Supplementary Information for

Compressive Mapping for Next-Generation Sequencing

Deniz Yorukoglu [1], Yun William Yu [1, 2], Jian Peng [1, 2, 3], and Bonnie Berger [1, 2]

[1] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA.

[2] Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA.

[3] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801, USA

Correspondence should be addressed to Bonnie Berger (bab@mit.edu)

Table of Contents

Table of Figures

Supplementary Figures

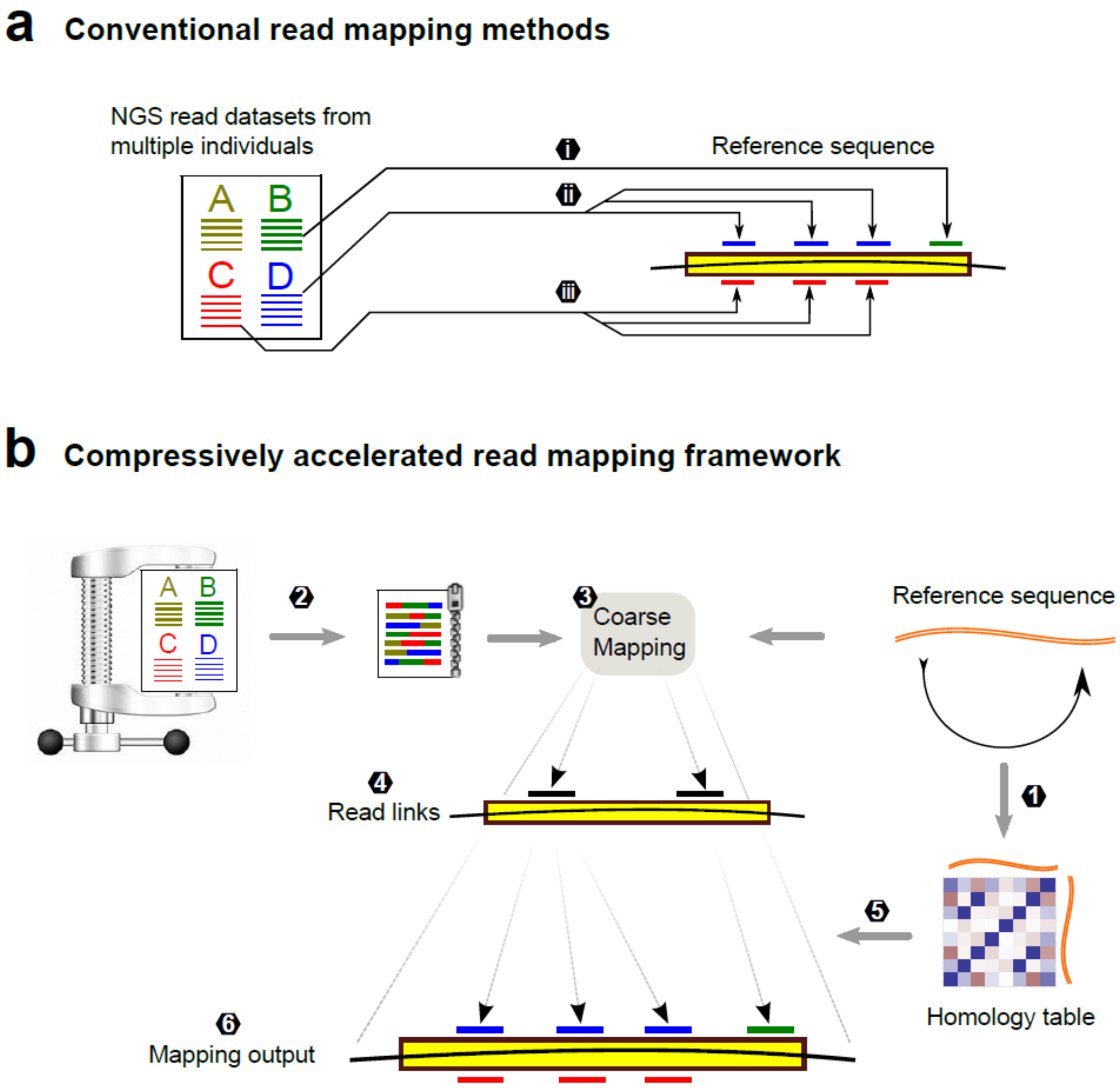
a
Conventional read mapping methods
NGS read datasets from multiple individuals
A
B
C
D
i
ii
iii
Reference sequence
b
Compressively accelerated read mapping framework
A
B
C
D
2
3
Coarse Mapping
Reference sequence
1
4
Read links
5
Homology table
6
Mapping output

Figure S1: Comparison between conventional read mapping methods and compressively-accelerated read mapping framework (CORA). (a) In order to generate read-mapping results, existing read mappers compare each read to the reference or a previously-constructed index for the reference. (i) Unique-read case: Some reads have a unique match to the reference. (ii) Multi-read case: Due to the repetitive structure of DNA sequences, a single read can often be aligned to more than one location with high similarity in the reference. Most existing approaches involve costly seed-extension or suffix-array traversal stages for each of these locations, requiring additional computational time reporting multiple mappings for each read (or comparing them to report a best mapping). Furthermore, in high depth-coverage datasets (especially when multiple individuals are mapped together), there can often be reads that are fully or

partially similar to other reads in the dataset. (iii) Redundancy within reads: As existing aligners cannot utilize inexact redundancy within reads, they process each read individually, potentially duplicating previous computations performed for earlier reads in the dataset. This inefficiency is particularly an issue for multi-reads sequenced from highly-repetitive regions in the genome, since they require sequence comparisons with a large number of loci in the reference. Compressively accelerated read mapping addresses the inefficiencies of mapping high-throughput NGS reads by capitalizing on redundancy within both read datasets and the reference. (b) CORA capitalizes on redundancy in both reads and reference. (1) As a preprocessing step, a high-resolution homology table is created for the reference sequence by mapping the reference to itself. The homology table contains all homologous pairs of loci in the reference above a similarity threshold, allowing fast direct access to similar locations in the reference during mapping. (2) The first step in compressive read-mapping is to compress the reads in order to eliminate full or partial redundancies across reads in the dataset. Compression is achieved through self-mapping of the read dataset. (3) Next, an off-the-shelf aligner can be used to perform a coarse mapping from the compressed read data—clusters of similar substrings—to the reference. (4) Each read link represents a cluster of substrings from one or more reads in the dataset and stores their differences from a locus in the reference. (5) Read links are further expanded to obtain final mapping results through traversal of the pre-computed homology table, and final mapping results are reported. (6) Far fewer comparisons are required for compressive read mapping due to efficient utilization of redundancy within read sequences as well as the reference.

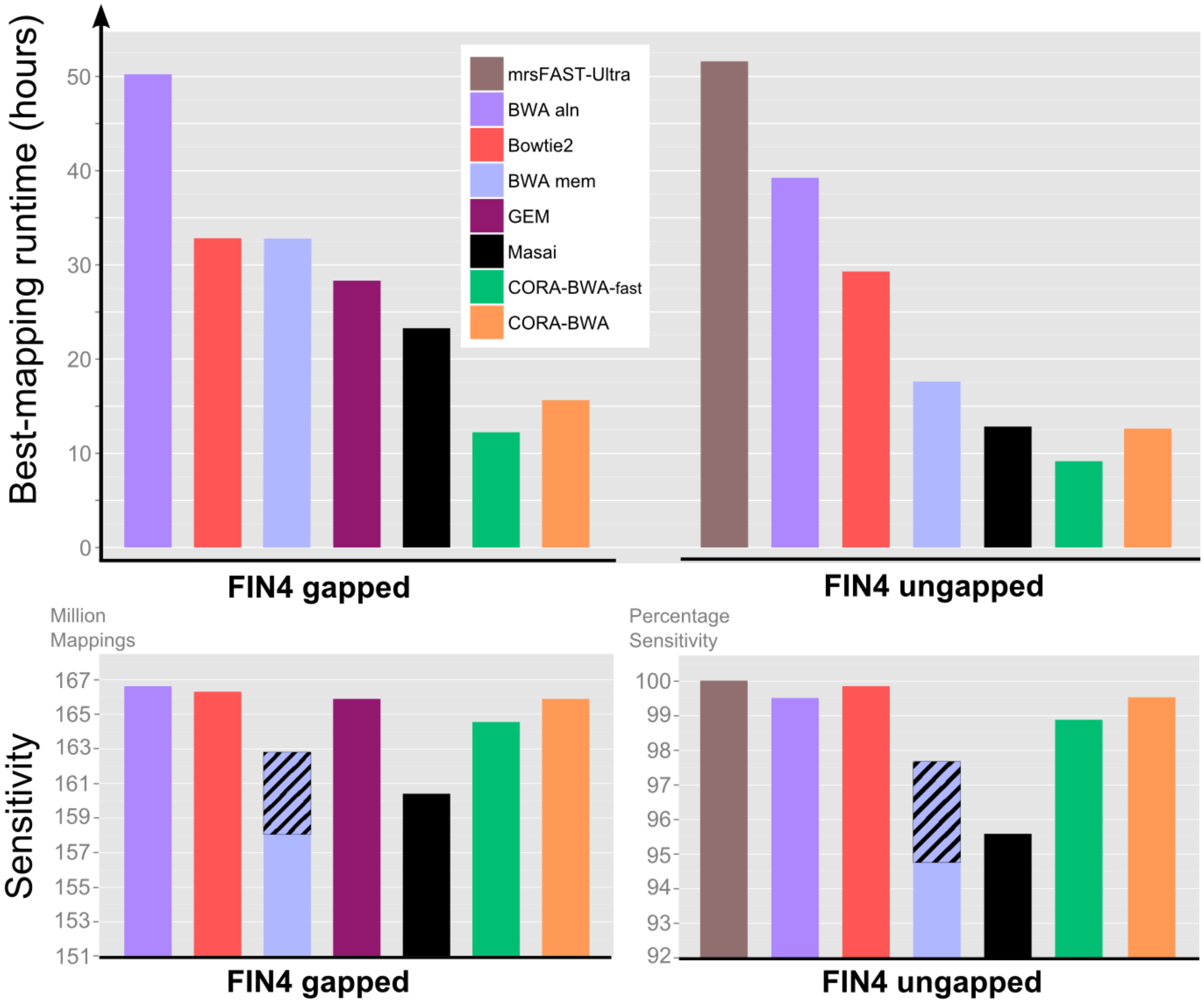

Figure S2: Runtime and sensitivity comparison results for whole genome ungapped (substitution-only) and gapped (with indels) best-mapping of 1000 Genomes Phase 1 Illumina 2x108bp paired-end read datasets of 4 Finnish individuals with ~16x read depth-coverage; similarity threshold is defined as Hamming distance of 4 for each end for the ungapped mapping and Levenshtein (edit) distance of 4 for each end for the gapped mapping benchmarks. Paired-end insert size interval is defined to be between 150 and 650 base pairs. We compared best-mapping runtimes of Bowtie2, BWA aln, BWA mem, mrsFAST-ultra (only for ungapped), GEM (only for gapped) and Masai against compressively accelerated version of BWA with two different modes: CORA-BWA and CORA-BWA-fast, which sacrifices some sensitivity allowing for faster best-mapping. The bars in the upper panel represent mappers' runtime performance, whereas the bars in the lower two panels indicate sensitivity performance: percentage sensitivity for ungapped mapping and number of mappings for gapped mapping. As BWA mem does not accept insert size intervals, we show two different mapping sensitivity measurements for it: sensitivity within the defined insert size interval and the increased sensitivity when mappings outside of the interval are included as well (the difference is indicated by the hatch pattern). Some of the results are estimated

from a down-sampled set of reads; detailed benchmark criteria as well as mapping parameters can be found in **Further details on experimental setup**. The plots indicate that compared to the fastest best-mappers we tested against, CORA-BWA mappers are at least ~2x faster with superior or comparable sensitivity. The only mapper that approached CORA-BWA in terms of best-mapping runtime was Masai for the ungapped mapping experiment, albeit with drastically lower sensitivity; even then CORA-BWA-fast was >1.4x faster than Masai using less memory. Moreover, compared to the original BWA aln, CORA-BWA generated best-mapping results with near-identical sensitivity, but >3.2x faster for gapped and >3.1x faster for ungapped mapping. Furthermore, comparisons with CORA's all-mapping runtime results in **Figure 1a** reveal that CORA can perform near-perfect sensitivity all-mapping faster than BWA, Bowtie2 and mrsFAST-Ultra can report best-mapping results. The peak memory usage of Bowtie2 was 3.2GB, BWA used 4.7GB and 6.2GB respectively for aln and mem, mrsFAST-Ultra used 4.7GB, whereas GEM and Masai's memory usages were 4.1GB and 23.2 GB respectively. CORA-BWA, at the maximum of collapsing, coarse-mapping, and homology table traversal stages, used 19.7GB of memory for the runs that only utilized the exact homology table, whereas it used 64.1GB for the runs that also loaded the inexact homology table into memory (e.g. ungapped mapping with CORA-BWA default mode).

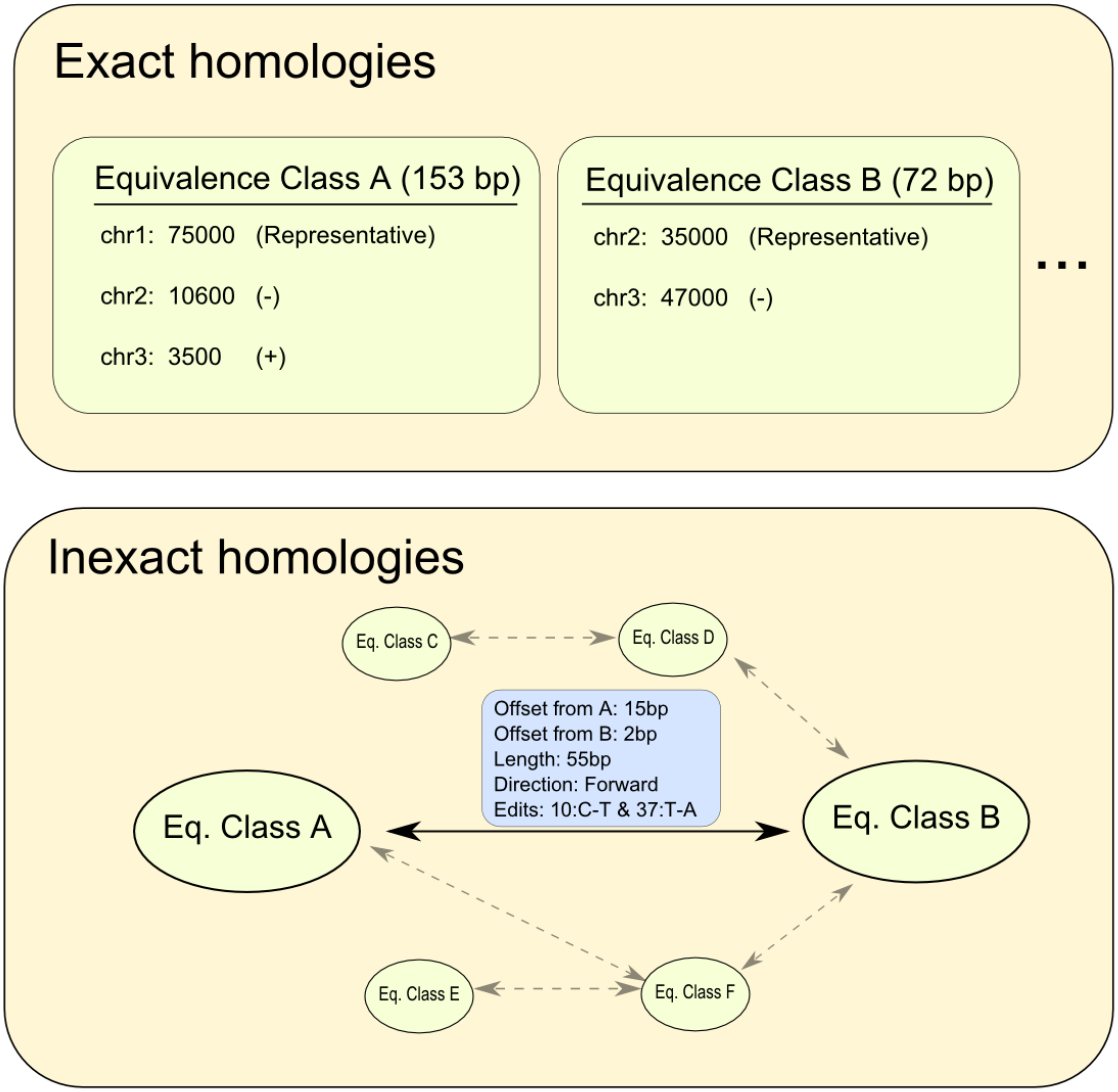

Figure S3: Compact representation of homology table with exact and inexact homologies. Identical homology blocks of a certain length are collected under the same equivalence class. Each equivalence class has a representative locus, as well as other loci that are oriented with respect to the representative locus. Inexact homology table is a graph with equivalence classes as nodes and partial similarities between equivalence classes as edges. Therefore, inexact homologies can only be defined between equivalence class representative loci. Each inexact homology edge identifies the offset from the beginning of the first equivalence class, the offset from the beginning of the second equivalence class, the length of the inexact homology block, forward or reverse-complement direction of homology, and the positions of differences or base substitutions (text above bold bi-directional arrow). An inexact homology block size indicates the final length of the inexact homology after two or more consecutive and concordant inexact k-mer homologies are merged together. Two consecutive inexact k-mers are defined as concordant if their target positions in the reference are also consecutive and their edit positions in the k-mer are concordant, i.e., containing the same (k-1)-mers.

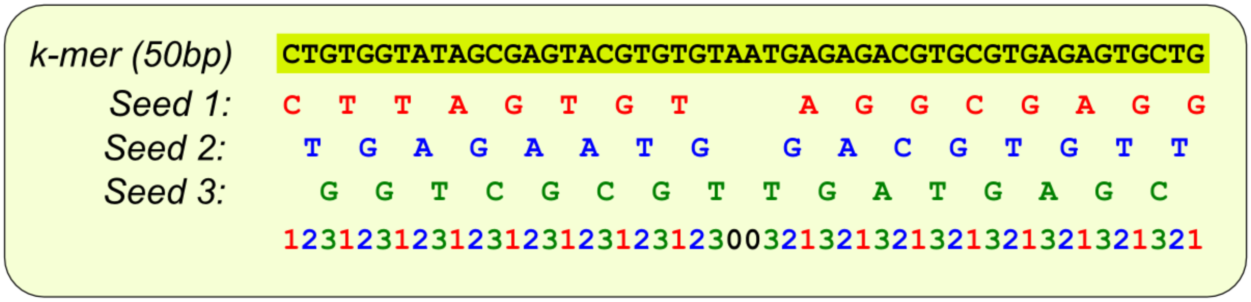
k-mer (50bp)
CTGTGGTATAGCGAGTACGTGTGTAATGAGAGACGTGCGTGAGAGTGCTG
Seed 1:
C T T A G T G T A G G C G A G G
Seed 2:
T G A G A A T G G A C G T G T T
Seed 3:
G G T C G C G T T G A T G A G C
12312312312312312312312300321321321321321321321321

Figure S4: Seed position selection scheme for inexact homology table construction. This seed selection scheme allows detection of all inexact homologies of a given k-mer length within a Hamming distance of 2 (by pigeonhole principle). For detection of higher numbers of mismatches, the number of seeds sampled can be increased. Rather than selecting consecutive positions for each seed, spacing out seed positions throughout the k-mer allows for more evenly-sized bins within the hash table. The positions sampled for each seed is designed to be closed under reverse complementation, so that hash tables created for each seed are disjoint. This allows for both space savings and easy parallelization of inexact homology table construction.

