(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 February 2014 (27.02.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/031393 A1

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/US2013/054808

(22) International Filing Date:
13 August 2013 (13.08.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/590,032 20 August 2012 (20.08.2012) US

(71) Applicant: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, Mail Stop 5OP7, Redwood Shores, California 94065 (US).

(72) Inventors: **SCHAUER, Justin**; 1320 Stevenson St., #C308, San Francisco, California 94103 (US). **AMBERG, Philip**; 30 E. Julian St., #116, San Jose, California 95112 (US). **HOPKINS, Robert David, II**; 815 Sea Spray Lane, Unit 314, Foster City, California 94404 (US). **LEXAU, Jon**; 8180 SW Miller Hill Road, Beaverton, Oregon 97007 (US).

(74) Agents: **BRANDT, Michael C.** et al.; 1 Almaden Boulevard, Floor 12, San Jose, California 95113 (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, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, 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, 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).

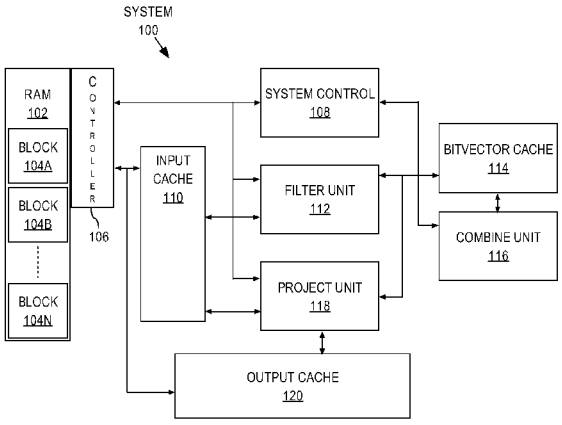
Published:

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

— *with amended claims (Art. 19(1))*

(54) Title: HARDWARE IMPLEMENTATION OF THE FILTER/PROJECT OPERATIONS

FIG. 1



(57) Abstract: Techniques are described for performing filter and project operations. In an embodiment, a set of predicates that specify criteria for filtering results to a query is received. Based on a particular predicate of the set of predicates, a predicate result for at least one portion of a particular column is generated. The predicate result identifies rows within the first column that satisfy the particular predicate. Rows are selected and returned as results to the query based at least in part on the predicate result. In an embodiment, the predicate result is a bitvector where each bit of the bitvector corresponds to a particular row within the particular column and identify whether the particular row satisfies the particular predicate.

WO 2014/031393 A1

HARDWARE IMPLEMENTATION OF THE FILTER/PROJECT OPERATIONS

FIELD OF THE INVENTION

[0001] The present disclosure relates generally to techniques for performing database operations and, more specifically, to techniques for performing filter and project operations.

BACKGROUND

[0002] The approaches described in this section are approaches that could be pursued, but not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated, it should not be assumed that any of the approaches described in this section qualify as prior art merely by virtue of their inclusion in this section.

[0003] A database comprises data and metadata that are stored on one or more storage devices, such as a set of hard disks. The data within a database may be logically organized according to a variety of data models, depending on the implementation. For example, relational database systems typically store data in a set of tables, where each table is organized into a set of rows and columns. In most cases, each row represents a distinct object, and each column represents a distinct attribute. However, other data models may also be used to organize the data.

[0004] In order to access and manipulate data in a database, a database management system (DBMS) is generally configured to receive and process a variety of database commands, often referred to as queries. In many implementations, the DBMS supports queries that conform to a Data Manipulation Language (DML) such as structured query language (SQL). When the DBMS receives a query, the DBMS performs one or more database operations specified by the query and may output a query result. Example database operations include filter, project, aggregation, and grouping operations, which are described in further detail below.

FILTER AND PROJECT OPERATIONS

[0005] Filter and project operations are database operations that output values from certain columns of certain rows, where the rows are filtered based on some criteria, known as predicates. In SQL, the project and filter operations use the SELECT and WHERE syntax. Specifically, SELECT statements indicate what data is projected (i.e. from which columns to retrieve output values) and WHERE clauses include predicates to filter the output (i.e.

indicate from which rows to retrieve output values). Examples of operators for the WHERE clause include, without limitation, the operators shown in Table 1 below.

