

MULTI-PASS SOFTWARE-ACCELERATED GENOMIC READ MAPPING ENGINE

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application claims the benefit of U.S. Provisional Application No. 63/317,859, filed March 8, 2022, the contents of which are incorporated by reference herein.

BACKGROUND

[0002] In some cases, genomic read mapping describes a method to identify the locus of a gene and the distances between genes. Computers can be used to analyze one or more sets of genomic data and correlate a collection of molecular markers, such as a series of nucleotides, with their respective positions on a given reference genome. In this way, a computer can be used to “map” the collection of molecular markers onto the reference genome.

SUMMARY

[0003] In some implementations, a system for multi-pass accelerated genomic read mapping includes one or more processing stages. Each of the one or more processing stages can include extracting one or more k-mers from a genomic data read and processing those k-mers to determine candidate alignment locations that indicate a location on a reference genome at which to align the genomic data read. In some implementations, a first processing stage includes generating a single filtered k-mer, generating a candidate alignment corresponding to reference genomic data, and evaluating the candidate alignment to determine if the alignment satisfies alignment criteria. If the candidate alignment generated in the first processing stage does not satisfy alignment criteria, a system can execute a second processing stage. If one or more candidate alignment locations generated in the second processing stage do not satisfy alignment criteria, a system can execute a third processing stage. In some implementations, processing stops when a candidate alignment that satisfies alignment criteria is evaluated.

[0004] In some implementations, alignment criteria includes a threshold amount of mismatches between a portion of the genomic data read and a portion of a reference genomic data read. In some implementations, a k-mer seed is defined as a sequence of sequential nucleotides where the number of nucleotides in the sequence for a given k-mer is defined by “k” and the nucleotides (or, more generally, bases) are represented by strings of letters from a defined vocabulary. For example, a given k-mer may represent the sequence “ATGCG” where the symbols: {A, C, G, T} represent the four types of nucleotides present in deoxyribonucleic acid (DNA), namely Adenine, Cytosine, Guanine, and Thymine.

[0005] In some implementations, a genomic data read includes data indicating a sequence of nucleotides. A sequence of nucleotides can include a sequence of symbols, each representing chemical compounds. For example, genomic data read can include symbols A, C, G, and T representing four types of nucleotides present in deoxyribonucleic acid (DNA), namely Adenine, Cytosine, Guanine, and Thymine. In ribonucleic acid (RNA), Thymine is replaced by Uracil (U).

[0006] In some implementations, a genomic signature includes a hash generated using a hash function applied to a k-mer. For example, a system can obtain a k-mer representing the sequence “ATGCG”. The system can apply a hash function on the data of the k-mer. In some implementations, the resulting hash is used as a key to query a hash table.

[0007] In some implementations, predetermined criteria used to select a subset of k-mer seeds includes value matching. For example, a system can perform one or more modulo operations on one or more genomic signatures generated based on a hash function applied to one or more k-mers. The system can compare the results of the one or more modulo operations to a value as part of a predetermined criteria. If the results match values specified in the predetermined criteria, the system can select the subset of k-mer seeds.

[0008] In some implementations, a reference sequence location includes a location of a k-mer within a reference genome. For example, the sequence “ATGCG” can appear starting at the 300th nucleotide in a reference genome. The sequence may appear in

one or more locations within the reference genome. The reference sequence locations for one or more k-mers can be stored in a hash table.

[0009] One innovative aspect of the subject matter described in this specification is embodied in a method that includes obtaining, by one or more computers, a first k-mer seed from a genomic data read; generating, by the one or more computers, a genomic signature based on a first k-mer seed; determining, by the one or more computers, a reference sequence location that matches at least a portion of the first k-mer seed using a hash data structure, where the hash data structure includes N data cells including a first portion storing a predetermined genomic signature and a second portion storing a value that corresponds to a location within a reference genomic sequence that matches at least a portion of the first k-mer seed from which the predetermined genomic signature was derived; determining, by the one or more computers, a number of mismatches based on comparing genomic data of the genomic data read to genomic data of the reference genomic sequence; based on determining the number of mismatches includes one or more mismatches, obtaining, by the one or more computers, a set of k-mer seeds from the genomic data read; and based on the set of k-mer seeds from the genomic data read, selecting, by the one or more computers, an actual alignment for the genomic data read.

[0010] Other implementations of this and other aspects include corresponding systems, apparatus, and computer programs, configured to perform the actions of the methods, encoded on computer storage devices. A system of one or more computers can be so configured by virtue of software, firmware, hardware, or a combination of them installed on the system that in operation cause the system to perform the actions. One or more computer programs can be so configured by virtue of having instructions that, when executed by data processing apparatus, cause the apparatus to perform the actions.

[0011] The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination. For instance, in some implementations obtaining the first k-mer seed from the genomic data read includes obtaining a first k bases of the genomic data read as the first k-mer seed.

[0012] In some implementations, actions include, based on determining the number of mismatches is zero, selecting the first k-mer seed as the actual alignment for the genomic data read.

[0013] In some implementations, actions include generating genomic signatures for each k-mer seed of the set of k-mer seeds; and selecting a subset of the set of k-mer seeds based on the genomic signatures.

[0014] In some implementations, selecting the subset of the set of k-mer seeds based on the genomic signatures includes performing one or more modulo operations including a modulo operation on each genomic signature of the genomic signatures; and selecting the subset of the set of k-mer seeds based on results of the one or more modulo operations and a predetermined criteria.

[0015] In some implementations, actions include determining, by the one or more computers, a reference sequence location for each k-mer seed of the subset that matches at least a portion of the given k-mer seed using the hash data structure; and generating a candidate location list including a reference sequence location for each k-mer seed of the subset.

[0016] In some implementations, actions include sorting the reference sequence locations of the candidate location list according to a number of k-mer seeds paired with the given reference sequence location in the hash data structure; and determining a number of mismatches for each of the reference sequence locations of the sorted candidate location list compared to the reference genomic sequence in the order of the sorted candidate location list.

[0017] In some implementations, actions include determining a number of mismatches for a first candidate location of the reference sequence locations of the sorted candidate location list satisfies a mismatch threshold; and selecting the first candidate location as the actual alignment.

[0018] In some implementations, the mismatch threshold includes a threshold number of mismatching nucleotide values.

[0019] In some implementations, actions include determining a number of mismatches for one or more candidate locations of the reference sequence locations of the sorted candidate location list do not satisfy a mismatch threshold; and based on determining the number of mismatches do not satisfy the mismatch threshold, obtaining a second set of k-mer seeds from the genomic data read.

[0020] In some implementations, actions include generating second genomic signatures for each k-mer seed of the second set of k-mer seeds; performing second one or more modulo operations including a modulo operation on each genomic signature of the second genomic signatures; and selecting a second subset of the second set of k-mer seeds based on results of the second one or more modulo operations and a predetermined criteria.

[0021] In some implementations, actions include determining a reference sequence location for each k-mer seed of the second subset that matches at least a portion of the given k-mer seed using the hash data structure; and generating a second candidate location list including a reference sequence location for each k-mer seed of the second subset.

[0022] In some implementations, actions include sorting the reference sequence locations of the second candidate location list according to a number of k-mer seeds paired with the given reference sequence location in the hash data structure; and determining a number of mismatches for each of the reference sequence locations of the sorted second candidate location list compared to the reference genomic sequence in the order of the sorted second candidate location list.

[0023] In some implementations, actions include determining a number of mismatches for a second candidate location of the reference sequence locations of the sorted second candidate location list satisfies a second mismatch threshold; and selecting the second candidate location as the actual alignment.

[0024] Another innovative aspect of the subject matter described in this specification is embodied in a method that includes obtaining, by one or more computers, a first k-mer seed from a genomic data read; generating, by the one or more computers, a genomic signature based on the first k-mer seed; determining, by the one or more

computers, a reference sequence location that matches at least a portion of the first k-mer seed using a hash data structure, wherein the hash data structure comprises N data cells comprising a first portion storing a predetermined genomic signature and a second portion storing a value that corresponds to a location within a reference genomic sequence that matches at least the portion of the first k-mer seed from which the predetermined genomic signature was derived; determining, by the one or more computers, a number of mismatches based on comparing genomic data of the genomic data read to genomic data of the reference genomic sequence; determining, by the one or more computers, the number of mismatches does not satisfy a mismatch threshold; based on determining the number of mismatches does not satisfy the mismatch threshold, obtaining, by the one or more computers, a second set of k-mer seeds from the genomic data read; and based on the second set of k-mer seeds from the genomic data read, selecting, by the one or more computers, an actual alignment for the genomic data read.

[0025] Other implementations of this and other aspects include corresponding systems, apparatus, and computer programs, configured to perform the actions of the methods, encoded on computer storage devices. A system of one or more computers can be so configured by virtue of software, firmware, hardware, or a combination of them installed on the system that in operation cause the system to perform the actions. One or more computer programs can be so configured by virtue of having instructions that, when executed by data processing apparatus, cause the apparatus to perform the actions.

[0026] The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination. For instance, in some implementations, 2. The method of claim 1, wherein obtaining the first k-mer seed from the genomic data read includes obtaining a first k bases of the genomic data read as the first k-mer seed.

[0027] In some implementations, selecting the actual alignment for the genomic data read based on the second set of k-mer seeds from the genomic data read includes generating genomic signatures for each k-mer seed of the second set of k-mer seeds;

and selecting a subset of the second set of k-mer seeds based on the genomic signatures.

[0028] In some implementations, selecting the subset of the second set of k-mer seeds based on the genomic signatures includes performing one or more modulo operations including a modulo operation on each genomic signature of the genomic signatures; and selecting the subset of the second set of k-mer seeds based on results of the one or more modulo operations and a predetermined criterion.

