



- (51) **International Patent Classification:**
G06F 17/30 (2006.01) *G06F 9/38* (2006.01)
- (21) **International Application Number:**
PCT/US2015/044502
- (22) **International Filing Date:**
10 August 2015 (10.08.2015)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (71) **Applicant:** HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).
- (72) **Inventors:** KIM, Mijung; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). LI, Jun; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). VISWANATHAN, Krishnamurthy; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US).
- (74) **Agents:** KIRCHEV, Ivan T. et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, CO 80528 (US).
- (81) **Designated States** (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY,

BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

- (84) **Designated States** (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

- as to the identity of the inventor (Rule 4.17(i))
- as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))

Published:

- with international search report (Art. 21(3))

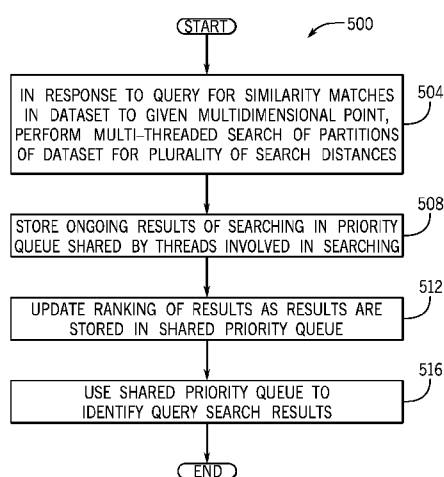
(54) **Title:** USING A SHARED PRIORITY QUEUE FOR MULTI-THREADED SEARCHING

FIG. 5

(57) **Abstract:** A technique includes, in response to a query for similarity matches in a dataset to a given multidimensional point, performing multi-threaded searching of partitions of the dataset for a plurality of search distances; and storing ongoing results of the searching in a priority queue shared by a plurality of threads involved in the searching. The technique includes updating a ranking of the results as results are stored in priority queue; and using shared priority queue to identify query search results.

USING A SHARED PRIORITY QUEUE FOR MULTI-THREADED SEARCHING

Background

[0001] Quite often, a relatively large volume of data is searched for purposes of identifying and retrieving the closest matches to a search query. For example, the data may be time series data that may be, for example, acquired by a sensor. Issues with the sensor may be identified by searching for certain patterns in the time series data. The time series data may be searched for patterns for various other purposes, such as classification, pattern detection, modeling and anomaly detection, as examples.

Brief Description Of The Drawings

[0002] Figs. 1 and 6 are schematic diagrams of multi-core machines to concurrently process multiple multi-dimensional queries according to example implementations.

[0003] Fig. 2 is an illustration of multi-threaded searching according to an example implementation.

[0004] Fig. 3A is an illustration of multi-threaded query processing according to an example implementation.

[0005] Fig. 3B is an illustration of bit vector processing to identify candidates for similarity matches according to an example implementation.

[0006] Fig. 4 is a flow diagram illustrating a bit vector-based candidate scanning technique for multi-threaded searching according to an example implementation.

[0007] Figs. 5 and 10 are flow diagrams illustrating techniques to use a priority queue for multi-threaded searching according to example implementations.

[0008] Fig 6 is a schematic diagram of a multi-core machine according to an example implementation.

[0009] Fig. 7 is a schematic diagram of a multi-threaded search system according to an example implementation.

[0010] Fig. 8 is an illustration of locality sensitive hash (LSH)-based index building according to an example implementation.

[0011] Fig. 9 is an illustration of an LSH-based index search according to an example implementation.

Detailed Description

[0012] Systems and techniques are described herein for purposes of processing a single query or multiple queries concurrently on a multiple core, or "multi-core" machine. In this context, a "multi-core" machine refers to a physical machine that contains a computing component with at least two independent central processing units (CPUs), or "processing cores." A given processing core is a unit that is constructed to read and execute machine executable instructions. In accordance with example implementations, the multi-core machine may contain one or multiple CPU semiconductor packages, where each package contains multiple processing cores.

[0013] The use of a multi-core machine to process a search query, as disclosed herein, allows relatively time efficient searching of a relative large volume dataset for purposes of identifying and retrieving matches to the search query. As an example, the dataset that is searched may be time series data, or data that is derived from, for example, the time sampling of a particular value. As an example, a sensor may acquire time series data. It is noted, however, that the techniques and systems that are disclosed herein may be applied to relatively large volume data other than time series data. For example, the systems and techniques that are disclosed herein may be applied to processing search queries on any high dimensional data (multimedia image or video data, as other examples).

[0014] Identifying and retrieving time series segments that are similar to a segment that is specified by a search query may be useful for such purposes as classification, pattern detection, modeling, fault diagnosis and anomaly detection, as well as for other purposes. Performing a relatively time efficient search may allow the construction of better models; better pattern detection; faster fault analysis; more rapid classifications and more timely detection of anomalies. Other and different advantages may be achieved in accordance with further implementations.

[0015] In accordance with example implementations, a multi-core processing machine may be used for purposes of performing multi-threaded searching to process a query. In general, the query describes a similarity search task for searching a database of points S based on a multi-dimensional query point q and returning a specified number of points from the database that are closest to the multi-dimensional point q , where "closeness" refers to a distance measure, such as Euclidean distance, or another search metric. In accordance with example implementations that are described herein, the goal for a given search task for a given query is to identify the K closest points in S to the multi-dimensional point q .

[0016] One way to perform multi-threaded searching for a given query is to allow multiple threads to search different partitions of the dataset, use the threads to independently identify a set of closest points to the multi-dimensional point q , and then merge the closest sets of points identified by the threads to derive the K closest points. Such an approach, however, may incur considerable processing delays and bandwidth, especially, if multiple queries are being concurrently processed.

[0017] In accordance with example implementations that are disclosed herein, a multi-core machine has features that enhance the processing of the above-described queries and particularly allows for low latency, concurrent processing of multiple queries. In particular, in accordance with example implementations, the multi-core machine has a priority queue that is shared by all of the threads performing a multi-threaded search. In this manner, the threads may concurrently update the shared priority queue as the threads are finding similar matches in a manner that maintains the top K search results found by all of the threads. In this context, "concurrently" refers to multiple threads updating the shared priority queue while the threads are in the process of determining ongoing results. As described herein, the concurrent updating may involve the use of a lock that controls when a given thread may update the shared priority queue. The use of the shared priority queue eliminates an operation to merge individual search results after the thread searching is complete for purposes of identifying the top K search results, while at the same time, to allow each thread to have an updated view of the current top K search results. This updated view allows each thread to more efficiently conduct its own search, by abandoning the search on the candidates that will not contribute to better search results than the currently identified top K search results. In accordance with example implementations, another feature of the multi-core

machine is a bit vector block, which enhances a thread's ability to quickly scan for data points that have been identified as candidate similarity matches and retrieve the data for these data points for further evaluation.

[0018] The techniques and systems described herein may be advantageous for searching a high-dimensional dataset. An example of such a dataset may be an image dataset. For an image similarity search, the image may be represented by a vector of float numbers, called "feature vector." The vector size is the dimension. The dimension size for the image may be in the hundreds, thousands, or even higher.

[0019] Referring to Fig. 1, as a more specific example, a multi-core machine 100, in accordance with some implementations, may process R queries 50 (example queries 50-1, 50-2... 50-R, being detected in Fig. 1, where R is equal to or greater than one) at a given time. To process a given query 50, the multi-core machine 100 assigns a group of threads 104 to process the query 50 from a thread pool 102. In accordance with example implementations, the multi-core machine 100 determines the particular number of threads 104 to process a given query 50 based on such criteria as a targeted query latency and targeted query throughput, which may be user-specified criteria.

[0020] As described further herein, in accordance with example implementations, the assigned threads 104 are assigned to search different, assigned partitions of a multi-dimensional dataset 109 and determine candidate similarity matches for the assigned partitions. In accordance with example implementations, a given thread 104 identifies candidate similarity matches in its assigned partition using hashing, such as hashing that involves using a locality-sensitive hash (LSH), for example. In accordance with example implementations, the assigned threads 104 identify the candidate similarity matches in an index retrieval phase, and subsequent to the index retrieval phase, each thread 104 scans the bit vector to identify candidate similarity matches so that the candidate similarity matches may be then evaluated to determine whether any of the candidate matches are a top K search result.

[0021] In accordance with example implementations, the multi-core machine 100 includes a bit vector block 108 to enhance scanning of the candidate similarity matches. In this manner, the assigned threads 104 for a given query search use a bit vector block 108 for

purposes of scanning candidate similarity matches. Fig. 1 depicts multiple bit vector blocks 108 for the scenario in which the multi-core machine 100 is concurrently processing multiple queries 50.

