MEMOCODE 2012 Hardware/Software Codesign Contest: DNA Sequence Aligner

Stephen A. Edwards
Department of Computer Science, Columbia University
New York, NY, USA
Email: sedwards@cs.columbia.edu

***Abstract*—This year's design contest problem is part of the DNA sequence alignment problem: exact substring matching. The challenge is to efficiently locate millions of 100-base-pair short read sequences in a 3-million-base-pair reference genome. Good solutions will combine judicious algorithm design with carefully designed data handling architecture.**

***Keywords*-DNA sequence alignment; string matching; hardware/software codesign;**

I. Introduction

This is the sixth MEMOCODE Design Contest, which has been held yearly since 2007. Every year, the contest organizers have proposed a new problem and teams from across the world have come together to design and build innovative hardware/software systems that solve the problem. Past years' problems were matrix multiplication [1], sorting encrypted data [2], Cartesian-to-polar interpolation [3], deep packet inspection [4], and network simulation [5].

This year's problem is DNA sequence alignment. Modern high-speed DNA sequencers can break an organism's genome into millions of short pieces and read the base pairs (perhaps 100 bases per piece) from these short sequences. The computational challenge is to reassemble these pieces into the original genome.

A typical first step in the alignment problem—the specific problem for this year's contest—is to find the locations where the short sequences appear in a reference genome. While this would rarely be the final answer desired by biologists, it is a useful initial filtering step.

Efficiently coping with large amounts of data is the key problem in this year's challenge. The problem itself is embarrassingly parallel and can easily be split into arbitrarily many small sub-problems; the challenge is how best to do this under limited memory and bandwidth. The human reference genome is about 700 Mb, but the short read sequence data can be over ten times larger. While such data volumes fit easily in modern mass storage (e.g., hard drives or flash memory), and has just become practical for high-capacity DRAMs, no single chip can store and process it.

Algorithm design is the other challenge. String matching is a well-studied classical computer science problem and many algorithmically efficient techniques are known, but many solutions do not scale up to the problem sizes considered here and are better-suited to classical sequential processors.

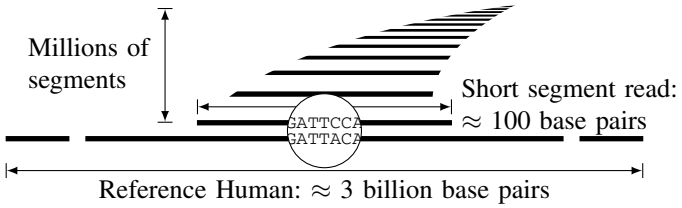

Figure 1. The DNA sequence alignment problem: (not to scale) find the locations, if any, where each segment read appears verbatim in the reference sequence. The number of sequence reads is typically expressed as *coverage*: the relative number of base pairs in the segment reads compared to the actual (human) genome. 2× as many is low; 20× is "deep."

II. The Problem

Fast DNA sequencing is at the cutting edge of biology research. The goal is to quickly and cheaply read an organism's entire genome, making it possible to check for disease-causing mutations, look for evolutionary patterns, and a host of other useful studies. Since the first human genome was sequenced in 2007 [6], many far faster reading techniques have been developed [7], leading to a deluge of data—the subject of this year's design competition.

The human genome consists of a sequence of roughly three billion base pairs (bp). There are four different base pairs (abbreviated G, A, T, and C), so each base pair can be encoded in two bits, setting the size of the human genome at about 700 Mb of data—about a single CD's worth.

Nobody knows how to read all the base pairs from a single lengthy DNA molecule in sequence; instead, sequencing systems work by making many copies of a single DNA molecule, breaking them up into many short pieces, then reading a short, fixed number of base pairs (say, 100) from each piece [7].

The alignment challenge is to reassemble these millions of short sequences into the original 3 Gbp sequence [8]. The whole process is a bit like taking thousands of copies of a single book, shredding them all, then grabbing a few handfuls of shredded paper at random and trying to reconstruct the text of the book. An additional complication: the copies are not truly identical—some typos may have occurred.