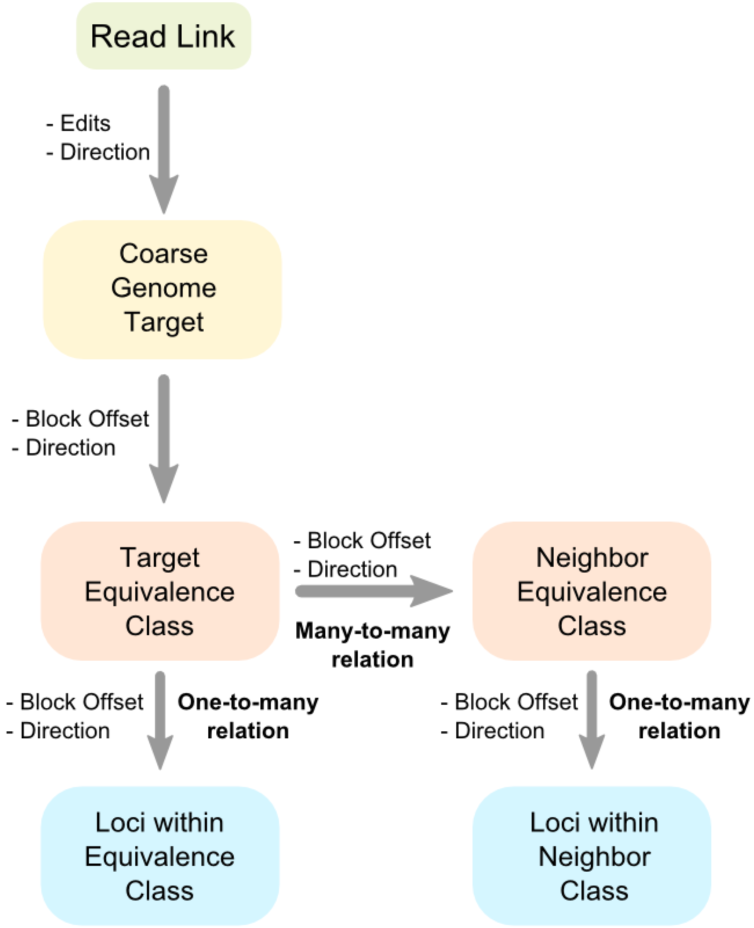

Figure S5: Homology table traversal scheme of CORA framework. Each read link, representing one or more reads, points to the genomic location determined by the coarse mapping stage in either forward or reverse complement direction, indicating homology within a small number of edits. This locus is either unique in the reference genome or associated with an equivalence class in the exact homology table. In the latter case, the anchor is linked to the equivalence class in the forward or reverse complement direction, with a block offset value indicating the starting location of a substring of the equivalence class representative that is identical to the coarse genome target. The representative of the equivalence class points to all members of the equivalence class for each valid offset (until the end of the block) in forward or reverse complement direction. Furthermore, the equivalence class representative can be linked to other equivalence classes through the inexact homology table. Each of these pointers contains the direction of homology (forward or reverse complementary), block offset in the compressed inexact homology block representation, and the edit script to convert one class representative to another. Provided that the juxtaposition of the edits from a read link to the anchor and from the anchor's class representative to a neighboring equivalence class still contains less than or equal to the user specified number of errors, read mappings associated with all members of these neighbor equivalence classes will also be reported.

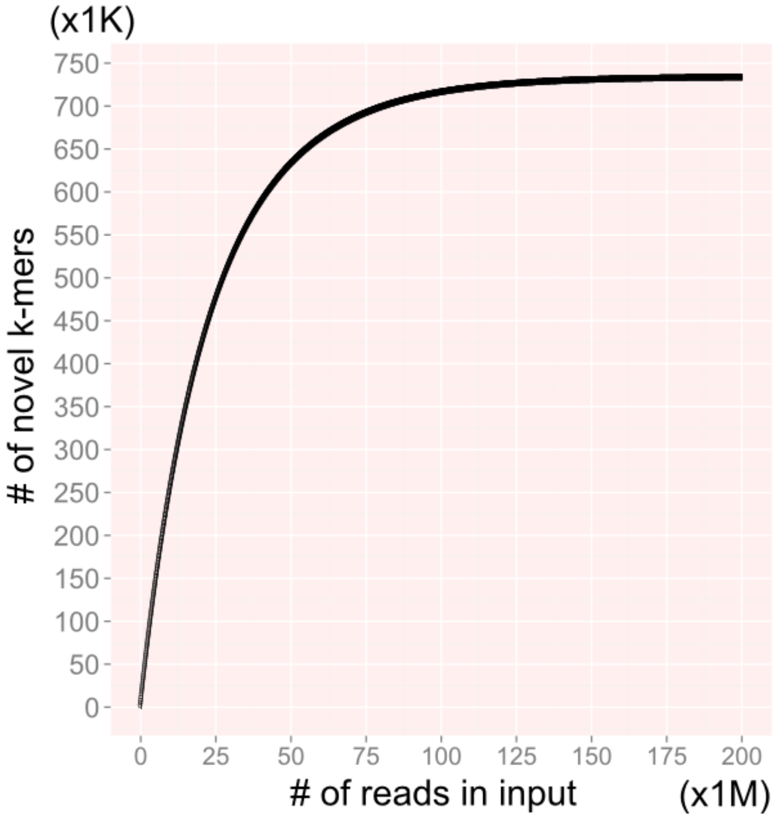

Figure S6: Estimation of redundancy within read datasets in absence of sequencing error. The plot above demonstrates how the number of k-mers processed by the coarse mapping stage scales with respect to total number of reads in the input dataset, for a high depth-coverage simulation of 100bp paired-end reads on hg19 chromosome 20, with 0.1% mutation rate and 0% sequencing error. After roughly 100 million reads in the input dataset, additional reads in the dataset do not affect the size of the coarse mapping stage, thus resulting in sublinear mapping scalability with the CORA framework.

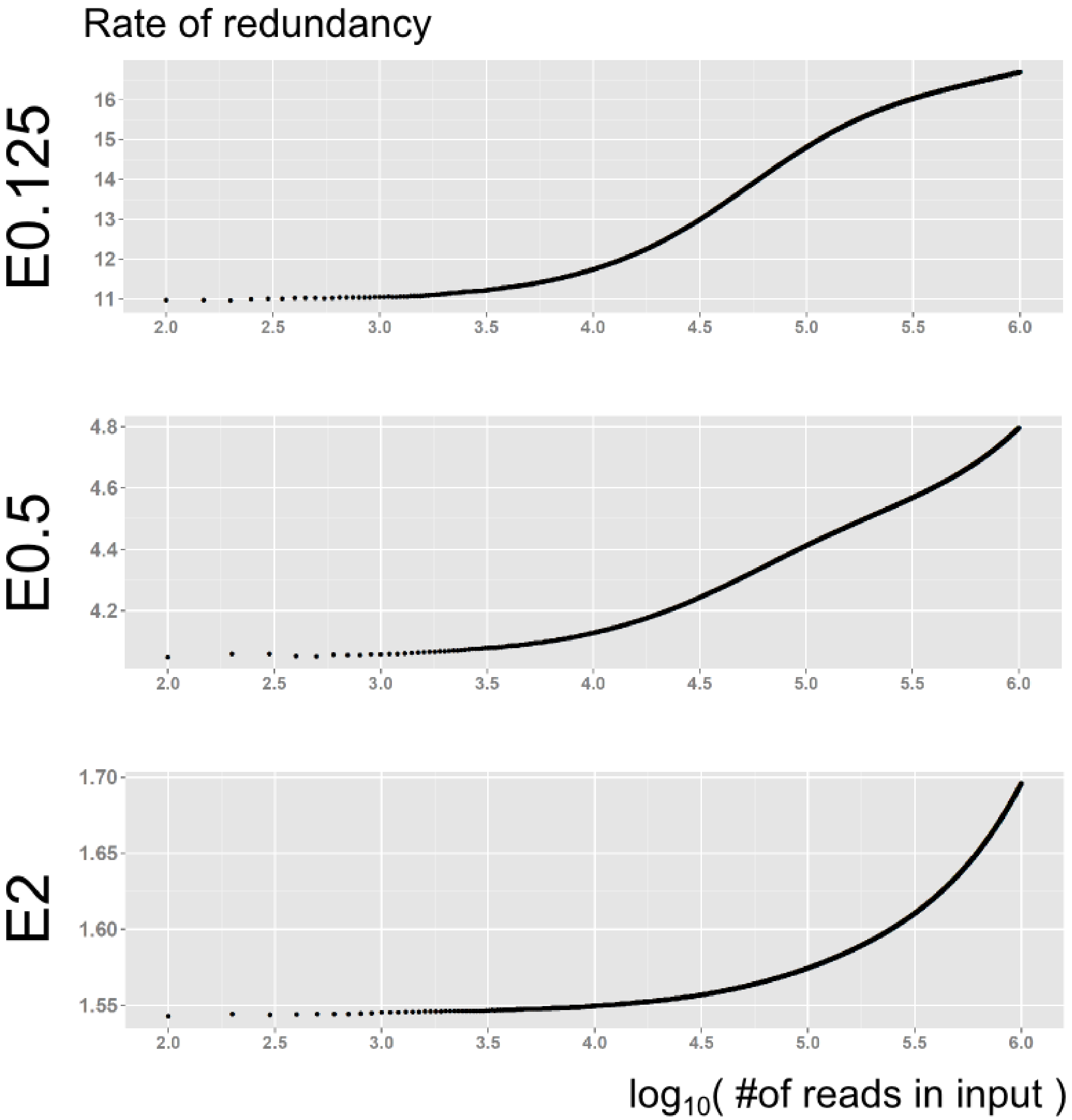

Figure S7: Sublinearity analysis of CORA framework on simulated paired-end read data from hg19 chromosome 20 with 0.1% mutation rate and 2%, 0.5%, and 0.125% sequencing errors (E2, E0.5, and E0.125), respectively. Rate of redundancy is calculated as the total number of k-mers in the read dataset divided by the number of k-mers processed by CORA during the coarse mapping stage. Results indicate that rate of redundancy monotonically increases even in the presence of sequencing errors; therefore, for all three datasets the CORA framework spends less coarse mapping time per additional read, indicating sublinear scalability regardless of sequencing errors in practice.

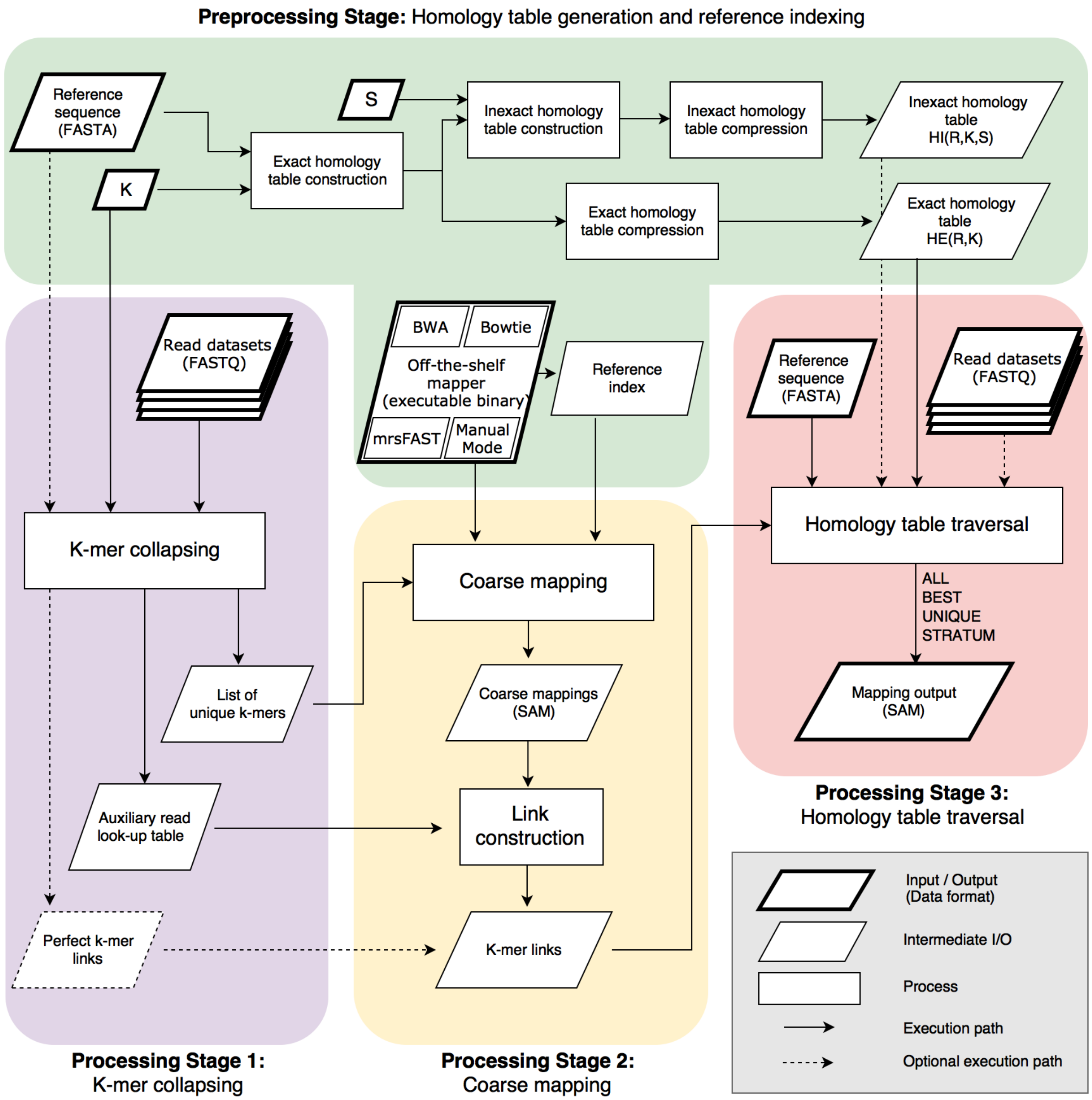
Preprocessing Stage: Homology table generation and reference indexing
Reference sequence (FASTA)
S
K
Exact homology table construction
Inexact homology table construction
Inexact homology table compression
Inexact homology table HI(R,K,S)
Exact homology table compression
Exact homology table HE(R,K)
BWA
Bowtie
Off-the-shelf mapper (executable binary)
mrsFAST
Manual Mode
Reference index
Read datasets (FASTQ)
K-mer collapsing
List of unique k-mers
Auxiliary read look-up table
Perfect k-mer links
Coarse mapping
Coarse mappings (SAM)
Link construction
K-mer links
Reference sequence (FASTA)
Read datasets (FASTQ)
Homology table traversal
ALL
BEST
UNIQUE
STRATUM
Mapping output (SAM)
Processing Stage 1: K-mer collapsing
Processing Stage 2: Coarse mapping
Processing Stage 3: Homology table traversal
Input / Output (Data format)
Intermediate I/O
Process
Execution path
Optional execution path

Figure S8: An overview of CORA framework and input/output relations between different stages of the CORA software. Green, purple, yellow and red panels describe pre-processing, k-mer collapsing, coarse mapping and homology table traversal stages of the CORA pipeline respectively. The grey panel on the lower right provides a legend for different types of boxes and arrows used in the diagram.

Green panel: The inputs for the preprocessing stage are a reference sequence in multi-FASTA format, k-mer length value K, distance threshold value S, and an executable binary for an off-the-shelf read mapper to be used in coarse-mapping. The off-the-shelf mapper's indexing algorithm is called in order to generate a reference index. Exact and inexact homology tables are constructed and compressed to be used in the homology table traversal stage.

Purple panel: K-mer collapsing stage takes in a set of FASTQ files together with the k-mer length value as input and generate a list of unique k-mers (with compact IDs that encode reads that contain the extracted k-mers). An auxiliary lookup table is generated for some of the k-mer IDs that are known to cause problems during coarse-mapping (e.g. very long IDs). Optionally, k-mer collapsing stage uses the reference genome in order to identify k-mers that are identical to a k-mer in the reference, in which case these k-mers will be separately reported as perfect k-mer links to the reference genome skipping the coarse mapping stage.

Yellow panel: The coarse mapping stage requires an off-the-shelf mapper executable (which could be built-in tools such as BWA-aln/BWA-mem, Bowtie/Bowtie2, mrsFAST/mrsFAST-Ultra but could also be manually described through manual mode) and its reference index in order to place each of the unique k-mers to a locus in the reference. Afterwards the coarse-mappings are converted to k-mer links within the link construction step of CORA. In the case that k-mer collapsing was performed with the reference sequence, the perfect k-mer links from the previous stage are merged with the k-mer links generated from the link construction stage.

Red panel: Homology table traversal stage takes in the k-mer links from the previous stage, reference sequence and the exact/inexact homology tables, in order to generate the final set of mapping (in SAM format) by traversing the homology table(s) following the k-mer links. While some mapping modes can be performed only using the exact homology table (e.g. fast best-mapping mode), other mapping modes would require the inexact homology table as well. Optionally the original read dataset is used in this stage, in order to print the SAM file with the original read names (as opposed to just their order information in the input file) and/or the quality scores.

A. Methods

A1. Overview of methods

CORA's acceleration relies on three key components (**Figure S1**): (1) identifying redundant k-mers across reads in the input read dataset first, as opposed to mapping each read directly to the reference; (2) mapping the reference to itself through a single preprocessing step, in order to create a comprehensive lookup table of similarities for fast retrieval; and (3) using a local neighborhood search in the Hamming and Levenshtein distance space of the self-mapped reference to speed up sequence comparison. The CORA framework is comprised of combining these advances with an off-the-shelf mapper for identifying inexact k-mer matches in the reference.

For preprocessing, we map the reference onto itself by building a high-resolution homology table of the reference genome sequence. The table stores the similarity information of all homologous or similar regions of a specified length in the reference genome, within a predefined Hamming distance. With this table, we can directly access all homologous or similar loci of any given locus, and thus are able to report all possible mapping locations at low computational cost, which is particularly useful for multi-reads (reads that map to multiple locations in the reference genome).

For the actual read mapping stages, we first map the reads to themselves in order to identify and compress shared k-mers, with the goal of constructing a compressed representation of the read dataset consisting of only unique k-mers. We then use an off-the-shelf short-read mapper (e.g., BWA [3] or Bowtie2 [4]) to coarsely map this compressed k-mer dataset to k-mer matches in the reference genome; we represent these matches in the form of links, or pointers, to the reference genome. Lastly, we generate the final set of mappings through simultaneous traversal of the homology table and multiple k-mer links of the read. We are thus able to identify all high-quality alignment positions in the reference with near-perfect sensitivity.

In the following sections, we present the technical details of these advances.