[0022] In accordance with example implementations, the bit vector block 108 contains binary indicators, or bits. Each bit is associated with a particular data point of the dataset 109 and indicates, or represents, whether the associated data point is a candidate similarity match or not. In this manner, by scanning a portion (several bits, for example) of the bit vector block 108, a thread 104 may readily determine whether the associated data points are candidate similarity matches (as indicated by corresponding bit values of "1," for example) or not (as indicated by corresponding bit values of "0," for example).

[0023] Due to the sparseness of candidate similarity matches among the data points, relatively large portions of the bit vector block 108 may be scanned quickly. In this manner, if a given thread 104 reads multiple bytes in a single read operation and the read value is "0," then the thread 104 learns that the corresponding set of associated data points are not candidate similarity matches. Thus, in accordance with example implementations, if the thread 104 reads a "0" for a corresponding portion of the bit vector 108, the thread may bypass that portion entirely without further scanning. If however, the thread 104 reads a nonzero value for a given portion of the bit vector 108, then thread 104 may step bit by bit through that portion, sorting out the associated data points that are and are not candidate similarity matches.

[0024] The threads 104 that are assigned to process a given query 50 may process the bit vector 108 in parallel, with each thread 104 processing a block of the bit vector 108, which corresponds to the data points in the thread's assigned partition. In accordance with example implementations, the bit vector is associated with a particular data type, such as a short integer or a long integer. In general, a relatively larger data type (a long integer, for example) may be used, as opposed to a relatively shorter integer type (a short integer, for example) for sparser data for purposes of bypassing relatively larger sets of data points to improve scanning efficiency.

[0025] In accordance with example implementations, another feature of the multi-core machine 100 that may be used to enhance concurrent query processing is a priority queue

(called a "shared priority queue 106" herein), which is shared by the threads 104 that are assigned to process a given query 50. Fig. 1 depicts multiple shared priority queues 106 for the scenario in which the multi-core machine 100 is concurrently processing multiple queries 50. In accordance with example implementations, a "priority queue" refers to memory region that contains a collection of data entries (here, search results or pointers to search results as well as their distances from the search query), the data entries are arranged in an order, and each data entry has a relative order, or priority. In accordance with example implementations, each data entry is a search result, and the "priority" of the data entry refers to the closeness (Euclidean distance or other search metric) of the search result to the data point q that is specified by the query 50. The priority queue 106, in accordance with example implementations, contains the current top K search results for the threads 104, and the "top priority" search result refers to the farthest of the top K results from the data point q, or the "top" of the queue 106. Therefore, in accordance with example implementations, if a given thread 104 has a search result that is closer than the top priority search result in the priority queue 106, then the thread 104 may remove the top priority search result and add the new search result to the priority queue 106.

[0026] In accordance with some implementations, the priority queue 106 may be formed by a data structure called a "heap." Other data structures may be used to form the priority queue 106, in accordance with further implementations.

[0027] As also depicted in Fig. 1, in accordance with example implementations, the multi-core machine 100 includes local memory nodes 110. A given local memory node 110 has a set of processing cores 112 (one or multiple central processing unit (CPU) cores) and a local memory that is readily accessible by the processing cores 112. In accordance with some implementations, the threads 104 that are assigned to process a given query 50 are software threads that execute on one or multiple processing cores 112 of a single, local memory node 110; the shared priority queue 106 are shared by the assigned threads 104, and the shared priority queue 106 is disposed in a local memory of the local memory nodes 110. For these example implementations, different queries 50 may be concurrently processed by threads 104 executing on the same local memory node 110 and/or executing on different local memory nodes 110. As a more specific example, in accordance with some implementations, the multi-core machine 110 may have a non-uniform memory access architecture (NUMA); and

the local memory nodes 110 may be NUMA nodes for example, with each NUMA node having a local memory.

[0028] Fig. 2 is an illustration 200 showing stages, or phases, used by the assigned threads 104 to process a given query 50. For the following example implementation, it is assumed that there are n threads 104 that are assigned to process the query 50. Referring to Fig. 2 in conjunction with Fig. 1, in particular, in accordance with example implementations, the multi-core machine 100 performs three phases to process the query 50 (in the following order): a hash computation phase 202, an index retrieval phase 208 and a candidate evaluation phase 214. In accordance with example implementations, all of the assigned threads 104 operate in parallel to perform each phase, with barriers being used to impose a wait for all threads to complete before the next phase begins.

[0029] More specifically, in the hash computation phase 202, the assigned threads 104 determine, or compute, hashes and use the hashes to look up indices to identify candidate similarity matches for their assigned partitions of the dataset 109. The threads 104 may employ the use of matrix multiplication. In particular, block-based multiplication may be leveraged by subdividing the original matrix multiplication into matrix multiplications on several smaller blocks. In this manner, the matrix multiplications on each block may be performed by an associated thread 104 such that the threads 104 perform the matrix multiplications in parallel, and then the smaller matrix multiplication results may be combined as illustrated at reference numeral 204. As also depicted in Fig. 2, a barrier 206 may be employed for purposes of imposing a wait for all threads 104 to perform the hash computation before the index retrieval phase 208 begins. The hash computation phase 202 produces L hash tables, as further described below.

[0030] For the index retrieval phase 208, the L hash tables are divided equally among the n threads 104, and each thread, thus gets assigned a subset of the L/n tables. The index retrieval 208 includes the threads 104 retrieving the indices of the data points from the hash tables in parallel (at reference numeral 207) and then searching 210 to find similarity match candidates based on the hash tables. In the index retrieval phase 208, the threads 104 update the bit vector block 108 with the corresponding binary values to indicate whether associated data points are candidate similarity matches or not. As illustrated at reference number 212 of Fig. 2, a barrier is in place to wait for all of the threads 104 to perform the index retrieval and

thus, provide a set of candidates before the query processing proceeds to the candidate evaluation phase 214.

[0031] Referring to Fig. 3A in conjunction with Fig. 2, in the candidate evaluation phase 214, the bit vector block 108 is used, where each bit represents a data point and indicates whether the associated data point is a candidate similarity match or not. As depicted in Fig. 3A, for the candidate evaluation phase 214, the threads 104 operate in parallel (as depicted at 216 in Fig. 2) to split the bit vector block 108 into n parts. Thus, each thread 104 (threads 104-1, 104-2 and 104- n , being depicted as examples in Fig. 3A) processes a sub block 302 of the bit vector block 108. If the bit vector block 108 is assumed to have N blocks (where each block refers to the chunk of data read in the scanning operation), then each sub block 302 contains N/n blocks 304. Each thread 104 in the candidate evaluation phase 214 thus, reads the sub blocks 304 one at a time, and if a given read value is "0," the thread ignores the corresponding data points. Otherwise, if a thread 104 reads a non-zero value for a sub block 304, the thread 104 further processes the bits of the sub block 304 for purposes of determining which of the corresponding data points are candidate for similarity matches.

[0032] As depicted in Fig. 3A, for candidate similarity matches, a given thread 104 provides a list 305 of candidates 306 and then evaluates the candidates using a search metric (a Euclidean distance, for example) for purposes of determining whether the thread 104 should update the top K results that are stored in the the global, or shared, priority queue 106. As depicted in Fig. 2, in accordance with example implementations, the parallel evaluation of the candidate similarity matches continues (as depicted at 217) by the threads 104 until the threads 104 have identified results for their respective data partitions. A barrier 218 may be imposed to cause all of the assigned threads 104 to wait for completion of the candidate evaluation.

[0033] Fig. 3B depicts bit vector processing 350 by a thread 104 (i.e., scanning of a portion of the bit vector block 108), in accordance with an example implementation. For this example, the thread 104 processes a bit vector block 302 that contains bits that are associated with corresponding data points. The thread 104 reads a first example sub block 304-1, and because at least one of the bits of the sub block 304 is nonzero, the read operation returns a nonzero value. The thread 104 then proceeds to processing 370 the sub-block 304-1 bit by bit: the bits that have nonzero values for this example are associated with candidates 306,

and as such, the thread 104 copies 360 the data point values to populate a candidate list 305 for further evaluation. For the example of Fig. 3B, the thread 104 processes the sub blocks 304 along processing path 372 to next read an example sub block 304-2, which contains bits that are all zeros. Therefore, the thread 104 reads a zero value, which means that the thread 104 proceeds along processing path 372 to read the next sub block 304, and so forth. Thus, in accordance with example implementations, the thread selectively bypasses (i.e., either processes or skips) the retrieval of values for data points corresponding to a subset of bit vectors based at least in part on whether the bits of the subset are the same (are all zero bits, for example).

[0034] Referring back to Fig. 3A, in the candidate evaluation phase 214 (Fig. 2), each thread 104 evaluates its candidate list 305 for purposes of determining whether one or multiple candidates 306 of the candidate list 305 should be stored in the shared priority queue 106 (i.e., evaluate whether a given candidate 306 is a current top-K result for the shared priority queue 106). In accordance with example implementations, each thread 104 reads the top of the queue 106 to determine whether its current candidate should be in the top-K or not, and whenever the candidate distance is less than the largest distance result in the queue 106, the thread 104 inserts the candidate into the queue 106. In accordance with example implementations, the multi-core machine 100 imposes a lock 320 on the shared priority queue 106 for purposes of avoiding conflicts when several threads 104 attempt to read/write to the global heap 106 at the same time.