Operator	Description
=	Equal
<>	Not Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	Specifies a set of exact values for the column

Table 1: Example predicate operators

[0006] An example filter and project query is shown in Table 2 below.

```
Query 1:
SELECT    SALESMAN, CUSTOMER, AMOUNT
FROM      sales
WHERE     AMOUNT > 200 and (SALESMAN = Pedro or SALESMAN = Alex)
```

Table 2: Sample filter/project query

This query filters the data in the sales table on the criteria that the salesman must be either Pedro or Alex, and the amount of the sale must be greater than 200. For each record in the sales table that meets these criteria, the query will return the associated salesman, customer, and amount specified in the record.

[0007] For instance, Table 3 below illustrates an example sales table.

SALE_ID	SALESMAN	CUSTOMER	AMOUNT
1	Pedro	Gainsley Corp.	400
2	Pedro	Lexau's Lexan	200
3	Alex	Lexau's Lexan	150
4	Michael	Lexau's Lexan	350
5	Alex	Gainsley Corp.	600
6	Alex	Lexau's Lexan	650
7	Pedro	Gainsley Corp.	470

Table 3: Example sales table

[0008] Given the example sales table of Table 3, Table 4 below illustrates the expected output of executing Query 1.

SALESMAN	CUSTOMER	AMOUNT
Pedro	Gainsley Corp.	400
Alex	Gainsley Corp.	600
Alex	Lexau's Lexan	650
Pedro	Gainsley Corp.	470

Table 4: Output of example query

AGGREGATION AND GROUPING OPERATIONS

[0009] Aggregation and grouping operations are database operations that provide summary statistics about data in specific columns. In SQL, grouping operations use the GROUP BY syntax to group results of aggregate functions by one or more columns. Table 5 below illustrates example aggregate functions that may be used in conjunction with GROUP BY statements.

FUNCTION NAME	DESCRIPTION
AVG	Returns the average value of a column
COUNT	Returns the number of rows in the column
FIRST	Returns the first value in the column
LAST	Returns the last value in the column
MAX	Returns the largest value in the column
MIN	Returns the smallest value in the column
SUM	Returns the sum of all values in the column

Table 5: Example aggregate functions

[0010] Example aggregation and grouping queries are shown below in Table 6.

Query 2: SELECT sum(AMOUNT) FROM sales
Query 3: SELECT SALESMAN, sum (AMOUNT) FROM sales GROUP BY SALESMAN

```
Query 4:
SELECT    SALESMAN, CUSTOMER, sum(AMOUNT)
FROM      sales
GROUP BY  SALESMAN, CUSTOMER
```

Table 6: Example aggregation queries

[0011] Query 2 requests the total dollar amount of sales the company has made. When Query 2 is executed, the DBMS performs aggregation but no grouping. The DBMS unconditionally sums all amounts in the sales table to return a final result. Given the example sales table of Table 3, Table 7 below illustrates the expected output of executing Query 2.

sum(AMOUNT)
2820

Table 7: Result table for Query 2

[0012] Query 3 requests the total dollar amount of sales grouped by the salesman who made the sale. When Query 3 is executed, the DBMS performs both grouping and aggregation. Specifically, the DBMS generates one aggregated result for each unique salesman in the sales table where the result is the total sales by the particular salesman. Given the example sales table of Table 3, Table 8 below illustrates the expected output of executing Query 3.

SALESMAN	**sum(AMOUNT)**
Pedro	1070
Alex	1400
Michael	350

Table 8: Result table for Query 3

[0013] Query 4 requests the total dollar amount of sales grouped by the salesman and the customer associated with the sale. When Query 4 is executed, the DBMS performs multi-column grouping and aggregation. In this case there will be one aggregated result for each unique salesman-customer pair, and the aggregated results are the total sales for that particular salesman-customer pair. Given the example sales table of Table 3, Table 9 below illustrates the expected output of executing Query 4.