A2. Homology table construction

Before performing compressive read mapping, we first preprocess the reference to generate the homology table, a high-resolution all-to-all similarity map of the reference genome onto itself, as well as construct the corresponding reference index of the mapper that CORA uses for its coarse mapping stage. For CORA's k-mer based compressive read mapping, a homology table that represents all homologies in the reference, exact as well as inexact, is most useful, as it enables fast retrieval of additional mappings of a k-mer once one mapping is identified in the coarse mapping stage.

For a chosen homology block length (e.g., 33-64 base pairs) and similarity measures (e.g., Hamming distances 0-3), the homology table of a reference genome contains links between all homologous loci satisfying the length and similarity requirements. In other words, instead of representing the sequences of k-mers in a given reference genome, a homology table represents pointers for each position in the genome indicating all of its neighboring k-mers. As such, the homology links form an inter-web of similar loci within the reference genome that enables CORA's k-mer-based compressively accelerated read mapping framework. Though relatively costly to generate (~18 hours for constructing a k=54bp homology table of the hg19 human reference genome with a Hamming distance of 2 on a 12-CPU Intel Xeon X5690 machine using 24 parallel threads), both homology table and reference index need be computed only once for a given reference genome, after which they can be repeatedly used for further compressive mapping runs on the same reference genome.

More formally, given a reference DNA sequence *R*, a substring of length *k*, and a mismatch error threshold *s*, a homology table *H(R, k, s)* is a compact data structure that stores the links among all similar *k*-base pair substrings (k-mers) of *R* (up to s mismatches), including reverse complements. To efficiently construct the homology table *H*, we first build an exact homology table *HE(R, k)* that stores the links among all identical *k*-mer occurrences in the reference under reverse complementation. Based on this exact homology table, we construct an inexact homology table *HI(R, k, s)* that stores the homology links among k-mers with at least one and at most *s* mismatches (**Figure S3**). Together, the exact homology table *HE(R, k)* and the inexact homology table *HI(R, k, s)* form *H(R, k, s)*. In other words, instead of storing neighborhood information for all positions in the genome, CORA compactly represents the neighborhood information between sets of identical k-mers.

The exact and inexact homology tables are generated, organized and stored separately in order to optimize runtime performance and memory/disk usage. Though we describe below a substitution-only version of the inexact homology table, it is straightforward to extend homology tables to other types of edits such as indels.

Exact homology table. We use an unordered hash table to construct the exact homology table. For each k-mer, we use its sequence as the key and its first occurring position in the reference genome as the value in the hash table. If the k-mer appears in other position(s) in the reference, an *equivalence class* is created to store all such positions. Each equivalence class is stored as a list of genomic positions and their directions with respect to the representative, which allows a quadratic number of exact homology links between pairs of genomic loci to be stored within linear space. This representation is further compacted by collapsing adjacent concordant equivalence classes. If there are two or more adjacent concordant equivalence classes (all loci within an equivalence class are shifted by one position with respect to another equivalence class), they are merged in a way that only the values of the first equivalence class with the smallest representative position are kept, with an additional block-length value, which indicates how many adjacent concordant equivalence classes are collapsed into a single one.

A formal definition of the exact homology table block merging can be given as follows: Denote $e_i = (c_i, p_i, d_i)$ as members of the equivalence class E, for i ranging from 0 to |E| - 1, c_i representing the chromosome ID of the ith element, p_i being a positive integer representing the chromosome position of the ith element, and d_i being either 1 or -1, respectively representing the forward or reverse complement direction of the homology of the ith element, with respect to the representative element e_0. The representative element is required to always have forward direction ($d_0 = 1$). Two equivalence classes E and F are merged to create a new equivalence class E_2 of block length 2 if and only if |E| = |F| and for all $e_i = (c_i, p_i, d_i) \in E$, there exists an $f_j = (c_j, p_j, d_j) \in F$, such that $c_i = c_j$, $d_i = d_j$, and $p_j = p_i + (d_i \text{ x } d_j)$. Blocks of length longer than 2 are inductively defined: If E and F can be merged, F and G can be merged, then E, F, and

G can be merged altogether into E_3, an equivalence class with a block length of 3, and so on.

Inexact homology table. Generating the inexact homology table, *HI(R, k, s)*, which represents all-to-all homologies of k-length substrings in R with between 1 and s mismatch errors involves a similar but more sophisticated approach. In order to generate the inexact homology table *HI(R, k, s)*, we use the exact homology table *HE(R, k)*, specifically the pre-computed equivalence classes, to reduce computation and storage requirements. Instead of constructing the inexact homologies between substrings from the reference, we need only do so among the equivalence class representatives in the exact homology table *HE(R, k)*. The remainder can be inferred directly using the equivalence class members. The inexact matches between equivalence class representatives are identified by a seed-and-extend procedure, which first checks if there is an exact match between seeds on two given k-mers, and then extends the remaining bases of the k-mer sequences to identify any mismatches. We use a special spaced-seeding scheme for performance and parallelization purposes (see **Seed position selection scheme**).

To construct *HI(R, k, s)* for a given R, k and s, we construct s+1 auxiliary hash tables, each corresponding to one of s+1 seeds extracted from equivalence classes from the exact homology table. The seeds are sampled according to the position selection scheme described above. In the hash table, we use the binary representation of a seed as the key, and the list of positions in the reference that contain the seed sequence as the value. Every time we find a hit, the current k-mer position in the reference is added to the list of the corresponding key.

Finally, we further compact the representation of the inexact homology table by merging any adjacent pairs of inexact matches with consecutive mismatch positions in forward and reverse complement alignment directions, similar to what we did for the exact homology table, but with the added requirement of concordant mismatch positions that are shifted by one.

Example homology table construction. In order to elucidate the structure of the homology tables, we provide below a toy genome example with a single chromosome and construct a homology table with a k-mer length of 4bp and a Hamming distance threshold of 1.

ACCGCAGCGGT

We initially build an exact homology table (HE) from the genome, identifying each 4-mer or its reverse complement, if lexicographically earlier. These correspond to, 1+:ACCG, 2+:CCGC, 3+:CGCA, 4–:CTGC, 5+:CAGC, 6+:AGCG, 7–:CCGC, 8–:ACCG, with each number representing the position of the 4-mer in the genome and the (+)/(–) sign indicating whether the default or the reverse complement 4-mer is considered. Note that positions 4, 7 and 8 have a (–) sign after them since their reverse complement 4-mers are lexicographically earlier than the forward ones.

Once the 4-mers are identified, identical 4-mers are merged into groups in order to construct the equivalence classes. This grouping results in 1+ and 8- in one equivalence class whereas 2+ and 7- are in another; all the remaining positions without identical 4-mers each have their own equivalence classes, i.e., which contain only themselves. Each equivalence class has a single representative, which is always in the forward direction (otherwise, all elements are in reverse complement to ensure this property), and other elements in the equivalence class assume directions relative to the representative. In our toy genome example, all six equivalence classes are [1: 8–], [2: 7–], [3], [4], [5], [6], the first number indicating the representative position, and the remaining numbers, non-representative equivalence class members with a direction in relation to their corresponding representatives. In order to conserve space, single item classes are not stored in the exact homology table HE, leaving [1: 8–] and [2: 7–].

To compact the exact homology table, all adjacent concordant equivalence classes are merged into block homologies that represent intervals longer than 4-mers. In our toy genome example [1: 8–] and [2: 7–] are adjacent (since 1 and 2 are adjacent) and concordant (since other elements in the two equivalence classes, 8- and 7- are also adjacent in reverse direction). We represent the merged exact homology block in the genome with an additional block length parameter appended to the equivalence class, in this case [1: 8–](2). The set of all compact equivalence classes of size larger than one (with a block length 1 or longer) constitutes the exact homology table, HE.

Note that the block merge operation can also be done with forward direction elements as well as with multiple items, as long as adjacency and concordance properties are satisfied. For example, an equivalence class [15: 35+, 73–] can be merged with [16: 36+, 72–] to form block equivalence class [15: 35+, 73–](2), whereas not with [16: 36+, 74–]. Similarly, multiple equivalence classes can be merged together. For example, [10: 20+], [11: 21+], [12: 22+] and [13: 23+] can be merged into a single block equivalence class [10: 20+](4).

In order to construct the inexact homology table, HI, we input the set of 4-mer equivalence classes formed during HE construction before we perform any block merge operations or elimination of single-item equivalence classes, which is {[1: 8–], [2: 7–], [3], [4], [5], [6]}. Among these, we identify all pairs of inexact homologies between class representatives within a Hamming distance of 1. These six homologies are: [1 → 6+: 2] [2 → 5+: 2], [3 → 6–: 4], [4 → 5–: 3], [5 → 7–: 2], and [6 → 8–: 2], where the first number represents the source position; the second number with the appended sign, the target and direction of homology; and the number after the semicolon, the 1-base error positions at which the two 4-mers differ, oriented by the direction of the first 4-mer. The reverse homologies ([6 → 1+: 2], [8 → 6–: 3], etc.) are not represented in HE, as they are redundant.

Similar to merging adjacent equivalence classes, we can also merge adjacent and concordant inexact homologies. In the case of inexact homologies, the concordance property requires the error positions to be concordant as well. For example, the homologies [3 → 6–: 4] and [4 → 5–: 3] are adjacent and concordant, since 3 and 4 as well as 6– and 5– are adjacent, and the error positions 4 and 3 are concordant with the adjacent 4-mers. In order to compact the inexact homology table we can represent this inexact block homology as [3→6–: 4](2), where 2 refers to the number of homologies merged. This example represents an inexact homology between the 5-mer that starts at position 3 (CGCAG) and the 5-mer, at position 5 (CAGCG), in reverse complement direction with a mismatch error at nucleotide 4. Similar to compacting HE, more than two HI homologies can be compacted as well as inexact homologies with multiple errors, as long as each error position satisfies the concordance property.

Thus, the set of homologies in HI for our toy genome example will be: {[1 → 6+: 2], [2 → 5+: 2], [3 → 6–: 4](2), [5 → 7–: 2], [6 → 8–: 2]}. HE and HI together constitute the homology table of the toy genome.

Seed position selection scheme. For inexact homology table construction, we use a special spaced-seeding scheme closed under reverse complementation for performance and parallelization purposes, as displayed in **Figure S4**.

When determining seeds for inexact homology table construction, we choose seed positions to be closed under reversal for two particular reasons. First, it allows the reverse complement of a seed to correspond to the exact same positions in the k-mer; thus we need only store the lexicographically smaller direction of any seed in the hash table and search for each seed once in the hash table. Second, this choice allows both simple parallelization and multi-pass implementation of the hash table, since data splitting can be performed easily without the need to search for a seed and its reverse complement in different hash tables. This choice allows for both space savings and ease of parallelization for inexact homology table construction.

Furthermore, as DNA sequences are highly repetitive, some bins in a hash table can get very large for frequently-occurring seeds. Rather than selecting consecutive positions for each seed, spacing out seed positions throughout the k-mer allows for more evenly-sized bins within the hash table. For this reason, we choose a spaced seeding scheme in which each position within a seed is sampled as far away as possible while maintaining that the positions are closed under reversal (**Figure S4**).

We experimentally evaluated the fitness of our seed position selection scheme by comparing it to a variety of schemes that are also closed under reversal. The schemes were based on selecting: (a) equally-spaced nucleotides from the k-mer which is built in CORA; (b) equally-spaced dinucleotides of a seed rather than each nucleotide (e.g. 112233112233 instead of 123123123123); (c) equally-spaced 4-mers (e.g. 111122223333); (d) equally-spaced 8-mers; and (e) random selection of seed positions that are not equally-spaced, satisfying only that seeds have equal length. We created the inexact homology table for chr20 of hg19 using each scheme three times. We noted that the slowest scheme was (d) with an average runtime of 680 seconds. As equally-spaced seed sub-blocks got shorter, the runtime also improved resulting in 639 seconds for (c), 568 seconds for (b), and 466 seconds for (a), which is the scheme we eventually used for CORA's homology table construction algorithm, which equally spaces all nucleotides of each seed. The randomly generated schemes were faster than (b) but slower than (a), averaging 557.6 seconds, with the best performing random scheme taking 549 seconds.

Implementation details and memory use. For smaller genomes, CORA's homology table construction algorithm can be run on the entire genome, processing all k-mers of the genome in a single pass. However, for larger genomes (e.g. mouse and human), the memory requirements for construction is expensive. Once the homology links are compacted, a homology table for the human genome can be reduced to a manageable size of ~15 GB; however the memory usage for the intermediate data structures would be on the order of several hundred GBs. In order to enable homology table construction for smaller memory machines, CORA employs a multi-pass homology table construction algorithm.

The basic idea behind the multi-pass construction of the homology table is to separate the total k-mer space into disjoint sets that are closed under reverse complementation, so that each pass of the multi-pass construction scheme can detect the exact homologies within the current set. For the sake of simplicity, we separated the set of k-mers based on the nucleotides on the two ends of the k-mer sequence. Note that if one end is used as such a separator for subsets, the other end also needs to be used due to reverse complementation. For instance, k-mers that start with 'A' should be in the same subset as k-mers that end with 'T', requiring at least two nucleotides to be used for the distribution of the k-mer. For instance, k-mers that start with 'A' and end with 'G' (in the form [A...G]) should be processed in the same pass with the reverse complement form [C…T]. For these k-mers, we construct the A-G hash table. In this way, we can construct the whole table with 10 passes. Specifically, these 10 passes include: 1) A-A and T-T; 2) A-C and G-T; 3) A-G and C-T; 4) A-T; 5) C-A and T-G; 6) C-C and G-G; 7) C-G; 8) G-A and T-C; 9) G-C; and 10) T-A. These 10 passes were designed to optimize empirical performance and memory usage. By incorporating more nucleotides from both ends of the k-mers, it is possible to separate the full k-mer space into smaller subsets and thus store the hash-table using even smaller memory size during each pass. In our implementation, we used the aforementioned 10-pass approach.

For the mapping of FIN datasets presented in the paper, we constructed the inexact homology table for the hg19 reference genome with a 10-pass approach with interleaved 24-split parallelization on 12 hyper-threaded processors. For parallelization of split signals, two bases from the ends of each seed are used, which gives 136 reverse complement agnostic sets– as opposed to 10 for one base from each end. These are collected into 24 balanced-set groups. The balancing is accomplished according to the general human GC ratio assumption (2 * |A or T| = 3 * |C or G|).

For generation of exact and inexact homology tables, CORA allows the user to specify the number of passes in the multi-pass construction scheme as well as the number of processes to be used for parallelizing each pass.

Below is a runtime/memory tradeoff analysis for construction of the homology table for the hg19 human reference genome for 54-mers with a Hamming distance of 2. This is the homology table we used for the FIN1-4 read mapping experiments presented in the main results. We carried out 5 different levels multi-pass runs comprising 6, 8, 10, 24 or 48 passes. The 6, 8 and 10-pass runs were performed using a 2-nucleotide signal (1 from each end of the k-mer), whereas the 24 and 48-pass runs were performed using a 4-nucleotide signal (2 from each end of the k-mer). For each run we present the peak memory used by the operating system (maximum memory resident set size), total process runtime (user + system), as well as the elapsed time using 12 hyper-threaded CPUs (**Table S1**).

Number of passes	Peak memory use (GB)	Elapsed time 12-CPU (hours)	Total CPU time (hours)
6	94.5	18.2	181.8
8	87.2	18.5	180.1
10	81.3	18.8	170.1
24	50.1	21.3	165.5
48	44.4	25.7	171.5

136	40.5	30.6	192.7

Table S1: Homology table construction runtime and memory costs for different multi-pass construction schemes.

We observe that peak memory usage decreases with more passes for constructing the homology table, since the computation is done in smaller chunks. The total process time also decreases with the number of passes due to smaller hash-tables which are less costly to maintain. On the other hand, the elapsed time increases as the time spent for I/O is higher for multi-pass runs due to the requirement that both the reference genome and the intermediate files need to be fully scanned for each pass, and these scans are not parallelized.

A3. Query-side compression and coarse mapping

In this phase of the CORA framework, the main goal is to reduce the redundant information within and across multiple large NGS read datasets, achieving a compact representation of the reads in the form of k-mer links to the reference; these represent a position in the reference genome similar to a substring within the read and the differences of the read from it. This representation of reads allows faster sequence matching as we need to check only corresponding edits during the homology table traversal stage rather than performing expensive sequence comparisons. The two main stages of the links table construction are (1) collapsing and (2) coarse mapping.

K-mer collapsing. The aim of the collapsing stage is to eliminate k-mer redundancies in the read dataset; this stage inputs a set of FASTA/FASTQ read datasets and converts them into a set of non-redundant k-mers.

In a simple scenario in which the reads are single-end and read length is very short (<50bp), we can assume the k-mer length to be equal to the read length. In this simple scenario, the k-mer collapsing scheme would simply involve identification of all identical reads (under reverse complementation) and collapse them into unique reads with compact IDs which encode the set of reads they correspond to in the input FASTA/FASTQ file. These IDs comprise a range of text-readable ASCII characters from '!' to '~' (with an alphabet size of 94), allowing off-the-shelf mappers to readily parse the read name format without any modifications and directly operate on CORA's compact read representation.

A direct whole-read collapsing approach as described above can be beneficial in terms of mapping speed for very short single-end read datasets (<60 bp) [13]; however, as the chances of a read to not contain any sequencing errors reduces drastically the longer the reads get (~22% for 75bp and ~5% for 150bp with a 2% sequencing error rate), the potential speed gain from whole-read compression also reduces drastically since even a single sequencing error in the read would virtually eliminate the chances of the read being an exact duplicate. The speed gains reduce even further when the reads are paired-end, as whole-read compression would require both mates to be merged together as a single k-mer for collapsing, resulting in even lower probability of the read being an exact duplicate.

In order to improve read k-mer collapsing performance of longer NGS reads (>70bp), as well as paired-end reads, instead of compressing the entire read sequence, the CORA framework splits the (paired-end) read into shorter k-mers of the same length (between 33-60 base pairs) and collapses them as independent sequences, producing a set of non-redundant k-mers. This collapsing stage is applied prior to the coarse mapping stage, and therefore the subsequent coarse mapping step is performed in a read agnostic manner (i.e., without the knowledge of which k-mer sequences come from which reads). For the k-mers of a read (and mates of a paired-end read) to be identified after coarse mapping, CORA assigns corresponding IDs to the splits and mates of the same read. Only after the coarse mapping stage are these collapsed k-mers reorganized based on their IDs and merged into whole reads in order to produce the final mapping results described in **Homology Table Traversal**. As a tool that can rapidly identify the positions of homologous k-mers in the reference, the homology table precisely complements the k-mer based compression of reads, together creating a very efficient method for NGS read mapping for large datasets with high redundancy.

The length of each sequence ID in the collapsed k-mer dataset is determined by the total number of reads to be mapped in a single mapping task. The compact encoding scheme CORA uses allows each read in a dataset consisting of ~19.5 million paired-end reads to be uniquely identified with 4 character sequence IDs, ~1.8 billion paired-end reads with 5 characters, and ~172 billion paired-end reads with 6 characters. As the sequence IDs are assigned the same order as they are listed in the input FASTA/FASTQ files, CORA also can optionally retrieve the original read name, quality scores, and sample information.

