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*—The MEMOCODE design contest for 2012 is exact substring matching: a simplified form of the DNA sequence alignment problem. The challenge is to efficiently locate millions of 100-base-pair short read sequences in a 3-million-base-pair reference genome. Contestants had a month to create a fast system that ran on a given set of test data. Entries were judged both on absolute time and the product of time and system cost. The two winning groups, which were invited to contribute papers describing their solutions, judiciously chose algorithms that exploited powerful hardware. The two winning entries employed a hash algorithm running on a Convey HC-1 FPGA/multicore hybrid with an aggressive memory system and a Burrows-Wheeler/hash hybrid running on a 12-core Intel system was second.**

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

I. Introduction

Every year since 2007, the organizers of the MEMOCODE Design Contest have proposed a design problem and invited teams from around the world to design and build innovative hardware/software systems to 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 comes from DNA sequence alignment. Modern high-speed DNA sequencers break an organism's genome into millions of short pieces and read the (e.g., 100) base pairs in each piece. The computational challenge is to reassemble these pieces into the organism's 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 does not answer the real biological question (i.e., details of the organism's complete genome sequence and how it differs from others), it is a useful initial filtering step.

Efficiently coping with large amounts of data was the key problem in this year's challenge. The problem itself is embarrassingly parallel and can easily be split into 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 is easily 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, currently no single chip can store and process it all.

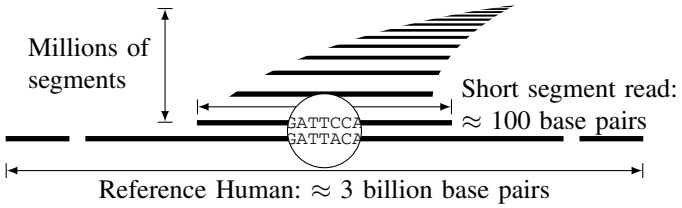

Figure 1. The DNA sequence alignment problem: (not to scale) find the locations, if any, where each segment read appears verbatim in the reference sequence. The number of sequence reads is typically expressed as *coverage*: the relative number of base pairs in the segment reads compared to the actual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

Algorithm design was the other challenge. While string matching is a well-studied computer science problem with many known efficient techniques, many do not scale up to the problem sizes considered here and are better-suited to sequential processors. The winning entries (Section VI) used hashing or the Burrows-Wheeler transform [6] to index the reference genome.

An earlier revision of this paper served as the problem specification for the teams. In this version, I describe the problem (Section II), the source and format of the data contestants were to use (Section III), the reference implementation I supplied (Section IV), the contest rules (Section V), and finally a quick summary of the entries and the results of the contest (Section VI).

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 [9], many far faster reading techniques have been developed [10], 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, current

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

Figure 2. Excerpts from the human reference genome [7] in FASTA format. Data lines are 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 [8] 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.

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 (e.g., 100) from each piece [10].

The alignment challenge is to reassemble these millions of short sequences into the original 3 Gbp sequence [11]. The whole process is like making thousands of copies of a single book, shredding them, and 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 really identical—a handful of misprints always occur.

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.

This year's design problem was 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 [12], [13], [14]—see the survey by Li and Homer [11]. 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 reference solution I supplied (see Section IV) was naïve and brute-force; good solutions were 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. The contest rules say time spent preprocessing (e.g., indexing) the reference sequence did not count since the same reference sequence can be used for any human DNA.

III. Data Issues

Contestants were instructed to use data collected as part of the 1000 Genomes project [15], 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. The following information comes mainly from the *README* files on their well-organized FTP site.

