



- (51) International Patent Classification:
G06F 12/00 (2006.01) *G06F 12/08* (2016.01)
- (21) International Application Number:
PCT/US2018/039121
- (22) International Filing Date:
22 June 2018 (22.06.2018)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
62/523,347 22 June 2017 (22.06.2017) US
- (71) Applicant: DATAWARE VENTURES, LLC [US/US];
701 E. Camino Alberca, Tucson, Arizona 85718 (US).
- (72) Inventors: ZHANG, Rui; c/o Dataware Ventures, LLC,
701 E. Camino Alberca, Tucson, Arizona 85718 (US). SN-
ODGRASS, Richard T.; PO Box 210077, Tucson, Arizona
85721 (US). CONVEY, Christian; c/o Dataware Ventures,
LLC, 701 E. Camino Alberca, Tucson, Arizona 85718 (US).
- (74) Agent: SOLOWAY, Norman P. et al.; c/o Hayes Soloway
PC, 4640 E. Skyline Drive, Tucson, Arizona 85718 (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, DJ, DK, DM, DO,
DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN,
HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP,
KR, KW, 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,

(54) Title: FIELD SPECIALIZATION TO REDUCE MEMORY-ACCESS STALLS AND ALLOCATION REQUESTS IN DATA-INTENSIVE APPLICATIONS

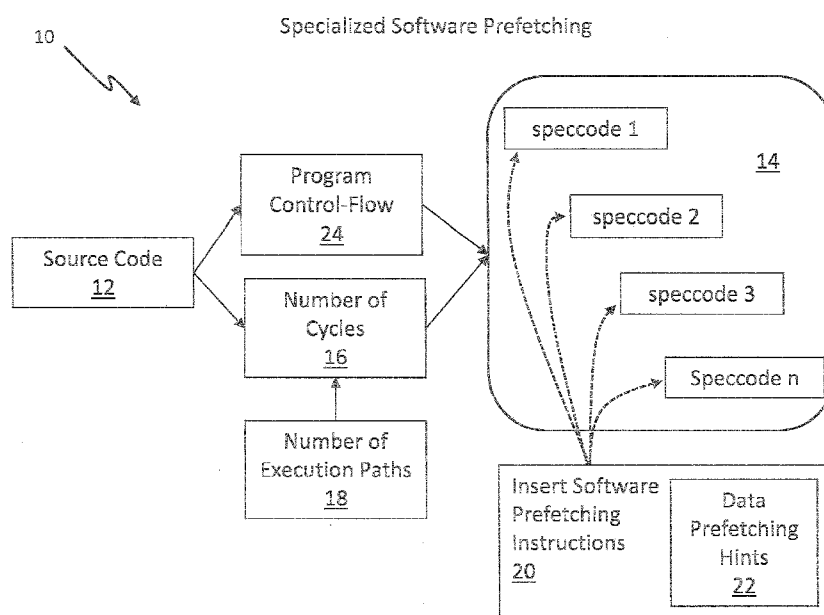


FIG. 1

(57) Abstract: A computer-implemented method and related systems for reducing memory access stalls and memory allocation requests in data-intensive applications are provided. Invariants associated with execution paths that access data in a memory of the data-intensive application are identified. At least one field specialization technique using at least one speccode segment is then applied. The speccode segment exploits the identified invariants, thereby reducing at least one of memory stalls and memory allocation requests in a data-intensive application. The field specialization technique may include specialized software prefetching, a data distribution-based hash function, process to CPU binding, memory segment reuse, or memory layout optimization, or any combination thereof.

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).

Published:

— *with international search report (Art. 21(3))*

FIELD SPECIALIZATION TO REDUCE MEMORY-ACCESS STALLS AND ALLOCATION REQUESTS IN DATA-INTENSIVE APPLICATIONS

SYSTEMS AND DATA-INTENSIVE APPLICATIONS

The present disclosure is generally related to field specialization to reduce memory-access stalls and memory allocation requests in database management systems and data-intensive applications.

A database management system (DBMS) is a collection of software programs that manage the storage and access of data. As the capacity of main memory increases and as its cost reduces, more memory is configured for use by DBMS servers and other data-intensive applications. The benefit of utilizing more memory is that larger volumes of data can be stored in the main memory, thereby significantly improving the efficiency of these applications. This is due to the fact that most or all of the data needed to evaluate a query or perform the requested analysis can be fetched directly from the main memory without having to access the hard disks, which are orders of magnitude slower in data access bandwidth.

However, even with this improvement, main-memory access continues to present a bottleneck in query evaluation in DBMSs and in data-intensive applications. Even though main memory is faster in data-access speed compared to disk access, main memory speed is still substantially slower than the rate at which instructions are being executed by the central processing unit (CPU). In particular, within these code paths, a large portion of the executed instructions performs data reads and writes. When data is read from or written into the main memory, the execution of these memory-related instructions will stall, waiting for the main memory to complete the requests.

Thus, a heretofore unaddressed need exists in the industry to address the aforementioned deficiencies and inadequacies.

Embodiments of the present disclosure provide systems and methods for reducing memory access stalls and memory allocation requests in data-intensive applications, such as in a database management system. In this regard, one embodiment of such a method, among others, can be broadly summarized by the following steps: Invariants associated with execution paths that access data in a memory of the data-intensive application are identified. At least one field specialization technique generating at least one specialized code (termed, "speccode") segment is then applied. The speccode segment exploits the identified invariants, thereby reducing memory stalls and/or memory allocation requests in data-intensive application. The field specialization technique may include specialized

1 software prefetching, a data distribution-based hash function, process to CPU binding,
2 memory segment reuse, or memory layout optimization, or any combination thereof.
3 In one embodiment the data-intensive application further comprises a database
4 management system.
5 In one embodiment where the at least one field specialization technique comprises
6 software prefetching, at least one of: a control-flow invariant; and a number of cycles
7 invariant are identified, preferably by inserting software prefetching instructions into the
8 at least one speccode segment, optionally by using at least one data prefetching hint to
9 determine a prefetching level.
10 In another embodiment the at least one field specialization technique comprises a data
11 distribution-based hash function, and preferably, further comprises transforming an input
12 value used as a key by a hash table with at least one of: a complete value distribution
13 hash function and a refined value distribution hash function, wherein the complete value
14 distribution hash function preferably relies on a value distribution of all rows of a
15 column of an underlying table upon which the hash table is built, and the refined value
16 distribution hash function relies on a value distribution of a subset of all rows of the
17 column of the underlying table upon which the hash table is built.
18 In another embodiment, the at least one field specialization technique comprises a CPU
19 binding of processes, wherein the CPU binding of processes preferably is dependent on:
20 a presence of inter-process data transfer; and
21 a size of the data transfer being substantial.
22 In yet another embodiment the at least one field specialization technique comprises a
23 memory segment reuse processes, wherein the memory segment reuse processes
24 preferably comprises:
25 use memory access analysis, including both static pointer alias analysis
26 and dynamic profiling to track all the reads and writes to a memory
27 segment;
28 determine if there is a portion of a lifetime of the memory segment where
29 it is not used;
30 determine whether two memory allocations of the memory segment are
31 adjacent during execution thereof;
32 determine a size of the memory segment; and
33 apply field specialization to reuse the memory segment.