In addition, through the use of a database of k-mers in the reference, CORA is able to identify exact k-mer matches to the reference during collapsing and print them to a different output together with the chromosome and position information of the reference k-mer match, further reducing the number of non-redundant k-mers to be processed in the coarse mapping stage. This optional feature is enabled in the experimental results presented in the paper.

Apart from compressive-mapping speed gained from k-mer based read compression, the flexibility of collapsing independently chosen k-mers within the read rather than the whole read enables a number of advantageous features of CORA's framework. As the k-mer length is independent from the read-length, reads with different lengths can be processed together in the same compressive mapping run. Independently processing k-mers also allows dynamic trimming of low quality reads on-the-fly during mapping, which further facilitates compression by reducing parts of the reads with lower redundancy. One caveat is that when overlapping k-mers are selected, the maximum mapping distance can be reduced if there are variants or sequencing errors in the overlapped region in the read. One solution to this potential problem would be to always pick non-overlapping k-mers in the reads, but leave a small number of bases in the read that are not covered by k-mers; these bases then can be checked during the mapping inference stage in order to verify whether the mapping similarity requirements are satisfied when considering the uncovered bases.

Selecting fewer but longer k-mers results in lower compression in the collapsing stage, increasing the time spent on coarse mapping; but longer k-mers decrease the average number of neighbors each reference locus has in the homology table, enabling faster homology table traversal. On the other hand, selecting many but shorter k-mers results in higher compression during collapsing and less time spent on coarse mapping; but shorter k-mers increase the number of homologies in the homology table, as well as time spent on homology table traversal.

Coarse mapping. Coarse mapping is the stage in which the CORA framework utilizes the short-read mapping capabilities of an off-the-shelf mapper. Any read mapping tool that can report a best mapping (i.e., a mapping with a minimum Hamming or Levenshtein (edit) distance among all possible mappings) of a read, such as BWA and Bowtie2, can be directly adapted to the CORA framework. Theoretically, CORA does not require a best mapping to be found. Finding any mapping within the distance threshold would suffice to guarantee perfect sensitivity assuming the homology table distance threshold is twice the search distance threshold. However, when the homology table is imperfect (the distance of the homology table is less than twice the distance threshold used for mapping), CORA benefits from coarse mapping being performed by a best-mapper. See **A5**, for a detailed theoretical analysis.

For off-the-shelf aligners that do not allow a best-mapping scheme, alternative strategies can be designed for integration with some caveats. If a mapper does not allow best-mapping but allows a user-specified limit on the number of reported mappings, a mapping mode that reports only a single mapping (not necessarily with minimum error) can be integrated into CORA, with minor loss in final read-mapping accuracy. If the off-the-shelf mapper provides user-specified limits on the number of errors within a read, it is also possible to perform an iterative re-alignment scheme, at each stage aligning previously unmapped reads with a higher error threshold. While this approach will preserve final read-mapping accuracy, it will yield less compressive acceleration due to the heavy cost of re-alignments within the coarse mapping stage.

Unlike most read mappers, CORA's coarse mapping stage does not aim to find actual mapping locations for each read's k-mers (which would ultimately provide little compressive acceleration), but instead identifies only one single good ‘representative’ position in the reference that a k-mer can be mapped to and be represented in the form of compact links (i.e. position and differences). For short k-mer sequences that are used within the coarse mapping stage (33-64bp), it is substantially faster to find a single good location in the reference than find all mappings within the substitution-error neighborhood.

For paired-end mapping, finding a proper mapping in the reference involves finding multiple good locations for each mate and merging them according to the user-defined allowable interval of insert size between the mates (e.g. 150-650bp). Coarse mapping for a paired-end read consists of merely finding a good location in the reference for each mate independently. Similarly, for reads that are divided into multiple k-mers, each k-mer is coarse mapped independently to the reference genome.

Through these conceptual advances in the mapping stage, the CORA framework achieves massive speed gains (typically 1-3 orders of magnitude for all-mapping and more than three times the speed-up for best-mapping), as compared to the original performance of the off-the-shelf mappers.

Constructing the links table from the coarse mapping output generated by the off-the-shelf mapper (in SAM format) involves simply scanning the SAM file and creating a link item for the read code on each line. Each link in the table contains the read name encoding described above, the position in the reference, as well as a list of mismatch offset positions from the beginning of the read together with its nucleotide change in the read. Since the links table contains the differences of each k-mer from the reference, it is possible to regenerate the original read sequences in the input, and thus compression of read data within the CORA framework is lossless.

Query-side compression enables sublinear-time coarse mapping. We described above that query-side compression consists of grouping shared k-mers across the multi-individual read dataset, which then are coarse mapped onto the reference genome. In order to estimate the speed-up directly gained from k-mer compression of reads, we analyze the number of unique k-mers present in a large set of reads drawn from multiple individuals' genomes. For simplicity of analysis, we will initially assume the read dataset contains a single haploid individual and the sequencer returns k-mers rather than full reads. We will later generalize the model to include multiple potentially overlapping k-mers in each read from multiple diploid/polyploid individuals.

Single k-mer read model. We use a generative model for each k-mer read with a likelihood of machine error in the base call, following the authors' previous work [24].

Let $\Sigma = \{A, C, G, T\} \cong \mathbb{Z}/4\mathbb{Z}$ be the alphabet from which the k-mer bases are drawn. Define 0.9 < p < 1 and q = 1-p, such that a base is read correctly with probability p and incorrectly with probability q. For simplicity, we assume that a sequencing error is equally likely among the three alternative nucleotides. Let the random variable $\sigma: [0,1] \rightarrow \Sigma$ be defined by

$$\sigma(\omega) = \begin{cases} 0, & \text{if } \omega \in [0, p) \\ 1, & \text{if } \omega \in [p, p + \frac{q}{3}) \\ 2, & \text{if } \omega \in [p + \frac{q}{3}, 1 - \frac{q}{3}) \\ 3, & \text{if } \omega \in [1 - \frac{q}{3}, 1] \end{cases}$$

Therefore, $\forall l \in \Sigma,\ l + \sigma = l$ with probability p. Given $x \in \Sigma^k$, define x_i as the ith letter (base) of x. For all $x \in \Sigma^k$, define independently the Σ^k-valued random variables R^x by $\forall i, R_i^x = x_i + \sigma_i$ where $\sigma_i, \ldots, \sigma_k$ are independent instances of σ. Thus, R^x can be thought of as a read of x, including machine errors.

We will model the sequencer by a list $R^{y_1}, \ldots, R^{y_c}$, where c is the total number of k-mers read and each y$_i$ is drawn uniformly randomly from the set G' of unique k-mers in G. We will make the simplifying assumption that elements in G' are uniformly randomly distributed in Σ^k. We will eliminate both of these assumptions later on.

Let K be the set of unique k-mers in the list $\{R^{y_1}, \ldots, R^{y_c}\}$. For some particular y, if $\delta(x, y) = d$, the Hamming distance of x from y, then $Pr(R^y = x) = p^{k-d} q^d 3^{-d}$ because each of the matched locations must be preserved, each of the mismatched locations must be altered, and the chance of a correct alteration at each mismatched location is 1/3. Thus, if y is randomly sampled from G', by the union bound (Boole's inequality), we can conclude

$$Pr(R^y = x) \leq \sum_{d=0}^{k} p^{k-d} q^d 3^{-d} \cdot Pr(\delta(y, x) = d)$$

By our assumption that non-redundant k-mers in the genome G are distributed uniformly and continuously in k-mer space,

$$Pr(\delta(y, x) = d) \approx \frac{\binom{k}{d} \cdot 3^d}{4^k}$$

Consider now the total contribution from k-mers in the genome that are distance d from x versus the total contribution from those that are distance d+1 from x:

$$\frac{p^{k-d}q^{d}\cdot\binom{k}{d}}{p^{k-d-1}q^{d+1}\cdot\binom{k}{d+1}}=pq^{-1}\cdot\frac{d}{k-d}$$

This implies that the contribution from k-mers further away from x decreases exponentially with distance, so long as q < 1/k. Thus, $Pr(R^{y}=x)\approx p^{k-d}q^{d}\cdot\binom{k}{d}\cdot 4^{-k}$, where $d=\delta(x,G)$. This implies that $Pr(x\in K)$ is largely determined by $d=\delta(x,G)$ and by the total number c of k-mers read:

$$1_{x}\approx 1-\left(1-p^{k-d}q^{d}\cdot\binom{k}{d}\cdot 4^{-k}\right)^{c}$$

where 1_{x} is an indicator random variable for x in K. By linearity of expectation,

$$E[|K|]=\ E\sum\nolimits_{x\in\Sigma^{k}}1_{x}\ \approx\sum\nolimits_{d}\left[1-\left(1-p^{k-d}q^{d}\cdot\binom{k}{d}\cdot 4^{-k}\right)^{c}\right]\cdot|x\in\ \Sigma^{k}\colon\delta(x,G)=d|$$

We can bound $|x\in\ \Sigma^{k}\colon\delta(x,G)=d|\leq\binom{k}{d}\cdot 3^{d}\cdot|G|$. Additionally, if G is uniformly distributed across Σ^{k}, this bound is also a good approximation when $\binom{k}{d}\cdot 3^{d}\cdot|G|<4^{k}$ because the Hamming shells of radius d around each point in G intersect minimally.

Thus, we can estimate the number of unique k-mers as

$$E[|K|]\approx\sum_{d=0}^{D}\left[1-\left(1-p^{k-d}q^{d}\cdot\binom{k}{d}\cdot 4^{-k}\right)^{c}\right]\cdot S_{d}$$

where

$$S_{d}=\binom{k}{d}\cdot 3^{d}\cdot|G|,\text{ for }0\leq d\leq D-1$$

and

$$S_{D}=4^{k}-(S_{1}+\cdots+S_{D-1})$$

This bound indicates that Hamming shells with shorter radii around k-mers sampled from the genome are saturated exponentially faster than larger radii, causing the set of unique k-mers to grow sublinearly even in the presence of sequencing errors.

To find the coarse mapping speedup from query-side compression in the single k-mer read scenario, we simply take $c/E[|K|]$, which is equivalent to the rate of redundancy. In the case that the non-redundant data scales sublinearly with the full data– as it is for most real-life biological datasets–, the coarse mapping time of CORA will also be sublinear in the number of reads.

Note that the assumptions that G' consists of only unique k-mers and that k-mers in G' are uniformly randomly distributed both correspond to the worst case scenario for the upper-bound on the number of unique k-mers:

(1) If the k-mers were drawn from a multiset of nonunique k-mers in the genome, sequenced k-mers would have a higher likelihood to be redundant on average compared to a set that contains the same k-mers with no duplicates.

(2) If G' is not uniformly randomly distributed (e.g there are inexact repeats in the genome), the shorter-radius Hamming shells of some k-mers in G' will overlap with an increased likelihood. Therefore the chances of each sampled k-mer with sequencing errors to be a unique k-mer is reduced.

Multiple k-mers per read. We can generalize the single k-mer read model described above to multiple k-mers per read, since contiguous k-mers in the read are also contiguously sampled from the genome. In the case that two k-mers overlap in the read, any sequencing error in the overlap region will be observed jointly in two separate k-mers. As this reduces the combination of possible k-mers that can be drawn from these two k-mers with a joint sequencing error distribution, the expected rate of unique k-mers will also be lower compared to the scenario in which the two k-mers are disjoint with independent sequencing error distributions.

Generalizing the model above to diploid/polyploid genomes from multiple individuals with SNPs, indels and other variants requires altering the unique k-mer set G' to encompass all of the unique k-mers in the complete genomes of the individuals sequenced. In a real biological population of genomes of the same species, it is expected that the total number of unique genome k-mers in G' will grow sublinearly as new individuals are added.

Practical observation of sublinear scaling. As the theory described above suggests, the CORA framework achieves sublinear scaling for the coarse mapping stage.

In the absence of sequencing errors, the total number of read sequences that are processed in the coarse mapping stage is in effect constant after about 100 million 2x100bp reads (**Supplementary Figure S6**).

In order to illustrate the sublinear-scaling property of the CORA framework in practice with sequencing errors present, we computed the rate of redundancy within simulated read datasets (experimental setup details in Section **B3**) with various sequencing error rates and read depth-coverage (**Supplementary Figure S7**). The rate of redundancy is calculated as the total number of k-mer sequences sampled from the reads divided by number of unique k-mers processed by CORA during the 'coarse mapping' stage. Results indicate that the rate of redundancy monotonically increases, even in the presence of sequencing errors. Moreover, the rate of increase in redundancy often alternates between slow and fast as we increase read depth coverage, which is supported by the saturation rate of various Hamming shell radii as described above. Across different sequencing error rates, the rate of increase is slow universally for lower coverage datasets (below 1M reads), since k-mers have not yet fully covered the reference sequence. As read depth-coverage increases further, more k-mers overlap with the previously encountered reads, resulting in a faster increase in the rate of redundancy, thus allowing CORA's compression scheme to produce fewer k-mers to be processed by the coarse mapping stage. This feature can be observed on all E2, E0.5, and E0.125 datasets between 10M and 100M reads. Depending on the sequencing error rate, the rate of redundancy may flatten again at a higher depth-coverage level (such as between 100M and 1 billion reads in E0.125 and roughly 100M reads in E.05). This observation is due to the fact that as rate of redundancy increases, the non-redundant k-mer set becomes dominated by k-mers containing sequencing errors which are less likely to be encountered multiple times. Ultimately, with enough depth-coverage, k-mers containing sequencing errors also become redundant, and the rate of increase in redundancy becomes steep again (as observed around 1 billion reads in E0.5).

Implementation details and memory use. We implemented read collapsing with an unordered hash table (unordered_map class in C++ standard library), consisting of DNA sequences (the lexicographically smaller of the forward sequence or its reverse complement) as hash keys and a string that represents the encoded collapsed read names as the value for each key.

We would like to note that, for simplicity of implementation, the initial version of the CORA framework software employs a relatively restricted k-mer selection scheme, requiring the k-mer length to be chosen as either full or half the read-length, with k-mers fully covering the bases of each mate of the read. The software will soon be updated to implement the flexible k-mer selection model described above.

For large datasets with many unique sequences, which would cause the hash table to exceed the memory limit, we use a multi-pass approach to keep the size of the in-memory hash table within specified memory bounds. This approach is akin to the multi-pass one we used in order to compute the exact homology table. The amount of memory used during the collapsing stage is directly dependent on the number of passes performed within the multi-pass hashing scheme.

Below we demonstrate the memory/runtime trade-off of CORA's collapsing stage for the human FIN1 dataset from the 1000 Genomes Project with paired-end 108bp reads and 54bp k-mer length, using a 4-nucleotide splitting scheme (e.g. AA-CT), ranging from 12 to 96 passes (**Table S2**).

Number of passes	Peak memory use (GB)	Total runtime for collapsing (sec)
12	54.19	5464.97
18	39.25	4783.58
24	31.61	4592.06
32	25.87	4565.42
40	23.39	4604.68
48	19.68	4635.77
96	19.68	5288.44

Table S2: Runtime and memory cost of CORA's collapsing stage for the FIN1 dataset with 54bp k-mer length for different multi-pass collapsing schemes.

Memory/runtime trade-off results indicate that memory use for the collapsing stage can be reduced by increasing the number of passes. Note that for the 4-nucleotide splitting scheme, there is no further memory reduction after 48 passes. However, the memory can be further reduced by using a 6-nucleotide splitting scheme instead (e.g., ATA-GGT with 3 nucleotides at each end of the k-mer), resulting in a slightly higher total runtime cost. In terms of runtime, too few passes are not desirable due to the incremental cost of hash-table operations for larger tables. On the other hand, too many passes are not desirable either, due to the overhead cost of I/O handling. We show that roughly 32 passes is ideal in terms of the runtime optimization for human datasets, resulting in less than 26GB memory use. However, the CORA software allows users to increase the number of collapsing stage passes in order to meet their runtime or memory use preferences.

As the CORA framework utilizes an off-the-shelf aligner (mapper) for its coarse mapping stage, the memory cost is directly dependent on memory needs of the aligner used. In the cases of BWA and Bowtie2, the memory cost of coarse mapping is dominated by the k-mer collapsing stage. Since the link construction stage involves scanning only a SAM file to generate the read links, its memory consumption is negligible.

A4. Homology table traversal

The main goal of this phase is to infer final paired-end read mapping results, making use of the pre-computed homology table of the reference, as well as the compact links table representing the read dataset.

As links and homology tables are all represented in terms of positions in the reference and the differences from it, rather than the read sequence itself, almost all of the computation in this phase of the CORA framework is performed in terms of edit operations rather than direct sequence comparison. Since a link generally has much fewer edits than the number of bases in the read, utilizing algebraic relations within this 'edit space' results in a much more efficient sequence comparison method than the base-to-base comparison approach used in conventional read mapping.

A schematic representation of the homology table traversal for a single read link is provided in **Figure S5**. Each link connects one or more k-mers to a position in the reference (the anchor of the link) in forward or reverse complement direction, indicating homology within a small number of edits. This anchor is either a unique k-mer in the reference genome or associated with an equivalence class within the exact homology table. In the latter case, the anchor is linked to the equivalence class in the forward or reverse complement direction, with a block offset value indicating the starting location (say offset = *p*) of a substring of the equivalence class representative that is identical to the anchor sequence. If an anchor does not belong to any existing equivalence class, it is considered to be the representative of its own single-item equivalence class.

A representative of an equivalence class points to all members of the equivalence class for each valid offset (until the end of the block) in the forward or reverse complement direction as described above. Among these pointers, only the ones with the offset *p* are relevant to the read link, and solely for these will the inferred mapping result be reported. Furthermore, the equivalence class representative of the anchor can be linked to other class representatives through the inexact homology table.

Each of these pointers contains the direction of homology (forward or reverse complement), block offset in the compressed inexact homology block representation, and the edit script to convert one k-mer to another. Provided that the juxtaposition of the edits from a read link to the anchor and from the anchor's class representative to the neighboring equivalence class still contains less than or equal to the Hamming distance threshold, read mappings associated with all members of this neighbor equivalence class will be reported.

For single k-mer reads (e.g., ≤ 64bp) that are collapsed as a whole read into read links, the final mapping output can be generated by the procedure described above. For paired-end and/or longer reads that contain multiple k-mers, all links relevant to the read are traversed to generate the final mapping(s). CORA achieves this traversal by loading the links table in multiple passes to avoid exceeding the memory limit. These links are then independently converted into mappings. In the end, these mappings are merged in order to generate the final set of paired-end mappings.

Traversal with indels. Even if the homology table data structure is built using a Hamming distance metric that only allows substitutions, it is possible to report gapped final mappings with indels (insertions and deletions), given that the coarse mapping is performed using an off-the-shelf tool that can perform gapped mapping and the error penalty assigned to insertions and deletions is greater than or equal to the penalty assigned to substitutions (e.g., Levenshtein distance that assigns identical penalties for substitutions, insertions, and deletions).

The homology table is initially built to provide Hamming neighbors of a k-mer in the reference for a fixed error distance threshold, E. However, it is also possible to recover homologies of (k-e)-mers in the reference (e ≤ E), which is guaranteed to discover all homologies within a Hamming error distance threshold of E-e, ignoring the matches/mismatches in the last e bases of homology table k-mers. Since e is often much smaller than k, most of the E-distance homologies of (k-e)-mers will also be recovered. This ability to search for (k-e)-mers in the homology table enables neighborhood search of k-mers that are coarse mapped with insertions in them. In the case that a k-mer is coarse mapped with a single base insertion in the k-mer link (which corresponds to a single base deletion from the reference), the k-mer is aligned to a (k-1)-mer in the reference. Using the above approach, CORA can recover most of the homologous regions of the (k-1)-mer.