SALESMAN	**CUSTOMER**	**sum(AMOUNT)**
Pedro	Gainsley Corp.	870
Pedro	Lexau's Lexan	200

Alex	Gainsley Corp.	600
Alex	Lexau's Lexan	800
Michael	Lexau's Lexan	350

Table 9: Result table for Query 4

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] The present disclosure is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0015] FIG. 1 is a block diagram illustrating an example system architecture for performing filter and project operations, according to an embodiment;

[0016] FIG. 2 is a flowchart illustrating an example process for performing filter and project operations, according to an embodiment;

[0017] FIGS. 3A to 3E are a series of block diagrams illustrating different states of a system in the process of performing filter and project operations, according to an embodiment;

[0018] FIG. 4 is a block diagram illustrating an example system architecture with an address generator for performing filter and project operations, according to an embodiment;

[0019] FIG. 5 is a block diagram illustrating an example system architecture for performing grouping and aggregation operations, according to an embodiment;

[0020] FIG. 6 is a flowchart illustrating an example process for performing grouping and aggregation operations according to an embodiment;

[0021] FIGS. 7A to 7E are a series of block diagrams illustrating different states of a system in the process of performing grouping and aggregation operations, according to an embodiment;

[0022] FIG. 8 is a block diagram of a computer system upon which embodiments may be implemented.

DETAILED DESCRIPTION

[0023] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

GENERAL OVERVIEW

[0024] Techniques are described herein for performing filter, project, grouping, and aggregation operations. In an embodiment, specialized hardware may be configured to perform these database operations. The specialized hardware may accelerate query processing by reducing the amount of data flowing to rate-limited parts of a computer system, which may help alleviate data bottlenecks. In particular, the specialized hardware may reduce the amount of data that needs to be stored in RAM during filter and project operations, thereby reducing the RAM input/output (I/O) operations needed to evaluate the query. In addition, the specialized hardware reduces instruction overhead that is present in most common general purpose processors for the execution of database operations.

[0025] Furthermore the specialized hardware may allow multiple data passes through a filter unit, which allows for flexibility in evaluating complex query predicates. Further still, the specialized hardware may allow for nonlinear evaluation and processing of predicates, such that the filter unit does not need to keep pace with a constant input stream of data.

[0026] According to other techniques described herein, the grouping and aggregation may be performed without a global sort of table data. Avoiding a global sort alleviates random memory access issues that occur while sorting a large list. For example, while sorting a list, one item may go in a group located in one block of memory, while the next item might belong to a group located in another block of memory. Writing the sorted table in this case would typically require closing and opening new memory pages.

[0027] In addition, techniques are described that allow groupings and aggregations to be performed on small chunks of memory, which allows random accesses to occur in fast, low power caches.

[0028] According to embodiments described herein, data structures such as bitvectors are generated to indicate which rows satisfy one or more predicates. In an example embodiment, a bitvector is generated for each predicate in a query to indicate which rows within the database satisfy the corresponding predicate. After a bitvector has been generated for each predicate, bitwise operators are used to combine the bitvectors to generate a final bitvector. The final bitvector indicates which rows satisfy all of the predicates in the set of predicates, and may be used in projection operations to select rows to output as results to the query.

[0029] In other embodiments, data structures such as bitvectors are generated to indicate which rows are part of the same group during grouping and aggregation operations. For example, a set of bitvectors may be generated where each bitvector in the group corresponds to a distinct group. The position of the bit within each of the bitvectors corresponds to a particular row. The bit value of each bit in the bitvector is set to a first bit value if the row

corresponding to the bit is part of the group represented by the bitvector or a second bit value if the corresponding row is not part of the group. Thus, the bitvectors may be used to easily identify rows that belong to the same group and may be used during performance of operations that aggregate values from columns of these rows.

EXAMPLE ARCHITECTURE FOR PROCESSING FILTER AND PROJECT OPERATIONS

[0030] FIG. 1 illustrates an example system architecture which may be configured to perform filter and project database operations, according to an embodiment. System 100 generally includes RAM 102, memory controller 106, system control 108, input cache 110, filter unit 112, bitvector cache 114, combine unit 116, project unit 118, and output cache 120.

[0031] RAM 102 stores N blocks of data, as illustrated by blocks 104A to 104N, where N may be any positive integer. RAM 102 may be implemented using any suitable computer data storage that allows random access to the stored data. Examples of RAM 102 may include without limitation dynamic RAM (DRAM) and static RAM (SRAM).