1 In yet another embodiment the at least one field specialization technique comprises
2 memory layout optimization, wherein memory layout optimization preferably comprises:
3 acquiring memory segments from specific locations within the memory
4 with a specialized allocator to produce allocated memory segments;
5 determining access patterns of data stored within the allocated memory
6 segments; and
7 optimizing a layout of data stored within the allocated memory segments
8 with the locations determined by the access patterns.

9 Other systems, methods, features, and advantages of the present disclosure will
10 be or will become apparent to one with skill in the art upon examination of the following
11 drawings and detailed description. It is intended that all such additional systems,
12 methods, features, and advantages be included within this description, be within the
13 scope of the present disclosure, and be protected by the accompanying claims.

14 Many aspects of the disclosure can be better understood with reference to the
15 following drawings. The components in the drawings are not necessarily to scale,
16 emphasis instead being placed upon clearly illustrating the principles of the present
17 disclosure. Moreover, in the drawings, like reference numerals designate corresponding
18 parts throughout the several views.

19 FIG. 1 is a diagrammatical illustration of specialized software prefetching, in
20 accordance with a first exemplary embodiment of the present disclosure.

21 FIG. 2 is a diagrammatical illustration of a distribution-based hash function, in
22 accordance with a second exemplary embodiment of the present disclosure.

23 FIG. 3 is a diagrammatical illustration of a CPU binding process using speccode,
24 in accordance with a third exemplary embodiment of the present disclosure.

25 FIG. 4 is a diagrammatical illustration of a memory segment reuse process using
26 field specialization, in accordance with a fourth exemplary embodiment of the present
27 disclosure.

28 FIG. 5 is a corresponding visual illustration of the allocated memory segments of
29 disjoint segments, in accordance with the fourth exemplary embodiment of the present
30 disclosure.

31 FIG. 6 is a corresponding visual illustration of the allocated memory segments of
32 overlapping segments, in accordance with the fourth exemplary embodiment of the
33 present disclosure.

1 FIG. 7 is a corresponding visual illustration of the allocated memory segments of
2 containing segments, in accordance with the fourth exemplary embodiment of the
3 present disclosure.

4 FIG. 8 is a diagrammatical illustration of the process of using a memory allocator
5 to optimize memory layout, in accordance with a fifth exemplary embodiment of the
6 present disclosure.

7 Many embodiments of the disclosure may take the form of computer-executable
8 instructions, including algorithms executed by a programmable computer. However, the
9 disclosure can be practiced with other computer system configurations as well. Certain
10 aspects of the disclosure can be embodied in a special-purpose computer or data
11 processor that is specifically programmed, configured or constructed to perform one or
12 more of the computer-executable algorithms described below.

13 The disclosure also can be practiced in distributed computing environments,
14 where tasks or modules are performed by remote processing devices that are linked
15 through a communications network. Moreover, the disclosure can be practiced in
16 Internet-based or cloud computing environments, where shared resources, software and
17 information may be provided to computers and other devices on demand. In a
18 distributed computing environment, program modules, or subroutines, may be located in
19 both local and remote memory storage devices. Aspects of the disclosure described
20 below may be stored or distributed on computer-readable media, including magnetic and
21 optically readable and removable computer disks, fixed magnetic disks, floppy disk
22 drive, optical disk drive, magneto-optical disk drive, magnetic tape, hard-disk drive
23 (HDD), solid state drive (SSD), compact flash or non-volatile memory, as well as
24 distributed electronically over networks, including the cloud. Data structures and
25 transmissions of data particular to aspects of the disclosure are also encompassed within
26 the scope of the disclosure.

27 Moreover, while field specialization is disclosed herein primarily with respect to
28 a DBMS, it should be understood that the concepts provided herein may be applied to
29 any program that manipulates data and in particular, performs complex analysis on that
30 data.

31 To overcome the aforementioned problems within the industry, in particular, to
32 alleviate the stalling of the execution of memory-related instructions, the present
33 disclosure provides various field specialization techniques to improve the efficiency of
34 in-memory processing by DBMSs and other data-intensive applications. These

1 specialization techniques employ speccode which may be created by one or more
2 specializers (“spiffs”) in the field which is inserted into DBMS code. The speccode may
3 be both smaller and faster than the original unspecialized code. The speccode may be
4 tailored to the runtime values of invariants associated with execution paths that access
5 data in the memory in order to minimize stalls during instruction executions due to these
6 memory accesses. These techniques may enable the CPU caches to fulfill as many data
7 read and write requests as possible, thereby minimizing main-memory access.

8 FIG. 1 is a diagrammatical illustration of a specialized software prefetching 10, in
9 accordance with a first exemplary embodiment of the present disclosure. Specialized
10 software prefetching is one technique which may be employed for reducing memory-
11 access stalls and memory allocations.

12 Modern CPUs are able to perform hardware prefetching when data is accessed in
13 a predictable pattern. A typical example of such a pattern may be a tight loop iterating
14 through a large array. When the data access pattern of the program is more complicated
15 than such a simple loop, hardware prefetching may not be as effective. Accordingly, in
16 such cases, inserting software-prefetching instructions 20 into the source code 12 of the
17 program can be an effective way to mask the memory-access overhead by having the
18 CPU carry out computations at the same time that do not depend on the data that is being
19 fetched from the memory.

20 Adding software prefetching instructions 20 requires that the data access pattern
21 of the program also be predictable. By knowing how many cycles 16 the computation
22 takes before the memory value is needed, the prefetching instruction can be added and
23 configured with an effective prefetching distance. Knowing the cycles that will be taken
24 by a code segment may be challenging, particularly because a code segment may consist
25 of many possible execution paths 18, sometimes also involving calls to other routines
26 that have their own execution paths. Each of these execution paths may present a
27 different number of cycles that will execute at runtime.

28 Field specialization can transform segments of source code 12 into multiple
29 speccodes 14, each corresponding to particular execution path(s) contingent upon
30 information known only at runtime. In this way, the number of cycles 16 taken by each
31 speccode during execution can be more straightforward to determine, enabling
32 prefetching instructions to be added to each of the speccodes 14, such that each newly-
33 added prefetching instruction is configured with a specific prefetching distance.

1 Additionally, there are multiple data prefetching hints 22 that can be specified for
2 data prefetching, including: load to level-1 cache, load to level-2 cache, load to level-3
3 cache, and to not load into the caches. Depending on the specifics of a memory access
4 pattern, the added prefetching instructions can thus be configured with such hints
5 directing the appropriate prefetching level.

6 While applying field specialization, two types of relevant invariants may first be
7 identified. To transform the original code segment into speccodes, the variables and/or
8 expressions that determine the control-flow 24 of the program, for example, such as in an
9 if statement, should be identified. When the values of these variables are in fact runtime-
10 invariant, the associated conditional evaluations logic can then be eliminated from the
11 speccodes and each speccode will represent the individual execution path(s) that
12 originally corresponded to distinct branches under the conditional evaluation logic. A
13 second type of invariant is the length of each resulting speccode, in terms of instruction
14 execution cycles 16. This length can be used to compute the prefetching distance for
15 each memory access within each individual speccode.

