KMI HASH Benchmark Summary

Summary Version

1.1

Purpose of Benchmark

KMI, K-mer Matching Interface, is a domain-specific library for genome assembly and mapping analysis applications. Genome assembly and mapping fundamentally rely on efficient matching of k-mers (which are short snippets of DNA sequences) on distributed memory machines. One purpose of this data-centric benchmark is to evaluate the performance of the architecture integer operations, specifically for hashing, and for memory-intensive genomics applications.

Note: This is a single node benchmark. Though KMI Hash is designed for Distributed memory parallelism using MPI, for CORAL, we are interested in the single node performance for this benchmark.

Characteristics of Benchmark

The benchmark performs the following steps:

1. Generate the sequence database (timed)
   Typically, the sequences are in the range of 100-250 in length depending on the sequencing technology used. For CORAL, we will assume sequences of length 100.

2. Construct the hash table using the generated sequences (timed)
   - Both, the sequence and its reverse complement will be added to the hash table
   - In the default KMI distributed memory implementation, the sequences are first globally sorted and re-distributed. Next, the re-distributed set is inserted into the local hash table.

3. Generate a set of queries (timed)
   - For CORAL, each query is currently of length 20
   - The reverse complement of the query is checked as well.
   - In the default KMI implementation, the queries are distributed among the various ranks.

4. Lookups: For each query, check if the query is present in the set of sequences
   a. The lookups are timed
   b. The “hits” are counted

   In the default KMI implementation, the queries are sent to an appropriate “home” rank which looks this up in its local hash table.

5. Compute and output the performance data

Mechanics of Building Benchmark

There is a makefile to build the benchmark.

Mechanics of Running Benchmark

The baseline implementation requires two MPI ranks on the single node to run.

Total number of strings (N) = 80,000,000
Total number of queries (M) = 8,000,000
String length (default) = 100
Query length (default) = 20

As an example: For 2 MPI rank on a single node:
n = 40,000,000
m = 4,000,000

Similarly, For 16 MPI ranks on a single node:
n = 5,000,000
m = 500,000

Medium problem: single node

Total number of strings (N) = 512,000,000
Total number of queries (M) = 64,000,000
String length (default) = 100
Query length (default) = 20

Large problem: single node

Total number of strings (N) = 2,400,000,000
Total number of queries (M) = 128,000,000
String length (default) = 100
Query length (default) = 20