[0032] Memory controller 106 is a memory controller which manages the flow of data going to and from RAM 102. For example memory controller 106 may process requests to read and write data to RAM 102. Memory controller 106 may be implemented using any suitable memory controller, including without limitation a double data rate DDR memory controller, a dual-channel memory controller, or a fully buffered memory controller.

[0033] Input cache 110, bitvector cache 114, and output cache 120 are memory caches that store data during query processing according to techniques described further below. By caching the data, the number of I/O operations processed by RAM 102 may be minimized, thereby reducing data bottlenecks.

[0034] System control 108, filter unit 112, combine unit 116 and project unit 118 (hereby referred to as "database units") function according to the techniques described in further detail below to perform filter and project operations. Each of these components and other database units described herein may be implemented as hardware or a combination of hardware and software. For example, one or more of these units may be implemented using a programmable logic device (PLD), such as a field programmable gate array (FPGA) or other type of gate array or reconfigurable circuit. As another example, one or more of these units may be implemented using a general purpose processor, such as an advanced RISC Machine (ARM) or other reduced instruction set computer (RISC) processor.

FILTERING DATA BASED ON PREDICATES

[0035] FIG. 2 illustrates an example process for performing filter and project operations, according to an embodiment. The process illustrated in FIG. 2 may be implemented on any suitable system such as system 100 illustrated in FIG. 1 or system 400 illustrated in FIG. 4.

[0036] Referring to FIG. 2, in step 202, a query that includes a set of one or more predicates is received by system control 108. For example, the query may include a SELECT statement with a WHERE clause that specifies one or more predicates. However, any other suitable syntax may be used depending on the particular implementation. In an embodiment, each predicate specifies a criterion that is used to filter data extracted from a database and output or otherwise returned as results to the query. The criterion may be specified using one or more operators including, without limitation, the operators illustrated in Table 1 above.

[0037] In an example embodiment, system control 108 parses the received query to determine which predicates the query includes and how the predicates should be programmed into filter unit 112. Steps 204 to 210 define a loop that is repeated for each predicate in the set of predicates. Thus, in the first iteration of step 204, filter unit 112 is programmed with a first predicate of the set of predicates. System control 108 may program the predicates into filter unit 112 in any suitable order. Techniques for selecting the order are described in further detail below. In an embodiment, programming the filter unit may comprise configuring an FPGA or other reconfigurable circuit to apply a filtering criterion dictated by the predicate. For example, filter unit 112 may be programmed using a hardware description language (HDL) to implement any of the predicate operators illustrated in Table 1 to compare one or more values with a predicate value.

[0038] Typically, predicates impose conditions of values from specific columns. A column upon which a predicate imposes a condition is referred to herein as a target column of the predicate. Thus, for the predicate (AMOUNT > 200), the AMOUNT column is the target column.

[0039] In step 206, values from columns are sent to filter unit 112. In an example embodiment, system control 108 sends a request to memory controller 106 to load values from one or more columns from RAM 102 into input cache 110. System control 108 may determine which columns should be loaded into the input cache 110 based on the predicate being evaluated. For example, system control 108 may cause the loading of values only from the target column of the predicate with which the filter unit 112 is currently programmed. For example, for the predicate AMOUNT>200, system control 108 may cause values from at least a portion of the AMOUNT column to be loaded from RAM 102 into input cache 110, as illustrated in the example implementation below.

[0040] The values from the target columns are then streamed from input cache 110 to filter unit 112. Upon receipt of the column data, filter unit 112 applies the predicate that was programmed at step 204 to generate a result identifying rows that meet the predicate. To determine which rows meet the predicate, filter unit 112 may apply one or more predicate operators to compare the value of a particular row of the received column with a predicate value specified in the query predicate.

[0041] In step 208, the results of the filter are stored as a data structure indicating which rows had target column values that met the predicate condition. In an embodiment, the data structure is a bit vector as described in further detail below. In an alternative embodiment, the data structure is encoded data generated from the bit vector. Any other suitable data structure may also be generated, stored, and used to indicate which rows had target column values that met the predicate condition.