16 Another technique which may be employed for reducing memory-access stalls
17 and memory allocations includes a data distribution-based hash function, which can
18 eliminate memory stalls originating from hash table lookup operations. Hash tables are
19 often utilized by data-intensive applications, including DBMSs. Examples are hash joins
20 and hash-based aggregate operations performed during query evaluation in a DBMS. A
21 typical hash table consists of many hash buckets, each containing a collection of entries.
22 In one example, one implementation, namely separate chaining, represents such a
23 collection using a linked list of entries. During the building of a hash table, an input
24 value is transformed, using a hash function, into a hash value. This hash value is then
25 used to identify a particular bucket in the hash table. When multiple entries are present in
26 the set associated with the current bucket, and when the hash table is probed, such as
27 when performing a hash join, the relevant set is scanned to identify matching rows.
28 Given that the entries in the set can be stored in arbitrary locations, accessing these
29 entries can lead to substantial memory stalls due to data-cache misses.

30 As an example, for a given hash join during query evaluation, the corresponding
31 hash table is often built based on particular columns from the tables that are being joined.
32 The data type of the column, width of the column, and the value range of this column can
33 all be determined and remain invariant during evaluation of that query. This metadata is
34 usually stored in the system catalog, which is consulted during query optimization.

1 FIG. 2 is a diagrammatical illustration of a distribution-based hash function 110,
2 in accordance with a second exemplary embodiment of the present disclosure. Field
3 specialization can be applied to specialize the hash function(s) invoked, such as during
4 evaluation of a particular query, by exploiting the values of invariants, as mentioned
5 above, that are associated with the particular join columns 120 used as the key of the
6 hash table. As an example, when the range of the values stored within a column 120 of
7 the hash table is known to be between two hundred (200) and three hundred (300), a
8 perfect hash function can be employed to map all one hundred (100) possible values into
9 a hash table 150 of one hundred (100) buckets. Such a hash function specialized on the
10 complete value distribution 130, where the value distribution of all rows for a given
11 column 120 in the underlying table, upon which the hash table is built is considered, may
12 guarantee that there will be no collisions, because each set has exactly one entry. Thus,
13 probing this hash table will be very efficient, as the need for (and thus the overhead
14 associated with) the traversal of the sets has been eliminated.

15 Moreover, there is an additional benefit, which is that using a specialized perfect
16 hash function 130 can reduce cache pressure. This is because a generic hash function will
17 likely convert consecutive values in a domain into a set of unordered values. As a result,
18 the hash table buckets associated with these consecutive values will be necessarily stored
19 in non-sequential locations within the memory. However, during probing, when the
20 values from the probe side are ordered, or at least partially ordered, accessing the hash
21 table via the identity hash function will significantly reduce memory stalls, especially if
22 the hash table is too large to fit in the CPU caches, because an identity hash function
23 guarantees the absence of duplicate hits from different probe values when these values
24 are (even just partially) sorted.

25 It is noted that the discussion relative to the specialized complete value
26 distribution hash function 130 assumes that hash table is built on all the rows present in
27 the join column in an underlying table. In a query, some hash join operations may only
28 take place after other plan operations. For instance, a hash join may be joining the rows
29 that result from table scan(s) with filters, effecting predicates within the query. When the
30 predicates are of high selectivity, in other words, if only a relatively small number of
31 rows are emitted by the table scan, using a hash function within the join that is
32 specialized on the original value distribution may no longer be the best choice. Instead a
33 refined hash function 140 tailored just to the distribution of the values based on which
34 the hash table is created may be a better solution. The refined value distribution 140 may

1 be based on certain intermediate values produced by particular query plan operators,
2 such as the value distribution of a subset of rows from a particular column produced by a
3 filtering operation issued by the query. The refined value distribution 140 can often be
4 obtained from the query plan as an estimate. Field specialization can provide the
5 capability and flexibility of letting specific hash table building routines be generated that
6 exploit the relevant data distribution, known only at runtime.

7 Note that the above discussion focuses on an example of utilizing consecutive
8 integers as hash values. Concerning hash function specializations, similar principles can
9 apply to more sophisticated value transformations, depending on the requirements of the
10 hash table. As a generalization, when the data type and range of the hash values—e.g.,
11 32bit integer, in range $[0, 2^{32})$ —are known and the data type and range of the input value
12 are known, one can devise a mapping between the two value domains effecting the value
13 transformation.

14 A third way to reduce memory stalls is by carefully managing CPU binding of
15 processes within a data-intensive application, using information available only at
16 runtime.

17 In a multi-process software application that employs a consumer-producer
18 execution model, data is transferred among multiple processes during the execution of
19 such an application. In this scenario, a consumer process will be placed by the operating
20 system (OS) scheduler within the operating system into the suspended state while
21 waiting for a producer process to send data. When the consumer process is woken up, the
22 scheduler will allocate a logical CPU to the consumer process to resume its execution.
23 (In a multicore CPU equipped with hyper-threading, a logical CPU corresponds to a
24 unique thread within a physical core on the CPU. For instance, on a quad-core CPU with
25 hyper-threading, there are eight logical CPUs within this physical CPU.) The scheduler
26 may select a different logical CPU on which the consumer will subsequently execute.
27 Switching to a different logical CPU can lead to substantial loss of data locality,
28 especially if the newly-selected logical CPU is located on a different physical processor
29 core. This is because running processes (which can be other consumers or producers that
30 are processing different sets of data) could be currently active on this selected logical
31 CPU, and therefore by scheduling the consumer onto a different logical CPU, the content
32 stored in the data caches on that CPU might be invalidated. As a result of this
33 scheduling, that (now active) process will have to read data from main memory, thereby
34 resulting in a memory stall. A subsequent context switch back to the original logical

1 CPU will probably result in further memory stalls. When such a suspend-resume cycle
2 occurs frequently during the application execution and when large amounts of data are
3 being sent and received, switching logical CPUs can thus lead to significant performance
4 degradation.

5 To address this shortcoming, identified pairs of consumer process and producer
6 process should be bound to a particular logical CPU to maximize data locality. It is noted
7 that the selection of the logical CPU by the OS scheduler can be determined by various
8 factors, including the availability of the logical CPUs, at the point of the context swap.
9 Therefore, in some cases, process-CPU binding may not be beneficial, as a logical CPU
10 may be busy performing other tasks by other processes. As an example concerning query
11 evaluation, it is often the case that for the same type of query-plan operator, such as hash
12 join, that appears multiple times in the plan tree, only certain ones will involve inter-
13 process data transfer.

14 FIG. 3 is a diagrammatical illustration of a CPU binding of processes using
15 speccode 210, in accordance with a third exemplary embodiment of the present
16 disclosure. With field specialization, a more effective process-CPU binding for this
17 particular situation can be realized by adding the affinity-setting system call to the
18 speccodes that specialize the stock routines in the consumers or the producers that
19 perform data manipulation. For example, following query evaluation of FIG. 2, as shown
20 in FIG. 3, it may be crucial to first identify the operators that require inter-process data
21 transfer (block 220). Second, the size of the data that will be transferred can be estimated
22 (block 230). If the size is substantial (block 240), then it may be beneficial for the
23 speccode to perform that binding (block 250).

24 A fourth way to reduce memory stalls is by carefully managing memory
25 allocation and reusing memory segments, when possible.

26 Memory allocation and deallocation in performance-critical execution paths can
27 result in significant performance penalty for two reasons. First, memory allocation and
28 deallocation are implemented with potentially complex logic. For example, a memory
29 management component may organize memory segments using a linked list. A memory
30 allocation with a certain size will require the traversal of this list until a memory segment
31 that satisfies the requested size is reached. During deallocation, the freed segment will be
32 added back onto the list. Additional manipulation may also be performed, such as
33 merging small free segments to form larger segments. Second, a memory allocation can
34 return a memory segment located at an arbitrary location in the memory. When such a

1 memory segment is read or written, memory-access stalls can occur, as the content of
2 that memory needs to be loaded into the CPU caches.