One thing that helps the alignment process is that human DNA sequences are largely identical, differing by only about 0.1% (roughly one base pair out of a thousand), and that a handful of individuals' genomes are already known.

```
>1 dna:chromosome chromosome:GRCh37:1:1:249250621:1
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
...
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNTAACCCTAACCCTAACCCTA
ACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTA
...
AACCAGGTCCAGGAAGAAGGTGCAAAGACAGCATTCCAGGTAAAAGAAACAGCTTGAACA
AAAAGTGTGTAGGGGAACCGCAAGCGGTCTTGAGTGCTGAGGGTACAATCATCCTTGGGG
AAGTACTAGAAGAAAGAATGATAAACAGAGGCCAGTTTGTTAAAAACACTCAAAATTAAA
...
```

Figure 2. Excerpts from the human reference genome [9] in FASTA format. Each data line is 60 characters long. "A," "C," "T," and "G" each represent a single base pair; "N" represents any base pair (i.e., is unknown). The ellipses (. . .) were added to this figure for clarity; they are not part of the actual file.

```
@ERR050082.521 HS18_6628:6:2303:13171:165808#3/2
ATTCTCCTCCAAGGCTGCAGAGGGGGCAGGAATTGGGGGTGACAGGAGAGCTGTAAGGTCTCCAGTGGGTCATTCTGGGCCCAGAGATGGGTGCTGAAGC
+
:DDFGFCFHEELJMJHIFDEOHHFIKDFIK;CEILH@NIAK:LHIKMIJ9HILBJJGJII7HIHJFJJGCJFGI?CIGJEG@JKJFGGFIDG>DC=D?>>
@ERR050082.686 HS18_6628:6:2202:16159:5695#3/2
TTAATTTCACCAGTGCCTTGTTAACTGATGTATCATATGCATGGATTTTTCTTTTTTTCTTTTTTTTTTGACTTTTATTTTAGGTTCGGGGGTACACGTG
+
:FDGEFGHJJKLJMJKMKJMMIGMLKMILKJNMLLILNLKKJLHKKMHJJGKLJJJIJHLKJIHEJJJJCAJKIIJDHJID@JKJF@GI;I:>DC@B<=6
```

Figure 3. Two short sequence reads [10] in FASTQ format. Each sequence consists of four text lines. The first is essentially a comment, which names the run and details about how this particular sequence was read. The second line is the sequence of base pairs, 100 per sequence in this case, coded as the usual ATGC. The third line "+" is essentially unused. The fourth line encodes the quality of each base pair read, which we will ignore.

This year's design problem is a data-intensive but algorithmically simple part of the DNA sequence alignment problem: given a reference human genome and millions of short read sequences, report how many times and where each short read sequence *appears exactly* in the reference genome (Figure 1). For the Unix-minded, this is something like running *fgrep* on a 700 Mb file with a million short patterns to match. A complete reassembly algorithm would use this as a starting point to filter out already-known parts of the genome then follow it with an inexact matching step.

The string-matching problem is very well-studied, and there has also been significant work on the sequence alignment problem, e.g., a number of open-source solutions already exist [11], [12], [13]—see the survey by Li and Homer [8]. The reason to consider it here is that the problem is not considered "solved" when the corpus (pun intended) and the number of search patterns is this large.

The supplied reference solution is naïve and brute-force; good solutions will be much more clever. Memory capacity constraints are paramount: while it may be possible to hold the whole reference genome in memory, holding all the short sequences may not be practical. Another fundamental question is what sort of indexing, if any, to do. While building and index takes time, it can save time in the long run.

III. Data Issues

We will use data collected as part of the 1000 Genomes project [14], whose goal is to read the DNA of 1000 individuals across the globe. They have collected a lot of DNA sequence data, made it public, and aligned some of it.

There are two mirrors of the data in the project, one for the Americas [15]; one for the rest of the world [16]. In the root directory of their FTP site, there are a number of README files from which most of the following information was derived.

A. The Reference Genome

The reference genome is a *gzip*-compressed file [9] in FASTA format: a text header followed by a text representation of the base pairs. Figure 2 shows a snippet of the (uncompressed) reference genome file.

B. Short Sequence Reads

The 1000 Genomes project has a lot of short sequence read data, all in gzip-compressed FASTQ format, an example of which is shown in Figure 3. The project has collected lots of short sequence read data from different individuals (its goal is 1000, as its name suggests), arranged in the *sequence_read* directories under the various directories under *ftp/data/*.

Because of coverage (the redundancy it implies) and accompanying accuracy information, the short sequence read data is much larger than the reference genome, even though both ultimately contain the same amount of information. For example, while the human reference genome [9] is 803 Mb (compressed), the data for the NA06985 individual totals about 36 Gb.

```
0:          0
1:          1
2:          2
3:          3
4:          3
5:          8
6:         10
7:      19920 + 2 others
8:       1130
9: -
```

Figure 4. Output from running the reference implementation on the supplied test case. For each sequence (their indexes are listed on the left), it lists the index, in base pairs, where the sequence was found in the reference genome, which may be 0 (in the case of sequence 9), 1 (sequences 0–6, and 8), or more (sequence 7).