[0029] In some implementations, actions include determining, by the one or more computers, a reference sequence location for each k-mer seed of the second set of k-mer seeds that matches at least a portion of a given k-mer seed using the hash data structure; and generating a candidate location list including a reference sequence location for each k-mer seed of the second set of k-mer seeds.

[0030] In some implementations, actions include sorting the reference sequence locations of the candidate location list according to a number of k-mer seeds paired with the given reference sequence location in the hash data structure; and determining a number of mismatches for each of the reference sequence locations of the sorted candidate location list compared to the reference genomic sequence in an order of the sorted candidate location list.

[0031] In some implementations, actions include determining a number of mismatches for a first candidate location of the reference sequence locations of the sorted candidate location list satisfies a second mismatch threshold; and selecting the first candidate location as the actual alignment.

[0032] In some implementations, the second mismatch threshold includes a threshold number of mismatching nucleotide values.

[0033] In some implementations, actions include generating second genomic signatures for each k-mer seed of the second set of k-mer seeds; performing second one or more modulo operations including a modulo operation on each genomic signature of the second genomic signatures; and selecting a second subset of the

second set of k-mer seeds based on results of the second one or more modulo operations and a predetermined criterion.

[0034] In some implementations, actions include determining a reference sequence location for each k-mer seed of the second subset that matches at least a portion of a given k-mer seed using the hash data structure; and generating a second candidate location list including a reference sequence location for each k-mer seed of the second subset.

[0035] In some implementations, actions include sorting the reference sequence locations of the second candidate location list according to a number of k-mer seeds paired with the given reference sequence location in the hash data structure; and determining a number of mismatches for each of the reference sequence locations of the sorted second candidate location list compared to the reference genomic sequence in an order of the sorted second candidate location list.

[0036] In some implementations, actions include determining a number of mismatches for a second candidate location of the reference sequence locations of the sorted second candidate location list satisfies a third mismatch threshold; and selecting the second candidate location as the actual alignment.

[0037] A third innovative aspect of the subject matter described in this specification is embodied in a method that includes obtaining, by one or more computers, a first k-mer seed from a genomic data read; generating, by the one or more computers, a genomic signature based on the first k-mer seed; determining, by the one or more computers, a reference sequence location that matches at least a portion of the first k-mer seed using a hash data structure, wherein the hash data structure comprises N data cells comprising a first portion storing a predetermined genomic signature and a second portion storing a value that corresponds to a location within a reference genomic sequence that matches at least a portion of the first k-mer seed from which the predetermined genomic signature was derived; determining, by the one or more computers, a number of mismatches based on comparing genomic data of the genomic data read to genomic data of the reference genomic sequence; comparing the number of mismatches to a mismatch threshold; and based on comparing the number of

mismatches to the mismatch threshold, selecting, by the one or more computers, an actual alignment for the genomic data read. In some implementations, selecting an actual alignment for the genomic data read occurs subsequent to comparing the number of mismatches to the mismatch threshold.

[0038] Other implementations of this and other aspects include corresponding systems, apparatus, and computer programs, configured to perform the actions of the methods, encoded on computer storage devices. A system of one or more computers can be so configured by virtue of software, firmware, hardware, or a combination of them installed on the system that in operation cause the system to perform the actions. One or more computer programs can be so configured by virtue of having instructions that, when executed by data processing apparatus, cause the apparatus to perform the actions.

[0039] The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination. For instance, in some implementations, actions include determining the number of mismatches satisfies the mismatch threshold, wherein selecting the actual alignment for the genomic data read includes selecting the reference sequence location that matches at least a portion of the first k-mer seed.

[0040] In some implementations, actions include determining the number of mismatches does not satisfy the mismatch threshold; and obtaining a second k-mer seed from the genomic data read, wherein selecting the actual alignment for the genomic data read includes selecting a reference sequence location that matches at least a portion of the second k-mer seed.

[0041] In some implementations, actions include determining the reference sequence location that matches at least a portion of the second k-mer seed using the hash data structure.

[0042] A fourth innovative aspect of the subject matter described in this specification is embodied in a method that includes extracting k-mer seeds from obtained genomic data; generating a filtered set of the k-mer seeds; and storing the filtered set of the k-mer seeds in a hash data structure.

[0043] Other implementations of this and other aspects include corresponding systems, apparatus, and computer programs, configured to perform the actions of the methods, encoded on computer storage devices. A system of one or more computers can be so configured by virtue of software, firmware, hardware, or a combination of them installed on the system that in operation cause the system to perform the actions. One or more computer programs can be so configured by virtue of having instructions that, when executed by data processing apparatus, cause the apparatus to perform the actions.

[0044] The foregoing and other embodiments can each optionally include one or more of the following features, alone or in combination. For instance, in some implementations, generating the filtered set of the k-mer seeds includes generating one or more hash values for each of the k-mer seeds; and filtering the k-mer seeds using the one or more hash values.

[0045] In some implementations, generating the filtered set of the k-mer seeds includes determining a number of occurrences of the k-mer seeds; and filtering the k-mer seeds using the number of occurrences.

[0046] In some implementations, the hash data structure comprises N data cells comprising a first portion storing a predetermined genomic signature for a k-mer seed of the k-mer seeds and a second portion storing a value that corresponds to a location within a reference genomic sequence that matches at least a portion of the k-mer seed from which the predetermined genomic signature was derived.

[0047] The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features and advantages of the invention will become apparent from the description, the drawings, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0048] FIG. 1 is a diagram showing an example of a system for generating a hash table for a multi-pass software accelerated genomic read mapping engine.

[0049] FIG. 2 is a diagram showing an example of a system for performing software-accelerated genomic mapping operations using the hash table generated using the system of FIG. 1 to select a candidate alignment location of a genomic read.

[0050] FIG. 3 is a graphical depiction of experimental results comparing data compression methods for genomic data.

[0051] FIG. 4 is a flow diagram illustrating an example of a process for generating a hash table for a multi-pass software accelerated genomic read mapping engine.

[0052] FIG. 5 is a flow diagram illustrating an example of a process for performing software-accelerated genomic mapping operations using the hash table generated using the system of FIG. 1.

[0053] FIG. 6 is a block diagram of system components that can be used to implement the systems of FIG. 1 or FIG. 2.

[0054] Like reference numbers and designations in the various drawings indicate like elements.

DETAILED DESCRIPTION

[0055] The present disclosure is directed to a multi-pass software accelerated genomic data mapping engine. The multi-pass software accelerated genomic data mapping engine provides an important technological improvement in that the multi-pass software accelerated algorithm enables an iterative evaluation of k-mer seeds against candidate alignments stored in a hash table. As a result of the iterative evaluation, the present disclosure is able to terminate the mapping process early before all of the k-mers of a particular genomic data read are evaluated. This technological improvement reduces the runtime of mapping and aligning genomic data reads of a biological samples to a reference genome. Accordingly, significant gains in efficiency can be achieved in any applications that use the genomic data mapping engine to perform mapping and aligning operations. Such operations may include, but are not limited to, for example, genomic data compression algorithms, which can achieve faster compression speeds and higher compression ratios relative to conventional software-accelerated mapping engines.

[0056] Based on the implementation, design tradeoffs can be made between speed, compression ratio, and size of the generated hash table. For example, in some implementations, a smaller k-mer seed size can result in generation of smaller hash table. Such implementations may be particularly beneficial when data storage on a computer running the multi-pass genomic mapping engine is limited. In such instances, however, it may be comparatively slower to identify an actual alignment relative implementations that utilize a larger k-mer seed size. Alternatively, in other implementations, a larger k-mer seed size can result in a larger hash table, but enable the identification of an actual alignment of a genomic data read faster. In either scenario, however, the present disclosure enables a technological improvement of either efficient use of storage space or reduced runtime. In certain implementations, an improvement can be achieved in both domains by finding the preferred balance of k-mer seed size and execution speed. For example, experimentation has shown that a k-mer seed size of 22 base calls can yield such balanced performance.

[0057] FIG. 1 is a diagram showing an example of a system 100 for generating a hash table for software accelerated genomic read mapping engine. The system 100 includes a computer 104. In some implementations, the computer 104 performs one or more operations of one or more of a K-mer engine 106, a filter engine 108, and a hash table engine 122. The computer 104 can be, for example, a tablet computer, a desktop computer, a server computer, multiple server computers, a nucleic acid sequencing device, or any other computing device(s). In other implementations, the computer 104 is communicably connected to one or more other computers configured to perform operations of one or more of the K-mer engine 106, the filter engine 108, and the hash table engine 122. These computers can be of the same, or different, type of computer as computer 104.

[0058] In some implementations, the computer 104 includes a portion of memory storage assigned to hash storage 134. In some implementations, the computer 104 is communicably connected to a computer configured with one or more memory devices for hash storage 134. For example, the computer 104 can be communicably connected to a server. The server can store the hash storage 134. The computer 104 can access the hash storage 134 on memory storage of the server.

[0059] The computer 104 obtains reference genome data 102. The reference genome data 102 can include a reference genome (which may also be referred to herein as a reference sequence) such as a DNA sequence assembled for an organism by one or more scientists. The reference genome data 102 can include data indicating a sequence of a plurality of nucleotides. For example, the reference genome data 102 can include symbols A, C, G, and T representing four types of nucleotides present in deoxyribonucleic acid (DNA), namely Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). In ribonucleic acid (RNA), Thymine is replaced by Uracil (U) and the reference genome data 102 would be included of symbols A, C, G, and U.

[0060] The genomic data referred to in the present disclosure can include, for example, and not as a limitation, nucleotide sequences, Deoxyribonucleic acid (DNA) sequences, or Ribonucleic acid (RNA). In some implementations, a reference sequence includes a synthetic sequence conceived, at least in part, to improve the compressibility of the reads in view of further processing.