3 It has been observed that in some execution paths within data-intensive
4 applications such as DBMSs, memory allocation is performed with a fixed size, or with a
5 size that the majority of the allocations do not exceed. For such memory allocation
6 requests, the same memory segment can sometimes be reused, rather than having to
7 deallocate and allocate memory repeatedly.

8 FIG. 4 is a diagrammatical illustration of a memory segment reuse process using
9 field specialization 310, in accordance with a fourth exemplary embodiment of the
10 present disclosure. In applying field specialization, a speccode can exploit the size of the
11 memory allocation request and thus reuse the memory segment. To enable such a
12 specialization, the analysis can (a) use memory access analysis, including both static
13 pointer alias analysis and dynamic profiling, to track all the reads and writes to a
14 memory segment (block 320), (b) determine whether there is a portion of the segment's
15 lifetime where that segment is not used (block 330), and (c) determine whether the two
16 allocations are adjacent during execution (block 340). It is noted that sometimes a value
17 of a variable--the invariant--will partition the possible paths so that one or more of the
18 paths allow this specialization. Then, the sizes of the segments need to be determined
19 (block 350) and field specialization can be applied to reuse the memory segment (block
20 360).

21 As described, optimization may be used to eliminate unnecessary memory
22 allocation and deallocation calls. The analysis for determining which allocation call(s)
23 can be eliminated may be differentiated in three scenarios.

24 The first scenario is disjoint segments, each allocating and then deallocating a
25 memory segment, which is illustrated in the following code snippet.

```
26 void foo(int a, int b) {  
27     int* A = malloc(sizeof(int));  
28     *A = a;  
29     printf("%d\n", *A);  
30     free(A);  
31     int* B = malloc(sizeof(int));  
32     *B = b;  
33     printf("%d\n", *B);  
34     free(B);  
35 }
```

1 FIG. 5 is a corresponding visual illustration of the allocated memory segments of disjoint
2 segments in this scenario, in accordance with the fourth exemplary embodiment of the
3 present disclosure.

4 The second scenario is overlapping segments, in which the first memory segment
5 is deallocated only after the second memory segment is allocated, which is illustrated in
6 the following code snippet.

```
7  
8     void foo(int a, int b) {  
9         int* A = malloc(sizeof(int));  
10        *A = a;  
11        printf("%d\n", *A);  
12        int* B = malloc(sizeof(int));  
13        *B = b;  
14        free(A);  
15        printf("%d\n", *B);  
16        free(B);  
17    }
```

18
19 FIG. 6 is a corresponding visual illustration of the allocated memory segments of
20 overlapping segments in this scenario, in accordance with the fourth exemplary
21 embodiment of the present disclosure.

22 The third scenario is containing segments, in which the second memory segment
23 is allocated and deallocated before the first memory segment is deallocated, which is
24 illustrated in the following code snippet.

```
25     void foo(int a, int b) {  
26        int* A = malloc(sizeof(int));  
27        *A = a;  
28        printf("%d\n", *A);  
29        int* B = malloc(sizeof(int));  
30        *B = b;  
31        printf("%d\n", *B);  
32        free(B);  
33        free(A);  
34    }
```

1 FIG. 7 is a corresponding visual illustration of the allocated memory segments of
 2 containing segments in this scenario, in accordance with the fourth exemplary
 3 embodiment of the present disclosure.

4 In all three scenarios, FIGS. 5-7, instead of allocating a new memory segment for
 5 B, the memory segment allocated for A can sometimes be reused. Two conditions must
 6 be satisfied to effect such an optimization. First, the size of memory segment A must be
 7 equal to or larger than that of B. It is noted that if the size of memory segment A is
 8 smaller than the size of B, an option would be to enlarge A's size, if the maximal size of
 9 the memory segment(s) allocated for B can be determined, perhaps using information
 10 obtainable only at runtime. Second, for scenarios 2 and 3, from the point where memory
 11 segment B is allocated, there must be no access--either read or write--to memory segment
 12 A. The second condition renders the memory segment A "dead" from B's point of view,
 13 which is equivalent to moving free (A) before B is allocated. In other words, scenarios 2
 14 and 3 can be transformed into scenario 1, as long as the analysis, again, perhaps using
 15 information known only at runtime, can determine that there is no access to A after B is
 16 first modified.

17 The impact of this memory-allocation optimization is as follows:

- 18 • When A and B are far apart in the program execution, retaining A's memory
 19 segment to be used later by B can increase the memory pressure during that
 20 interval.
- 21 • Retaining A's memory segment means that the same cache lines will be
 22 referenced as opposed to frequent cache misses due to the allocation of B,
 23 especially if B is allocated in a loop, as shown in the following code snippet.

```

24 void foo(int a, int b) {
25     int* A = malloc(sizeof(int));
26     *A = a;
27     printf("%d\n", *A);
28     free(A);
29     for (int i = 0; i < 100; ++i) {
30         int* B = malloc(sizeof(int));
31         *B = b;
32         printf("%d\n", *B);
33         free(B);
  
```

$$\begin{array}{l} 1 \\ 2 \end{array} \quad \begin{array}{l} \} \\ \} \end{array}$$

- 3 • Retaining A's memory segment means fewer calls to malloc and free.

Note that the analysis required for transforming scenarios 2 and 3 to scenario 1 closely resembles the variable liveness analysis, which determines the scope of a variable and can help in performing register allocation during machine code generation. Once it has been determined via liveness analysis that two variables' scopes do not intersect, the second variable can be stored using the same register that has been allocated for the first variable. Similarly, if it can be determined via liveness analysis that the life cycle of two memory segments does not intersect, the second memory segment can essentially be replaced by the first memory segment, that is, the first segment is reused.

12 A fifth way to reduce memory stalls, and therefore achieve greater efficiency of
13 in-memory processing by data-intensive applications, is to specialize the memory
14 allocator to optimize memory layout. FIG. 8 is a diagrammatical illustration of the
15 process of using a memory allocator to optimize memory layout 410, in accordance with
16 a fifth exemplary embodiment of the present disclosure. As shown in FIG. 8, a
17 specialized allocator may acquire memory segments from specific locations within the
18 memory (block 420). Access patterns of data stored within the allocated memory
19 segments may then be determined (block 430) and the layout of data stored within the
20 allocated memory segments can be optimized with the locations determined by the
21 access patterns to the stored data (block 440). An example from DBMSs is to allocate
22 entries in a hash table carefully. Specifically, the hash values (which are often stored as a
23 field member within a composite struct type) for the hash entries in the sets associated
24 with the hash table buckets can instead be collected and then consecutively stored,
25 allowing efficient access to these frequently-read values, thus reducing memory stalls.

26 It is noted that the approaches described within this disclosure may all utilize
27 information available at runtime to generate specialized code that experiences fewer
28 memory stalls and fewer memory allocation and deallocation operations within DBMSs
29 and other data-intensive applications, thereby often resulting in substantial performance
30 improvement, even when all the data is already available in main memory. Any of the
31 approaches of this disclosure may be used individually or in any combination, all of
32 which are considered within the scope of the present disclosure.