[0035] Thus, referring to Fig. 4, in accordance with example implementations, a technique 400 includes performing (block 404) multi-threaded processing of a hash-based index to identify data points from a plurality of data points, which are candidates for similarity matches for a query. Pursuant to the technique 400, the plurality of data points are represented (block 408) by a plurality of associated binary indicators, where each binary indicator represents whether an associated data point is one of the candidates. According to the technique 400, values for the data points that are candidates are selectively retrieved (block 412) based at least in part on the indicators; and the retrieved values are processed (block 416) to evaluate the candidates to determine a result for the query. In this context, the "selective retrieval" refers to the values being retrieved or not being retrieved, depending on

the corresponding binary indicators (depending on the states of the binary indicators, for example).

[0036] Referring to Fig. 5, in accordance with example implementations, the processing of the query may also involve a technique 500. Pursuant to the technique 500, in response to a query for similarity matches in a data set to a given multidimensional point, a multi-threaded search of partitions of the data set is performed (block 504) for a plurality of search distances from the given multidimensional point. The technique includes storing (block 508) ongoing results of the searching in a priority queue, which is shared by a plurality of threads involved in the searching. Pursuant to the technique 500, a ranking of the results is updated (block 512) as the results are stored in the shared priority queue, and the shared priority queue is used to identify the query results, pursuant to block 516.

[0037] Referring to Fig. 6, in accordance to example implementations, the multi-core machine 100 is a physical machine that is constructed from actual machine executable instructions, or "software," and actual hardware. The hardware of the multi-core machine 100 includes S local memory nodes 110 (local memory nodes 110-1 . . . 110-S, being depicted as examples in Fig. 6), which may be part of, for example, a particular multi-core central processing unit (CPU) package. As depicted in Fig. 6, each local memory node 110 contains Q CPU processing cores 112 (processing cores 112-1, 112-2. . . 112-Q, being depicted in Fig. 6 for each node 110) and a local memory 114.

[0038] The processing cores 112 may experience relatively rapid access times to an associated local memory 614 of the same local memory node 110, as compared to, for example, the times to access the memory 614 of another local memory node 110. In this manner, access to a memory 614 of another local memory node 110 occurs through a memory hub 620 of the machine 100 or another interface, which introduces delays. The local memory 614, in general, is a non-transitory memory, which includes non-transitory memory storage devices, such as semiconductor devices, phase change devices, random access memory (RAM) devices, dynamic RAM (DRAM) devices, resistive memory devices, flash memory devices, a combination of one or more of these devices, and so forth.

[0039] In accordance with example implementations, each local memory node 110 contains a memory controller (not shown). Fig. 6 also depicts a persistent memory 630 (a non-volatile

memory, such as flash memory, for example) that may be accessed by the processing cores 112 via the memory hub 620. For the example implementation of Fig. 6, the shared priority queue 106 is disposed in the local memory 614, one or multiple processing cores 112 of a given local memory node 110 may execute threads to process a given query, and this processing involves the use of the shared priority queue 106, as described herein.

[0040] Referring to Fig. 7 in conjunction with Fig. 6, in accordance with example implementations, a search system 700 that may be implemented on the multi-core machine 100 includes a coordinator 640; N multi-threaded logical processing units 710 (example logical processing 710-1, 710-2. . . 710-N, being depicted as examples in Fig. 7) and a persistent memory store 760. In accordance with example implementations, the coordinator 640 and processing units 710 are components that are formed by the execution of machine executable instructions, or "software," by one or more of the processing cores 112 of a given local memory node 110 (see Fig. 6).

[0041] The machine executable instructions may be stored, for example, in the local memory 614 of the local memory node 110, as represented at reference numeral 615 in Fig. 6. Thus, in accordance with example implementations, instructions that are stored in a non-transitory computer readable storage medium may, when executed, cause a processor-based machine, or computer, to, in response to a query for similarity matches in a dataset to a given multidimensional point, perform multi-threaded searching of partitions of the dataset for a plurality of search distances; and use a priority queue for the multi-threaded searching, where using the priority queue includes storing ongoing results of the searching in a priority queue shared by a plurality of threads involved in the searching; updating a ranking of the results as results are stored in priority queue; and using the shared priority queue to identify query search results.

[0042] In accordance with further example implementations, one or multiple of the processing units 710 and the coordinator 640 may be formed from hardware (dedicated integrated circuits, for example).

[0043] The persistent memory store 760 may be formed by storage provided by the persistent memory 630 (see Fig. 6), for example. As depicted in Fig. 7, a dataset 762 to be

searched by queries 50 as well as parameters 764 for the query searches may be stored in the persistent memory store 760, in accordance with example implementations.

[0044] In accordance with example implementations, the processing unit 710 processes a given query 50, and as such, the processing units 710 for the example implementation of Fig. 7, are concurrently processing N queries 50. The coordinator 640 launches a given processing unit 710 for a given query 50 and sets up the parameters employed by the processing unit 710 for the searching. In accordance with example implementations, the processing unit 710 imposes the barriers between the phases of the search (as depicted in Fig. 2), controls the locking of the priority queue 106, communicates commands to start and stop the multi-threaded searching for a given query, communicates query parameters to the processing unit 710, assign threads 104 from the thread pool 102 (see Fig. 1) to the processing units 710, communicates search or query results 220, from the processing units 710, and so forth.

[0045] For example implementations that are disclosed herein, the processing units 710 use locality sensitive hashing (LSH) to perform the searching, although other search techniques may be used, in accordance with further example implementations. An LSH-based index is a data structure that stores a collection D of points in a d-dimensional Euclidean space in such a manner that given a query point, the data structure, with relatively high probability, returns points in D that are within distance R of the query point; and the data structure does not return too many points that are at distance greater than cR. Here, "R" represents a search distance, and "c" represents a scaling factor, such that "c" determines the space that is occupied by the LSH-based index. The LSH-based index is applied to the problem of finding the closest time series segment to a given time series segment in the following manner, in accordance with example implementations. A processing unit 710 builds its index table located in the native memory that can be directly accessible by the processing unit 710, in accordance with example implementations.

[0046] The following describes index hashing and the use of the hashing to identify candidate similarity matches, in accordance with example implementations. The length of the search pattern may be fixed, and every point in the dataset 762 may be indexed. Assuming that "D" refers to a set of candidate data points, a sequence of locality sensitive hash indexes are built, with each being designed for a different value of search distance called

"R." It is assumed that $R_0 < R_1 < R_2$, etc., denote the values of R for the indices. The entire time series data is partitioned into different corresponding partitions, and a separate set of LSH-based indices is constructed for each of the partitions. These indices can share the same R values, in accordance with example implementations.

[0047] Given a query, a search is first performed within each partition, with each of the indices, which corresponds to search distance R_0 for points that are within distance R_0 . After all of the points from all the partitions discovered with the initial search are obtained, a check is performed for purposes of determining if the number of points obtained is greater than or equal to "K" (where "K" represents the number of closest points desired to be returned). If this is not the case, the procedure is repeated for each partition for the search distance R_1 and so on. In accordance with example implementations, the search is terminated after at least K points are found. As the search progresses, each of the candidate points that are retrieved are compared with the query point, and the K closest points are returned.

[0048] In accordance with example implementations that are disclosed herein, the search problem is solved for a fixed length, or dimension, for the query search pattern, in this case called "d." In further implementations, the problem may be solved for multiple query lengths, with separate indexes being constructed for each of the query lengths.

[0049] Fig. 8 is an illustration 800 of locality sensitive hashing -based index building according to an example implementation. Each point 818 to be indexed is a d-dimensional real vector. In general, the LSH-based index building generates k random zero-mean, unit variance Gaussian d-dimensional real vectors 824, and computes the inner product of 818 with each of the k random vectors. The results of the k inner products are quantized to obtain a k dimensional integer vector 828. The parameter k is selected so that a point at a distance more than $c \cdot R$ from a given point 818 is mapped to a different vector 828 with high probability. The points are stored in an LSH index-based collection 840 of hash tables (L example hash tables 840-1, 840-2 . . . 840-L, being depicted as examples in Fig. 3). Each hash table in the collection 840 contains groupings, or buckets 860, of the points; and the buckets 860 are indexed by the vectors 828. Closer points fall within the same group, or "bucket 860," within the hash table 840. The above-described process is repeated L times, which produces L hash tables 840.