[0061] The computer 104 provides the reference genome data 102 to the k-mer engine 106. The k-mer engine 106 generates one or more k-mers based on the obtained reference genome data 102. For example, the k-mer engine 106 can use a moving window to extract portions of the reference genome data 102. The moving window can have a length indicating the number of nucleotides extracted from the reference genome data 102 into a given k-mer. The moving window can be of a fixed or variable width producing k-mers of a fixed or variable length. The moving window can start at a beginning of the reference genome data 102 and move to an end of the reference genome data 102.

[0062] In some implementations, the k-mer engine 106 extracts a k-mer at regular intervals. For example, the k-mer engine 106 can use a moving window starting at a first position in the reference genome data 102 to obtain a first k-mer. The k-mer engine 106 can move the window by an interval, such as one or more nucleotides in the reference genome data 102, to a second position in the reference genome data 102 to obtain a second k-mer representing that second position.

[0063] Within this specification, a given k-mer is defined as a sequence of sequential nucleotides of a genomic sequence where the number of nucleotides in the genomic sequence for a given k-mer is defined by “k” and the nucleotides of the genomic sequence are represented by strings of As, Cs, Gs, and Ts for DNA sequences or As, Cs, Gs, and Us for RNA sequences. The nucleotides may represent a nucleotide of a references sequence or a base call generated by a nucleic acid sequencer that corresponds to a nucleotide of a sample sequence.

[0064] The k-mer engine 106 generates k-mers 102a-d from the reference genome data 102. Each of the k-mers 102a-d represent a portion of the reference genome data 102. For example, each of the k-mers 102a-d correspond to a particular subset sequence present in the reference genome data 102 having a nucleotide length of k , where k is any positive integer greater than 0. For example, a k-mer of the k-mers 102a-d can represent a nucleotide sequence of length k nucleotides including nucleotides “AAGTAT”. The genomic data 102 can include at least one instance of the sequence “AAGTAT”. In some implementations, the k-mer engine 106 generates k-mers that are 22 nucleotides long. In some implementations, the k-mer engine 106 generates k-mers that are 16 nucleotides long. In general, longer k-mers with more nucleotides occur less frequently in genomic sequences making the positions more unique. More unique positions increases the likelihood that a determined candidate alignment location is accurate.

[0065] Selection of a particular k-mer seed length can be made on an implementation-by-implementation basis based on the technical improvements sought by the particular implementation. For example, selection of a k-mer size of 22 nucleotides can result in a hash table that is larger in size than a hash table generated using a selected k-mer size of 16 nucleotides, but may result in faster candidate alignment determinations in a manner that is more accurate. Alternatively, selection of k-mer size of 16 can result in a hash table that is smaller in size than a hash table generated using a selected k-mer size of 22 nucleotides, but may result in slower candidate alignment determinations in a manner that is less accurate. Any even number of nucleotides between 16 and 22, less than 16, or greater than 22 can also be selected and be used to achieve similar technological benefits and/or tradeoffs.

[0066] The k-mer engine 106 provides the k-mers 102a-d to the filter engine 108. The filter engine 108 includes a filter hash engine 110 and abundance filtering engine 114. In some implementations, the filter hash engine 110 of the filter engine 108 generates a hash value for each of the k-mers 102a-d. For example, the filter hash engine 110 can include one or more hash functions. The filter hash engine 110 can apply the one or more hash functions to each k-mer of the k-mers 102a-d. Based on applying the one or more hash functions to each k-mer, the filter hash engine 110 can generate a hash value for each k-mer.

[0067] In some implementations, the filter hash engine 110 uses the generated hash values 112 to filter one or more k-mers. For example, the filter hash engine 110 can perform one or more modulo operations. The filter hash engine 110 can obtain a modulo value determined by a user or automated process. The filter hash engine 110 can perform one or more modulo operations with the obtained modulo value. In some implementations, the filter hash engine 110 generates a result from a modulo operation and uses the result to determine whether to filter a given k-mer. For example, the filter hash engine 110 can perform one or more modulo operations on the hash value generated for k-mer 102a. The filter hash engine 110 can obtain a result from the one or more modulo operations and compare the result to predetermined criteria. Based on comparing the result to the predetermined criteria, the filter hash engine 110 can determine whether to filter out a corresponding k-mer or not.

[0068] In some implementations, predetermined criteria includes one or more values of a modulo operation result. For example, the filter hash engine 110 can generate a result from a modulo operation on a hash value of a k-mer of the k-mers 102a-d and compare the result to one or more values. The one or more values can include the value 0. The filter hash engine 110 can compare the result to 0 and filter the corresponding k-mer based on whether or not the result matches one or more of the values. In some implementations, results that match the one or more values are filtered. In some implementations, results that do not match the one or more values are filtered.

[0069] In the example of FIG. 1, the filter hash engine 110 filters out k-mer 102b based on the generated hash of the hash values 112 and one or more modulo operations. The filter hash engine 110 retains k-mers 102a and 102c-d.

[0070] The filter engine 108 performs abundance filtering engine 114. In some implementations, the filter engine 108 provides the filtered k-mer set obtained from the filter hash engine 110 to the abundance filtering engine 114. In this way, the filter engine 108 can more efficiently determine occurrences by only determining occurrences for the remaining k-mers after the filtering of the filter hash engine 110.

[0071] In some implementations, the abundance filtering engine 114 includes generating a number of occurrences 116 for each k-mer 102a-d. In some implementations, the abundance filtering engine 114 includes generating a number of occurrences 116 for each of the filtered k-mers, such as k-mers 102a and 102c-d. For example, the filter engine 108 can compare the nucleotide sequence of each k-mer to sequences in the reference genome data 102. The filter engine 108 can increment an occurrence counter for a given k-mer based on a number of sequences in the reference genome data 102 matching the given k-mer. For example, the filter engine 108 can compare a first nucleotide representation of the k-mer 102a with a first nucleotide of the reference genome data 102. The filter engine 108 can compare a second nucleotide representation of the k-mer 102a with a second nucleotide of the reference genome data 102. If one or more representations of the k-mer 102a match a sequence of the reference genome data 102, the filter engine 108 can increment an occurrence counter for the k-mer 102a.

[0072] In some implementations, the filter engine 108 filters one or more k-mers based on a number of times a k-mer occurs in a corresponding reference sequence. For example, the filter engine 108 can generate occurrences 116. The filter engine 108 can compare each occurrence value of the values 116 to a threshold value. Based on the comparison, the filter engine 108 can filter one or more k-mers. For example, the filter engine 108 can filter out k-mers that occur more than, or equal to, a certain number of times within a reference sequence, such as the reference genome data 102. In this way, the filter engine 108 can identify a set of k-mers that is more unique, in that they

appear less often in a corresponding reference sequence, than another set with higher numbers of occurrences. In some implementations, k-mers that occur less often in a reference sequence are more indicative of accurate candidate alignments because there are fewer false positives. The k-mers can improve the efficiency of subsequent candidate alignment determination.

[0073] The filter engine 108 compares the number of occurrences of the k-mer 102c with a threshold and, based on the comparison, filters out the k-mer 102c. In some implementations, a threshold occurrence value is used. For example, the filter engine 108 can determine to filter out a k-mer if it occurs more than a number of times, e.g., 20, within the reference genome data 102.

[0074] The filter engine 108 generates index data 118 and 120 for the filtered set of k-mers 102a and 102d. The index data 118 and 120 includes a location of one or more occurrences enumerated in occurrence values 116 and data representing the k-mers 102a and 102d. The index data 118 includes the reference location 128 indicating one or more occurrences of the sequence corresponding to k-mer 102a in the reference genome data 102. The index data 120 includes the reference location 130 indicating one or more occurrences of the sequence corresponding to k-mer 102d in the reference genome data 102. In some implementations, the index data 118 and 120 include only the first occurrence, among one or more possible occurrences in a reference sequence, of a k-mer.

[0075] The filter engine 108 provides the index data 118 and 120 to the hash table engine 122. The hash table engine 122 obtains the index data 118 and 120 and generates hash values for the k-mers 102a and 102d corresponding to the index data 118 and 120. In some implementations, the hash table engine 122 generates hash table 132. For example, the hash table engine 122 can generate one or more hash values and store them in memory of the computer 104 or a device communicably connected to the computer 104.

[0076] The hash table engine 122 generates one or more hash values for each of the filtered k-mers 102a and 102d. In some implementations, the hash table engine 122 generates a hash signature using a hash signature function 124 and a hash table value

using a hash table value function 126. For example, the hash table engine 122 can obtain the data representing the sequences of k-mers 102a and 102d. The hash table engine 122 can apply both the hash signature function 124 and the hash table value function 126 to the sequences of k-mers 102a and 102d.

[0077] In some implementations, a signature generated from the hash signature function 124 is used as a key in the hash table 132. For example, the hash table engine 122 can generate a signature value for the k-mer 102a based on the index data 118. The hash table engine 122 can use the hash signature function 124 to generate the signature value based on the k-mer 102a. In some implementations, the signature generated by the hash table engine 122 is a compressed version of the k-mer 102a. For example, the k-mer 102a can be represented, as a sequence of nucleotides, in 16 bits for a k=16 k-mer with 16 nucleotides. The signature generated by the hash table engine 122 and the hash signature function 124 can be represented by 8 bits. This key compression significantly reduces the required size of the hash table 132. The decreased size increases look up time, storage time, and overall performance.

[0078] In some implementations, the hash table engine 122 generates a hash table item for each of the filtered k-mers 102a and 102d to be stored in the hash table 132. For example, the hash table engine 122 can generate a hash table value using the hash table value function 126 applied to each of the k-mers 102a and 102d to generate a hash table value for storing data corresponding to the k-mers 102a and 102d in the hash table 132. In some implementations, a hash table value represents an index in the hash table 132. The hash table engine 122 can store data corresponding to the given k-mer, such as a signature of a k-mer generated by the hash signature function 124, as well as one or more locations of the k-mer occurrence within a genomic read, such as the reference genome data 102.