It should be noted that any process descriptions or blocks in flow charts should be understood as representing modules, segments, portions of code, or steps that include

1 one or more instructions for implementing specific logical functions in the process, and
2 alternate implementations are included within the scope of the present disclosure in
3 which functions may be executed out of order from that shown or discussed, including
4 substantially concurrently or in reverse order, depending on the functionality involved, as
5 would be understood by those reasonably skilled in the art of the present disclosure.

6 It should be emphasized that the above-described embodiments of the present
7 disclosure, particularly, any “preferred” embodiments, are merely possible examples of
8 implementations, merely set forth for a clear understanding of the principles of the
9 disclosure. Many variations and modifications may be made to the above-described
10 embodiment(s) of the disclosure without departing substantially from the spirit and
11 principles of the disclosure. All such modifications and variations are intended to be
12 included herein within the scope of this disclosure and the present disclosure and
13 protected by the following claims.

CLAIMS

1
2 What is claimed is:

- 3 1. A computer-implemented method for reducing memory access stalls and memory
4 allocation requests in data-intensive applications, the method comprising:
5 identifying invariants associated with execution paths that access data in a memory of the
6 data-intensive application; and
7 applying at least one field specialization technique using at least one speccode segment,
8 whereby the speccode segment exploits the identified invariants, thereby reducing at
9 least one of memory stalls and memory allocation requests in data-intensive application.
- 10 2. The method of claim 1, wherein the data-intensive application further comprises
11 a database management system.
- 12 3. The method of claim 1 or claim 2, wherein the at least one field specialization
13 technique comprises software prefetching.
- 14 4. The method of claim 3, wherein at least one of: a control-flow invariant; and a
15 number of cycles invariant are identified.
- 16 5. The method of claim 3, further comprising inserting software prefetching
17 instructions into the at least one speccode segment.
- 18 6. The method of claim 5, further comprising using at least one data prefetching hint
19 to determine a prefetching level.
- 20 7. The method of any one of claims 1-6, wherein the at least one field specialization
21 technique comprises a data distribution-based hash function.
- 22 8. The method of claim 7, further comprising transforming an input value used as a
23 key by a hash table with at least one of: a complete value distribution hash function and a
24 refined value distribution hash function.
- 25 9. The method of claim 8, wherein the complete value distribution hash function
26 relies on a value distribution of all rows of a column of an underlying table upon which
27 the hash table is built, and the refined value distribution hash function relies on a value
28 distribution of a subset of all rows of the column of the underlying table upon which the
29 hash table is built.
- 30 10. The method of any one of claims 1-9, wherein the at least one field specialization
31 technique comprises a CPU binding of processes.
- 32 11. The method of claim 10, wherein the CPU binding of processes is dependent on:
33 a presence of inter-process data transfer; and
34 a size of the data transfer being substantial.

- 1 12. The method of any one of claims 1-11, wherein the at least one field
2 specialization technique comprises a memory segment reuse processes.
- 3 13. The method of claim 12, wherein the memory segment reuse processes
4 comprises:
5 use memory access analysis to track all reads and writes to a memory segment;
6 determine if there is a portion of a lifetime of the memory segment where it is not used;
7 determine whether two memory allocations of the memory segment are adjacent during
8 execution thereof;
9 determine a size of the memory segment; and
10 apply field specialization to reuse the memory segment.
- 11 14. The method of any one of claims 1-13, wherein the at least one field
12 specialization technique comprises memory layout optimization.
- 13 15. The method of claim 14, wherein memory layout optimization comprises:
14 acquiring memory segments from specific locations within the memory with a
15 specialized allocator to produce allocated memory segments;
16 determining access patterns of data stored within the allocated memory segments; and
17 optimizing a layout of data stored within the allocated memory segments with the
18 locations determined by the access patterns.
19