IV. The Reference Implementation

The *2012-memocode-contest.tar.gz* compressed archive file contains a primitive DNA sequence aligner implemented in C along with some small test cases. It compiles and runs under 32- and 64-bit Linux and other Unix-derived operating systems. The full program uses low-level Unix I/O facilities (e.g., *open()*, *stat()*, and *mmap()*), but the core string searcher (*match()* in *align.c*) only uses C library routines *malloc()* and *memcmp()*, both of which could be replaced.

The reference implementation does exact substring matching, looking for identically-sized short read sequences (e.g., 100 base pairs each) against a (long) reference genome. At the end, for each sequence, it reports how many times the sequence was found and, if it was found more than once in the reference genome, the index of the last match, although returning the index of any match is allowed when there is more than one. Figure 4 shows the output from the reference implementation running on the included testcase.

Both the reference genome and the sequences are stored on disk and in memory in a packed (but not compressed) form in which each byte holds four base pairs, two bits per base, with the LSB holding the first base, the next base in the next two bits, and so forth. The reference implementation includes simple format conversion programs *fasta2bin* and *fastq2bin* that convert textual FASTA and FASTQ files into this packed binary format, documented in the source files.

The reference implementation is a brute-force string matcher: it maintains a buffer through which the entire reference genome is shifted one base pair at a time. The contents of this buffer is repeatedly compared to every short read sequence; matches are recorded. The complexity in this implementation arises from the packed representation, chosen to enable multiple base pairs to be compared in parallel and to keep the memory footprint within a few gigabytes.

The goal of this contest is to improve upon this naïve implementation. Sources of improvement include parallelism, indexing, and more-easily-searched data structures.

V. The Contest

The reference implementation defines the inputs and outputs for this contest. Solutions must start with the packed binary form of the reference genome and sequence read data and report their results in the textual format defined in Figure 4 and in the reference implementation. Each team's results must match those from the reference implementation exactly except for sequences that appear more than once in the reference genome (e.g., sequence 7 in Figure 4). In these cases, a solution must report one matching index, but may choose one non-deterministically.

A. Test Data

Teams will use the human reference genome from the 1000 Genomes website [9], but will not be given the read sequences until the end of the contest. The read sequences will be one of the individuals from the 1000 Genomes project and each read sequence will be exactly 100 base pairs. Teams may use data for the NA06985 individual [10] for testing purposes. Thus, a solution may be tuned to only work with the given human reference genome but should accept fairly arbitrary short read sequence data.

B. Metrics

Two metrics will be considered when judging the performance of designs: runtime and cost.

Runtime is the wall-clock time from when delivery to the platform of the packed, binary short sequence data begins to when all the sequence matching information (e.g., as in Figure 4) has been returned to mass storage. In particular, time taken to run *gunzip* and *fasta2bin*, and *fastq2bin* to process sequence data is not counted in the total time.

Furthermore, any time spend loading or indexing the reference genome data is not counted in the total. Effective designs may thus assume the reference genome rarely changes but the short sequence read data is fresh each time.

While teams are expected to match all the short sequence reads for a particular individual, if a team so chooses (e.g., to reduce memory requirements), it may choose to search for only a fraction of the supplied short sequence data. The team must negotiate with the judges over what fraction of the sequence data it will use (e.g., 10%); the judges will select the actual sequences. The runtime assigned to the team will be derated according to the fraction, e.g., if a team chooses to search for only 25% of the sequences, its judged runtime will be quadrupled from the measured value.

The *cost* of the system is the publicly listed academic price, or if that is unavailable, the publicly listed retail price, or if that is unavailable, a price as estimated by the judges.

For the purposes of cost calculation, a host workstation is not counted if all it does is supply the source sequence read data (e.g., by running *gunzip* and *fastq2bin* and receive the results. It may also be used to index the reference genome

without being considered part of the total system cost. However, if the workstation performs additional indexing or preprocessing of the sequence data, it will be considered part of the system for cost purposes.

C. The Unlimited Class

One way to win the contest is to have the shortest net runtime, as defined above. In this class, platform cost, power consumption, and other metrics will be ignored.

D. The Normalized Class

In this class, the winner will be the one with the smallest runtime-cost product; a team who can sequence everything in 20 hours a $100 platform will be equivalent to a $200 platform that took 10 hours. It is possible for a single team to win in both the unlimited and normalized classes.