In the case of a single nucleotide deletion in the k-mer link (insertion in the reference), the k-mer corresponds to a (k+1)-mer in the reference. In this case, the homology table can enumerate the k-mer homologies for any k-mer contained within this (k+1)-mer and then determine whether the (k+1)-mer homology is within the valid error distance threshold by checking the similarity of the last base in the (k+1)-mer.

For multiple insertions and deletions in the k-mer link, the homology table traversal strategy is determined by t = #deletions - #insertions. If t < 0, the first strategy employed ignores the last t bases in the k-mer homologies; if t > 0, the second strategy employed checks for t additional bases at the end. If t = 0, the initial version of the k-mer based homology table traversal is performed, while handling shifted bases due to insertions and deletions within the k-mer when juxtaposing k-mer homology edits with the edits from the k-mer link. The t value is also important when merging multiple k-mer link mappings into a whole read during the final mapping stage. If one of these k-mers have a positive or negative t value, k-mer merging is performed between accordingly adjusted mapping positions.

Mapping recovery. Although the links table is a lossless representation of the read sequences in the original dataset, there are three main reasons why a particular read mapping may be missed during standard traversal of the homology table: (a) Using an inexact homology table with a distance threshold lower than twice the mapping error is used; and (b) loss of sensitivity during coarse mapping due to imperfect sensitivity of the coarse mapper or (c) non-uniform distribution of errors among k-mers sampled from the read during collapsing. In order to achieve near-perfect sensitivity, CORA employs a recovery scheme which recovers a portion of the missed mappings.

In order to recover mapping locations of reads with unmapped k-mers during coarse mapping, CORA keeps track of all k-mers that are unmapped during the coarse mapping stage. Then the homology table traversal generates all mapping loci for the other k-mers of the read. If more than a certain percentage (e.g., 75%) of its k-mers constitute a viable mapping (i.e., only missing the unmapped k-mers but satisfy the error distance and min/max read insert length conditions for the others), each unmapped k-mer is extended using a base-by-base comparison method. In the case of gapped mapping base-by-base comparison is replaced by a banded dynamic programming algorithm in order to detect indels.

As the mapping recovery stage is performed independently from homology table traversal, the dynamic programming algorithm within this stage can handle larger error distances than the homology table allows for (e.g. 10 or more mismatches), as well as affine gap penalties to allow for dynamic trimming of reads (when the k-mer is at the end) or alternatively capture indels longer than the error distance threshold.

When constructing a homology table, in order to guarantee perfect sensitivity for obtaining all matches of a k-mer in the reference genome within a Hamming distance of D, the homology table should ideally use a distance threshold of 2D. This is due to the fact that, even when a read k-mer is within distance D of both the coarse mapping locus and final mapping locus, the distance between the final mapping locus and the coarse mapping locus could be as large as 2D, if the set of edits from the read to the coarse mapping locus and the read to the final mapping locus are disjoint. However, as homology tables with large error rates take substantially longer to construct and consume larger memory, for practical performance reasons, the threshold of the homology table can be chosen to be less than 2D (such as D, which is used for the main experimental results in the paper), resulting in imperfect sensitivity. In order to recover mappings that have distance higher than D to the coarse mapping, CORA again employs a nucleotide-level string comparison given that a large percentage of the remaining k-mers constitute a viable mapping. In Section **B2**, we provide an experimental analysis on the runtime and sensitivity trade-off of the recovery scheme for a variety of homology table distance thresholds and perform a comparative analysis on experiments when the recovery scheme is enabled and disabled.

In Section **B2**, we provide an experimental analysis on the runtime and sensitivity trade-off of the recovery scheme for a variety of homology table distance thresholds and perform a comparative analysis on experiments when the recovery scheme is enabled and disabled.

Implementation details and memory use. For the human (hg19) read-mapping experiments presented in this study for a k-mer length of 54bp, the exact homology table requires ~0.8 GB of memory, whereas the inexact homology table that represents all reference homologies of Hamming distance 2 requires ~22.2GB of space in memory. For CORA's fast best-mapping mode, only the exact homology table is required, whereas for all-mapping mode or a more sensitive best-mapping mode, CORA requires both exact and inexact homology tables to be represented in the memory. Apart from the homology table, CORA also utilizes lookup tables for the reference in order to identify the nucleotide at each locus in the reference as well as corresponding indices in the homology tables. This table requires an additional 2.4 GB of memory for best mapping modes that do not utilize the inexact homology table and 24.8 GB for other mapping modes (the higher memory usage is due to index values needed for the inexact homology table). Among the best mapping modes, when the inexact homology table is used, both sensitivity and runtime is slightly higher than the modes that do not utilize the inexact homology table (**Figure S2**).

Finally CORA allocates an adjustable amount of memory to represent the read links in RAM for fast access. For the FIN1 dataset, the maximum memory size needed for allocation is 2.76 GB (as opposed to 29.1 GB, which is the original FASTQ file size). However, this amount can be reduced by a factor of X through a multi-pass option that scans the read link file X times.

For speed optimization on high-coverage read datasets, CORA uses a memoization scheme. Memoization is a widely-used optimization technique in computer science that aims to gain speed through storing the output of expensive function calls.

For the read links that represent more than a certain number of k-mers in compressed form, CORA traverses the homology table once and saves the inferred genomic positions for the link in a lookup table. Then for each of the k-mers within the link, this lookup table is used instead of re-computing the genomic neighbors using the homology table. In whole genome real-data experiments **(Figures 1 and S2)**, we chose this threshold to be 20 which resulted in a good balance between runtime improvement and additional memory required for the lookup table. Below we present a breakdown of runtime and peak memory usage (maximum resident set size) of homology table traversal for best- and all-mapping, as well as a variety of memoization threshold values. While CORA's memoization scheme results in moderate speed gains for best-mapping with a negligible increase in peak memory usage, it provides drastic speed gains for all-mapping (when the threshold is set to 10 or 20) with a relatively small increase in peak memory use (**Table S3**). All experiments reported on human datasets (**Figures 1 and S2**) with memoization threshold 20 can be run on a machine with ~64GB of free RAM.

Memoization Threshold		No memoization	20	10	5
Mapping mode					
Best mapping (Fast mode)	Runtime (seconds)	1331	1200	1125	1191
	Peak memory	8.90 GB	9.29 GB	9.30 GB	9.67 GB
All-mapping	Runtime (seconds)	11260	7743	7797	7134
	Peak Memory	48.3 GB	51.3 GB	54.3 GB	72.3 GB

Table S3: Runtime and memory cost of CORA's homology table traversal stage on FIN1 dataset for best and all-mapping for different memoization thresholds.

A5. Using homology table improves asymptotic complexity of seed-and-extend based mapping.

We now show that pairing a seed-and-extend based mapping method with a homology table reduces the asymptotic complexity of mapping on average by a factor of the redundancy of the reference genome with respect to the read sequence. Note that this complexity analysis is independent of the size of the read dataset or the redundancy within. As shown earlier, in the case of coarse mapping in CORA, the asymptotic complexity is further reduced by a factor of the redundancy within the read dataset itself due to k-mer compression of reads. Furthermore, even though the measure of redundancy of the genome with respect to the read is correlated with the redundancy within the reference genome, the former is lower in cases such as uniquely mapped or unmappable reads as we quantify later.

Though we initially assume an ungapped all-mapping model, we extend this analysis later to the gapped mapping scenario. The complexity analysis also holds in the best-mapping version of the problem, except a corner case which we elaborate on later. We always consider the perfect sensitivity versions of the problem in our complexity analysis, and therefore exclude heuristic or greedy algorithm based mappers that generate imperfect sensitivity mapping results. We also assume a strict seed-and-extend mapping model, where each seed is separately searched in the reference genome, without any precomputation on the read dataset (such as CORA's k-mer compression scheme) and all different seed hit loci are extended.

For our analysis below, we need to define a mapping method that reports an arbitrary mapping locus in the reference genome within a specified distance threshold of the read without any preference for its locus or actual distance, or none if there is no mapping locus within the distance threshold. We will call this an ‘any-mapper.’ For the following analysis, we will assume that we will be pairing a seed-and-extend based any-mapper with the homology table in order to build an all-mapper, and compare its runtime complexity to a seed-and-extend based all-mapper. We will assume that the underlying complexity of seed-and-extend operations are consistent across the mappers.

We define the seed operation as sampling a shorter substring from the read and mapping it exactly (i.e. with no errors) to all matching locations in the reference genome which then become candidate loci for the extend stage. The extend stage is defined as identifying which of the candidate loci are valid mappings within the similarity threshold (in the case of all mapping) or identifying the locus with the highest similarity score (in the case of best mapping). To achieve this, the seeds are extended towards either direction of the reference match.

For ease of analysis, we assume that the read is single-end and is of fixed length k. We will refer synonymously to a read and the k-mer it represents from here on.

Let us define $\mathrm{R}(\mathrm{q},\mathrm{d}) = \{\mathrm{p} \in \mathrm{G}\colon\ \delta\ (\mathrm{p},\mathrm{q}) \leq \mathrm{d}\}$, the local k-mer neighborhood in our genome G of the query k-mer q for Hamming distance d>0. Note that $|\mathrm{R}(\mathrm{q},\mathrm{d})|$ serves also as a good measure of the redundancy of the reference genome with respect to the read q. This aligns with our natural intuition for redundancy as describing the self-similarity of G with respect to q, as $|\mathrm{R}(\mathrm{q},\mathrm{d})|$ is also the output size of a perfectly sensitive all-mapper that maps q to G with a distance threshold of d. We exclude unmappable reads from this complexity analysis as $|\mathrm{R}(\mathrm{q}, \mathrm{d})|$ = 0; however, since the use of an any-mapper or a homology table in the case of unmappable reads will not have an effect on the runtime, its asymptotic complexity behavior is similar to the uniquely aligned reads with $|\mathrm{R}(\mathrm{q}, \mathrm{d})|$ = 1.

Consider an all-mapper τ employing the seed-and-extend strategy as follows to return $\mathrm{R}(\mathrm{q}, \mathrm{d})$ with guaranteed perfect sensitivity:

(1) Break the read q of length k up into length s seeds.

(2) Find all locations in the genome that match these seeds in order to enumerate tentative mapping locations.

(3) Extend each seed hit to check if the tentative mapping location is within distance d of q.

(4) Report positions of the mapping locations within distance d.

Note that the read q must be broken up into at least d+1 seeds to ensure that at least one seed will have a perfect match to the reference index [25]. Thus, a seed length s ≤ $\lfloor \mathrm{k}/(\mathrm{d}+1) \rfloor$ is required in order to guarantee perfect sensitivity assuming uniform seed lengths.

Let us define the complexity function of this seed search operation (2) of the seed-and-extend based read mapper to be f(s,|G|). When considering all of the seed search operations, the complexity of the seed stage overall will be O(d × f(s,|G|)). If we assume that the data structure used will allow accessing each additional seed hit in the genome in constant time, the time required to access all additional seeds will be dominated by the extend stage time, thus we can ignore it.

The computational cost in step (3) is finding whether each seed match in the reference is a valid read mapping with at most d mismatches. Let us define the complexity function of this extend operation to be g(k,|G|,d). In the case of a hash table implementation, the runtime complexity of the extend operation will be O(k) as we need to compare k-s bases between the read and the reference locus. As we need to perform the extend stage for each seed hit, the total complexity of this stage is O(#of seed matches × g(k,|G|,d)). The cost of reporting the mapping positions will be dominated by #of seed matches, thus we can ignore it for the all-mapper.

Now let us consider pairing an any-mapper m with a homology table H to build an all-mapper. Given a query string q, m(q) returns either the location of some substring of G within distance d of q, or that there is no such location. We construct the any-mapper similarly to the seed-and-extend all-mapper above, except that we short-circuit the extend stage once a location within distance d of q is found. In expectation, the runtime of this any-mapper will be $\mathrm{O}\left(\frac{\#\text{of seed matches} \times \mathrm{g}(\mathrm{k},|\mathrm{G}|,\mathrm{d})}{|\mathrm{R}(\mathrm{q},\mathrm{d})|}\right)$, assuming we process the seed hits in a randomized order.

Now, let us consider the computational cost of using a homology table to generate all of the mappings of q to the reference within distance d, given m(q) from the any-mapper. Below, we redefine the homology table in a slightly different way than the description in Section **A2**. This change is due to the fact that, here we are concerned solely about the asymptotic complexity of the problem rather than practical concerns when building the CORA framework, such as memory/disk-space or preprocessing time costs. For example, we exclude here the block-merging step of the actual homology table used in the CORA framework, which in practice saves a lot space with little or no effect on real runtime, but might increase the worst case complexity of search.

We will construct the homology table for distance 2d as follows:

(1) Bin the locations in the genome together by exact k-mer matches. Because each bin contains all locations with a particular k-mer, we will refer synonymously to the bin and the k-mer it represents.

(2) For every pair of bins b_1 and b_2, create a link between them if the distance between their k-mers is $\leq 2d$. In that link, store the position of the mismatch(es).

Then our homology-table augmented all-mapper algorithm will be as follows:

(1) Use the any-mapper to get m(q), which is within distance d of q.

(2) Look up m(q) in the homology table.

(3) In the homology table, for every link from m(q)'s bin, check if following that link will lead to a bin with k-mers within distance d of q.

(4) Traverse the appropriate links and return R(q,d).

By the triangle inequality, we are guaranteed to get all of R(q,d) using this scheme, so long as the homology table has all links with distance $\leq 2d$. Note that because homology table links store the positions of the mismatches, we need only check the mismatches to determine the Hamming distance between the new bin and q. Thus, homology table traversal takes $\mathrm{O}(\mathrm{d} \times |\mathrm{R}(\mathrm{m}(\mathrm{q}),2\mathrm{d})|)$ time. Total runtime for the homology-table augmented all-mapper is thus $\mathrm{O}\left(\mathrm{d} \times \mathrm{f}(\mathrm{s}, |\mathrm{G}|) + \frac{\#\text{of seed matches} \times \mathrm{g}(\mathrm{k},|\mathrm{G}|,\mathrm{d})}{|\mathrm{R}(\mathrm{q},\mathrm{d})|} + \mathrm{d} \times |\mathrm{R}(\mathrm{m}(\mathrm{q}), 2\mathrm{d})|\right)$.

This complexity can be further reduced if we modify the homology table to have a precomputed mismatch table for each bin, that specifies which other bins will be within distance d, depending on the locations of differences between m(q) and q. For example, a bin can point towards different sets of neighboring bins, in the case that the 5th position in the k-mer has a mismatching base 'G' versus the 26th position has a mismatching base 'T'. In the naïve version, this would require us to have a table of size $\binom{\mathrm{k}}{\mathrm{d}} \times 3^{\mathrm{d}}$ for each bin, even though more space efficient data structures are possible that exploit the sparsity of this table. With this expanded homology table, total runtime complexity reduces to $\mathrm{O}\left(\mathrm{d} \times \mathrm{f}(\mathrm{s}, |\mathrm{G}|) + \frac{\#\text{of seed matches} \times \mathrm{g}(\mathrm{k},|\mathrm{G}|,\mathrm{d})}{|\mathrm{R}(\mathrm{q},\mathrm{d})|} + |\mathrm{R}(\mathrm{q}, \mathrm{d})|\right)$, assuming output costs consist of reporting only the position in the genome in constant time for each mapping.

The initial complexity of the all-mapper is $\mathrm{O}\big(\mathrm{d} \times \mathrm{f}(\mathrm{s}, |\mathrm{G}|) + \#\text{of seed matches} \times \mathrm{g}(\mathrm{k}, |\mathrm{G}|, \mathrm{d})\big)$. We did not add an additional term for the output cost |R(q,d)| as #of seed matches = Ω(|R(q,d)|). If d × f(s,|G|) = O(g(k,|G|,d)) as well as $|R(q,d)|^2$ = O(#of seed matches × g(k,|G|,d)), we can claim that an any-mapper paired with the homology table is faster than an all-mapper by a factor of |R(q,d)|, which corresponds to the redundancy of the reference with respect to the read. Assuming that f(s,|G|) = O(s = $\left\lfloor\frac{\mathrm{k}}{\mathrm{d}+1}\right\rfloor$) and g(k,|G|,d) = Ω(k), the first condition is satisfied. Both of these assumptions are valid, since a hash-table is able to satisfy the first condition; and since we assume no precomputation on the read dataset, the remaining bases in the read, apart from the seed loci, and error positions are unknown prior to the extend stage, thus requiring Ω(k) operations for a k-mer with a seed hit.

For the second condition, we need to show that #of seed matches = Ω ($|R(q,d)|^2$). In the generalized case, where we have d+1 seeds of length s = $\lfloor k/(d+1) \rfloor$, the maximum number of unique d-away k-mers we can construct from q is $\binom{k}{d} \times 3^{d}$. Whereas the total number of possible unique k-mers that contain at least one exact seed in their corresponding k-mer position is $4^{k} - \left(4^{\left\lfloor \frac{k}{d+1} \right\rfloor} - 1\right)^{d+1} = \Omega\left(4^{\frac{k}{2}}\right)$. If the reference is assumed to be constructed in a non-adversarial way by selecting k-mers from a unique k-mer list with repetitions, by expectation the second type of k-mers will appear exponentially more often than the first type of k-mers, for 0 < d < $\frac{k}{2}$. In this model, it is easy to show that the #of seed hits grows faster than quadratic in the number of valid mappings, satisfying our second condition.

By using the homology table, if the runtime is dominated by the number of seed matches, we are able to get acceleration proportional to the local redundancy of our query in the genome. Amortized across all reads and their seeds, this runtime will by expectation correspond to the average redundancy of the reference genome with respect to the read dataset. This quantity is a function of the similarity between the donor genome and the reference genome as well as the sequencing error rate.

In the perfect sensitivity version of the best-mapping problem, in which a best mapping with minimal distance to the reference genome is guaranteed to be found provided that one exists within the predetermined distance threshold d, the seed stage will be performed as earlier for both best-mapper and any-mapper. The extend stage will also still be the same for an any-mapper. If we make sure that the precomputed mismatch table in the homology table points to the interval of minimal distance mapping, the traversal stage can find a random best-mapping in constant time. In the case that there are no perfect matches, the complexity of the extend stage will not be changed as each seed hit needs to be extended to make sure that there is not a mapping with a lower number of errors. In the case that there is no more than a constant number of perfect matches for the read in the reference, the complexity of the extend stage for perfect sensitivity best-mapping will be the same as perfect sensitivity all-mapping, as the expected number of seed hits that need to be extended is not changed. Even in the presence of relatively frequent perfect matches, the any-mapper will be faster, but not necessarily as much as a factor of |R(q,d)|.

In the gapped mapping version of the problem the extend stage will be more costly due to the detection of insertions and deletions while extending. However, the seed stage will stay the same as we are still guaranteed to find all read mappings if d+1 disjoint seeds are sampled from the read of length s ≤ $\lfloor k/(d+1) \rfloor$. Since the cost of the gapped extend stage is lower bounded by the ungapped extend stage, our analysis still holds in the gapped all-mapping and gapped perfect sensitivity best mapping problems.