A. The Reference Genome

The reference genome is a *gzip*-compressed file 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.

```
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).

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.

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 is 803 MB (compressed), the data for the NA06985 individual totals about 36 GB.

IV. The Reference Implementation

The reference implementation I supplied to contestants contains a brute-force 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 was 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.

V. The Contest

The reference implementation defined the inputs and outputs for the contest. Solutions had to 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 had to 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, the solution could report any matching index, i.e., may choose one non-deterministically.

A. Test Data

Teams used the human reference genome from the 1000 Genomes website [7], but were only told which read sequences to use at the end of the contest. They were, however, told the read sequences would be one of the individuals from the 1000 Genomes project and each read sequence will be exactly 100 base pairs. Teams were told to use data from the NA06985 individual for testing purposes. Thus, solutions could be tuned to only work with the given human reference genome but had to accept fairly arbitrary short read sequence data.

B. Metrics

Two metrics—runtime and cost—were considered when judging the performance of designs.

Runtime was 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 was not counted in the total time.

Furthermore, any time spent loading or indexing the reference genome data was not counted in the total. Solutions could thus assume the reference genome rarely changed but the short sequence read data is fresh each time.

While teams were asked to match all the short sequence reads for a particular individual, certain teams opted (e.g., to reduce memory requirements) to search for only a fraction of the supplied short sequence data and have their final

Table I
SUBMISSION STATISTICS, ORDERED BY RUNTIME

Group	Platform	Cores/ Threads	Memory (GB)	Algorithm	Runtime (s)	Cost (USD$)	Time-Cost ($\$ \times$ s)
Iowa State University	Convey HC-1		64	Hashed reference sequence	8.9×10^{-1}	67,100	6.0×10^{4}
IPM HPC, Iran	2 Intel Xeon X5650	12/24	48	Burrows-Wheeler/hash hybrid	2.2×10^{1}	2,355	5.2×10^{4}
IPM HPC, Iran	Intel Xeon X5650	6/12	16	Burrows-Wheeler	3.0×10^{2}	3,000	9.0×10^{5}
Iowa State University	23 Nvidia GTX 480	23/11040	35	Prefix table	1.8×10^{4}	27,800	5.1×10^{8}
Shahid Beheshti University, Iran	Intel Core i7 2600K	4/8	8	Hash/tree hybrid	2.4×10^{4}	1,800	4.3×10^{7}
IPM HPC, Iran	Nvidia Tesla C2050	1/448	9	Parallel brute-force search	6.2×10^{6}	1,850	1.1×10^{10}
KAIST, Korea	8 Nvidia Tesla M2050	8/3584	24	Hashed reference sequence	6.7×10^{6}	8,000	5.4×10^{10}
Sungkyunkwan University, Korea	Nvidia GT 530	1/96	1	Parallel brute-force search	3.1×10^{8}	50	1.6×10^{10}
University of Tehran, Iran	Altera DE2-115 FPGA		0.1	Parallel brute-force search	7.2×10^{8}	305	2.2×10^{11}

Notes: The Intel chips are general-purpose multicore processors: the X5650 has 6 cores; the Core i7 2600K has 4.
The Nvidia chips are general-purpose graphics processors (GPGPUs).
The Convey HC-1 combines a Xilinx Virtex 5 FPGA with an Intel Xenon dual-core processor and 8 DDR2 memory controllers.
Most runtimes were extrapolated from reported numbers based on the fraction of short sequences run.
Figures for memory represent aggregate system DRAM capacity; some numbers are estimates based on self-reported system descriptions.

runtimes de-rated by the fraction of sequences they actually ran, e.g., if a team chose to search for only 10% of the read sequences, their final runtime was multiplied by ten.

The *cost* of each solution was its price in US dollars, specifically the lesser of its academic or retail price, or if neither were available, a number estimated by the judges.

For the purposes of cost calculation, a host workstation was not counted if all it did was supply the source sequence read data (e.g., by running *gunzip* and *fastq2bin* and receive the results). It could also be used to index the reference genome without being considered part of the total system cost. However, if the workstation performed additional indexing or preprocessing of the sequence data, it was considered part of the system for cost purposes.

C. The Unlimited Class

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

D. The Normalized Class

In the second class, the winner was the one with the smallest runtime-cost product; a team whose solution could sequence everything in 20 hours on a $100 platform was equivalent to a $200 platform that took 10 hours. A single team could win in both the unlimited and normalized classes.

E. Schedule

Teams were given the reference design and this problem description on March 1st, 2012. On April 1st, 2012, teams were 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 were expected to report the total time it took their solution to process all the binary-formatted short read sequence data to when it completed reporting match information in the form reported by the reference implementation. Any indexing or preprocessing of the reference genome could be performed before April 1st without it counting towards runtime.

F. Suggested Platforms

Teams were told to consider, but were 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. In the unlimited class, runtime was the only criterion; in the normalized class, total system cost was also considered.

VI. RESULTS

Table I lists the final submissions, ordered by total runtime. The two winning teams are listed first: one from Iowa State University, which won the unlimited class, and one from the Institute for Research in Fundamental Sciences (IPM) in Iran, which won in the normalized class.