E. Schedule

Teams will be given the reference design and this problem description on March 1st, 2012. On April 1st, 2012, teams will be told which individual from the 1000 Genomes project they are to sequence (i.e., which subdirectory of the 1000genomes website to download and start processing). Within a week, teams are expected to report the total time it took from when the platform started processing the binary-formatted data for the read sequences to when it has completed reporting match information in the form reported by the reference implementation. Again, any indexing or preprocessing of the reference genome can be performed before April 1st without it counting towards runtime.

F. Suggested Platforms

Teams may wish to consider, but are not limited to, FPGA-based development platforms such as Xilinx's XUPV5, or Altera's DE4, GPGPUs such as those supporting NVIDIA's CUDA platform, the Sony Playstation 3 (i.e., using the CELL processor), or even clusters of off-the-shelf X86-style servers. Again, in the unlimited class, runtime will be the only criterion; in the normalized class, total system cost will also be considered.

G. Hints

Start by reading Li and Homer's survey [8] on existing algorithms, which classifies algorithms into those using hash tables, suffix/prefix tries. Langmead et al. [11] has a nice description of how they used the Burrows-Wheeler indexing algorithm to do alignment.

Indexing, parallelism, and divide-and-conquer will probably help. Indexing, or at least merging, the short read sequences to minimize the number of exact matches to be attempted seems like an excellent idea. This problem is embarassingly parallel so it should be easy to search, say, multiple groups of sequences simultaneously. Finally, since the string matching problem is not especially sequential, breaking it into multiple pieces that comfortably fit into available memory should work very well.

References

[1] F. Brewer and J. C. Hoe, "MEMOCODE 2007 co-design contest," in *Proceedings of the International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, Nice, France, May 2007, pp. 91–94.

[2] P. Schaumont, K. Asanovic, and J. C. Hoe, "MEMOCODE 2008 co-design contest," in *Proceedings of the International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, Anaheim, California, Jun. 2008, pp. 151–154.

[3] F. Brewer and J. C. Hoe, "2009 MEMOCODE co-design contest," in *Proceedings of the International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, Cambridge, MA, Jul. 2009, pp. 66–68.

[4] M. Pellauer, A. Agarwal, A. Khan, M. C. Ng, M. Vijayaraghavan, F. Brewer, and J. Emer, "Design contest overview: Combined architecture for network stream categorization and intrusion detection (CANSCID)," in *Proceedings of the International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, Grenoble, France, Jul. 2010, pp. 69–72.

[5] D. Chiou, "MEMOCODE 2011 hardware/software codesign contest: NoC simulator," in *Proceedings of the International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, Cambridge, UK, Jul. 2011, pp. 73–76.

[6] S. Levy, G. Sutton, P. C. Ng, L. Feuk, A. L. Halpern *et al.*, "The diploid genome sequence of an individual human," *PLoS Biology*, vol. 5, no. 10, pp. 2113–2144, Oct. 2007. [Online]. doi: 10.1371/journal.pbio.0050254

[7] M. L. Metzker, "Sequencing technologies—the next generation," *Nature Reviews. Genetics.*, vol. 11, no. 1, pp. 31–46, 2010. [Online]. doi: 10.1038/nrg2626

[8] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in Bioinformatics*, vol. II, no. 5, pp. 473–483, 2010. [Online]. doi: 10.1093/bib/bbq015

[9] 1000 Genomes Project, "Human reference genome." [Online]. Available: ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/technical/reference/human_g1k_v37.fasta.gz

[10] ——, "Sequence read for individual na06985." [Online]. Available: ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/data/NA06985/sequence_read/ERR050082.filt.fastq.gz

[11] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, Mar. 2009. [Online]. doi: 10.1186/gb-2009-10-3-r25

[12] R. Li, C. Yu, Y. Li, T. Lam, S. Yiu, K. Kristiansen, and J. Wang, "SOAP2: an improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–7, Jun. 2009. [Online]. doi: 10.1093/bioinformatics/btp336

[13] C. Liu, T. Wong, E. Wu, R. Luo, S. Yiu, Y. Li, B. Wang, C. Yu, X. Chu, K. Zhao, R. Li, and T. Lam, "SOAP3: Ultrafast GPU-based parallel alignment tool for short reads," *Bioinformatics*, Jan. 2012, to appear. [Online]. doi: 10.1093/bioinformatics/bts061

[14] "The 1000 Genomes Project." [Online]. Available: http://www.1000genomes.org

[15] 1000 Genomes Project, "FTP mirror for the Americas." [Online]. Available: ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/

[16] ——, "FTP mirror for the world." [Online]. Available: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/