[0042] In step 210, system control 108 determines whether there are predicates remaining in the set of predicates that have not been evaluated yet by filter unit 112. If there are remaining predicates that have not been evaluated, then the process returns to step 204, and system control 108 programs filter unit 112 with one of the remaining predicates. If all the predicates within the set of predicates have been already been evaluated, then the process continues with step 212, which is described in further detail below.

GENERATING BITVECTORS AT THE FILTER UNIT

[0043] In an embodiment, the filter unit generates a bitvector for each predicate result. Each bit of the bitvector corresponds to a row and indicates whether the corresponding row satisfies the query predicate that is associated with the bitvector. For example, the position of the bit within the bitvector may correspond to the position of the corresponding row within a table. Thus, the third bit in a bitvector may correspond to the third row of a table.

[0044] The value of each bit represents a Boolean value, where a first bit value indicates that the corresponding row satisfies the predicate condition and a second bit value indicates that the corresponding row does not satisfy the predicate condition. For example, the third bit in a bit vector is “1” if the third row of the table satisfies the predicate that is associated with the bitvector, and is “0” if the third row of the table does not satisfy the predicate that is associated with the bitvector.

COMBINING RESULTS

[0045] After a results for each predicate have been generated according to steps 202 to 210, a final result is generated in step 212 by combining the results of each predicate in a

manner dictated by the query. The final result is a data structure, such as a final bitvector, that identifies the set of rows that meet all the predicates in the query.

[0046] In an embodiment, the final result is a final bitvector where, each bit of the final bitvector corresponds to a particular row of the table that is targeted by the query. Just as the value of each bit in the predicate-specific bitvectors indicates whether the corresponding row satisfies the predicate associated with the bitvector, the value of each bit in the final bitvector indicates whether the corresponding row satisfies all predicates. Thus, a first bit value (e.g. "1") is used to indicate that the corresponding row satisfies all predicates in the query, and a second bit value (e.g. "0") is used to indicate that the corresponding row does not satisfy all predicates.

[0047] In an example embodiment, system control 108 programs combine unit 116 with instructions on how to combine the bitvectors. For example, combine unit 116 may be programmed to perform one or more bitwise operations based on the logical operators specified in the query to combine the result bitvectors. In the case of Query 1, for example, system control 108 would program combine unit 116 to perform a bitwise OR operation, and then a bitwise AND operation, to produce the final bitvector, as illustrated in the example implementation below.

PROJECTING FILTERED DATA

[0048] In step 214, project unit 118 uses the final result to select rows to output from a projected column. In the case where a final bitvector is used, project unit 118 processes the final bitvector bit by bit to identify the rows that satisfy all query predicates (i.e., those rows whose corresponding bit, within the final bitvector, is set to the first bit value). In step 216, project unit 118 retrieves column data for the projected column from input cache 110 and outputs the rows identified by the final result to output cache 120. Outputting a row in this context refers to storing the row that satisfies the predicate in output cache 120. Once the row is output, it may be returned, for example to a user or application program, at any time as a result to the query.

[0049] In step 218, system control 108 or project unit 118 determines whether all columns indicated by the query have been projected. For example, in the case of Query 1, the projected columns include the SALESMAN, CUSTOMER, and AMOUNT columns. If there are any projected columns remaining, then the process returns to step 214, and the final result is applied to a column in the remaining set. Applying the final result may comprise using the final bitvector as a mask or translating the final bitvector into memory addresses, as described

further below. This process repeats until all columns indicated by the query have been projected.

[0050] The process of projecting rows based on the final result may vary depending on the particular implementation. In one embodiment, if not already stored in input cache 110 from the filtering process, then rows of values from a projected column are loaded into input cache 110. Project unit 118 determines, based on the final result, which of these rows of values should be stored in output cache 120. In an embodiment, project unit 110 uses a final result bitvector as a mask that controls which rows are output at step 216 from project unit 118 to output cache 120. For example, each row of a projected column may be streamed from input cache 110 to project unit 118. Project unit 118 may then apply the bitvector as a mask to stream to output cache 120 only those rows that satisfy all query predicates. Alternatively, an address generator unit may be used, as described in further detail below, to provide project unit 118 with only those rows that should be output.