Estimated runtimes of the solutions spanned nearly nine orders of magnitude: the fastest system took less than a second; the slowest would have taken nearly 23 years. System costs varied more modestly (less than a factor of 1000), but the time-cost product still varied about six orders of magnitude.

Teams ultimately employed four different kinds of platforms: traditional Intel multicore server-class machines, Nvidia general-purpose graphics processor units (GPGPUs), a low-end Terasic/Altera DE2-115 FPGA board, and a Convey HC-1—a high-end multicore/FPGA hybrid with a very aggressive memory system. The costs for these ranged from fairly low (a single GPU card with a retail price of about $50) to quite high (the Convey retails for $67,100).

Memory capacity and algorithm choice appear to have been the determining factors. In general, solutions using

more memory reported shorter runtimes. While a basic constraint was the need to hold the whole reference sequence in memory at once, the more successful solutions used vastly more memory than absolutely necessary to store large indexes of the reference sequence. The winning group, which used the HC-1, built a 22.5 GB minimal perfect hash table for every 100 bp subsequence in the reference; other successful groups also built elaborate data structures for representing the reference sequence. By contrast, groups that implemented brute-force search had much slower solutions. One exception was the entry from KAIST: although it also used a hash table, the team reports their solution was hampered by excessive time spent copying data between CPU and GPU memory.

Despite this appearing to be an "embarrassingly parallel" problem, which would suggest it would be easy to use a lot of hardware to solve it quickly, the time-cost results suggest otherwise. For such problems, doubling the amount of hardware should halve the time while doubling the cost, suggesting the time-cost product should stay constant, but instead we observed many orders of magnitude difference in time-cost products across the solutions. The two slowest solutions (from SKKU and the University of Tehran) illustrated this: both used the brute-force-searching algorithm, but per dollar, the FPGA solution was still dramatically slower, perhaps because of memory bottlenecks.

Instead, algorithm choice was key to winning this contest. Two of the top three entries used the Burrows-Wheeler Transform (BWT) [6] to index the reference genome. The BWT, a reversible string permutation, was originally intended for data compression, since it tends to group identical characters into runs; Ferragina and Manzini [16] later adapted it to string searching; Lam et al. [17] showed its utility in DNA sequence matching.

VII. Conclusions

Nine teams from six institutions ultimately submitted working solutions to the problem, whose speeds ranged over nearly eight orders of magnitude. Algorithm selection, coupled with sufficient memory resources and bandwidth, ultimately determined the outcome.

Acknowledgments

I would like to thank all the entrants, without whom the contest would not have taken place. I am in awe of how much work Table I represents.

My colleague Itsik Pe'er provided invaluable guidance by first describing the problem to me and then pointing me to the relevant literature.

List of Teams and Members

(Order follows that of Table I)

Iowa State University, US
Chad Nelson
Kevin Townsend
Bhavani Satyanarayana Rao
Jungmin Park
Dr. Phillip Jones
Dr. Joe Zambreno

Institute for Research in Fundamental Sciences (IPM), Iran
High Performance Computing Center (HPC)
Aryan Arbabi
Milad Gholami
Mojtaba Varmazyar

Institute for Research in Fundamental Sciences (IPM), Iran
High Performance Computing Center (HPC)
Armin Ahmadzadeh
Seyed Koosha Mirhosseini
Mohsen Zare Zardeyni

Iowa State University, US
Mihir Awatramani
Mengduo Ma
Michael Patterson
Stanley Ho
Dr. Joe Zambreno
Dr. Phillip Jones

Shahid Beheshti University, Iran
Design Automation Group
Amir Haji-Ali Khamse
Mehran Goli
Mohsen Faryabi
Armin Belghadr
Hamed Fatemi
Bahareh Pourshirazi
Ronak Zahrhoon
Ali Jahanian

Institute for Research in Fundamental Sciences (IPM), Iran
High Performance Computing Center (HPC)
Ahmad Lashgar

Korea Advanced Institute of Science and Technology
Miri Kim
Kiun Choi

Sungkyunkwan University (SKKU), Korea
Parallel Architecture and Programming Laboratory (PAPL)
Euijin Kwon
Jinmin Kim
Beomwon Jung
Namhyung Kim

University of Tehran, Iran
Mehrdad Biglari
Mahdi Jelodari
Nazanin Farahpour
Arman Pouraghily
Zain Navabi

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] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Digital Equipment Corporation, Palo Alto, California, Tech. Rep. SRC–RR–124, May 1994. [Online]. Available: http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.html

[7] 1000 Genomes Project, "Human reference genome." [Online]. Available: ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/technical/reference/human_g1k_v37.fasta.gz

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

[9] 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.

[10] M. L. Metzker, "Sequencing technologies—the next generation," *Nature Reviews. Genetics.*, vol. 11, no. 1, pp. 31–46, 2010.

[11] 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.

[12] 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.

[13] 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.

[14] 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: Ultra-fast GPU-based parallel alignment tool for short reads," *Bioinformatics*, Jan. 2012, to appear.

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

[16] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proceedings of Foundations of Computer Science (FOCS)*, Redondo Beach, CA, Nov. 2000, pp. 390–398.

[17] T. W. L. amd W. K. Sung, S. L. Tam, C. K. Wong, and S. M. Yiu, "Compressed indexing and local alignment of DNA," *Bioinformatics*, vol. 24, no. 6, pp. 791–797, March 15 2008.