[0050] Fig. 9 is an illustration 900 of an LSH-based index search of data in response to a query using hash tables 840 that are constructed from the data, in accordance with example implementations. At query time, the same identical LSH hash function described in connection with Fig. 2 is applied to a multi-dimensional search query point 924, to determine L vectors 928, one for each of the L hash tables in the collection 340. The resulting search of a given hash table may identify a bucket 360 that contains a corresponding set of candidate matches, which is also referred to herein as a "bucket 950." Fig. 9 also illustrates merging of the buckets 950 derived from searches of multiple hash tables into a merged bucket 960 of candidate results. The merged candidate results 960 may further be searched (using a straightforward exhaustive search, as an example) based on distance to determine the closest matches.

[0051] The search may involve the evaluation of the query point 924 at different distances search distances R_0 , R_1 , R_2 , and so forth. Thus, multiple searches may be performed, each at a different search distance R. The parameter L is selected so that if the set of high-dimensional points that are indexed contains any point within distance R of the query point, then the point is retrieved with a relatively high probability.

[0052] To summarize, in accordance with example implementations, a technique 1000 (see Fig. 10) to process a query for similarity matches in a dataset to a given multidimensional point includes, in response to the query, performing multi-threaded searching of partitions of the dataset for a plurality of search distances, pursuant to block 1002. The technique includes, pursuant to block 1006, using a shared priority queue for the multi-threaded searching, including, for each thread, selectively unlocking the shared priority queue and updating the shared priority queue based on ongoing search results, and using the shared priority queue to identify the query search results.

[0053] Other implementations are contemplated, which are within the scope of the appended claims. For example, although locality-sensitive hashing is discussed herein, other hashing-based techniques may be employed to identify candidate similarity matches, in accordance with further implementations. For example, in accordance with further example implementations, angular hashing may be used.

[0054] While the present techniques have been described with respect to a number of embodiments, it will be appreciated that numerous modifications and variations may be applicable therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the scope of the present techniques.

WHAT IS CLAIMED IS:

- 1 1. A method comprising:
2 performing multi-threaded processing of a hash-based index to identify data points
3 from a plurality of data points which are candidates for similarity matches for a query;
4 representing the plurality of data points by a plurality of associated binary indicators,
5 wherein each binary indicator of the plurality of indicators represents whether an associated
6 data point of the plurality of data points is one of the candidates;
7 selectively retrieving values for the data points that are candidates based at least in
8 part on the plurality of binary indicators; and
9 processing the retrieved values to evaluate the candidates to determine a result for the
10 query.
- 1 2. The method of claim 1, wherein representing the plurality of data points as a plurality
2 of associated binary indicators comprises representing the plurality of data points as a block
3 of bit vectors.
- 1 3. The method of claim 2, further comprising scanning the block of bit vectors,
2 comprising reading a subset of the bit vectors, and selectively bypassing retrieval of values
3 for data points corresponding to the subset based at least in part on whether the bits of the
4 subset are the same.
- 1 4. The method of claim 2, wherein the retrieving of values for the data points comprises
2 using different threads to scan different portions of the block of bit vectors.
- 1 5. A system comprising:
2 a multi-core machine comprising a plurality of processing threads partitioned to
3 search partitions of a dataset in response to a query for similarity matches in the dataset to a
4 given multidimensional point;
5 a priority queue; and

6 a coordinator to select a group of the processing threads to process a query;
7 wherein processing threads of the selected group of processing threads:
8 perform searching in partitions of a dataset to identify candidate similarity
9 matches to a given multidimensional point using indices constructed from multidimensional
10 points of the partitions;
11 evaluate results of the searching; and
12 selectively update the priority queue based on the evaluation of the ongoing
13 results.

1 6. The system of claim 5, wherein the processing threads of the selected group of
2 processing threads comprise threads executing on a Non-Uniform Memory Access
3 Architecture (NUMA) node, and the priority queue is stored in a local memory of the NUMA
4 node.

1 7. The system of claim 5, wherein the processing threads of the selected group of
2 processing threads perform the searching before the evaluating results of the searching and all
3 of the processing threads complete searching before any of the processing threads of the
4 selected group of processing threads begins evaluating results of the searching.

1

1 8. The system of claim 5, wherein the processing threads of the selected group of
2 processing threads perform evaluating results of the searching while concurrently selectively
3 updating the priority queue.

1 9. The system of claim 5, wherein the processing threads use locality-sensitive hashes to
2 identify the candidate similarity matches to the given multidimensional point.

1 10. An article comprising a non-transitory computer readable storage medium to store
2 instructions that when executed by a processor-based machine cause the processor-based
3 machine to:

4 in response to a query for similarity matches in a dataset to a given multidimensional
5 point, perform multi-threaded searching of partitions of the dataset for a plurality of search
6 distances; and

7 use a priority queue for the multi-threaded searching, comprising:

8 storing ongoing results of the searching in a priority queue shared by a
9 plurality of threads involved in the searching;

10 updating a ranking of the results as results are stored in priority queue; and

11 using the shared priority queue to identify query search results.

1 11. The article of claim 10, the storage medium storing instructions that when executed
2 by the processor-based machine causes the processor-based machine to identify a predefined
3 number of highest ranked search results.

1 12. The article of claim 10, the storage medium storing instructions that when executed
2 by the processor-based machine cause the processor-based machine to, for given thread of the
3 plurality of threads, selectively unlock the priority queue and update the priority queue while
4 the queue is unlocked.

1 13. The article of claim 10, the storage medium storing instructions that when executed
2 by the processor-based machine causes the processor-based machine to use locality-sensitive
3 hashes to perform the multi-threaded searching.

1 14. The article of claim 10, the storage medium storing instructions that when executed
2 by the processor-based machine causes the processor-based machine to:
3 use hash computation to look up indices to identify candidates for similarity matches
4 to the given multidimensional point;
5 evaluate the candidates; and
6 selectively update the priority queue based on the evaluation.

- 1 15. The article of claim 10, the storage medium storing instructions that when executed
- 2 by the processor-based machine causes the processor-based machine to determine Euclidean
- 3 distances between the candidates and the multidimensional point and evaluate the candidates
- 4 based at least in part on the distances.