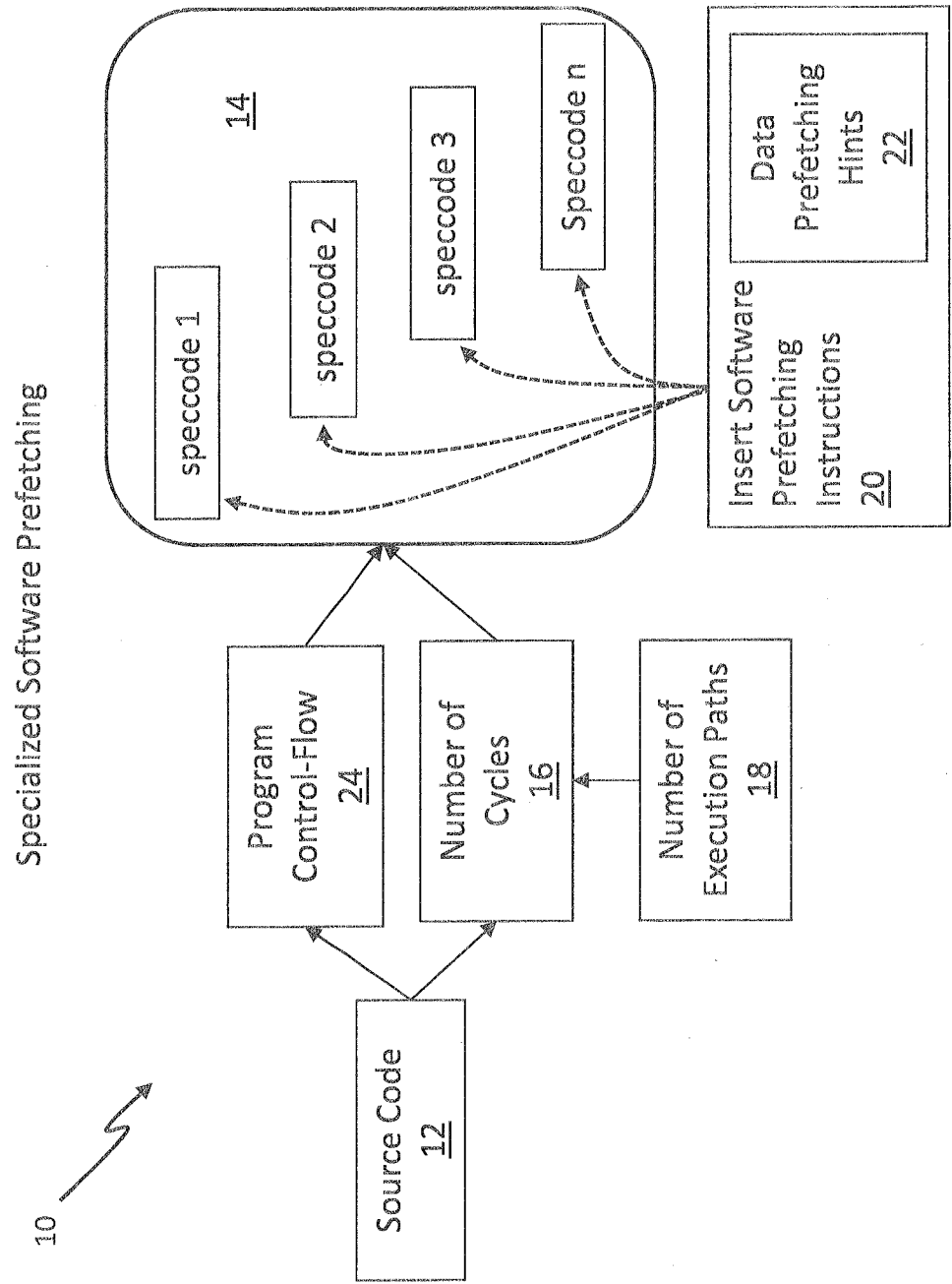


FIG. 1

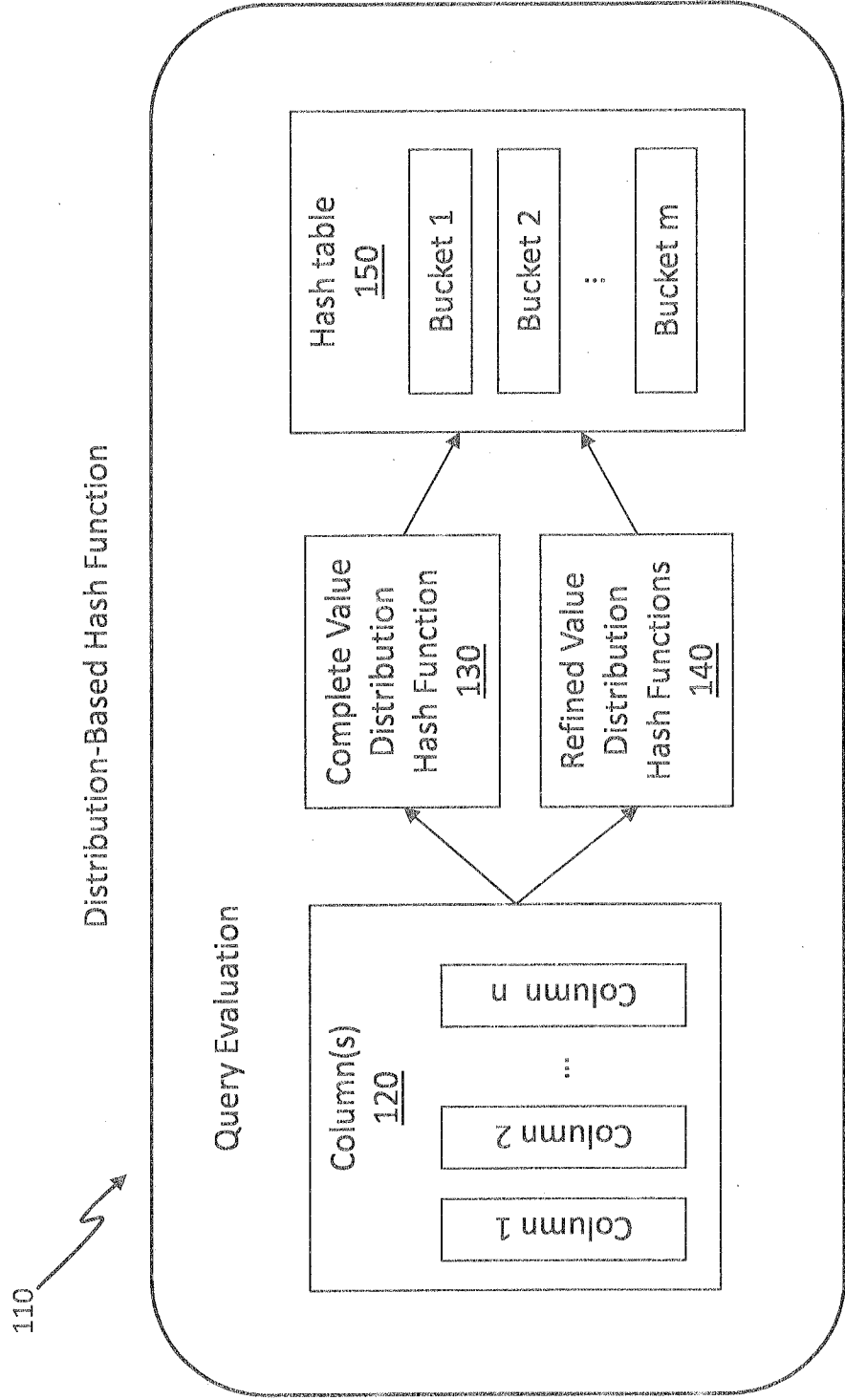


FIG. 2

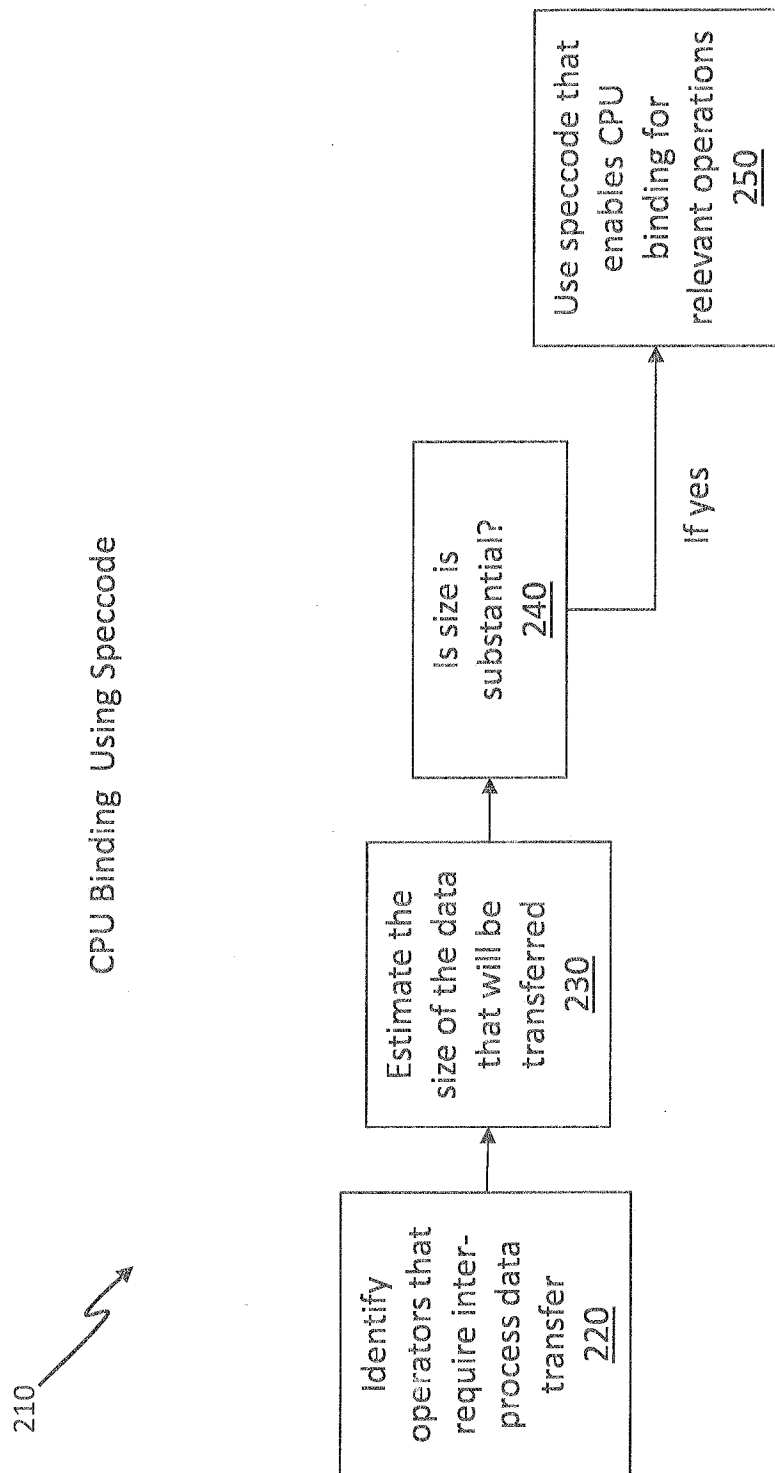


FIG. 3

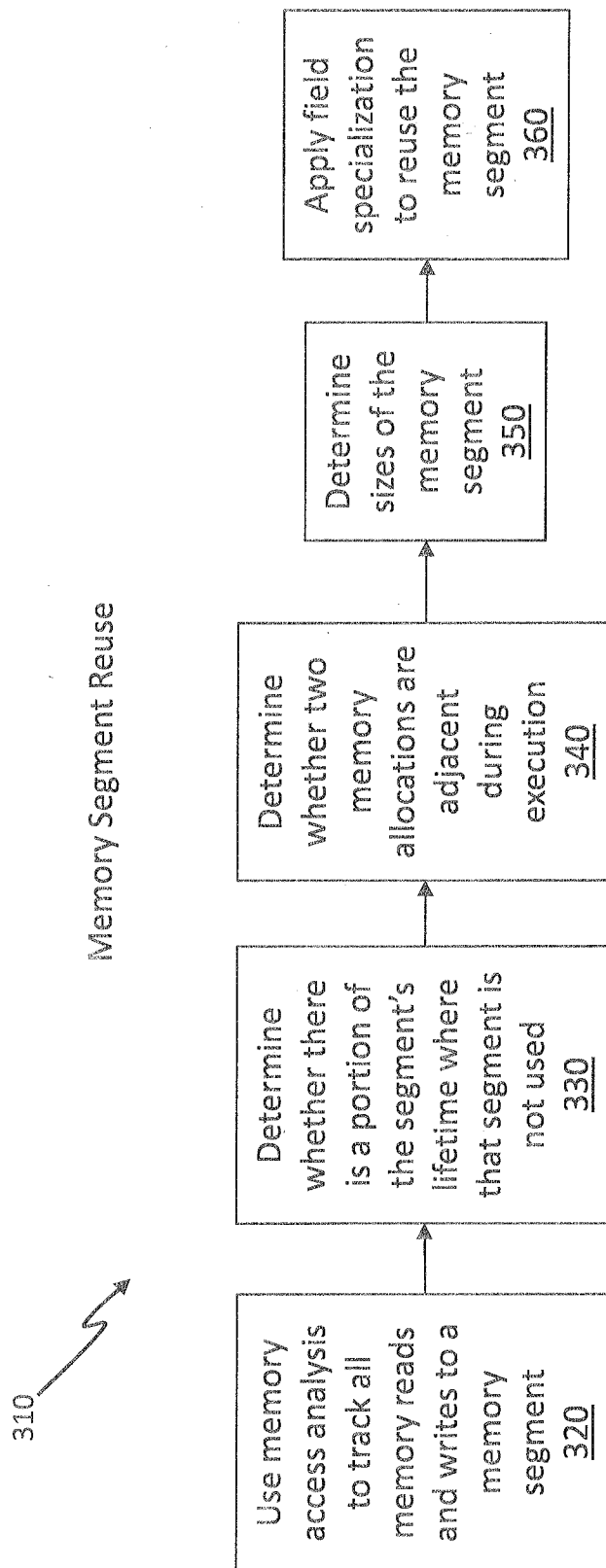


FIG. 4

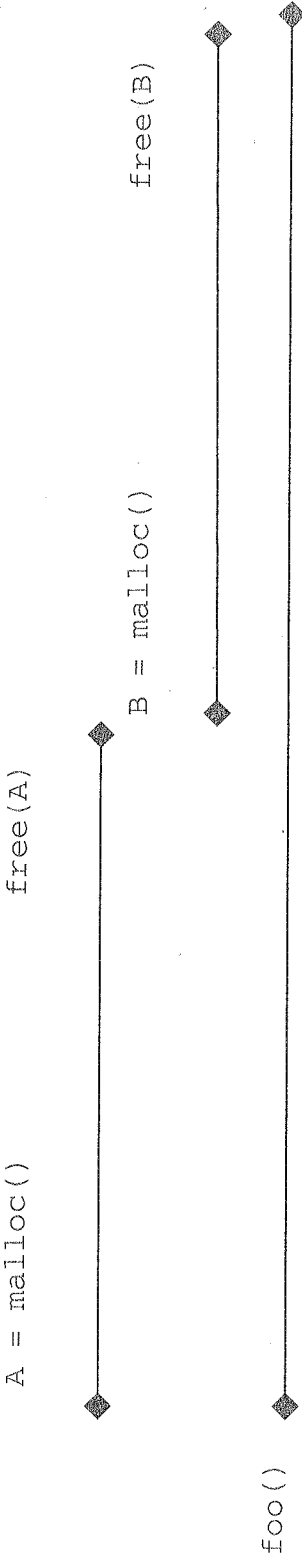


FIG. 5

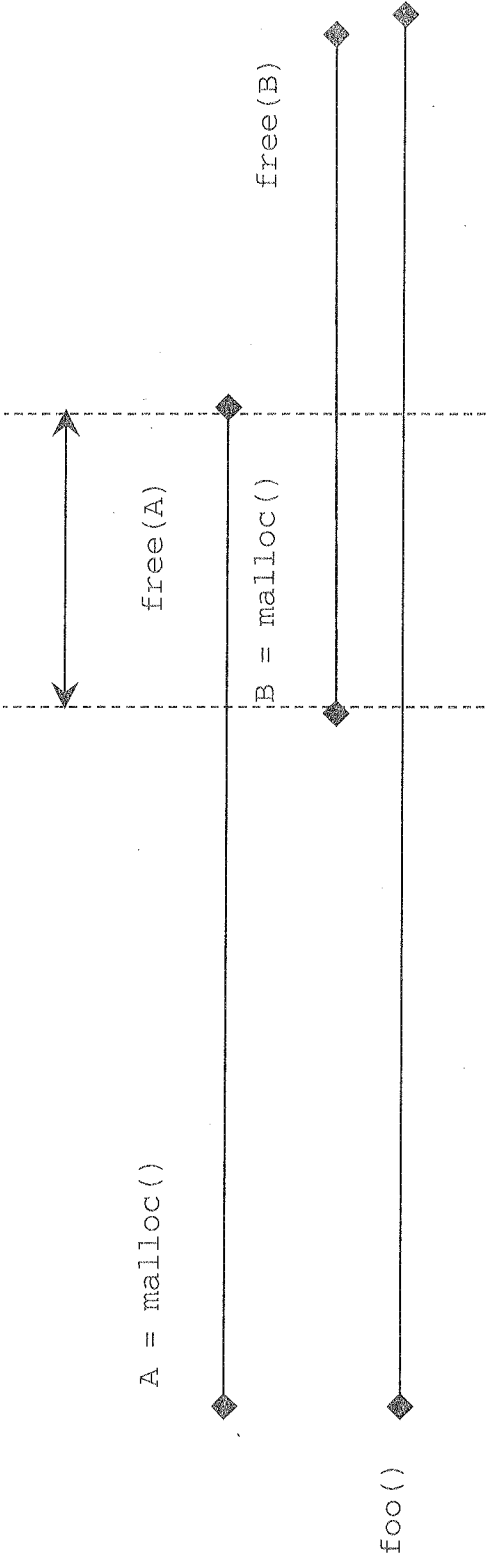


FIG. 6

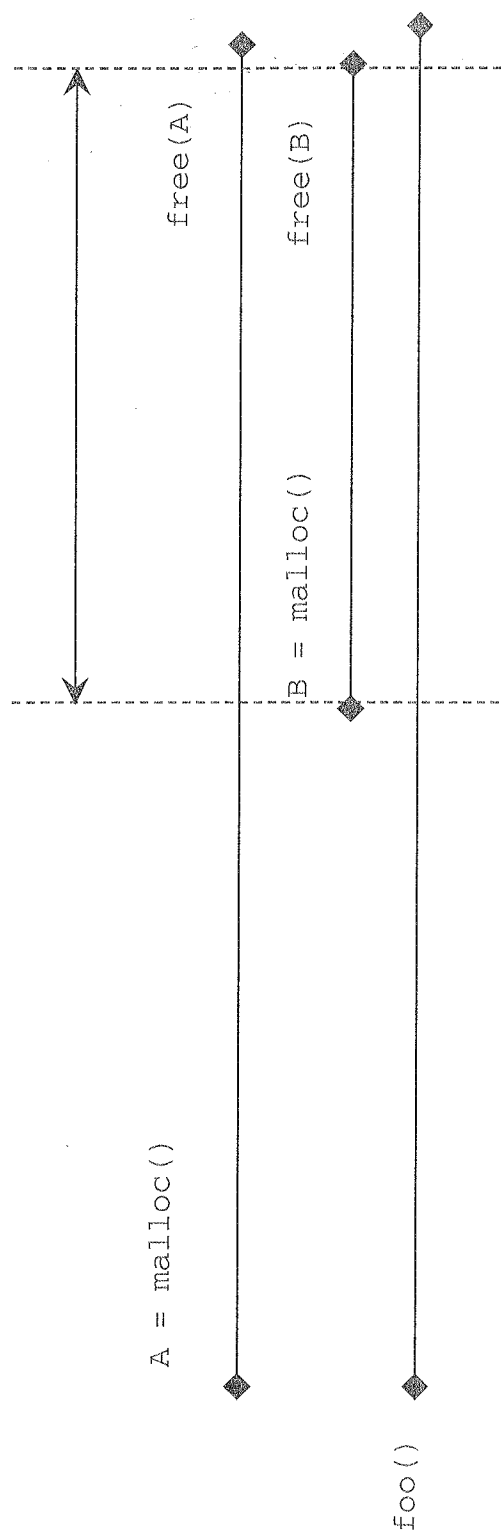


FIG. 7

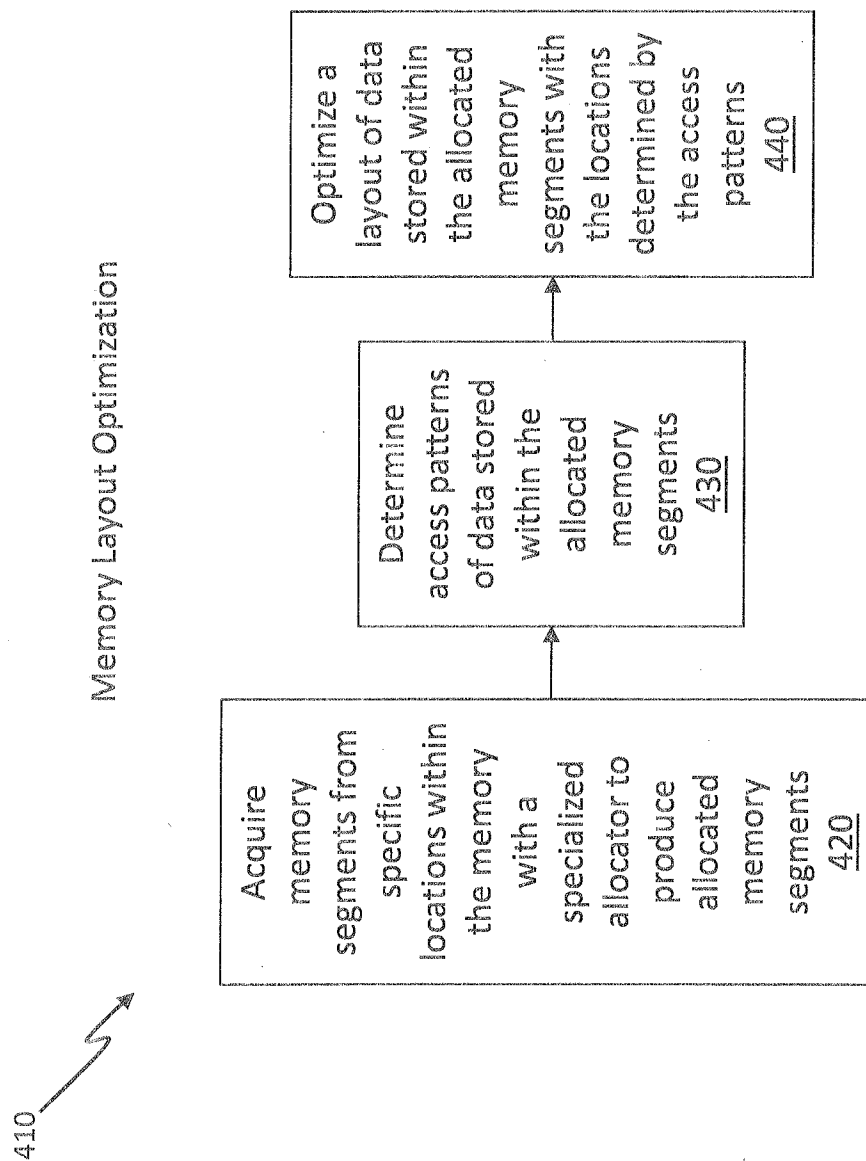


FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 18/39121

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☒ Claims Nos.: 7-15
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.
- ☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.
- ☐ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 18/39121