[0079] The hash table engine 122 stores the reference locations 128 and 130 in the hash table 132. In some implementations, the hash table engine 122 stores the reference locations 128 and 130 as the value within a hash table item for each of the filtered k-mers 102a and 102d. For example, the hash table engine 122 can generate a hash table item that includes a location, such as reference locations 128 and 130, with a

key generated by the hash signature function 124, at an index of the hash table 132 generated by the hash table value function 126.

[0080] The hash table engine 122 provides data corresponding to the hash table 132 to the hash storage 134. In some implementations, the hash table engine 122 sends data for each of the index data 118 and 120 to the hash storage 134. In some implementations, the hash table engine 122 sends data for one or more other entries of the hash table 132. In some implementations, the hash storage 134 includes memory storage on the computer 104. In some implementations, the hash storage 134 includes memory storage of a device communicably connected to the computer 104.

[0081] FIG. 2 is a diagram showing an example of a system 200 for performing software-accelerated genomic mapping operations using the hash table generated using the system of FIG. 1 to select a candidate alignment location of a genomic read. The system 200 includes the computer 104, the k-mer engine 106, the filter engine 108, the candidate list engine 206, the hash storage 134, the sorting engine 218, and the evaluation engine 210.

[0082] In some implementations, the computer 104 operates in multiple stages. For example, a first stage can include generating one or more k-mers, generating one or more candidate alignments based on the one or more k-mers, and evaluating the candidate alignments. Second and third stages can include generating one or more k-mers, generating one or more candidate alignments based on the one or more k-mers, sorting the list of one or more candidate alignments, and evaluating the candidate alignments in the order of the sorted list. In the example of FIG. 1, the first stage is shown as path 1. The second stage is shown as path 2. The third stage is shown as path 3.

[0083] The computer 104 obtains genomic read data 202. The genomic read data 202 can include a sequence of base calls generated by a nucleic acid sequencing device by sequencing a biological sample obtained from an organism. The organism can include a human, an animal, an insect, a reptile, a plant, or any other organism. Each base call of the read can correspond to a nucleotide and be represented by an A, C, T, or G. In the case of an RNA sequence read, each base call of the read can

correspond to a nucleotide that is represented by an A, C, G, or U. The computer 104 can be used to determine the alignment between the genomic read data 202 and the reference genome data 102. Once properly aligned to the reference genome, variants between the genomic read data 202 and the reference genome data 102 can be analyzed in order to infer a number of conclusions about the organism from which the genomic read data 202 was sequenced. These conclusions can include, e.g., types of treatment that may be best suited for the organism for one or more particular ailments.

[0084] The computer 104 provides the genomic read data 202 to the k-mer engine 106. As discussed in reference to FIG. 1, the k-mer engine 106 obtains one or more k-mers based on the obtained genomic read data 202 in the same manner described with reference to obtaining k-mers from a reference genome 102, described above. In the first stage, the k-mer engine 106 generates k-mers 204 including k-mers 204a and 204b. The first k-mer that is not filtered out can be evaluated to determine if a corresponding candidate alignment satisfies one or more criteria. In some implementations, the k-mer engine 106 generates a single k-mer. For example, the filter engine 108 can determine that the first k-mer obtained by the k-mer engine 106 from the genomic read data 202 satisfies one or more filter criteria, such as hash filtering or abundance filtering. The filter engine 108 can then send data corresponding to that single k-mer to the candidate list engine 206.

[0085] In some implementations, the computer 104 filters k-mers before hash table look up. For example, the computer 104 can use the same filtering techniques used on the k-mers 102a-d to generate a portion of the hash table 132. In some implementations, applying the filter before hash table look up increases efficiency and reduces runtime relative to conventional processes. For example, the hash table 132 is generated based on k-mers that were filtered according to a particular filter method, such as filters applied by the filter engine 108. Newly generated k-mers, such as the k-mers 204, can be generated in order to find matching k-mers in the hash table 132. By using the same filtering method on the k-mers used to generate the hash table 132 and the k-mers used to look up k-mers in the hash table 132, the computer 104 prevents newly generated k-mers that are definitely not in the hash table 132 from being used to query the hash storage 134. For example, without applying the same filter processes, a

k-mer that was filtered out during the generation of the hash table 132 may not be filtered out during a candidate alignment determination stage, such as the first processing stage of FIG. 2. A resulting query on that k-mer would waste processing resources and increase runtime.

[0086] In the example of FIG. 2, the filter engine 108 filters out k-mer 204a and retains k-mer 204b. The filter engine 108 can use one or more filtering techniques, including the filtering techniques described in reference to FIG. 1, to filter k-mers generated by the k-mer engine 106 from the genomic read data 202.

[0087] In some implementations, the filter engine 108 uses particular parameters to filter the k-mers based on the stage of processing. For example, in the first stage of processing, the filter engine 108 can use a first modulo value to perform one or more modulo operations. In a second stage of processing, the filter engine 108 can use a second modulo value to perform one or more modulo operations. In subsequent stages, the filter engine 108 can use different modulo values. As discussed in FIG. 1, the modulo value and modulo operations can be used to filter one or more k-mers. In general, the filter engine 108 can adjust filter parameters to increase a number of selected k-mers over time. In this way, stages with fewer k-mers are processed before stages with more k-mers to increase efficiency and decrease run time of the system.

[0088] In some implementations, the filter engine 108 decreases a modulo value and maintains or increases one or more result values indicating not to filter a k-mer. For example, a modulo value for a first stage of processing can be 32. As discussed in FIG. 1, the filter engine 108 can apply a modulo function to a hash generated based on a k-mer. An example expression to generate a result can include: *generated_hash* mod 32. In some implementations, if the result is 0, the k-mer is further processed and if the result is not 0, the k-mer is not further processed. In this case, 0 is a result value indicating not to filter a k-mer.

[0089] In some implementations, the filter engine 108 increases the number of k-mers to process by increasing the number of result values indicating not to filter a k-mer. For example, the filter engine 108 can not filter, and further process a given k-mer, if a

corresponding hash value modulo a given modulo value, is equal to 0 or 1, greater than 3, less than 5, among others.

[0090] The filter engine 108 can increase the number of k-mers to process by decreasing the modulo value and maintaining one or more result values indicating not to filter a k-mer. For example, the filter engine 108 can not filter, and further process a given k-mer, if a corresponding hash value modulo a first modulo value, is equal to 0 or other value. In this way, the filter engine 108 selects one out of the number corresponding to the first modulo value. If the first modulo value is 32, the filter engine 108 will, on average, select one out of every 32 k-mers generated by the k-mer engine 106. To increase the number of k-mers to process, such as in a later stage of processing, the filter engine 108 can decrease the first modulo value to a second modulo value.

[0091] The filter engine 108 sends the k-mer 204b to the candidate list engine 206. In some implementations, the filter engine 108 does not send the k-mer 204a. The candidate list engine 206 queries the hash storage 134 based on the k-mer 204b. In some implementations, the candidate list engine 206 hashes the k-mer 204b to generate a hash value. The hash value can be used as a query key to search the hash table 132 stored on the hash storage 134.

[0092] As discussed in FIG. 1, the hash storage 134 stores indexes and reference locations for one or more k-mers extracted from the reference genome data 102 in the hash table 132. If the k-mer 204b is present within the hash table 132, the candidate list engine can obtain a corresponding reference location. In some implementations, the key in the hash table is a hash of the k-mer of the reference genome. By using the same hash function as was used to generate the keys of the hash table 132, the candidate list engine 206 can generate keys for new k-mers that, if matching keys of the hash table 132, indicate that the k-mer was present in the reference genome data 102 and the hash table 132 includes a location of the k-mer within the reference genome data 102.

[0093] In some implementations, the candidate list engine 206 queries the hash storage 134 based on the k-mer 204b and obtains a reference location of a

corresponding k-mer in the reference genome data 102. By comparing the location of the k-mer 204b within the genomic read data 202 and the reference location of a corresponding k-mer in the reference genome data 102, the candidate list engine 206 can generate a candidate alignment location. The candidate alignment condition can indicate a position within the reference genome data 102 where the genomic read data 202 aligns.

[0094] In some implementations, the candidate list engine 206 provides a first candidate alignment location 208a corresponding to the k-mer 204b, to the evaluation engine 210. The evaluation engine 210 obtains data of the reference genome data 102 and data of the genomic read data 202. The evaluation engine 210 compares one or more nucleotides of the reference genome data 102 and data of the genomic read data 202. In some implementations, the evaluation engine 210 starts with a first nucleotide of the reference genome data 102 and data of the genomic read data 202 at the first candidate alignment location. If the nucleotides do not match, the evaluation engine 210 can increment a mismatch counter. The evaluation engine 210 can evaluate all nucleotides in a portion of the genomic read data 202 or stop after a number of mismatches are counted. In some implementations, the evaluation engine 210 evaluates each and every nucleotide in the genomic read data 202.

[0095] In some implementations, the evaluation engine 210 generates an evaluation result based on the first candidate alignment location 208a. For example, the evaluation engine 210 can generate a value indicating a number of one or more mismatches between the genomic read data 202 and the reference genome data 102 at the first candidate alignment location 208a. Each mismatch can indicate a nucleotide of the genomic read data 202 at a position relative to the first candidate alignment location 208a being different than a corresponding nucleotide of the reference genome data 102 at the same position relative to the first candidate alignment location 208a (e.g., FIG. 2 shows a comparison graphically in item 209 where the nucleotide “C” of the genomic read data 202 is different than the nucleotide “T” in the reference genome data 102 at the same position relative to the candidate alignment location (CAL)).