1 / 10

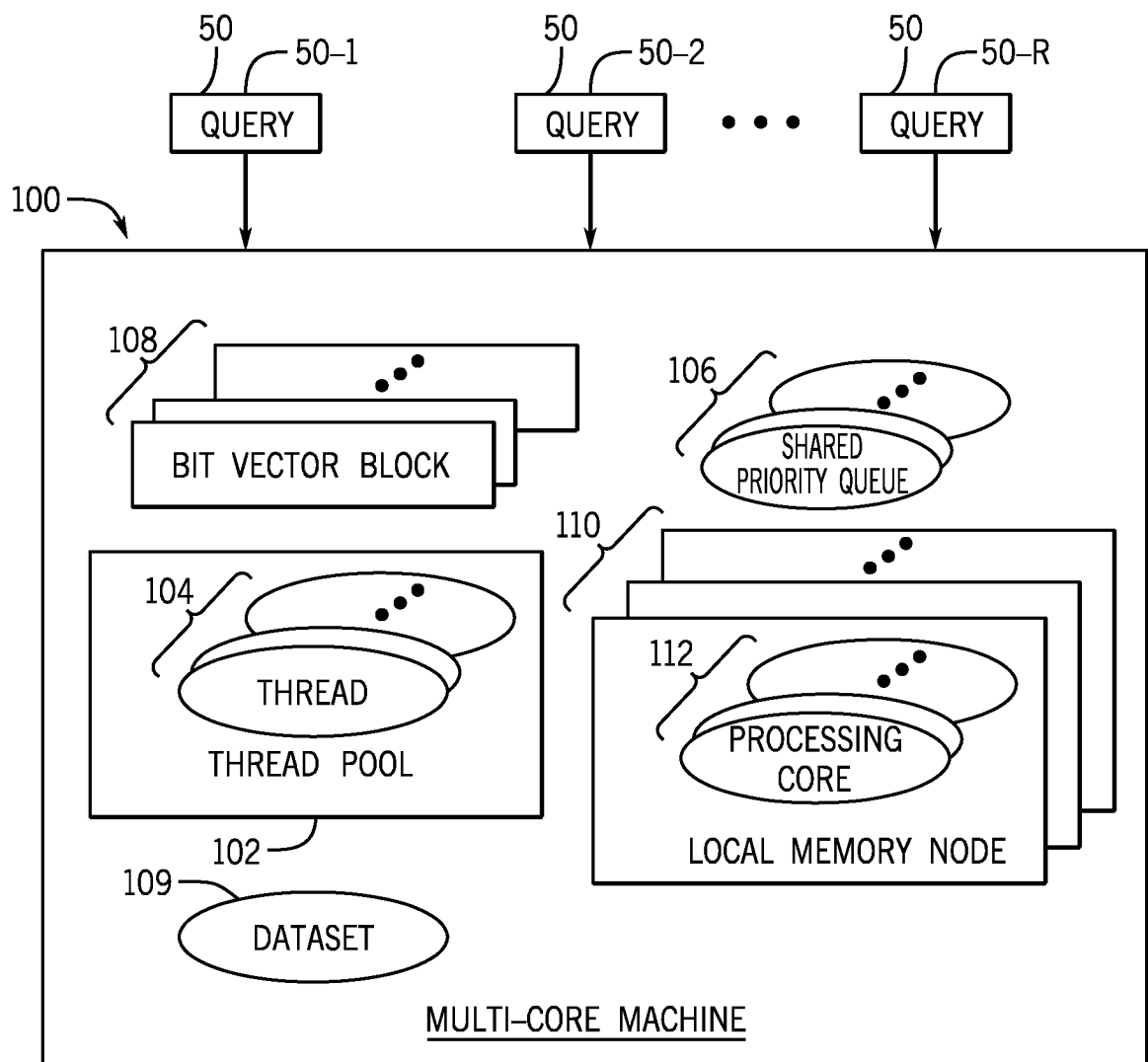


FIG. 1

2 / 10

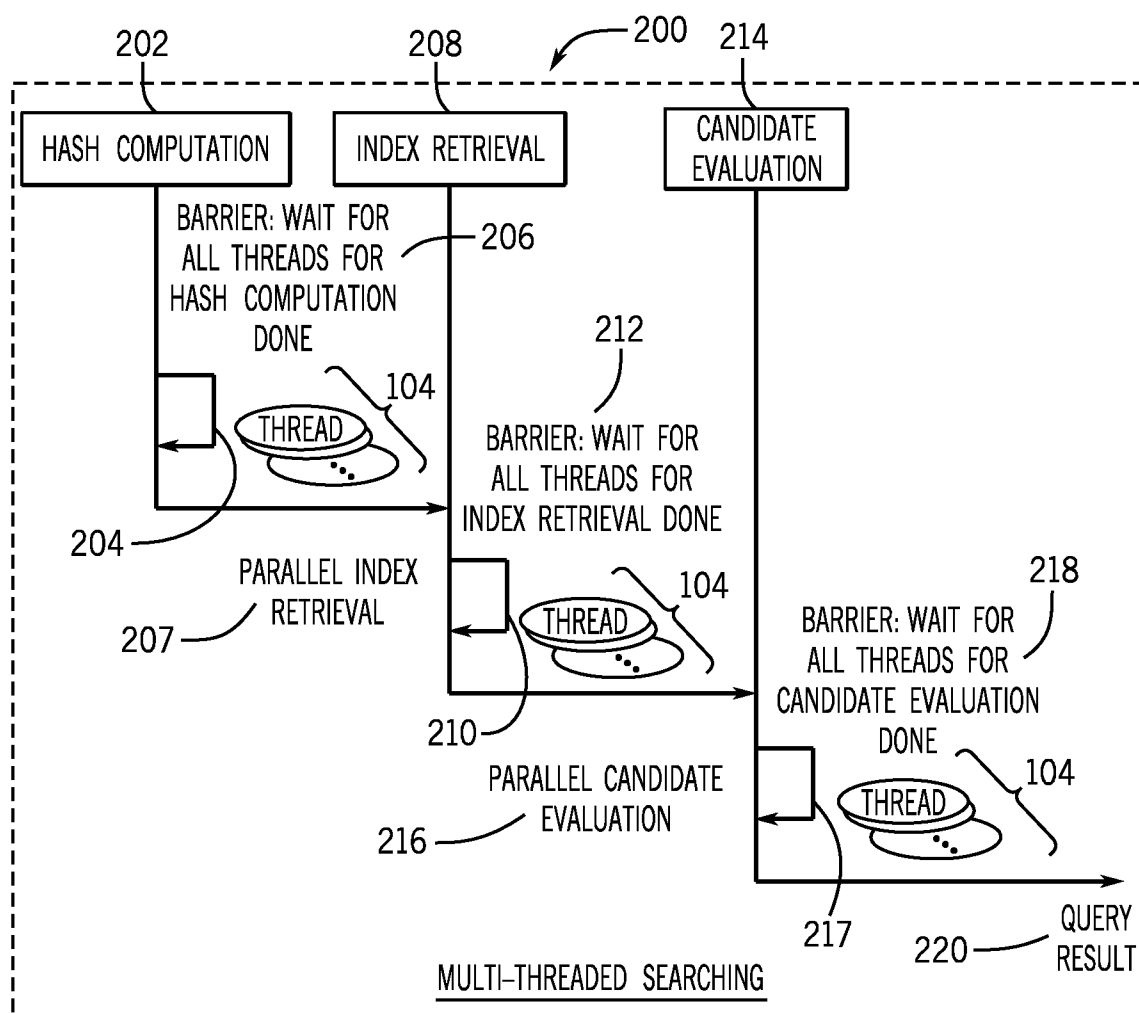


FIG. 2

3 / 10

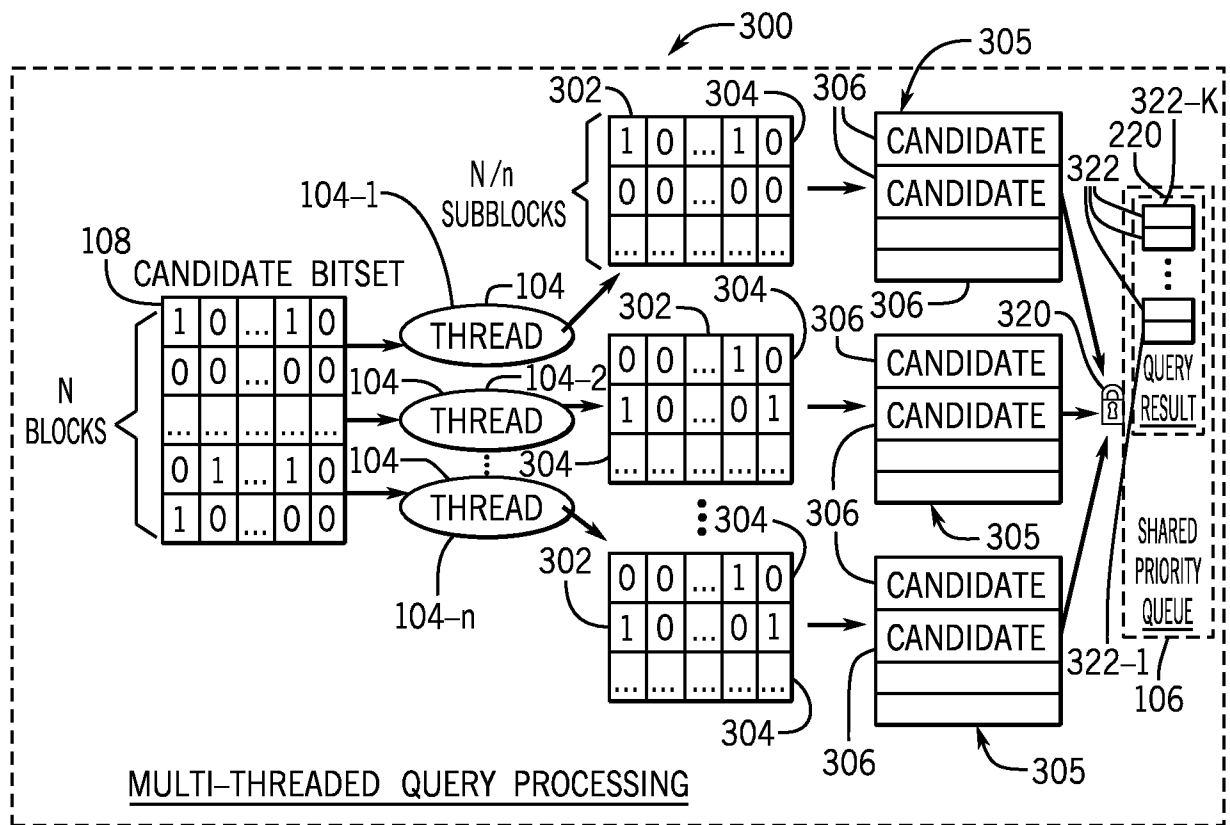


FIG. 3A

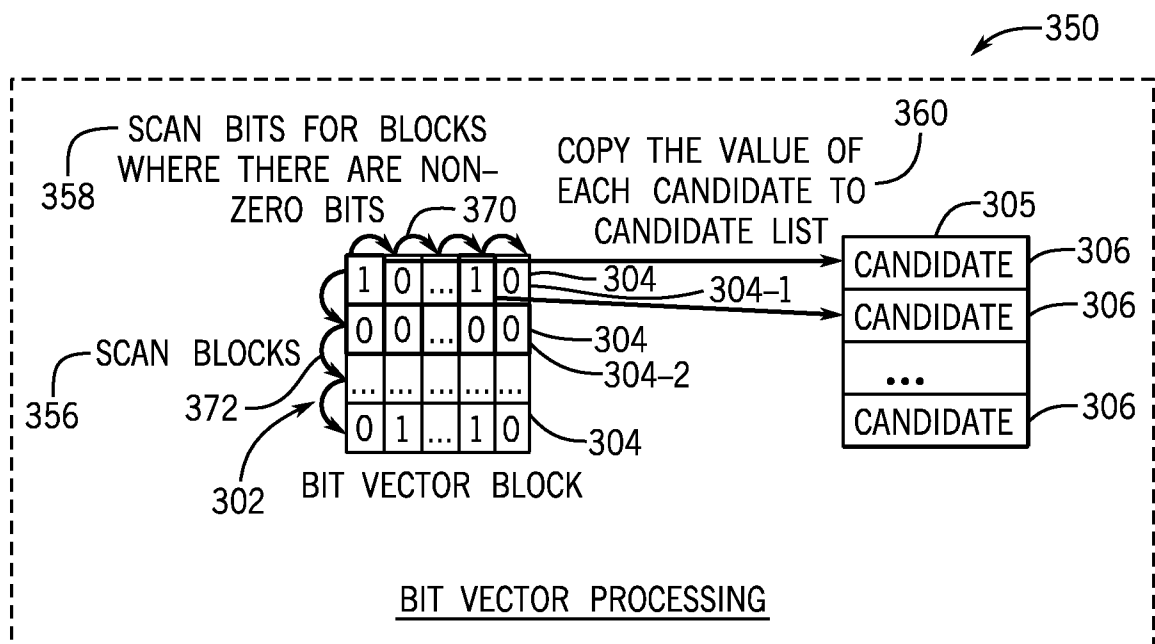


FIG. 3B

4 / 10

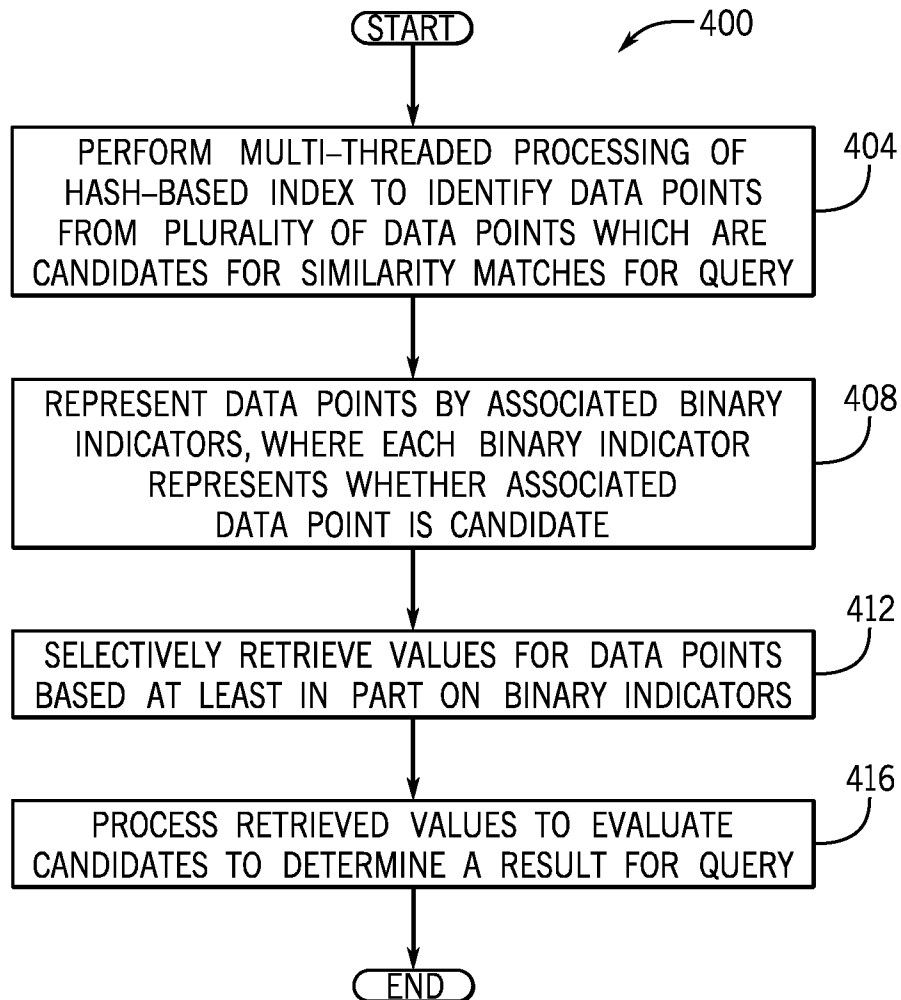


FIG. 4

5 / 10

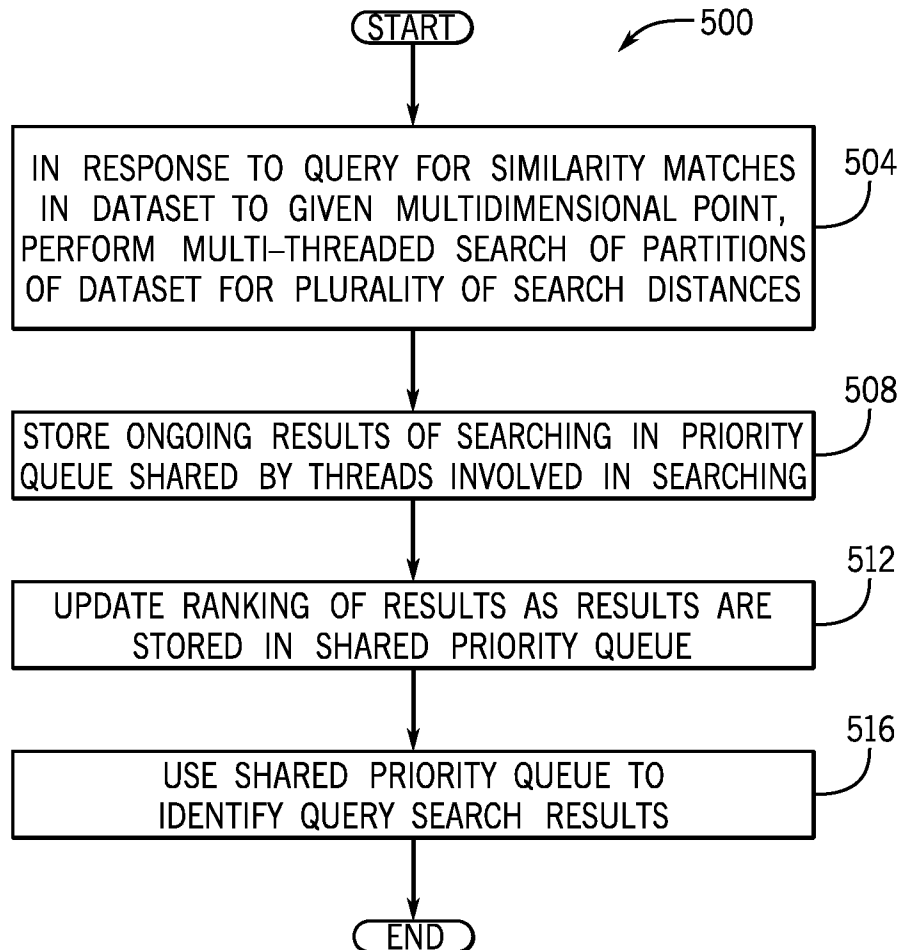


FIG. 5

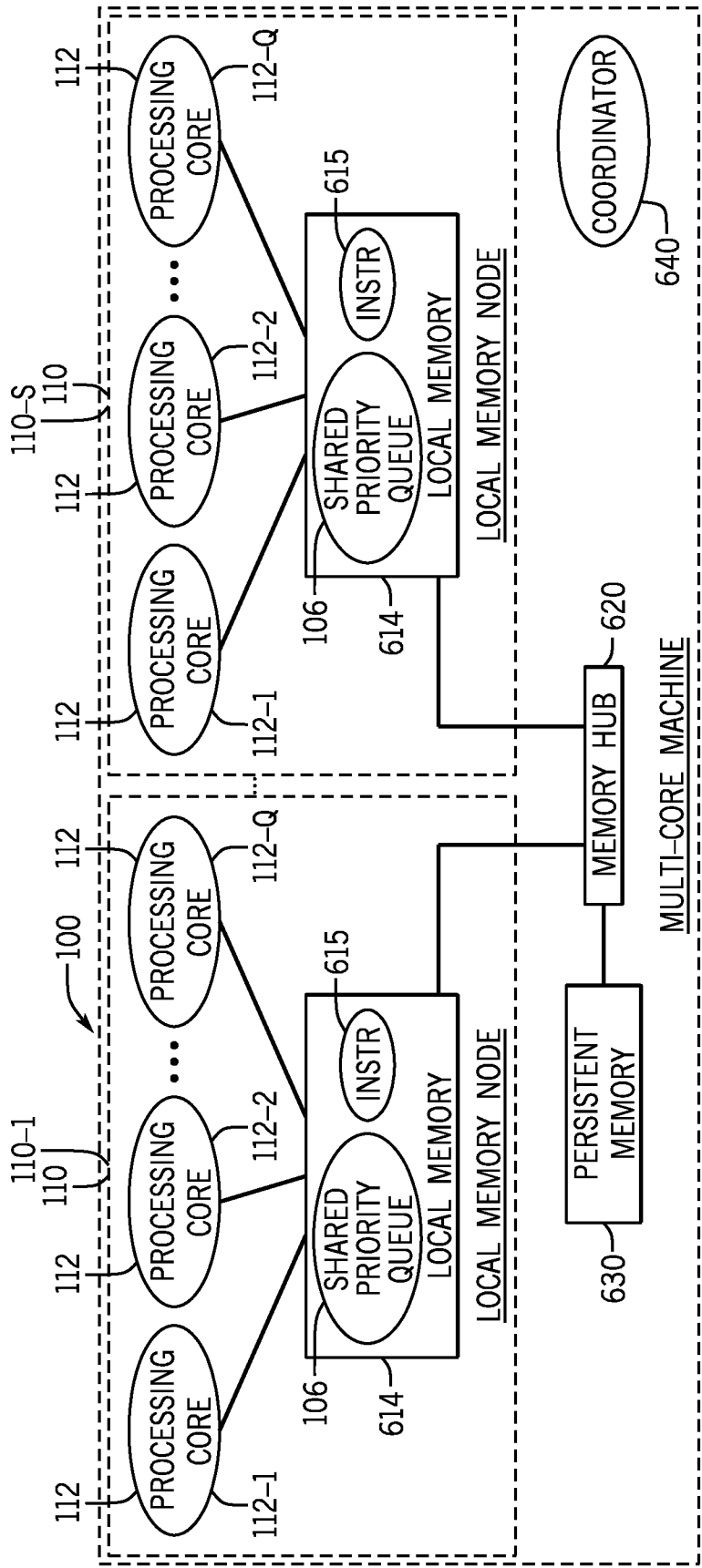


FIG. 6

7 / 10

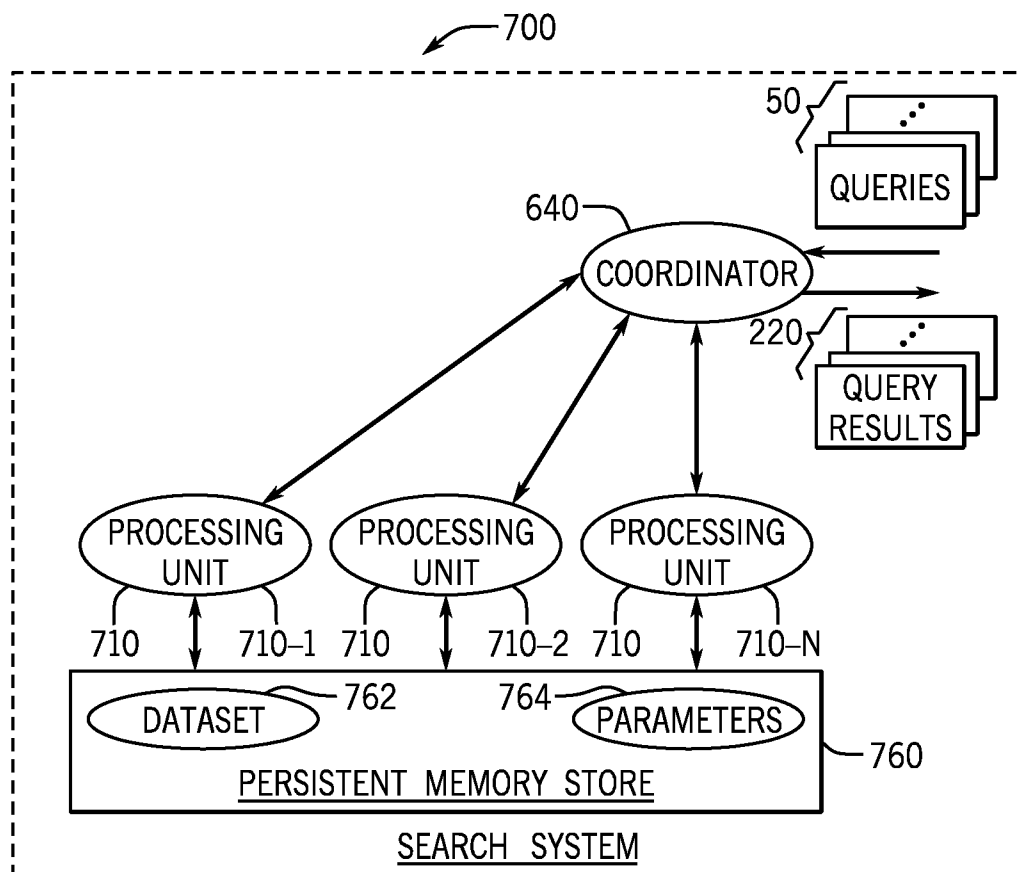


FIG. 7

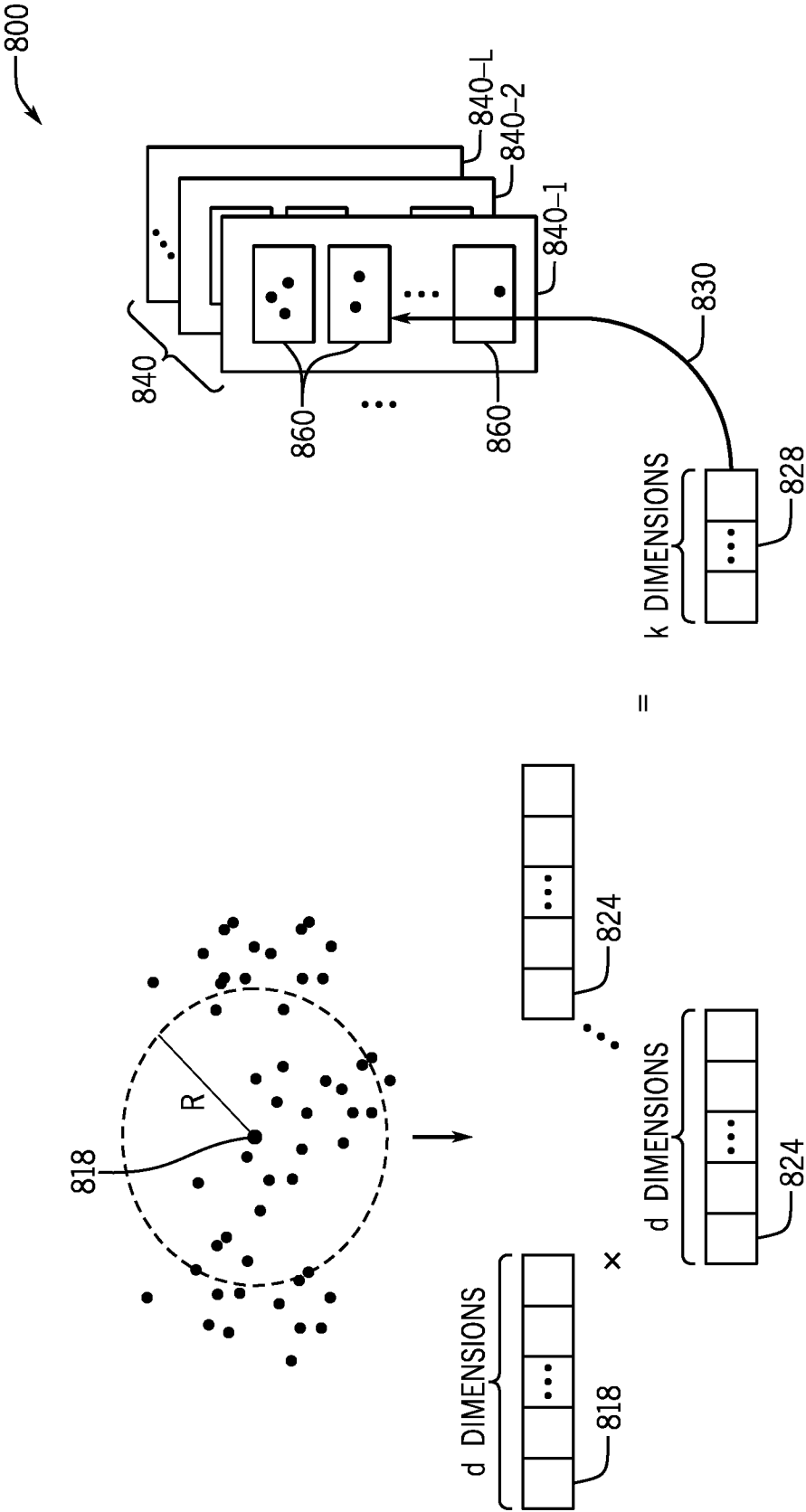


FIG. 8

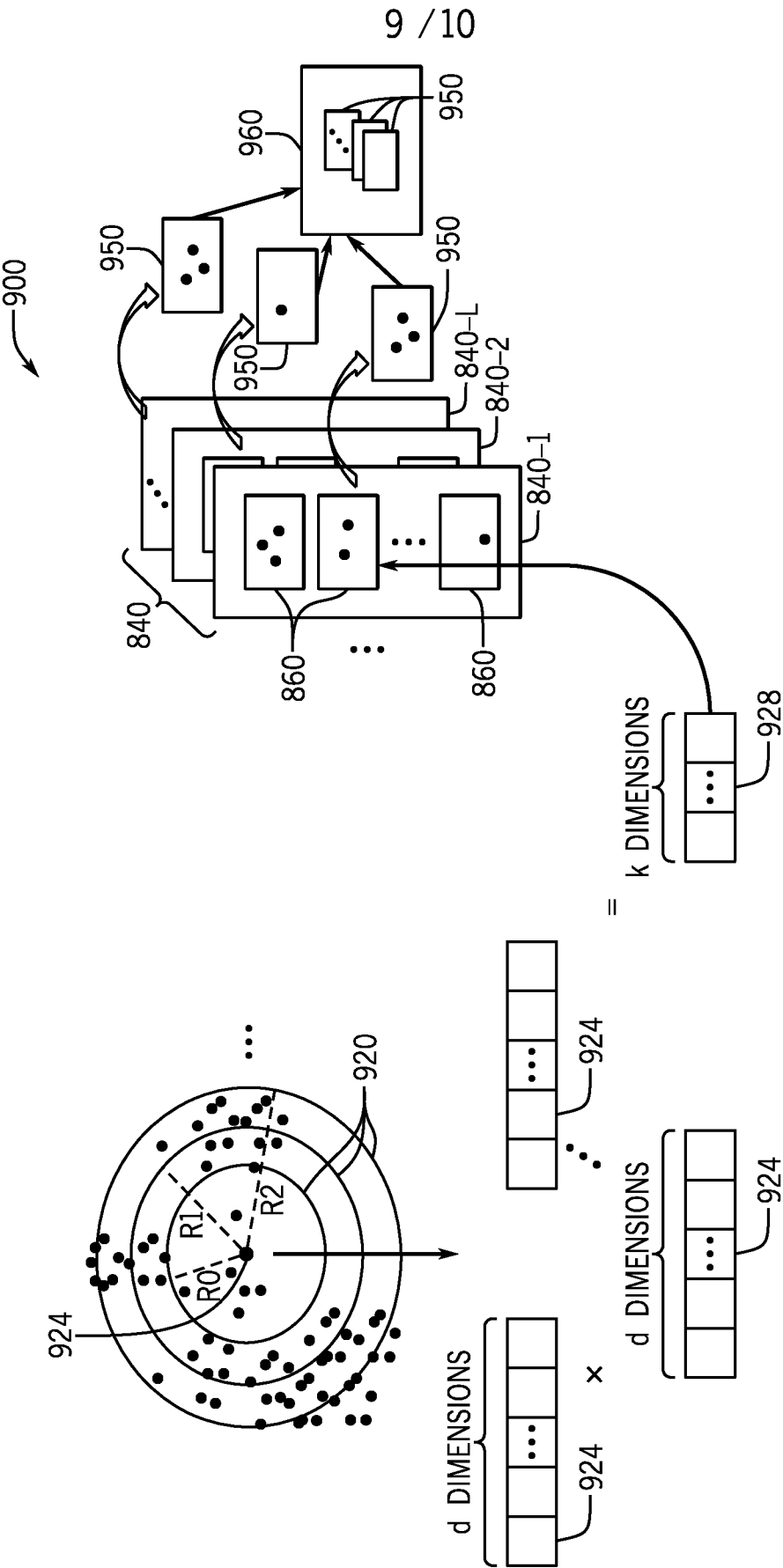


FIG. 9

10 / 10

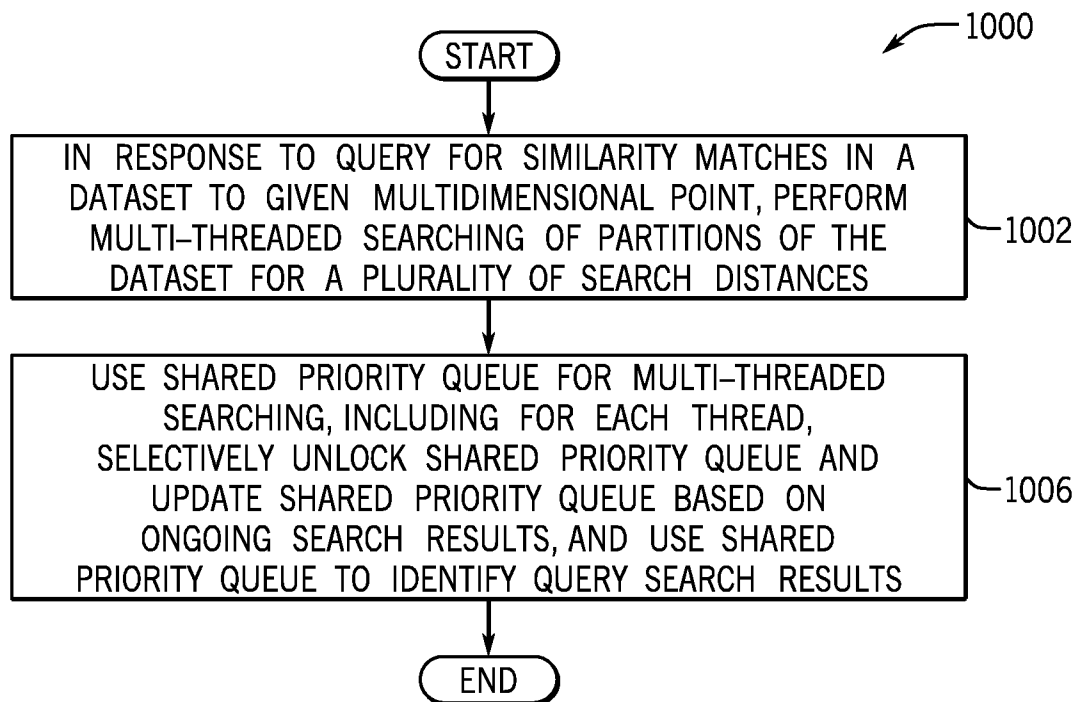


FIG. 10

A. CLASSIFICATION OF SUBJECT MATTER**G06F 17/30(2006.01)i, G06F 9/38(2006.01)i**