A. CLASSIFICATION OF SUBJECT MATTER

IPC(8) - G06F 12/00, G06F 12/08 (2018.01)

CPC - G06F 9/383, G06F 9/3844, G06F 9/30047, G06F 9/3455, G06F 9/3836, G06F 12/0862, G06F 2212/6028

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)

See Search History Document

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

See Search History Document

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

See Search History Document

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X -- Y	WO 2016/161130 A1 (DATAWARE VENTURES, LLC), 06 October 2016 (06.10.2016), entire document, especially Abstract; page 1, ln 24-49; page 5, ln 20-27; page 6, ln 4-25; page 14, ln 4-12	1-2 ----- 3-6
Y	US 2004/0133747 A1 (Coldewey), 08 July 2004 (08.07.2004), entire document, especially Abstract; para [0024]-[0025], [0050]	3-6
A	US 2003/0145314 A1 (Nguyen et al.), 31 July 2003 (31.07.2003), entire document	1-6
A	US 6,560,693 B1 (Puzak et al.), 06 May 2003 (06.05.2003), entire document	1-6

☐ 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

23 August 2018

Date of mailing of the international search report

14 SEP 2018

Name and mailing address of the ISA/US

Mail Stop PCT, Attn: ISA/US, Commissioner for Patents

P.O. Box 1450, Alexandria, Virginia 22313-1450

Facsimile No. 571-273-8300

Authorized officer:

Lee W. Young

PCT Helpdesk: 571-272-4300

PCT OSP: 571-272-7774