[0096] In some implementations, the computer 104 compares evaluation results of the evaluation engine 210 to a threshold. For example, the computer 104 can compare the number of mismatches to a mismatch threshold. If the number of mismatches satisfies the threshold, such as being less or equal to the threshold or less than the threshold, among others, the computer 104 can stop processing and select the first candidate alignment location 208a as the actual alignment of the genomic read data 202. If the number of mismatches does not satisfy the threshold, the computer 104 can execute additional processing stages. The computer 104 can generate and provide a notification to a device of a user or a display of the computer 104. The notification can include information indicating that the first processing and evaluation stage did not result in an alignment that satisfied criteria, such as a threshold. The computer 104 can obtain and include details of the alignment, filtered k-mers, evaluation results, among others, in the notification.

[0097] In some implementations, the computer 104 executes one or more processing stages after a first stage. For example, the computer 104 can determine that the first candidate alignment location 208a does not satisfy a threshold. The computer 104 can determine that there are no matching k-mers from the reference genome data 102 in the hash storage 134. In response to one or more of these determinations, the computer 104 can execute subsequent processing stages.

[0098] In some implementations, the computer 104 executes a second processing stage. For example, after a first processing stage does not produce an alignment that satisfies criteria, the k-mer engine 106 can provide additional k-mers 214 to the filtering engine 108 to determine a new set of filtered k-mers to process. In some implementations, the filter engine 108 filters the k-mers 214 using a set of filtering parameters different from filtering parameters used in the first processing stage. For example, the filter engine 108 can adjust filtering parameters, such as resulting values to include in further processing, and modulo values to increase or decrease the number of k-mers used in subsequent processing.

[0099] In some implementations, the filter engine 108 provides the filtered k-mers 214a and 214c-d to the candidate list engine 206. The filter engine 108 can filter out the

k-mer 214b using one or more filtering techniques as described herein. The candidate list engine 206 can query the hash storage 134 to obtain a reference location for each k-mer of the filtered k-mers 214a and 214c-d. In some implementations, the candidate list engine 206 queries the hash storage 134 using one or more hashes representing each of the filtered k-mers 214a and 214c-d. For example, the candidate list engine 206 can generate a hash value by applying a hash function to each of the filtered k-mers 214a and 214c-d to generate a hash function result. The candidate list engine 206 can query the hash storage 134 to find hash table items with a key that matches the hash function results. Matching keys can indicate identical k-mers from the reference genome data 102.

[00100] In some implementations, the candidate list engine 206 queries the hash storage 134 based on the filtered k-mers 214a and 214c-d and obtains reference locations for corresponding k-mers in the reference genome data 102. By comparing the location of each k-mer of the filtered k-mers 214a and 214c-d with the obtained reference locations, the candidate list engine 206 can generate a candidate alignment location. The candidate alignment condition can indicate a position within the reference genome data 102 where the genomic read data 202 aligns. The candidate list engine 206 can generate one or more candidate alignment conditions for each of the filtered k-mers 214a and 214c-d.

[00101] In some implementations, the candidate list engine 206 generates one or more candidate alignment locations. For example, the candidate list engine 206 can generate a second candidate alignment location 208b and a third candidate alignment location 208c. K-mers used to generate alignments are referred to as supporting the alignment. One or more k-mers of the filtered k-mers 214a and 214c-d can support the same candidate alignment location. For example, both the k-mer 214a and the k-mer 214d support the second candidate alignment location 208b. The k-mer 214c supports the third candidate alignment location 208c.

[00102] In some implementations, the candidate list engine 206 provides a list of one or more candidate alignment locations to the sorting engine 218. The sorting engine 218 can obtain the candidate alignment locations and sort the candidate alignment

locations. In some implementations, the sorting engine 218 sorts candidate alignment locations based on the number of supporting k-mers. For example, because both k-mers 214a and 214d support the second candidate alignment location 208b and only the k-mer 214c supports the third candidate alignment location 208c, more k-mers support the second candidate alignment location 208b compared to the third candidate alignment location 208c. Because more k-mers support the second candidate alignment location 208b compared to the third candidate alignment location 208c, the sorting engine 218 can sort the second candidate alignment location 208b above the third candidate alignment location 208c.

[00103] In some implementations, the sorting engine 218 provides a sorted list 220 to the evaluation engine 210. For example, the sorting engine 218 can generate the sorted list 220 including the second candidate alignment location 208b sorted before the third candidate alignment location 208c. The sorted list 220 can be stored in any suitable data format including a linked list, array, queue, or the like.

[00104] In some implementations, the evaluation engine 210 obtains data from the sorting engine 218. For example, the sorting engine 218 can provide the sorted list 220 to the evaluation engine 210. The evaluation engine 210 can evaluate each item of the sorted list 220 in the order of the sorted list 220. For example, the evaluation engine 210 can evaluate the second candidate alignment location 208b prior to evaluating the third candidate alignment location 208c. The computer 104 can compare the evaluation results for each item of the sorted list 220 to an evaluation threshold to determine whether to select the alignment as the actual alignment. If the alignment satisfies the evaluation threshold, the computer 104 can prevent further processing by the evaluation engine 210 and output the alignment location as an output candidate alignment location 212.

[00105] In some implementations, the computer 104 adjusts an evaluation threshold based on a processing stage. For example, for a first processing stage, the computer 104 can require a lower number of mismatches for a selected actual alignment compared to a later processing stage. For a processing stage after the later processing stage, the number of mismatches can be maintained, increased, or decreased.

[00106] In some implementations, the computer 104 selects the alignment of the first processing stage only if the alignment results in 0 mismatches. In some implementations, the computer 104 compares an evaluation result to a threshold number of mismatches, e.g., 8, and based on the evaluation result satisfying the number of mismatches, e.g., less than or equal to, or less than, the threshold, the computer 104 selects the corresponding alignment location as the output candidate alignment location 212. In some implementations, the evaluation engine 210 stops during a processing stage and outputs the last processed alignment if the alignment satisfies a mismatch threshold. For example, the mismatch threshold for the first processing stage can be 0 requiring the evaluation of the first candidate alignment location 208a to have no mismatches. The mismatch threshold for the second processing stage can be 8, requiring the evaluation of the second candidate alignment location 208b or the third candidate alignment location 208c to have less than, or less than or equal to, 8 mismatches. The mismatch threshold for the third processing stage can be 4, requiring the evaluation of a subsequent candidate alignment location to have less than, or less than or equal to, 4 mismatches.

[00107] In some implementations, the evaluation engine 210 generates evaluation results for each candidate alignment of the sorted list 220. In some implementations, none of the evaluation results for each candidate alignment of the sorted list 220 satisfies a predetermined criteria, such as a mismatch threshold. In this case, the computer 104 can start a third processing stage by generating a new set of k-mers and processing the k-mers as shown for the k-mer set 214 and filtered k-mers 214a and 214c-d.

[00108] In some implementations, the k-mer engine 106 and the filter engine 108 generate k-mers for the third processing stage similar to the k-mers 214 according to one or more parameters. For example, the k-mer engine 106 can extract k-mers from the genomic read data 202 of a size “k” corresponding to a seed length parameter. The filter engine 108 can filter the extracted k-mers according to one or more filter parameters. For example, the filter engine 108 can use one or more filtering techniques, such as hash filtering or abundance filtering. In some implementations, the modulo value used to generate results of modulo operations changes in the third

processing stage compared to the second processing stage. For example, the modulo value can decrease from 32 to 8. The filtering engine can effectively decrease filtering by selecting one out of every 32 k-mers in the second processing stage and selecting one out of every 8 k-mers in the third processing stage. In general, the parameters to generate k-mers in subsequent processing stages can be adjusted by the user or automated process based on the results of the processed k-mers in one or more prior processing stages.

[00109] In some implementations, the computer 104 generates a filtered set of k-mers in a third processing stage after a first and second processing stage fail to select an alignment that satisfies conditions. The filtered set of k-mers can be used to generate a candidate alignment list by the candidate list engine 206 in the same way as the filtered k-mers 214a and 214c-d. The alignment list can be sorted and evaluated in the same way as the sorted list 220.

[00110] Based on one or more of the processing stages 1 through 3, the computer 104 selects an alignment location that satisfies an alignment criteria. In some implementations, the alignment criteria includes a mismatch threshold. The computer 104 can provide the alignment location as the output candidate alignment location 212. The computer 104 can provide data indicating the output candidate alignment location 212 to a memory storage device, a display, a device of a user, another communicably connected device, among others.

[00111] In some implementations, a first computer performs one or more operations performed by the computer 104 as discussed in FIG. 1 and a second computer performs one or more operations performed by the computer 104 as discussed in FIG. 2. For example, at least two distinct computer devices can perform one or more operations performed by the computer 104. In some implementations, a first computer performs hash table generation including performing one or more operations discussed in reference to FIG. 1. In some implementations, a second computer, distinct from the first computer, performs genomic mapping including performing one or more operations discussed in reference to FIG. 2.

[00112] FIG. 3 is a graphical depiction of experimental results comparing data compression methods for genomic data. FIG. 3 shows a comparison between a spring compression method and two versions that use a hash table compression method. The spring compression method is a compression method available on GitHub. The spring compression method is a compression tool for FASTQ files containing up to 4.29 Billion reads. Key 306 shows the graphical representation for each of the three methods.

[00113] In some implementations, a FASTQ file is a text file that includes the sequence data from the clusters that pass filter on a flow cell. If samples were multiplexed, the first step in FASTQ file generation is demultiplexing. Demultiplexing assigns clusters to a sample, based on the cluster's index sequence(s). After demultiplexing, the assembled sequences are written to FASTQ files per sample. If samples were not multiplexed, the demultiplexing step does not occur, and, for each flow cell lane, all clusters are assigned to a single sample.