Note that for uniquely aligned reads as well as unmappable reads, we do not expect any speedup of more than a constant factor. However, even in these cases compressive acceleration is possible due to k-mer compression of reads in the CORA framework.

Moreover, in our asymptotic analysis in this section, we did not include mapping methods based on greedy algorithms that only check a subset of the seed hits for mapping (in particular best-mapping), resulting in imperfect sensitivity mapping results. Thus, other practically useful speed improvements of methods that cannot guarantee perfect sensitivity (e.g. Bowtie2, BWA, Masai, GEM, etc.) are left outside of this complexity analysis.

A6. Related work and novelty of CORA framework

Here we describe existing work and how each differs from CORA's compressive read-mapping framework.

In their pioneering work, Veeneman et al.[13] describe a read-mapping methodology, Oculus, which wraps an existing aligner and aims to accelerate read mapping via elimination of duplicate reads in the input read dataset prior to mapping; the method reconstructs final mappings by regenerating additional mapping results for the unmapped, eliminated reads using results from their mapped copies. In essence this approach is similar to a primitive version of CORA's single-end read-mapping without using a homology table, sampling only a single whole read for the collapsing stage. Though a simplistic approach like this can provide modest compressive acceleration for highly redundant read datasets that are single-end and have short read-length, it strikingly fails to provide any meaningful acceleration for longer read-length paired-end read datasets, such as the 2x108bp read datasets from 1000 Genomes Project used in FIN1, FIN2, FIN4 experiments presented in this study. In contrast, CORA can provide orders of magnitude speed increase over existing aligners on the FIN4 dataset, whereas the total number of duplicate paired-end reads in this dataset is less than 3%, which represents a hard upper bound on the acceleration Oculus can provide. Moreover, Oculus requires time to de-duplicate the input and expand the output mappings for eliminated reads, resulting in even smaller margins for acceleration, which is roughly 2%. As opposed to Oculus, CORA's acceleration stems from the fact that k-mers shorter than read length are used for compression (with exponentially increased odds for duplicates), splitting a longer read into multiple smaller chunks and constructing the whole paired-end read mapping results after coarse mapping through the use of the homology table rapidly enumerating multiple mapping positions for each k-mer in the reference genome. Without the use of a homology table, such a k-mer based compressive mapping approach would be prohibitively costly.

Apart from identifying exact duplicates, Mahmud and Schliep explored wrapping existing aligners using a whole-read clustering scheme (TreQ-CG) where reads that are similar within a distance threshold are clustered together [32]. While some level of acceleration is gained from this approach, it comes with a significant loss in mapping sensitivity. Furthermore, whole-read inexact clustering is a costly solution, causing CORA to be able to run its entire mapping pipeline several times over before TreQ-CG completes even its initial read clustering stage.

In addition to end-to-end compression of reads through de-duplication or clustering, some read-mappers [10,11,12] have explored the idea of jointly representing seeds or putting them in a compact data structure, in order to accelerate the seed-matching stage of mapping. Below we summarize the idea of clustering seeds prior to seed-matching step and describe the conceptual and practical differences between the CORA framework and joint seed-matching approaches.

Inspired by the efficient seed-and-extend alignment algorithm of BLAST[26], almost all short-read mapping algorithms are designed with two main computational stages: the seed stage and the extend stage. The seed stage can be broadly described as sampling a short subsequence (or multiple subsequences) of the read and matching it exactly (or with very high similarity) to multiple tentative locations in the reference genome which then become candidate loci for the extend stage. The main computational cost of the seed stage is searching for seed sequences in a data structure that represents the reference genome (e.g. a hash table or suffix array). The extend stage can be broadly described as identifying which of the candidate loci are valid mappings within the similarity threshold (all mapping) or identifying the locus with the highest similarity score (best mapping) through extending the seeds towards either direction of the reference match (as well as filling in gaps in the case of spaced seeds). The main computational cost of the extend stage is to perform similarity comparisons between the reference and the bases in the reads that are not contained in the seed, which quite often corresponds to the majority of the bases in the read, as well as the majority of the computational cost of read-mapping. Most read mapping methods already employ a preprocessed reference index that allows fast identification of seed matches in the reference genome, thereby speeding-up the seed stage of mapping. Furthermore, some methods such as mrsFAST-Ultra, Masai and BWA-SW employ a seed clustering scheme that also process the seed-set on-the-fly in order to reduce redundancy in seed-matching comparisons across multiple seeds from different reads, which aim to further accelerate seed matching stage.

Conceptually, there are three key differences between CORA's k-mer read compression scheme and joint seed-matching schemes that exist in the literature. Firstly, the length of the k-mers that CORA collapses are chosen to be significantly longer (40-60bp) than the seed lengths that are typically used by read-mappers (8-20bp). Therefore, the type of redundancy that CORA leverages is based on longer homologous regions across both the reference genome and the reads sequenced from these regions across multiple individuals; on the other hand, existing methods rely on a large number of spurious hits that short seeds would normally produce, often increasing the computational cost of the extend stage. Longer k-mer lengths are essential to the CORA framework, as CORA's k-mer read compression scheme is complemented by a homology table that enables very efficient all-mapping via rapid enumeration of homologous regions in the reference genome. Building a homology table with such short seeds would be intractable, as it would be infeasible to pre-process, store and enumerate the exponentially larger number of homologies. Secondly, by joint seed-matching, read mappers aim to accelerate only the seed stage of mapping (which requires relatively less costly computations in the form of data structure access operations compared to costly nucleotide-level comparisons of the extend stage), whereas CORA's k-mer read compression accelerates the entirety of the coarse mapping stage, which includes both the seed stage and the costlier extend stage. Thirdly, while the seed-matching stage requires identifying many regions in the reference genome in order to produce candidate mapping loci for the extend stage (otherwise the sensitivity of mapping would suffer), it suffices for CORA's coarse mapping stage to only produce any single region in the reference genome, that is, within a specified distance of the compressed k-mer. As we analyzed above, the asymptotic complexity of this search operation is lower than seed-and-extend based inexact mapping by a factor of the redundancy in the reference genome with respect to the reads.

In practice CORA's runtime performance is superior to mrsFAST-Ultra with near-perfect sensitivity and to Masai with much better sensitivity. BWA-SW is primarily designed to efficiently map longer reads (>200bp) and costlier than other BWA methods for FIN1-4 datasets.

Overall CORA's k-mer based read compression, complemented by its compressive homology table, represents a novel methodological advance in read mapping. This advance is also apparent in the superior practical performance, in terms of both speed and sensitivity, of our implementation of the CORA compressive read-mapping acceleration framework as compared to existing tools.

A7. Software features

We provide an implementation of CORA at http://cora.csail.mit.edu, which can readily perform rapid gapped read-mapping for paired-end Illumina read datasets, given a FASTA/FASTQ input file, reporting a SAM format mapping output file. We discuss features of our implementation below.

Integration of other mapping tools into CORA. For easy integration of other mapping tools into the CORA framework, we implemented a manual coarse mapping mode: Users can give a series of commands to run for the coarse mapping stage of CORA (specifying the executable files, input, output and reference), and CORA automatically calls these commands to perform the coarse mapping stage, without requiring any code changes in the mapper or CORA framework.

Using existing mappers within CORA framework. We implemented the CORA framework in the C++ programing language. It does not have any dependencies apart from the user-specified coarse mapping tool to be incorporated into the framework. Virtually all short-read mapping tools can be plugged into the CORA framework with minimal or no changes to the implementation. Currently integrated tools and mapping modes include BWA[3] aln and mem, Bowtie2[4], mrsFAST[5], and mrsFAST-Ultra[10].

The current implementation of CORA can perform mappings of paired-end read datasets with uniform read-lengths (between 2x36bp and 2x150bp) within a Hamming distance threshold. In addition, users specify an allowable insert length interval for mate-pairs. CORA performs end-to-end mapping of reads, which corresponds to the global alignment of each read to a locus in the reference. The current implementation of CORA does not make use of quality scores within the alignment (all substitutions or indels have equal weight within the Hamming/Levenshtein distance metric); however, quality scores can optionally be printed for downstream use in a sequence analysis pipeline.

Integration of CORA into pipelines and other mapping tools. CORA can be readily integrated into NGS processing pipelines that work with the commonly used SAM format for representing read mappings. However, it is also possible to integrate key components of CORA, such as k-mer based read compression and/or homology table traversal, into other mapping tools through the CORA framework libraries. For example, through the use of CORA's libraries, an external method can perform k-mer based compression on a set of sequences or can access the homology table in order to query the homologous locations of a k-mer in the reference genome directly without the need to run the entire CORA pipeline.

Distance metric options. CORA allows the user to specify the distance metric used for mapping, Hamming or Levenshtein (edit) distance, without any modification needed to the homology table. Current version of CORA software allows the user to set the distance threshold up to 6 edits/substitutions all and best-mapping. For best-mapping, the CORA package has available mapping inference modes that can also handle distances higher than 6.

Mapping reporting options. CORA allows the user to specify whether he/she wishes to output all-mappings, best-mappings, unique-mappings or best stratum mappings, the latter of which corresponds to outputting all mappings within the highest tier. Furthermore, CORA enables reporting of the original read names and quality scores from the input read dataset, or alternatively assigns numbers to the reads that represent their order in the read dataset.

Future extensions to CORA framework. In future updates to the CORA framework, we will be extending the k-mer based compression of the read dataset, with a flexible k-mer selection scheme independent of read length, allowing varied read lengths in the input datasets as well as soft clipping (dynamic trimming) for reads with low-quality ends. Finally, we will be adding support for multi-reference alignment (mapping) by augmenting the homology table with links to individuals' variants within user-specified VCF (Variant Call Format) files.

B. Supplementary Results

B1. Datasets

Real NGS datasets used. For the experimental results shown in this paper (**Figure 1, Figure S2**), we used three real-life whole-genome sequencing datasets (mentioned below as FIN1, FIN2 and FIN4) with varying depth-coverage obtained from the 1000 Genomes Project [27]. These datasets include Illumina sequencing reads from 7 Finnish individuals: HG00173, HG00174, HG00176, HG00177, HG00178, HG00179, and HG00180. Among these, FIN1 contains only HG00173, FIN2 contains two individuals, HG00174 and HG00176, and FIN4 contains the remaining 4 individuals HG00177, HG00178, HG00179, and HG00180. Finnish individuals were selected from the 1000 Genomes Project due to the availability of long and uniform length paired-end read sets with good coverage. For the experiments, we have only used reads from the 7 Finnish individuals, which are paired-end and 108 base pairs on each end. The depth of coverage of FIN1 is 4.25x, FIN2 is 8.04x and FIN4 is 15.87x. The hg19 human reference genome multi-fasta dataset is used as the reference with the default reference indexing scheme used in all comparisons between mapping tools. The hg19 reference genome dataset we used contained contigs for all autosomes and sex chromosomes, chromosome M as well as 29 alternative contigs for various chromosomes and 39 additional "chrUn" haplotypes that were not placed in any reference chromosome.

Chromosome 20 read datasets used in the experiments in **B2** are from 32 Finnish individuals in the 1000 Genomes Project, with their reads restricted to chromosome 20. The individuals included are: HG00171, HG00173, HG00174, HG00176, HG00177, HG00178, HG00179, HG00180, HG00182, HG00183, HG00185, HG00186, HG00187, HG00188, HG00189, HG00190, HG00266, HG00267, HG00269, HG00270, HG00272, HG00306, HG00311, HG00312, HG00357, HG00361, HG00366, HG00367, HG00369, HG00372, HG00373, and HG00377. The original read datasets all contained paired-end 108bp long reads. For the experiments with the paired-end read lengths of 70bp, 80bp, 90bp and 100bp, we trimmed each mate from the end that is closer to the fragment center. The total number of reads in these 32 datasets is ~37.27 million, with total depth coverage of 127.7x, 118.3x, 106.5x, 94.6x, 82.8x for the 2x108bp, 2x100bp, 2x90bp, 2x80bp and 2x70bp datasets, respectively.

Simulated NGS datasets used. For the simulation results shown in **Figure 1c, S6 and S7**, we used the human reference genome (hg19) with a fixed mutation rate of 0.1%, in order to capture the k-mer redundancy profile of the diploid human genome in the presence of mutations. Using this mutated reference, we simulated 20 million paired-end reads (2x100bp) from chromosome 20 with varying sequencing error rates: 2%, 1%, 0.5%, 0.25%, and 0.125%. We used SAMtools wgsim tool[28] for simulating reads.

Mouse dataset. In addition to human read mapping experiments, we also present supplementary results on the mouse genome. For mouse experiments, we used the original mm9 reference genome (NCBI build 37) as downloaded from UCSC genome bioinformatics site. Our mouse read datasets are taken from the Mouse Genomes Project [29]: ERR118246, ERR118251, ERR118256, and ERR118261, which each consist of paired-end 100bp Illumina HiSeq 2000 reads. The accession code for the datasets is ERA123494. The merged dataset, including all four datasets, contains ~122 million reads with ~9x depth coverage.

B2. Experiments on real NGS human data

We implemented compressively-accelerated versions of BWA aln and Bowtie2 (denoted as CORA-BWA and CORA-Bowtie2). We chose BWA and Bowtie2 for our experiments because they are widely used and also the primary means by which many biotechnology labs map large NGS read datasets. We also perform runtime and sensitivity comparisons to other mapping tools, including mrsFAST-Ultra, BWA mem, GEM[30], and Masai. All mapping experiments are run on a single CPU of a 12-CPU Intel Xeon X5690 machine with 94GiB RAM.

To determine whether compression yields acceleration, we compared the read alignment performance of our CORA-based compressively-accelerated mappers against other methods using 1000 Genomes sequencing datasets from multiple individuals. These datasets consist of paired-end reads (108 bp) from the 1000 Genomes Phase 1 Illumina sequencing read data (see **Datasets**). Three of these datasets contain respectively 1, 2, and 4 different Finnish individuals with roughly 4x, 8x and 16x depth-coverage. We mapped these three read datasets onto the whole human reference genome (hg19) with four different alignment strategies: all-mapping with indels (Levenshtein distance), all-mapping without indels (Hamming distance), best-mapping with indels, and best-mapping without indels. We tested eight read mapping methods (BWA, CORA-BWA, Bowtie2, and CORA-Bowtie2, BWA mem, mrsFAST-Ultra, GEM and Masai) and measured their runtime and sensitivity. While we excluded Masai and GEM from the all-mapping experiments in the main text due to their very low sensitivity (~10%) as compared to other tools, we do however provide a detailed analysis of their all-mapping performance in **Additional experimental results**.

Whereas all CORA runtimes are reported on the full datasets, some of the other runtime and sensitivity results were estimated using downsampled read sets (see **Further details on experimental Setup**).

All-mapping performance comparison with existing methods. CORA's compressive mapping framework achieved from 6 times to 3 orders of magnitude speed-up compared to existing all-mappers with comparable sensitivity (**Figure 1**).

For the gapped alignment of the FIN1 dataset with roughly 4x coverage, we estimated that BWA aln would take more than 21 days to find paired-end all-mappings within a Levenshtein distance of 4 for each end, whereas CORA-BWA took less than 10 hours, more than 54 times faster than the original BWA mapper. For the FIN4 dataset, CORA-BWA was more than 62 times faster than BWA aln, indicating that CORA's compressive acceleration increases with higher read-depth coverage. For the ungapped mapping of FIN4, CORA-BWA was still faster by an order of magnitude compared to BWA. CORA-BWA achieved these massive accelerations of BWA while also substantially improving the sensitivity of gapped and ungapped all-mapping (**Figure 1b**).

For the gapped alignment of the FIN1 dataset, we estimated that finding all-mappings using Bowtie2 would require several years to complete, whereas our compressively-accelerated version of Bowtie2 was able to complete within 10 hours, effectively boosting Bowtie2's all-mapping efficiency by three orders of magnitude, while again improving sensitivity.

For ungapped all-mapping on the FIN4 dataset CORA-BWA achieved ~6 times the speed up as compared to mrsFAST-Ultra, with near-perfect sensitivity. CORA-BWA reported mappings ~3.5 times faster even when mrsFAST-Ultra printed read mappings in an unordered fashion, creating computational debt for the downstream analysis. Furthermore, CORA's acceleration as compared to mrsFAST-Ultra is significantly greater for higher redundancy reference genomes (e.g. mouse), effectively producing all-mapping results about an order of magnitude faster (see **Experiments on mouse data**). CORA's superior all-mapping speed as compared to mrsFAST-Ultra is remarkable, given that mrsFAST-Ultra is cache-optimized for all-mapping and the current implementation of CORA-framework does not employ any machine architectural optimizations.

Sensitivity results for BWA, Bowtie2, mrsFAST-Ultra, CORA-BWA, and CORA-Bowtie2 are given in **Figure 1b** for the all-mapping alignment strategy. For ungapped all-mapping, sensitivities are computed with respect to the complete all-mapping datasets accepted as ground-truth, generated by exhaustive search[5]. CORA-based versions of BWA and Bowtie2 achieved almost perfect sensitivity, whereas the original mappers exhibited a significant loss in sensitivity. For gapped all-mapping, as no method could be used as a perfect sensitivity ground truth, we compared the number of gapped mapping results different mappers reported. Both CORA-BWA and CORA-Bowtie2 have significantly higher sensitivity than BWA and Bowtie2's gapped all-mappers. This improvement is due to CORA's high-resolution homology table, particularly its representation of all reference-related homologies relevant to read-mapping, as well as its recovery scheme for missing 'read links' (**Supplementary Methods**).

We did not include Masai and GEM mappers in our main all-mapping comparisons as they reported all-mapping results with extremely low sensitivity (~10%)

on the FIN4 dataset. See **Additional experimental results** for a detailed comparative analysis of GEM and Masai's gapped and ungapped all-mapping performance with CORA-BWA.

Best-mapping performance comparison with existing methods. While relatively modest compared to its acceleration of all-mapping, CORA still achieved substantial best-mapping performance improvements compared to existing state-of-the-art best-mappers for both gapped and ungapped mapping in terms of speed, sensitivity or both (**Figure S6**).

Remarkably even CORA's all-mappers **(Figure 1a)** reported mapping results faster than BWA, Bowtie2 and mrsFAST-Ultra's best-mappers. This is a considerable strength of the CORA framework, as it enables NGS analysis pipelines based on all-mapping that are even faster than existing pipelines based on best-mapping.

Runtime and sensitivity analysis for varied k-mer length and error rate. We performed additional experiments on real NGS data in order to demonstrate mapping runtime and sensitivity performance of CORA for varied k-mer lengths and error rates.

We aligned chromosome 20 reads for 32 Finnish individuals from 1000 Genomes Project onto hg19 chromosome 20 with a Hamming distance of 2, 4, and 6. The paired-end read lengths used were 108bp, 100bp, 90bp, 80bp and 70bp. For these error rates and read lengths, we constructed homology tables with the corresponding k-mer lengths of 54bp, 50bp, 45bp, 40bp and 35bp, with a Hamming distance threshold of 1, 2, and 3, respectively. **Table S4** shows the runtime of these experiments in seconds.

