = std::make_pair(refid, pos);
   fIndexMap[key] = row;
}

int64_t RAMNTupleIndex::GetRow(int32_t refid, int32_t pos) const
{
   if (fIndexMap.empty() && !fIndex.empty()) {
      const_cast<RAMNTupleIndex *>(this)->RebuildMap();
   }

   auto key = std::make_pair(refid, pos);
   auto low = fIndexMap.lower_bound(key);
```

Code snippet from `RAMNTupleRecord.cxx` showing the usage of `fIndexMap`

5) Add support for Stat(), View(), Split(), Merge(), and Sort()

Functionalities mirror SAMtools

All functionalities draw parallel to SAMtools documentation such as: https://www.htslib.org/doc/samtools-view.html

Summary

Benchmark
- Refactor existing Benchmark
- Make new Benchmark cross-format

Improvements
- Improve data compression
- Optimize indexing and search capabilities

New Features
- Add support for Stat(), View(), Split(), Merge(), and Sort()

Thank You!