[00114] In some implementations, for a single-read run, one Read 1 (R1) FASTQ file is created for each sample per flow cell lane. For a paired-end run, one R1 and one Read 2 (R2) FASTQ file is created for each sample for each lane. FASTQ files are compressed and created with the extension *.fastq.gz.

[00115] In some implementations, for each cluster that passes filter, a single sequence is written to the corresponding sample's R1 FASTQ file, and, for a paired-end run, a single sequence is also written to the sample's R2 FASTQ file. In some implementations, each entry in a FASTQ files includes 4 lines: (i) a sequence identifier with information about the sequencing run and the cluster (the exact contents of this line vary by based on conversion software used), (ii) the sequence (e.g., the base calls; A, C, T, G and N), (iii) a separator (e.g., a plus (+) sign), and (iv) the base call quality scores (e.g., Phred +33 encoded, using ASCII characters to represent the numerical quality scores).

[00116] FIG. 3 shows results from the compression of a 49 GB fastq.gz data file. Chart 302 shows a comparison of final compressed size generated by each of the spring compression, hash table compression v1, and hash table compression v2. The spring compression generates a compressed version of the 49 GB fastq.gz data file that is 40

GB. The hash table compression v1 generates a compressed version of the 49 GB fastq.gz data file that is 4.3 GB. The hash table compression v2 generates a compressed version of the 49 GB fastq.gz data file that is 5.5 GB.

[00117] The hash table compression v2 includes the method of compression described herein, e.g., FIG. 1 and corresponding description. The hash table compression v1 is a similar method that also uses a hash table but with different parameters. For example, v1 includes a seed length parameter of 16. V2 includes a seed length parameter of 22. V2, and the increase in seed length, allows v2 to have more seeds in the reference genome with unique positions than v1 and increases the number of total distinct seeds. The trade-off is that this implementation of v2 requires more memory usage than v1.

[00118] In some implementations, the hash table compression v1 with a seed length of 16 extracts one or more seeds from a reference genome 16 nucleotides in length. In some implementations, v1 inserts 143×10^6 k-mers into the hash table. The hash table can have 2^{28} cells (e.g., 5B per cell), resulting in load factor of 0.53 and total hash size of $2^{28} \times 5B = 1280$ MB.

[00119] In some implementations, the hash table compression v2 with a seed length of 22 extracts one or more seeds from a reference genome 22 nucleotides in length. In some implementations, v2 inserts 299×10^6 k-mers into the hash table. The hash table can have 2^{29} cells (e.g., 5B per cell) resulting in load factor of 0.55 and a total hash size of $2^{29} \times 5B = 2560$ MB.

[00120] The increased memory usage is shown in FIG. 3 where the hash table compression v2 generates a compressed version of the 49 GB fastq.gz file that is 5.5 GB while the hash table compression v1 generates a compressed version of the 49 GB fastq.gz file that is 4.3 GB.

[00121] However, the hash table compression v2 results in faster compression as shown in chart 304. Both the hash table compression v1 and the hash table compression v2 are quicker than the spring compression method (e.g., 55930 seconds). The hash table compression v2 is quicker than the hash table compression v1 (e.g., 5641 seconds compared to 9208 seconds).

[00122] In some implementations, increases in efficiency are achieved by the hash table compression v2 because of the mapping engine's process of generating the input to the compression algorithm. For example, the computer 104 can generate mapped reads faster and with a tailored level of accuracy for compression algorithms that results in better compression performance over prior methods, such as hash table compression v1 and spring compression.

[00123] FIG. 4 is a flow diagram illustrating an example of a process 300 for generating a hash table for a multi-pass software accelerated genomic read mapping engine. The process 400 may be performed by one or more electronic systems, for example, the computer 104 of system 100 in FIG. 1.

[00124] The process 300 includes obtaining genomic data (402). For example, the computer 104 can obtain the reference genome data 102. The reference genome data 102 can include symbols A, C, G, and T representing four types of nucleotides present in deoxyribonucleic acid (DNA), namely Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). In ribonucleic acid (RNA), Thymine is replaced by Uracil (U) and the reference genome data 102 would be included of symbols A, C, G, and U.

[00125] The process 300 includes extracting one or more k-mers from the genomic data (404). For example, the k-mer engine 106 can obtain one or more extraction parameters from a user or automated process. The parameters can include a seed length indicating the length of seeds to extract into extracted k-mers. The k-mer engine 106 can generate one or more k-mers from the reference genome data 102.

[00126] The process 300 includes filtering the one or more k-mers (406). For example, the filter engine 108 can perform one or more filtering methods on the extracted k-mers to generate a subset of k-mers for inclusion in the hash table 132. In some implementations, the filter engine 108 includes a filter hash engine 110 and an abundance filtering engine 114.

[00127] The process 300 includes storing a filtered set of one or more k-mers in a hash table (408). For example, the hash table engine 122 can generate a hash table based on one or more of the filtered k-mers generated by the filter engine 108. The hash table engine 122, or other component of the computer 104 or connected computer or device,

can perform hash operations to generate one or more hash function results. The hash function results can be used to store data corresponding to each of the k-mers within the hash table 132. For example, a hash value can be used as a signature in the hash table 132 and a hash value can be used as a hash table index in the hash table 132. In some implementations, the hash values are generated using different hash functions.

[00128] FIG. 5 is a flow diagram illustrating an example of a process 500 for performing software-accelerated genomic mapping operations using the hash table generated using the system of FIG. 1. The process 500 may be performed by one or more electronic systems, for example, the computer 104 of system 200 in FIG. 2.

[00129] The process 500 includes obtaining a first k-mer seed from a genomic data read (502). For example, the computer 104 can obtain the genomic read data 202. The genomic read data 202 can include a sequence of base calls generated by a nucleic acid sequencing device by sequencing a biological sample obtained from an organism. The organism can include a human, an animal, an insect, a reptile, a plant, or any other organism. Each base call of the read can correspond to a nucleotide and be represented by an A, C, T, or G. In the case of an RNA sequence read, each base call of the read can correspond to a nucleotide that is represented by an A, C, G, or U.

[00130] The process 500 includes generating a genomic signature based on the first k-mer seed (504). For example, the candidate list engine 206 can generate a key for a given k-mer based on a hash function used to generate keys for k-mers of genomic reference data. The candidate list engine 206 can query a previously generated hash table storing the genomic reference data, such as the reference genome data 102, using the generated key for the given k-mer. In some implementations, the hash result for the key is generated by applying a hash function to the data representing the given k-mer.

[00131] The process 500 includes determining a reference location based on the genomic signature (506). For example, the candidate list engine 206 can query a previously generated hash table storing the genomic reference data, such as the reference genome data 102, using the generated key for the given k-mer. The hash table can store reference locations indicating a position of a stored k-mer within a

reference sequence. The candidate list engine 206 can use the location information of the stored k-mer to determine a location of a mapped read, such as the genomic read data 202, in relation to reference genomic data, such as the reference genome data 102.

[00132] The process 500 includes determining a number of mismatches (508). For example, the evaluation engine 210 can obtain data of the reference genome data 102 and data of the genomic read data 202. The evaluation engine 210 can compare one or more nucleotides of the reference genome data 102 and data of the genomic read data 202 to determine one or more mismatches based on a starting location determined by the candidate list engine 206.

[00133] The process 500 includes selecting an actual alignment for the genomic data read (510). For example, in a first processing stage or pass, the evaluation engine 210 can determine that a number of mismatches satisfies a mismatch threshold and the computer 104 can select the location corresponding to the first generated k-mer seed as the actual alignment for the genomic read data 202. In some implementations, one or more subsequent processing stages are performed based on the evaluation engine 210 generating evaluation results that do not satisfy alignment quality criteria, such as a mismatch threshold. During subsequent processing stages, such as second or third stages, the computer 104 can adjust parameters of the extraction of seeds, hash table look up, evaluation criteria, among others, to select the actual alignment for the genomic read data 202.

[00134] FIG. 6 is a diagram illustrating an example of a computing system used for hash table generation and selecting a candidate alignment location of a genomic read based on a reference genomic read stored in a hash table. The computing system includes computing device 600 and a mobile computing device 650 that can be used to implement the techniques described herein. For example, one or more components of the system 100 or 200 could be an example of the computing device 600 or the mobile computing device 650, such as the computer 104, devices that access information from the computer 104, or a server that accesses or stores information regarding the operations performed by the computer 104.

[00135] The computing device 600 is intended to represent various forms of digital computers, such as laptops, desktops, workstations, personal digital assistants, servers, blade servers, mainframes, and other appropriate computers. The mobile computing device 650 is intended to represent various forms of mobile devices, such as personal digital assistants, cellular telephones, smart-phones, mobile embedded radio systems, radio diagnostic computing devices, and other similar computing devices. The components shown here, their connections and relationships, and their functions, are meant to be examples only, and are not meant to be limiting.

[00136] The computing device 600 includes a processor 602, a memory 604, a storage device 606, a high-speed interface 608 connecting to the memory 604 and multiple high-speed expansion ports 610, and a low-speed interface 612 connecting to a low-speed expansion port 614 and the storage device 606. Each of the processor 602, the memory 604, the storage device 606, the high-speed interface 608, the high-speed expansion ports 610, and the low-speed interface 612, are interconnected using various busses, and may be mounted on a common motherboard or in other manners as appropriate. The processor 602 can process instructions for execution within the computing device 600, including instructions stored in the memory 604 or on the storage device 606 to display graphical information for a GUI on an external input/output device, such as a display 616 coupled to the high-speed interface 608. In other implementations, multiple processors and/or multiple buses may be used, as appropriate, along with multiple memories and types of memory. In addition, multiple computing devices may be connected, with each device providing portions of the operations (e.g., as a server bank, a group of blade servers, or a multi-processor system). In some implementations, the processor 602 is a single threaded processor. In some implementations, the processor 602 is a multi-threaded processor. In some implementations, the processor 602 is a quantum computer.