		2 x 70bp	2 x 80bp	2x 90bp	2 x 100bp	2 x 108bp
Hamming 2	Full runtime	779.02	843.40	905.61	954.09	1101.18
	Coarse mapping	237.10	314.26	369.37	462.74	560.19
Hamming 4	Full runtime	1977.11	1863.28	1775.27	1920.49	1895.81
	Coarse mapping	994.67	1065.52	1095.38	1298.04	1390.94
Hamming 6	Full runtime	3698.55	3395.73	3391.54	3541.02	3471.02
	Coarse mapping	1306.36	1795.29	2305.34	2597.97	2799.05

Table S4: All-mapping and coarse mapping runtimes of CORA-BWA when mapping paired-end 70-108bp read-length NGS datasets onto hg19 chromosome 20 within a Hamming distance of 2, 4, and 6.

We also estimated the mapping sensitivity for a subset of the experiments above, for the paired-end 90bp, 100bp and 108bp read datasets across Hamming distance thresholds of 2, 4 and 6 (**Table S5**).

Homology table distance threshold	**Search distance threshold**	**2 x 90bp**	**2 x 100bp**	**2 x 108bp**
Hamming 2	Hamming 2	99.523%	99.433%	99.289%
Hamming 4	Hamming 2	99.996%	99.996%	99.997%
	Hamming 4	99.506%	99.380%	99.218%
Hamming 6	Hamming 2	99.999%	99.999%	99.999%
	Hamming 4	99.910%	99.857%	99.791%
	Hamming 6	99.421%	99.250%	99.024%

Table S5: All-mapping percentage sensitivity results of CORA-BWA when mapping paired-end 90-108bp read-length read datasets onto hg19 chromosome 20 within a Hamming distance of 2, 4, and 6. For mapping experiments with Hamming distance 4 and 6, we also measured the percentage sensitivity of

CORA when recovering mappings with a Hamming distance of 2. For experiments with Hamming distance 6, we also looked at the sensitivity of mapping with Hamming distance 4.

For the experiments in which the homology table distance thresholds were twice or more than the search distance threshold, CORA performed mapping with almost 100% sensitivity. This observation is in accordance with the mathematical proof given earlier. In practice, a small sensitivity loss of <0.004% is observed due to potential misses in the coarse mapping stage using BWA aln.

Recovery scheme analysis for varied k-mer length and error rate. We also ran CORA without the recovery scheme for the original 108bp paired-end read dataset in order to analyze the tradeoff between runtime and sensitivity in comparison to the runs above with the recovery scheme enabled (**Table S6**).

Homology table distance threshold	**Performance measurement**	**2 x 108bp**
Hamming 2	Hamming 2 search sensitivity	96.323%
	Full runtime (seconds)	1082.37
Hamming 4	Hamming 2 search sensitivity	99.930%
	Hamming 4 search sensitivity	89.485%
	Full runtime (seconds)	1854.053
Hamming 6	Hamming 2 search sensitivity	99.982%
	Hamming 4 search sensitivity	97.922%
	Hamming 6 search sensitivity	84.123%
	Full runtime (seconds)	3410.816

Table S6: Runtime and sensitivity results of all-mapping with CORA-BWA when mapping a paired-end 2x108bp NGS read dataset onto hg19 chromosome 20 when recovery scheme is disabled for Hamming distance thresholds of 2, 4 and 6.

Performance results indicate that the recovery scheme becomes more important as the homology table distance threshold increases, especially if the search distance threshold is larger than half the homology table distance threshold. The runtime was nominally improved when the recovery scheme was disabled.

B3. Experiments on simulated human data

For the runtime results that demonstrate CORA's enhanced performance with improvements in quality of sequencing (**Figure 1c**), we simulated 20 million paired-end reads (2x100bp) from chromosome 20 of the human reference genome (hg19) with a fixed mutation rate of 0.1% and varying sequencing error rates of 2%, 1%, 0.5%, 0.25%, and 0.125%. We used SAMtools wgsim tool for simulating paired-end reads.

For sequencing read simulation, we employed SAMtools version 0.1.19 with default insert size and distribution parameters –d 500 –s 50, as well as –R 0 –X 0 parameters in order to simulate read errors within the Hamming distance metric.

To demonstrate CORA's sublinear coarse mapping scaling property, we further simulated four datasets with one billion paired-end reads from chromosome 20 with 2%, 0.5%, 0.125%, and 0% sequencing error rates (denoted as E2, E05, E0125, and E0 datasets).

Even though any read-mapping method must scale at least linearly with the number of lines in the input dataset as well as the output, between the input reading and output writing stages, the CORA framework achieves sublinear scaling for costly sequence similarity computations. In fact, in the absence of sequencing errors, the total number of read sequences that are processed in the coarse mapping stage is in effect constant above 500x read depth-coverage (**Supplementary Figure S6**). Notably, the rate of redundancy in read k-mers monotonically increases even in the presence of sequencing errors (**Supplementary Figure S7**).

B4. Experiments on mouse data

We benchmarked CORA-BWA in comparison to mrsFAST-Ultra and BWA aln/mem for substitution-only all-mapping and best-mapping for mouse read datasets using a single CPU of a 20-CPU Intel Xeon E5-2650 (2.3GHz) machine with 384GB 2133MHz RAM.

For all-mapping, we ran CORA-BWA on the full read dataset, whereas we ran mrsFAST-Ultra on a 1/10 downsampled dataset due to prohibitive runtime. All-mapping sensitivity comparisons were measured on the downsampled dataset, based on the assumption that mrsFAST-Ultra produces mappings with perfect sensitivity.

For best-mapping, we ran CORA-BWA on the full dataset with both fast best-mapping mode, which uses only the exact homology table, and original best-mapping mode, which makes use of both exact and inexact homology, whereas we ran BWA aln and BWA mem on the 1/10 subsampled dataset.

The mouse read dataset consisted of 4 individuals with 100bp paired-end read datasets with a total of 8.96x depth coverage. We mapped these read datasets onto the mm9 mouse reference genome within a Hamming distance of 4 for each end and with an allowed insert size interval of 150-650bp. CORA's homology table of mm9 reference genome was constructed with a k-mer length of 50bp and a Hamming distance threshold of 2. As the homology table is precomputed, its runtime requirement is not included in our comparisons.

When scaled to the whole read dataset from the 1/10 downsampled runs, mrsFAST-Ultra's runtime corresponds to 63.4 hours for mapping and 72.2 hours for sorting the dataset; 135.6 hours for the full mapping pipeline for the all-mapping reads to be used for downstream analysis. Scaling for mapping and sorting components were performed separately as described in **B6**. In terms of sensitivity, we assumed mrsFAST-Ultra's ungapped all-mapping sensitivity to be 100%. In comparison, CORA-BWA performed all-mapping on the full dataset in 14.8 hours (readily in read sorted order), capturing 99.64% of the mappings mrsFAST-Ultra produced. This result indicates that, CORA can produce read sorted all-mapping results ~9.2x faster than the full mrsFAST-Ultra pipeline with near-perfect sensitivity (~4.3x faster if sorting is not included).

For the 1/10 downsampled run, BWA aln completed reporting substitution-only best-mapping results in 181 minutes and 55 seconds (BWA mem took 348 minutes and 6 seconds), corresponding to 30.3 hours when scaled to the full run. In comparison, CORA completed performing best-mapping on the full dataset in 12.7 hours for the original best mapping mode and 6.78 hours for its fast best-mapping mode. BWA aln reported best-mapping results with ~95.02% sensitivity in the downsampled read dataset, whereas CORA-BWA's original best-mapping mode reported mappings with 99.41% sensitivity for the same set of reads, ~2.4x times faster than BWA aln. CORA's fast best-mapping mode, which does not make use of the inexact homology table, reported mapping results with 92.5% sensitivity, ~4.47x times faster than BWA aln. In comparison, BWA mem's sensitivity was much lower at ~83.1%.

B5. Additional experimental results

In the main experimental results provided for real NGS datasets, we compared CORA mappers with the state-of-the-art mappers in terms of runtime and sensitivity. A number of results were excluded from the main all-mapping results for fairness of comparison, as they reported very low sensitivity mapping results compared to other tools included and also consistently crashed on the datasets we tested other tools on. We also excluded tools that did not have a command-line option for the type of mapping performed. In this section we list such results or experiments we excluded from the main set of results.

First, mrsFAST-Ultra was excluded from gapped mapping experiments as it is a substitution-only mapping that cannot detect read mappings with insertions and deletions. Similarly, BWA-mem was excluded from all-mapping experiments as it does not have an all-mapping mode. Moreover, GEM was excluded from ungapped mapping experiments as it does not have a user option for limiting non-mismatch edits.

All-mapping with GEM and Masai. GEM and Masai's all-mapping results were excluded from the main result plots, both due to the fact that they consistently crashed on the 1000 genomes datasets used in our main experimental results, making benchmarking challenging, and their all-mapping sensitivity was substantially lower than comparative tools in the maximum sensitivity all-mapping mode; thus a fair head-to-head comparison was not feasible at a practically useful sensitivity level. We provide detailed results here for both Masai and GEM's all-mapping performance and compare them to CORA's all-mapping results.

For our benchmarking experiments with Masai, we used version 0.7.1 for Linux x86_64. As precomputation, we indexed the hg19 reference genome using Masai's default indexing algorithm. We used the command line arguments '–mm all' for all mapping, '-e 4' for mapping with distance 4 for each end, and '-ll 400 –le 250' which determines the valid paired-end insert size interval as [150,650]. For the ungapped (substitution-only) experiments, we declared the flag '-ng'; for gapped mapping experiments we did not declare this flag. The remainder of the parameters were chosen by default.

Similar to BWA, Masai requires three separate executions for paired-end mapping: single-end mapping for each end and a third execution for inferring paired-end mappings from the two intermediate files. Masai crashed during each of these 3 stages on the full FIN4 dataset, for both substitution-only and gapped mapping runs, throwing 'std::bad_alloc' errors. While the last stage crashed after saturating the entire 96GB of memory, the first two stages crashed after using ~60 GB of memory.

For this reason, we uniformly downsampled the FIN4 dataset in order to gauge Masai's all-mapping speed and sensitivity. When we ran it on a 1/100 downsampled dataset for ungapped (substitution-only) all-mapping, Masai was able to successfully report mapping results in 33 minutes and 51 seconds, corresponding to 56.4 hours for the full run. The percentage sensitivity of Masai for the downsampled set was 10.1%. In comparison, CORA-BWA takes 14.4 hours to run on the full dataset while producing mappings with 99.7% sensitivity. For ungapped mapping, Masai produces ~10% of the ungapped mappings that CORA-BWA reports while taking ~3.9x times the time CORA takes to report the mappings.

For gapped all-mapping on the 1/100 downsampled dataset, Masai produced mapping results in 67 minutes and 27 seconds, which corresponds to 114 hours for the full run. Strikingly, Masai's sensitivity at detecting alignments was substantially lower than other all-mappers we benchmarked in that it produced 4.57 million valid mappings for the 1/100 dataset (with the criteria of valid gapped mappings defined earlier in *B2*). In comparison, CORA-BWA takes 30.95 hours for mapping the full FIN4 dataset and produces 45 million valid mappings for the same set of reads included in the subsampled Masai run. Overall, Masai produces 10.1% of the mappings CORA-BWA produces while taking ~3.68x times the time to compute the mappings.

For our benchmarking experiments with GEM, we used GEM-mapper build 1.376. As precomputation, we indexed the hg19 reference genome using GEM's default indexing algorithm. We consecutively ran gem-mapping and gem-2-sam algorithm to create a pipeline that takes in FASTQ read input and reports SAM mapping output. We used command line arguments '-E 4' for setting the edit distance threshold to 4 for each end, '-b' to align both ends, and '-s 4' for the mapper to recover all possible mapping edit distance strata that cover 4 edits. We also defined '--min-insert-size 150' and '--max-insert-size 650'. Our full FIN4 data runs as well as subsampled runs consistently crashed using the GEM mapper, reporting a 'wrong alignment' crash error after processing ~1% of the reads. Fortunately, the GEM mapper reports mapping results for each read as it computes mapping loci, so it is possible to give a rough estimate of its sensitivity and runtime on the full dataset. Our estimated runtime of the GEM mapper on the full FIN4 dataset is 81.7913 hours. The number of valid gapped mappings GEM produces for the processed reads is 3.385 million. In comparison, CORA-BWA produces 37.185 million valid gapped mappings for the same dataset. Overall, GEM produces 9.1% of the mappings that CORA-BWA reports while taking ~2.64x times more time.

Note that for seed-and-extend read mapping algorithms, improving sensitivity affects runtime superlinearly[31], as increasing sensitivity requires sampling shorter seeds from the read with drastically higher chance occurrences. Thus, we can suppose that a more sensitive Masai algorithm, using similar data structures but performing a deeper index search with shorter seeds, would be at least ~36x slower than CORA-BWA for a similar level of sensitivity; a more sensitive GEM algorithm would be at least ~29x slower for gapped all-mapping. Moreover, it is likely these tools would be substantially slower.

Oculus. Results from Oculus, a mapping acceleration tool that wraps off-the-shelf aligners and leverages compression of fully identical reads, were excluded from the main results. This decision was due to the fact that Oculus' requirement that paired-end reads be fully identical to be compressed prevents any meaningful acceleration to be gained for the datasets we presented. We tested Oculus on the FIN1 dataset with BWA aln. The total runtime was 31.4 hours with 16 minutes spent within Oculus. The estimated saved time reported by Oculus was 37.8 minutes, which corresponds to a <2% runtime improvement. Note that the number of duplicate reads in the dataset was 3%, which represents a hard upper bound on the rate of acceleration that can be achieved by Oculus for any mapping tool or mode.

B6. Further details on experimental setup

We compared mapping speed and sensitivity of BWA, Bowtie2, compressively accelerated BWA (CORA-BWA) and compressively accelerated Bowtie2 (CORA-Bowtie2) for the all-mapping and best-mapping alignment strategies. For our comparisons, we used Bowtie2 version 2.1.0, BWA version 0.7.5a, mrsFAST-Ultra version 3.3, GEM version 1.376, Masai version 0.7.1 and Oculus version 0.1.2.

Mapping criteria and evaluation of sensitivity. For mapping benchmarks presented, we used the Hamming distance (substitution distance) threshold of 4 for ungapped and Levenshtein distance (edit distance) threshold of 4 for gapped. We required both mates of the paired-end reads to be aligned end-to-end with an insert size interval between 150 and 650 base pairs, which specifies the distance from the beginning position of the first mate in the reference to the second's (i.e. the alignment gap length between mates + single read length).

We specified this alignment criteria for both best-mapping and all-mapping, so that we can compare CORA's all-mapping performance not only with other all-mapping methods but also the best-mapping performance of existing tools.

For the all-mapping benchmarks, we measured each mapper's ability to report all of the mappings that satisfy the criteria given above. For example, any paired-end read alignment with an insert size between 150bp and 650bp and with 4 substitutions for each end is a valid alignment to be reported for ungapped all-mapping (or x insertions, y deletions and z substitutions for gapped all-mapping, such that x+y+z = 4).

Note that in our gapped mapping experiments the penalty for each mismatch and each base of a deletion/insertion is chosen to be equal, as specified by the Levenshtein distance metric, which is both the most basic gapped distance metric and the only one uniformly supported by the mappers we tested. In the cases where the default gapped mapping mode of a mapper did not exactly correspond to Levenshtein distance metric, we specified a gap opening penalty of 0 and a gap extension penalty equal to the mismatch penalty for the mappers to ensure a consistent comparison across different methods. While this is a fair comparison benchmark that evaluates a desired property of mappers –the ability to sensitively detect high-quality mappings of reads within a specified distance threshold– it does not evaluate different mappers' ability to detect mappings outside of the specified distance threshold (mappings containing more mismatches or covering indels longer than the threshold value). In order to have an estimate of what percentage of variants this would exclude from our analysis, we recomputed the NA12878 genotyping sensitivities presented in Yu et al.[33] for both GATK and SAMtools pipelines (GATK 'best-practices' bundle used as gold-standard) [6,28]. When genotyping performance of BWA's default mapping results was compared to the subset of these mappings within Levenshtein distance of 4—our gapped mapping benchmark criteria—we see that 2.7% of the SNPs as well as 7.5% of the indels in the gold-standard change from true-positive calls in the former to false-negative calls in the latter. However, the decrease in sensitivity caused by excluding mappings outside of Levenshtein distance of 4 also result in a substantial increase in precision: False discovery rate within the top 2.5 million SNP calls is 24.6% higher in the full mapping dataset compared to the filtered—in the case of top 150 thousand indels, this increase in FDR is 16.1%. In other words, by selecting a Levenshtein distance threshold of 4, our benchmarks restrict the solution space to a smaller yet higher quality set of mappings that result in higher precision but lower recall variant calls.

Sensitivity for ungapped all-mapping results is calculated as the percentage of these valid paired-end read alignments that are correctly reported by the mapping method. Note that sensitivity for all-mapping is equivalent to the accuracy, as read mapping tools in general (including CORA-accelerated versions) do not falsely report a mapping within the specified similarity threshold. For ungapped all-mapping sensitivity benchmarks, mapping results of mrsFAST-Ultra is accepted as ground truth as it can perform ungapped mapping with perfect sensitivity.

Sensitivity for gapped all-mapping results is not calculated as percentage results as none of the mappers we tested were able to give all-mapping results with perfect sensitivity for a gapped alignment strategy. For that reason, we compared the number of valid non-redundant mappings that each gapped mapper was able to map within the specified insert length interval and error distance. We define a mapping to be redundant if there is another mapping for the same read at the same loci for both mates with a potentially different traversal of the dynamic programming matrix for either end. For example, if there is a read mapping with no errors at position x, any other mapping for the same read at position x is considered redundant. Moreover, mappings for the same read that start at position x+k with k inserted bases (or k silenced bases) at the beginning of the read are also considered redundant. Furthermore, if an insertion or deletion sequence spans a micro-repeat region in the read or the reference, indels spanning different instances of the repeat are all considered redundant apart from one of them with a minimal error distance. Since there can be many non-trivial configurations of these redundant mapping variants as well as their combinations, we employed a simpler scheme to obtain a non-redundant all-mapping set.

1) Each reported paired-end mapping was merged into a single line in the SAM output.
2) Merged SAM output was sorted by read name, then chromosome, then first mate mapping position and finally second mate mapping position in the same chromosome.
3) Starting from the second line of the sorted file, if the previous line's read name and chromosome are identical with the current line, the positions of the first mates are at most d apart between the two lines, and the positions of the second mates are also at most d apart (d being the Levenshtein distance threshold of mapping), the mapping in the current line is considered redundant and eliminated.

Though this simple elimination scheme does not necessarily pick the lowest distance mapping, it is adequate for measuring the total number of non-redundant all-mappings. In very few cases, this elimination scheme can cause false elimination of non-redundant mappings. However, we estimated this effect to be negligible (<0.001%).

The ungapped best-mapping read alignment scheme is defined as reporting only one of the valid paired-end mappings with the lowest cumulative Hamming distance with respect to both ends. In the case of multiple valid paired-end mappings with the lowest Hamming distance, any one of the mappings with the lowest distance can be selected as a valid best-mapping output. Sensitivity for ungapped best-mapping results is calculated as the percentage of reads that have at least one valid mapping to the reference, and one of the lowest Hamming distance mappings is reported by the alignment method. For ungapped best-mapping sensitivity benchmarks, the lowest

Hamming distance tier of mrsFAST-Ultra's mapping results is accepted as ground truth as it can perform ungapped mapping with perfect sensitivity.