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 17/30; G06F 9/38

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) & Keywords: multi-threaded, shared priority queue, search query, LSH, KNN-algorithm, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	AMELIE MARIAN et al., "Adaptive Processing of Top-k queries in XML," In: 2005 IEEE 21st International Conference on Data Engineering, pp. 162-173, 05-08 April 2005 See left column of page 166, line 15 - right column of page 168, line 17; and figure 4.	1-15
A	JIA PAN et al., "Bi-level Locality Sensitive Hashing for K-Nearest Neighbor Computation," In: 2012 IEEE 28th International Conference on Data Engineering, pp. 378-389, 1-5 April 2012 See left column of page 380, line 49 - left column of page 384, line 16; and figure 1.	1-15
A	CAROLINA BONACIC et al., "Building Efficient Multi-Threaded Search Nodes," Proceedings of the 19th ACM international conference on Information and knowledge management, pp. 1249-1258, 26-30 October 2010 See left column of page 1251, line 1 - left column of page 1253, line 13; and figures 1-2.	1-15
A	US 2011-0093491 A1 (PETER ZABBACK et al.) 21 April 2011 See paragraphs [0022]-[0023]; and figures 2 and 4.	1-15

☒ Further documents are listed in the continuation of Box C.☒ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

11 May 2016 (11.05.2016)

Date of mailing of the international search report

11 May 2016 (11.05.2016)

Name and mailing address of the ISA/KR

International Application Division

Korean Intellectual Property Office

189 Cheongsu-ro, Seo-gu, Daejeon, 35208, Republic of Korea

Facsimile No. +82-42-481-8578

Authorized officer

NHO, Ji Myong

Telephone No. +82-42-481-8528



INTERNATIONAL SEARCH REPORT

International application No.

PCT/US2015/044502

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2014-0229473 A1 (MICROSOFT CORPORATION) 14 August 2014 See paragraphs [0053]-[0057] and figure 3.	1-15

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2015/044502Patent document
cited in search reportPublication
datePatent family
member(s)Publication
date

US 2011-0093491 A1

21/04/2011

US 8195648 B2

05/06/2012

US 2014-0229473 A1

14/08/2014

WO 2014-126822 A1

21/08/2014