[00137] The memory 604 stores information within the computing device 600. In some implementations, the memory 604 is a volatile memory unit or units. In some implementations, the memory 604 is a non-volatile memory unit or units. The memory 604 may also be another form of computer-readable medium, such as a magnetic or optical disk.

[00138] The storage device 606 is capable of providing mass storage for the computing device 600. In some implementations, the storage device 606 may be or include a computer-readable medium, such as a floppy disk device, a hard disk device, an optical disk device, or a tape device, a flash memory or other similar solid-state memory device, or an array of devices, including devices in a storage area network or other configurations. Instructions can be stored in an information carrier. The instructions, when executed by one or more processing devices (for example, processor 602), perform one or more methods, such as those described above. The instructions can also be stored by one or more storage devices such as computer- or machine readable mediums (for example, the memory 604, the storage device 606, or memory on the processor 602). The high-speed interface 608 manages bandwidth-intensive operations for the computing device 600, while the low-speed interface 612 manages lower bandwidth-intensive operations. Such allocation of functions is an example only. In some implementations, the high speed interface 608 is coupled to the memory 604, the display 616 (e.g., through a graphics processor or accelerator), and to the high-speed expansion ports 610, which may accept various expansion cards (not shown). In the implementation, the low-speed interface 612 is coupled to the storage device 606 and the low-speed expansion port 614. The low-speed expansion port 614, which may include various communication ports (e.g., USB, Bluetooth, Ethernet, wireless Ethernet) may be coupled to one or more input/output devices, such as a keyboard, a pointing device, a scanner, or a networking device such as a switch or router, e.g., through a network adapter.

[00139] The computing device 600 may be implemented in a number of different forms, as shown in the figure. For example, it may be implemented as a standard server 620, or multiple times in a group of such servers. In addition, it may be implemented in a personal computer such as a laptop computer 622. It may also be implemented as part of a rack server system 624. Alternatively, components from the computing device 600 may be combined with other components in a mobile device, such as a mobile computing device 650. Each of such devices may include one or more of the computing device 600 and the mobile computing device 650, and an entire system may be made up of multiple computing devices communicating with each other.

[00140] The mobile computing device 650 includes a processor 652, a memory 664, an input/output device such as a display 654, a communication interface 666, and a transceiver 668, among other components. The mobile computing device 650 may also be provided with a storage device, such as a micro-drive or other device, to provide additional storage. Each of the processor 652, the memory 664, the display 654, the communication interface 666, and the transceiver 668, are interconnected using various buses, and several of the components may be mounted on a common motherboard or in other manners as appropriate.

[00141] The processor 652 can execute instructions within the mobile computing device 650, including instructions stored in the memory 664. The processor 652 may be implemented as a chipset of chips that include separate and multiple analog and digital processors. The processor 652 may provide, for example, for coordination of the other components of the mobile computing device 650, such as control of user interfaces, applications run by the mobile computing device 650, and wireless communication by the mobile computing device 650.

[00142] The processor 652 may communicate with a user through a control interface 658 and a display interface 656 coupled to the display 654. The display 654 may be, for example, a TFT (Thin-Film-Transistor Liquid Crystal Display) display or an OLED (Organic Light Emitting Diode) display, or other appropriate display technology. The display interface 656 may include appropriate circuitry for driving the display 654 to present graphical and other information to a user. The control interface 658 may receive commands from a user and convert them for submission to the processor 652. In addition, an external interface 662 may provide communication with the processor 652, so as to enable near area communication of the mobile computing device 650 with other devices. The external interface 662 may provide, for example, for wired communication in some implementations, or for wireless communication in other implementations, and multiple interfaces may also be used.

[00143] The memory 664 stores information within the mobile computing device 650. The memory 664 can be implemented as one or more of a computer-readable medium or media, a volatile memory unit or units, or a non-volatile memory unit or units. An

expansion memory 674 may also be provided and connected to the mobile computing device 650 through an expansion interface 672, which may include, for example, a SIMM (Single In Line Memory Module) card interface. The expansion memory 674 may provide extra storage space for the mobile computing device 650, or may also store applications or other information for the mobile computing device 650. Specifically, the expansion memory 674 may include instructions to carry out or supplement the processes described above, and may include secure information also. Thus, for example, the expansion memory 674 may be provide as a security module for the mobile computing device 650, and may be programmed with instructions that permit secure use of the mobile computing device 650. In addition, secure applications may be provided via the SIMM cards, along with additional information, such as placing identifying information on the SIMM card in a non-hackable manner.

[00144] The memory may include, for example, flash memory and/or NVRAM memory (nonvolatile random access memory), as discussed below. In some implementations, instructions are stored in an information carrier such that the instructions, when executed by one or more processing devices (for example, processor 652), perform one or more methods, such as those described above. The instructions can also be stored by one or more storage devices, such as one or more computer- or machine-readable mediums (for example, the memory 664, the expansion memory 674, or memory on the processor 652). In some implementations, the instructions can be received in a propagated signal, for example, over the transceiver 668 or the external interface 662.

[00145] The mobile computing device 650 may communicate wirelessly through the communication interface 666, which may include digital signal processing circuitry in some cases. The communication interface 666 may provide for communications under various modes or protocols, such as GSM voice calls (Global System for Mobile communications), SMS (Short Message Service), EMS (Enhanced Messaging Service), or MMS messaging (Multimedia Messaging Service), CDMA (code division multiple access), TDMA (time division multiple access), PDC (Personal Digital Cellular), WCDMA (Wideband Code Division Multiple Access), CDMA2000, or GPRS (General Packet Radio Service), LTE, 6G/6G cellular, among others. Such communication may occur, for example, through the transceiver 668 using a radio frequency. In addition,

short-range communication may occur, such as using a Bluetooth, Wi-Fi, or other such transceiver (not shown). In addition, a GPS (Global Positioning System) receiver module 670 may provide additional navigation- and location-related wireless data to the mobile computing device 650, which may be used as appropriate by applications running on the mobile computing device 650.

[00146] The mobile computing device 650 may also communicate audibly using an audio codec 660, which may receive spoken information from a user and convert it to usable digital information. The audio codec 660 may likewise generate audible sound for a user, such as through a speaker, e.g., in a handset of the mobile computing device 650. Such sound may include sound from voice telephone calls, may include recorded sound (e.g., voice messages, music files, among others) and may also include sound generated by applications operating on the mobile computing device 650.

[00147] The mobile computing device 650 may be implemented in a number of different forms, as shown in the figure. For example, it may be implemented as a cellular telephone 680. It may also be implemented as part of a smart-phone 682, personal digital assistant, or other similar mobile device.

[00148] A number of implementations have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the disclosure. For example, various forms of the flows shown above may be used, with steps re-ordered, added, or removed.

[00149] Embodiments of the invention and all of the functional operations described in this specification can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Embodiments of the invention can be implemented as one or more computer program products, e.g., one or more modules of computer program instructions encoded on a computer readable medium for execution by, or to control the operation of, data processing apparatus. The computer readable medium can be a machine-readable storage device, a machine-readable storage substrate, a memory device, a composition of matter effecting a machine-readable propagated signal, or a combination of one or

more of them. The term “data processing apparatus” encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them. A propagated signal is an artificially generated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal that is generated to encode information for transmission to suitable receiver apparatus.

[00150] A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program does not necessarily correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

[00151] The processes and logic flows described in this specification can be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

[00152] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive

instructions and data from a read only memory or a random access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Moreover, a computer can be embedded in another device, e.g., a tablet computer, a mobile telephone, a personal digital assistant (PDA), a mobile audio player, a Global Positioning System (GPS) receiver, to name just a few. Computer readable media suitable for storing computer program instructions and data include all forms of non volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[00153] To provide for interaction with a user, embodiments of the invention can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input.

[00154] Embodiments of the invention can be implemented in a computing system that includes a back end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the invention, or any combination of one or more such back end, middleware, or front end components. The components of the system can be interconnected by any form or medium of digital data communication,

e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

[00155] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

[00156] While this specification contains many specifics, these should not be construed as limitations on the scope of the invention or of what may be claimed, but rather as descriptions of features specific to particular embodiments of the invention. Certain features that are described in this specification in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

[00157] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

[00158] In each instance where an HTML file is mentioned, other file types or formats may be substituted. For instance, an HTML file may be replaced by an XML, JSON, plain text, or other types of files. Moreover, where a table or hash table is mentioned,

other data structures (such as spreadsheets, relational databases, or structured files) may be used.

[00159] Particular embodiments of the invention have been described. Other embodiments are within the scope of the following claims. For example, the steps recited in the claims can be performed in a different order and still achieve desirable results.

[00160] What is claimed is:

CLAIMS

1. A method for software-accelerated genomic data read mapping, the method comprising:
 - obtaining, by one or more computers, a first k-mer seed from a genomic data read;
 - generating, by the one or more computers, a genomic signature based on the first k-mer seed;
 - determining, by the one or more computers, a reference sequence location that matches at least a portion of the first k-mer seed using a hash data structure, wherein the hash data structure comprises N data cells comprising a first portion storing a predetermined genomic signature and a second portion storing a value that corresponds to a location within a reference genomic sequence that matches at least the portion of the first k-mer seed from which the predetermined genomic signature was derived;
 - determining, by the one or more computers, a number of mismatches based on comparing genomic data of the genomic data read to genomic data of the reference genomic sequence;
 - determining, by the one or more computers, the number of mismatches does not satisfy a mismatch threshold;
 - based on determining the number of mismatches does not satisfy the mismatch threshold, obtaining, by the one or more computers, a second set of k-mer seeds from the genomic data read; and
 - based on the second set of k-mer seeds from the genomic data read, selecting, by the one or more computers, an actual alignment for the genomic data read.
2. The method of claim 1, wherein obtaining the first k-mer seed from the genomic data read comprises:
 - obtaining a first k bases of the genomic data read as the first k-mer seed.