Sensitivity for gapped best-mapping results is not calculated as percentage results as none of the mappers we tested were able to give best-mapping results with perfect sensitivity for a gapped alignment strategy. For this reason, sensitivity results are reported as the total number of reads that the mappers reported a valid paired-end mapping for within the specified Levenshtein distance threshold and insert size interval.

In the specific case of BWA mem—as it does not allow the user to specify an insert size interval—we measured the sensitivity for two sets of criteria: (1) using the default criteria described above and (2) including additionally mappings BWA mem reported outside of the insert size interval so long as other tools also found a mapping for the same read within the insert size interval. The reason for this two-tiered sensitivity analysis is that, while our original benchmark provides a head-to-head comparison of all tools according to the same mapping criteria it also indirectly penalizes BWA mem's lack of a user option for insert size intervals. As a way to ameliorate this situation, our second sensitivity analysis aims to be more generous to BWA mem by incorporating its mappings outside of the insert size interval to the analysis as well, while still being fair to other mappers by not incorporating mappings where there are no mappings for the same read within the paired-end insert size interval (please see **Experimental setup for best-mapping benchmarks** for details).

Experimental setup for all-mapping benchmarks.

The all-mapping scheme for BWA is employed as follows:

0) Reference genome is indexed using default parameters of BWA.

1) First mate of the paired-end read is aligned with the “bwa aln” program using parameters “–n 4” (specifying the maximum Hamming/Levenshtein distance for each valid read alignment as 4), and “-N” for identification of all possible valid alignments for each mate. For ungapped mapping, “–o 0” was specified for preventing gap openings and indels. For gapped mapping “-d 0 –i 0” was defined in order to sensitively detect all indel mappings for the entire read.

2) Second mate is aligned using the same command and parameters.

3) Paired-end alignments are found with the “bwa sampe” program using parameters “-s” for disabling Smith-Waterman for the unmapped mate in order to restrict the reported mappings to a pre-defined Hamming or Levenshtein distance threshold for both mates, “-c 0” for eliminating chimeric read mappings, “-a 650” for reporting paired-end read alignments with at most 650 base pairs insert size (bwa sampe does not support a lower bound for insert size), and “-n 1M -N 1M –o 100M” for enabling maximum possible sensitivity of read mappings (M stands for one million).

In addition to these three stages, as BWA sampe reports only a single paired-end mate in its output and the remaining read mappings are reported as single-end mappings without any pair information, we implemented a paired-end read mapping extraction tool for BWA that efficiently parses the locations reported by sampe in XA field for each end, sorts each list of single-end mappings and performs a linear scan to report all pairs of locations with matching chromosomes and with mates within the user-specified insert size interval. In our reported runtimes, we indicated the cost of this paired-end mapping extraction tool with the lighter color on top of the BWA bars. We did not include its costs for indexing the genome. For downsampled runs, we extrapolated the full runtime of BWA assuming linear scaling with the read dataset size.

The all-mapping scheme for Bowtie2 is employed as follows:

0) Reference genome is indexed using default parameters of Bowtie2.
1) Bowtie2 aligner is run to perform paired-end alignment of the read dataset onto the reference genome using parameters "--no-mixed" in order to suppress alignments that are not paired-end, "-a --end-to-end --ignore-quals" in order to find end-to-end mappings of each mate using the Hamming or Levenshtein distance metric and report all paired-end mappings, "--np 6 --mp 6 --score-min L,0,-0.25" for limiting the maximum number of allowed number of substitutions/insertions/deletions to only 4 bases for each 108bp long mate, and "--minins 150 --maxins 650" specifying the insert size interval. For ungapped mapping, "--rdg 1000,1000 --rfg 1000,1000" was defined in order to eliminate all indels. For gapped mapping "--rdg 0,6 --rfg 0,6" was defined to set equal weight between indels and substitutions corresponding to Levenshtein distance.

In our reported runtimes for Bowtie2, we have not included its cost for indexing the genome. For downsampled runs, we extrapolated the full runtime assuming linear scaling with the read dataset size.

The ungapped all-mapping scheme for mrsFAST-Ultra is employed as follows:

0) Reference genome is indexed using default parameters of mrsFAST-Ultra.

1) Parameters ‘-e 4 --min 150 --max 650’ are specified for the insert length interval for the paired-end mapping as well as Hamming distance of 4 for each read end.

mrsFAST-Ultra reports all-mappings in an unsorted order (w.r.t. read or chromosome order) whereas all other tools we benchmarked, including CORA, report read-ordered mappings. Printing all-mapping results in an unsorted order creates computational debt for the downstream analysis tools, as they have to directly or indirectly perform a sorting task in order to identify all of the mappings of each read. We sorted mrsFAST-Ultra’s output by read names using linux sort. In order to perform efficient sorting we used the local temporary directory and declared LC_ALL=C environment variable, which dramatically improves performance for sorting files with ASCII characters. For downsampled runs, we extrapolated the mapping portion of mrsFAST-Ultra’s runtime linearly; the sorting portion of its runtime has complexity $\Theta((N/M)\log_R(N/M))$ for the R-Way merge sort algorithm that linux sort implements, where N and M correspond to data and memory size, respectively, and R is the number of files the merge operation is performed on simultaneously. As we did not measure the constants in the complexity function, including R, M or the constant factor before the complexity function, we estimated the cost of sorting for two separate file size as two independent data points for extrapolation. These two files were 1/10 downsampled and 1/20 downsampled mapping output files, both of which are larger than the RAM size of the machine. If X is the cost of sorting the 1/20 downsampled file and Y is the cost of sorting the 1/10 downsampled file, the cost of sorting the full file is estimated to take $20X + 10\ (Y-2X)\ \log_2 20$, which is independent from R, M or the constant factor before the complexity function. In the main figure plots we indicated the cost of this sorting process with the lighter color on top of the mrsFAST-Ultra runtime bars.

Note that this sorting operation merely positions the all-mappings of each read to be adjacent in the file and does not preserve the original order of the read names or perform any sorting on the chromosome or the positions. Ideally a mapper should also preserve the order of the original reads as well as print the mappings of each read in proper order (with respect to chromosomes and positions); however, we assume resolving these will not incur significant computational costs for downstream analyses. Furthermore, we ignore any potential super-linear disk-operation costs that might arise from disk inefficiencies due to reading/writing larger files on disk. We also attempted to convert the sam file into bam format first and sort using samtools, but obtained a larger runtime cost even without extracting the sorted bam file.

Details of the all-mapping scheme employed for GEM and Masai experiments are given in **B5**.

For compressively-accelerated versions of BWA and Bowtie2, we generated the hg19 homology table using 54bp k-mers. We specified that compression is performed using 54bp k-mers (corresponding to half of each mate) and coarse mapping, using BWA aln and Bowtie2. We specified the homology table traversal stage to output all mappings within the Hamming distance limit of 4 for substitution-only mapping and the Levenshtein (edit) distance limit of 4 for gapped mapping. In reported runtimes, we have not included the preprocessing cost for homology table creation, nor the reference indexing costs incurred by BWA aln and Bowtie2. The runtime cost of k-mer based read compression is included in the total runtimes reported.

While all of the CORA all-mapping runtimes were measured from full dataset runs, some of the other tools were estimated using a downsampled read set. Downsampling was performed uniformly and consistently throughout the experiments: All 1/10 downsampled read sets are identical for the same dataset across different mapping experiments and the 1/10 downsampled read set is a superset of the 1/100 downsampled read set and so on.

Both Bowtie2's gapped and ungapped runtimes were estimated using a 1/1000 downsampled read set due to prohibitive runtime cost. BWA's gapped runtimes were estimated using 1/100 downsampled read sets. The number of mappings for the gapped all-mappers was estimated from the same 1/1000 downsampled read set for FIN4.

The runtime of mrsFAST-Ultra mapping and sorting was measured using a 1/10 downsampled read set. Sensitivity percentages and runtimes for BWA's ungapped all-mapping were measured from a 1/10 downsampled read set, whereas Bowtie2's ungapped all-mapping was measured from a 1/1000 downsampled read set. CORA mappers' ungapped sensitivity percentages were measured from the full read set.

Experimental setup for best-mapping benchmarks.

The best-mapping scheme for BWA aln is employed as follows:

0) Reference genome is indexed using default parameters of BWA.
1) First mate of the paired-end read is aligned with "bwa aln" program using parameters "–n 4" (specifying the maximum Hamming distance for each valid read alignment as 4). For ungapped mapping strategy "–o 0" was used in

order to prevent gap openings and indels. For gapped mapping we defined “-d 0 –i 0”.

2) Second mate is aligned using the same command line parameters as the first mate.
3) Paired-end alignments are found with “bwa sampe” program using parameters “-s” for disabling Smith-Waterman for the unmapped mate in order to restrict the reported mappings to pre-defined Hamming distance threshold for both mates, “-a 650” for reporting paired-end read alignments with at most 650 base pairs insert size (sampe does not support lower bound for insert size), and “–n 0 –N 0” in order to report only one paired-end mapping per read.

The best mapping scheme for BWA mem is employed as follows:

0) Reference genome is indexed using default parameters of BWA.
1) BWA mem aligner is run with parameters “-A 1 -B 1 -O 0 -E 1 -L 1 -U 1000 -T 100” for gapped mapping, which assigns a penalty of 1 for each edit operation, a score of 1 for each match operation, and a threshold score of 100 corresponding to Levenshtein distance of 4. For ungapped mapping, “-A 1 -B 1 -O 1000 -E 1000 -L 1000 -U 1000 -T 100” is defined in order to prevent any insertions and deletions and set the distance threshold as Hamming distance of 4. For both alignment strategies, mate rescuing is disabled by specifying “-S” which results in a major increase in speed with negligible loss in sensitivity, “-t 1” is defined to run BWA mem on a single processor, and all other parameters were selected as default.

As BWA mem does not provide a user option for specifying paired-end insert size interval, we measured and reported two levels of sensitivity for it (**Figure S2**). BWA mem's base sensitivity—which is lower—was measured using the default criteria employed to evaluate the other best mappers, whereas a second sensitivity measure—more tolerant for mappings outside the specified insert size interval—was computed by also taking into account BWA mem's mappings outside of the [150,650] insert size interval so long as the edit distance of the mapping reported—summation of both mates' edit distances—was within the distance threshold of the original benchmarks and other tools also found a mapping for the same read within the [150,650] insert size interval. In the case of ungapped mapping, we used the set of reads with valid mrsFAST-Ultra mappings for this comparison since mrsFAST-Ultra reports ungapped mappings with perfect sensitivity. In the case of gapped mapping, the union of the set of reads with valid BWA aln, Bowtie2 or mrsFAST-Ultra mappings were used.

The best-mapping scheme for Bowtie2 is employed as follows:

0) Reference genome is indexed using default parameters of Bowtie 2
1) Bowtie2 aligner is run to perform paired-end alignment of the read dataset onto the reference genome using parameters "--no-mixed" in order to suppress alignments that are not paired-end, "--end-to-end --ignore-quals" in order to perform end-to-end mapping of each mate using Hamming or Levenshtein distance metric for mapping similarity, "--np 6 --mp 6 --score-min L,0,-0.25" for limiting the maximum number of allowed number of substitutions to only 4 base-pairs for each 108bp long mate, "--minins 150 --maxins 650" specifying the insert size interval. For ungapped mapping "--rdg 1000,1000 --

rfg 1000,1000” parameters were used. For gapped mapping, the same parameters were defined as “--rdg 0,6 --rfg 0,6”.

The ungapped best-mapping scheme for mrsFAST-Ultra was performed as follows:

0) Reference genome is indexed using default parameters of mrsFAST-Ultra.
1) Parameters ‘-e 4 --min 150 --max 650 --best’ are specified for best-mapping with the specified insert length interval for the paired-end mapping as well as Hamming distance of 4 for each read end.

As mrsFAST-Ultra prints best-mapping output in read sorted order, there is no sorting cost added for best-mapping.

The best-mapping scheme for Masai was performed as follows:

0) Reference genome is indexed using default parameters of Masai.
1) The FASTQ file containing the first mates are aligned with masai_mapper algorithm was call with ‘-e 4’ parameter that specifies Hamming/Levenshtein distance threshold. In the case of ungapped mapping we additionally specified parameter ‘-ng’ which prevents indels.
2) The FASTQ file containing the second mates is aligned using the same command line parameters as the first FASTQ file.
3) Paired-end alignments are found using masai_output_pe algorithm, with the arguments ‘-ll 400 –le 250’ which specify the insert size interval as [150,650].

For both gapped and ungapped alignments, Masai crashed with a bad_alloc error on the full FIN4 dataset after using 144.2GB memory in stage 3. For this reason, we performed downsampled runs for Masai, extrapolating full runtime.

The gapped best-mapping scheme for GEM was performed as follows:

0) Reference genome is indexed using default parameters of GEM.
1) gem-mapper algorithm was called with the command-line arguments “-E 4 –b” to map both ends within a Levenshtein distance of 4 , “-T 1” for restricting GEM to a single processor, --min-insert-size 150 --max-insert-size 650” in order to specify the insert size interval.
2) gem-2-sam algorithm was called in order to convert the mappings from GEM’s internal format to SAM format.

For best-mapping runs, the CORA-bwa framework is run with the same parameters as the all-mapping runs except for the last stage where the final mappings are inferred and only a single mapping with the lowest Hamming or Levenshtein distance is reported. For ungapped best-mapping, we performed runs with two separate speed levels: fast mode for which only the exact homology table is traversed; the default mode for which both exact and inexact homology tables are traversed. For gapped best-mapping, we also performed runs with two separate speed levels: fast sensitivity mode for which only the exact homology table is traversed and the banded dynamic-programming algorithm is not employed; default mode for which the banded dynamic-programming algorithm is also employed.

All best-mapping runtimes were measured from full dataset runs, apart from GEM and Masai which consistently crashed on the full dataset. We estimated Masai runtimes from a 1/10 uniformly downsampled read.

Similar to the all-mapping runs, best-mapping runs with GEM also crashed early in the read list, throwing a “wrong alignment” error. We removed a portion of the reads that GEM could not process and reran GEM, but it crashed again soon after. For this reason, we extrapolated the full runtime and sensitivity of GEM from only the mapped portion of the dataset until the first crash, which corresponds to ~1% of the full read dataset.

All sensitivity percentages for ungapped best-mapping were estimated from the 1/10 downsampled FIN4 dataset. While all sensitivity results for gapped best-mapping were estimated from the 1/100 downsampled read set apart from GEM. We estimated GEM’s sensitivity on the read set that it mapped and compared its sensitivity to CORA-BWA’s sensitivity on the same set of reads. We assumed this sensitivity ratio to be fixed when we estimated GEM’s sensitivity for the full dataset.

C. Supplementary Discussion

CORA’s relative runtime advantage over existing mappers increases substantially with both read depth-coverage and additional individuals in the dataset because the compressive representation enables much less time to be spent per read and sublinear runtime in practice (**Supplementary Figures S6 and S7**). In particular, if multiple individuals are included within a dataset, CORA’s compressive framework ensures their reads can be processed all at once, as opposed to separately for each individual. In this instance, CORA can reuse mapping computations performed for previous individuals in the dataset in order to avoid redundant calculations. Furthermore, CORA’s runtime will substantially improve as sequencers generate higher-quality reads, as its compressive framework achieves gains inversely related to the sequencing error rate (**Figure 1c**), which decreases as higher-quality sequencing technologies emerge.

The CORA pipeline has been tested with BWA aln, BWA mem, Bowtie2, GEM, Masai and mrsFAST-Ultra without any modification to their code, but can be readily used with other off-the-shelf mappers as well. CORA's speed and accuracy is loosely tied to the performance of the off-the-shelf tool used in its coarse mapping stage. However, CORA is expected to give substantial speed gains for other existing short-read mappers as well. In the future, we also plan to combine mrsFAST-Ultra’s cache-oblivious architecture with the full acceleration capabilities of CORA.

As read mapping is typically the most costly step in NGS analysis pipelines (e.g., GATK [6]), any improvement to existing mappers will immediately accelerate sequence analysis studies on large read datasets. CORA’s faster and more accurate alignments for all-mapping allow for similar improvements in genotyping (repeat region analysis, structural variation and SNP detection, and copy-number variation analysis). Though we demonstrate the capabilities of the CORA framework on genome sequencing data, it can readily be used to accelerate mapping of exome and metagenome sequencing datasets, though larger metagenome references will require additional computational resources. CORA’s underlying compressive acceleration principles can also benefit RNA-seq mapping pipelines.

Though CORA is designed to accelerate mapping paired-end Illumina sequencing reads with relatively low sequencing errors, it is possible to extend the framework for sequencing technologies with higher error rates (e.g. PacBio sequencing), using three different strategies: introducing higher error rate homology blocks in the homology table, approximate compression of k-mers in the read dataset, or sampling shorter and more k-mers per read.

The flexibility of the CORA framework enables it to adopt the functionalities of the coarse mappers that it uses. For example, CORA can perform alignment with indels, provided that the coarse mapper used within CORA is able to report coarse mappings with indels. Other coarse mapper functionalities can also be incorporated into the CORA framework without a large code update to CORA framework, such as SNP-aware alignment algorithm of mrsFAST-Ultra, which reduces reference mapping bias, or dynamic trimming of read ends that allow low quality reads to be trimmed during alignment. Similarly, CORA's indel detection capabilities can be improved by utilizing other mappers' capabilities to capture longer indels in the coarse mapping stage.

Given the sheer quantity and variety of read-mapping tools and their non-basic functionalities, it is intractable to incorporate all of them into the CORA framework, even though the mappers themselves can be incorporated to CORA framework as the coarse mapping stage using the manual coarse mapping mode without any code changes. However, we will also periodically release updates incorporating additional important functionalities to CORA. Full CORA software will also be released with open source license together with developer libraries for incorporation of CORA's k-mer read compression and homology table components to the newly developed methods and pipelines, enabling biomedical developer community to incorporate key functionalities of CORA into their methods as well as their custom functionalities into the CORA pipeline.

As state-of-the-art NGS technologies continue to improve and generate ever-increasing quantities of data, we expect CORA to produce even more substantial acceleration, becoming a key component in revolutionizing how the biomedical community handles sequencing data in the upcoming years.

Supplementary References

24. Yu, Y.W. et al. *Research in Computational Molecular Biology*, 285-399 (2014).

25. Baeza-Yates, R.A. & Perleberg, C.H. *Information Processing Letters*, **59(1)**, 21-27 (1996).

26. Altschul, S.F. et al *Journal of Molecular Biology*, **215(3)**, 403-10 (1990).

27. The 1000 Genomes Project Consortium. *Nature* **491**, 56-65 (2012).

28. Li, H. *et al. Bioinformatics* **25(16)**, 2078-2079 (2009).

29. Keane T.M. et al. *Nature* **477**; 7364;289-94 (2011).

30. Marco-Sola, S., et al. *Nature Methods* **9(12)**, 1185-1188 (2012).

31. Schatz M. *Bioinformatics* **25**(11):1363-1369 (2009).

32. Mahmud P. & Schliep A. *arXiv*, 1404.2872 (2014).

33. Yu, Y.W. et al. *Nature Biotechnology* **33**, 240-243 (2015).