3. The method of claim 1, wherein selecting the actual alignment for the genomic data read based on the second set of k-mer seeds from the genomic data read comprises:

generating genomic signatures for each k-mer seed of the second set of k-mer seeds; and

selecting a subset of the second set of k-mer seeds based on the genomic signatures.

4. The method of claim 3, wherein selecting the subset of the second set of k-mer seeds based on the genomic signatures comprises:

performing one or more modulo operations including a modulo operation on each genomic signature of the genomic signatures; and

selecting the subset of the second set of k-mer seeds based on results of the one or more modulo operations and a predetermined criterion.

5. The method of claim 1, comprising:

determining, by the one or more computers, a reference sequence location for each k-mer seed of the second set of k-mer seeds that matches at least a portion of a given k-mer seed using the hash data structure; and

generating a candidate location list including a reference sequence location for each k-mer seed of the second set of k-mer seeds.

6. The method of claim 5, comprising:

sorting the reference sequence locations of the candidate location list according to a number of k-mer seeds paired with the given reference sequence location in the hash data structure; and

determining a number of mismatches for each of the reference sequence locations of the sorted candidate location list compared to the reference genomic sequence in an order of the sorted candidate location list.

7. The method of claim 6, comprising:

determining a number of mismatches for a first candidate location of the reference sequence locations of the sorted candidate location list satisfies a second mismatch threshold; and
selecting the first candidate location as the actual alignment.

8. The method of claim 7, wherein the second mismatch threshold comprises a threshold number of mismatching nucleotide values.

9. The method of claim 1, comprising:
generating second genomic signatures for each k-mer seed of the second set of k-mer seeds;
performing second one or more modulo operations including a modulo operation on each genomic signature of the second genomic signatures; and
selecting a second subset of the second set of k-mer seeds based on results of the second one or more modulo operations and a predetermined criterion.

10. The method of claim 9, comprising:
determining a reference sequence location for each k-mer seed of the second subset that matches at least a portion of a given k-mer seed using the hash data structure; and
generating a second candidate location list including a reference sequence location for each k-mer seed of the second subset.

11. The method of claim 10, comprising:
sorting the reference sequence locations of the second candidate location list according to a number of k-mer seeds paired with the given reference sequence location in the hash data structure; and
determining a number of mismatches for each of the reference sequence locations of the sorted second candidate location list compared to the reference genomic sequence in an order of the sorted second candidate location list.

12. The method of claim 11, comprising:

determining a number of mismatches for a second candidate location of the reference sequence locations of the sorted second candidate location list satisfies a third mismatch threshold; and

selecting the second candidate location as the actual alignment.

13. A method for software-accelerated genomic data read mapping, the method comprising:

obtaining, by one or more computers, a first k-mer seed from a genomic data read;

generating, by the one or more computers, a genomic signature based on the first k-mer seed;

determining, by the one or more computers, a reference sequence location that matches at least a portion of the first k-mer seed using a hash data structure, wherein the hash data structure comprises N data cells comprising a first portion storing a predetermined genomic signature and a second portion storing a value that corresponds to a location within a reference genomic sequence that matches at least a portion of the first k-mer seed from which the predetermined genomic signature was derived;

determining, by the one or more computers, a number of mismatches based on comparing genomic data of the genomic data read to genomic data of the reference genomic sequence;

based on determining the number of mismatches includes one or more mismatches, obtaining, by the one or more computers, a set of k-mer seeds from the genomic data read; and

based on the set of k-mer seeds from the genomic data read, selecting, by the one or more computers, an actual alignment for the genomic data read.

14. The method of claim 13, wherein obtaining the first k-mer seed from the genomic data read comprises:

obtaining a first k bases of the genomic data read as the first k-mer seed.

15. The method of claim 13, comprising:
based on determining the number of mismatches is zero, selecting the first k-mer seed as the actual alignment for the genomic data read.

16. The method of claim 13, wherein selecting the actual alignment for the genomic data read based on the set of k-mer seeds from the genomic data read comprising:
generating genomic signatures for each k-mer seed of the set of k-mer seeds;
and
selecting a subset of the set of k-mer seeds based on the genomic signatures.

17. The method of claim 16, wherein selecting the subset of the set of k-mer seeds based on the genomic signatures comprises:
performing one or more modulo operations including a modulo operation on each genomic signature of the genomic signatures; and
selecting the subset of the set of k-mer seeds based on results of the one or more modulo operations and a predetermined criterion.

18. The method of claim 16, comprising:
determining, by the one or more computers, a reference sequence location for each k-mer seed of the subset that matches at least a portion of the given k-mer seed using the hash data structure; and
generating a candidate location list including a reference sequence location for each k-mer seed of the subset.

19. The method of claim 18, comprising:
sorting the reference sequence locations of the candidate location list according to a number of k-mer seeds paired with the given reference sequence location in the hash data structure; and

determining a number of mismatches for each of the reference sequence locations of the sorted candidate location list compared to the reference genomic sequence in an order of the sorted candidate location list.

20. The method of claim 19, comprising:
determining a number of mismatches for a first candidate location of the reference sequence locations of the sorted candidate location list satisfies a mismatch threshold; and
selecting the first candidate location as the actual alignment.

21. The method of claim 20, wherein the mismatch threshold comprises a threshold number of mismatching nucleotide values.

22. The method of claim 19, comprising:
determining a number of mismatches for one or more candidate locations of the reference sequence locations of the sorted candidate location list does not satisfy a mismatch threshold; and
based on determining the number of mismatches does not satisfy the mismatch threshold, obtaining a second set of k-mer seeds from the genomic data read.

23. The method of claim 22, comprising:
generating second genomic signatures for each k-mer seed of the second set of k-mer seeds;
performing second one or more modulo operations including a modulo operation on each genomic signature of the second genomic signatures; and
selecting a second subset of the second set of k-mer seeds based on results of the second one or more modulo operations and a predetermined criterion.

24. The method of claim 23, comprising:

determining a reference sequence location for each k-mer seed of the second subset that matches at least a portion of the given k-mer seed using the hash data structure; and

generating a second candidate location list including a reference sequence location for each k-mer seed of the second subset.

25. The method of claim 24, comprising:

sorting the reference sequence locations of the second candidate location list according to a number of k-mer seeds paired with the given reference sequence location in the hash data structure; and

determining a number of mismatches for each of the reference sequence locations of the sorted second candidate location list compared to the reference genomic sequence in an order of the sorted second candidate location list.

26. The method of claim 25, comprising:

determining a number of mismatches for a second candidate location of the reference sequence locations of the sorted second candidate location list satisfies a second mismatch threshold; and

selecting the second candidate location as the actual alignment.

27. A method for software-accelerated genomic data read mapping, the method comprising:

obtaining, by one or more computers, a first k-mer seed from a genomic data read;

generating, by the one or more computers, a genomic signature based on the first k-mer seed;

determining, by the one or more computers, a reference sequence location that matches at least a portion of the first k-mer seed using a hash data structure, wherein the hash data structure comprises N data cells comprising a first portion storing a predetermined genomic signature and a second portion storing a value that corresponds to a location within a reference genomic sequence that matches at least a

portion of the first k-mer seed from which the predetermined genomic signature was derived;

determining, by the one or more computers, a number of mismatches based on comparing genomic data of the genomic data read to genomic data of the reference genomic sequence;

comparing the number of mismatches to a mismatch threshold; and

based on comparing the number of mismatches to the mismatch threshold, selecting, by the one or more computers, an actual alignment for the genomic data read.

28. The method of claim 27, comprising:

determining the number of mismatches satisfies the mismatch threshold,

wherein selecting the actual alignment for the genomic data read comprises selecting the reference sequence location that matches at least a portion of the first k-mer seed.

29. The method of claim 27, comprising:

determining the number of mismatches does not satisfy the mismatch threshold; and

obtaining a second k-mer seed from the genomic data read,

wherein selecting the actual alignment for the genomic data read comprises selecting a reference sequence location that matches at least a portion of the second k-mer seed.

30. The method of claim 29, comprising:

determining the reference sequence location that matches at least a portion of the second k-mer seed using the hash data structure.

31. A method for generating a hash data structure for a multi-pass software accelerated genomic read mapping engine, the method comprising:

extracting k-mer seeds from obtained genomic data;

generating a filtered set of the k-mer seeds; and
storing the filtered set of the k-mer seeds in a hash data structure.

32. The method of claim 31, wherein generating the filtered set of the k-mer seeds comprises:

generating one or more hash values for each of the k-mer seeds; and
filtering the k-mer seeds using the one or more hash values.

33. The method of claim 31, wherein generating the filtered set of the k-mer seeds comprises:

determining a number of occurrences of the k-mer seeds; and
filtering the k-mer seeds using the number of occurrences.

34. The method of claim 31, wherein the hash data structure comprises N data cells comprising a first portion storing a predetermined genomic signature for a k-mer seed of the k-mer seeds and a second portion storing a value that corresponds to a location within a reference genomic sequence that matches at least a portion of the k-mer seed from which the predetermined genomic signature was derived.

35. A non-transitory computer-readable medium storing one or more instructions executable by a computer system to perform operations of one or more of claims 1-34.

36. A system, comprising:
one or more processors; and
machine-readable media interoperably coupled with the one or more processors and storing one or more instructions that, when executed by the one or more processors, perform operations of one or more of claims 1-34.