HYBRID COLUMNAR BLOCK PROCESSING

[0051] In an embodiment, the system stores at least a portion of the database in RAM 102 in a hybrid-columnar fashion. Hybrid-columnar storage breaks the database into blocks, where each block has a fixed number of rows for one or more columns. For example, a first block may store the first 50 rows for one or more columns, the second block the next 50 rows, and a third block the next 30 rows. Within each block, data is stored in a column-oriented fashion. In other words, the elements within a column are stored contiguously within the block. Storing the data in hybrid-columnar fashion allows the system to easily process columns in manageable block sizes.

[0052] In an embodiment, the filtering and projection operations described above may be performed on a per-block basis. For example, if a column is spread over multiple blocks, the filtering and project operations may be applied to a first portion of the column residing in a first block to generate a first result set. The process may repeat for each subsequent block until the entire column has been processed. The result set generated at one stage does not need to wait for a subsequent stage before being returned. For instance, the result set generated for one block may be returned before or during processing of a subsequent block.

EXAMPLE SYSTEM IMPLEMENTATION OF FILTER/PROJECT OPERATIONS

[0053] FIGS. 3A to 3E are a series of block diagrams illustrating different states of a system in the process of performing filter and project operations, according to an

embodiment. In particular, these figures illustrate system 100 processing Query 1 shown in Table 2 above.

[0054] FIG. 3A is a block diagram of system 100 at the start of the filter operation. Referring to FIG. 3A, the SALESMAN, CUSTOMER, and AMOUNT columns reside in RAM 102 in a hybrid-columnar fashion and are split between block 302 and block 310. Specifically, a first portion of the SALESMAN column, CUSTOMER column, and AMOUNT column as shown by S1 304, C1 306, and, A1 308, respectively, reside in block 302. A second portion of the SALESMAN column, CUSTOMER column, and AMOUNT column, as shown by S2 312, C2 314, and A2 316, respectively, reside in block 310.

[0055] When system control 108 receives Query 1, system control 108 determines how to evaluate the predicates and causes the appropriate columns to be loaded into input cache 110. As illustrated by FIG. 3B, the AMOUNT > 200 predicate is evaluated first. Accordingly, system controller 108 programs filter unit 112 with predicate 330. The first portion of the AMOUNT column, A1 308, is loaded into input cache 110 and sent to filter unit 112. Filter unit 112 then evaluates the column data of A1 308 value by value (where each value corresponds to a distinct row) using predicate 330 to generate bitvector Ap1 340, which indicates the rows of A1 308 that satisfy the AMOUNT>200 predicate. The first bitvector, Ap1 340, is shown as Ap1 in Table 10 below. Bitvector cache 114 stores this bitvector for subsequent processing.

AMOUNT	Ap1
400	1
200	0
150	0
350	1
600	1
650	1
470	1

Table 10: The first bitvector, Ap1, showing rows where AMOUNT > 200

[0056] After the AMOUNT > 200 predicate has been evaluated, the next two predicates, SALESMAN = Pedro, and SALESMAN = Alex, are then processed serially in a similar fashion. For example, the SALESMAN column S1 304 may be loaded into input cache 110. System control 108 programs filter unit 112 to apply the "=" operation to the "Pedro" value in the first instance and "Alex" value in the second instance. Filter unit 112 evaluates the SALESMAN column row by row in each instance to generate a second and third bitvector.

FIG. 3C is a block diagram of the system after all predicates have been processed. Bitvector Sp2 342 represents the bitvector generated in response to evaluating the SALESMAN = Pedro predicate and bitvector Sp3 344 represents the bitvector generated in response to evaluating the SALESMAN = Alex predicate. These bitvectors are shown in Table 11 below.

AMOUNT	Ap1	SALESMAN	Sp2	Sp3
400	1	Pedro	1	0
200	0	Pedro	1	0
150	0	Alex	0	1
350	1	Michael	0	0
600	1	Alex	0	1
650	1	Alex	0	1
470	1	Pedro	1	0

Table 11: The second bitvector, Sp2, shows rows where SALESMAN = Pedro and the third bitvector, Sp3, shows rows where SALESMAN = Alex

[0057] In an embodiment, the bitvector generated by filter unit 112 is as many bits long as there are rows in a block. For instance, the length of the bitvectors shown in Tables 10 and 11 above correspond to the number of values, for each column, are stored in the Block 1 302. Thus, bitvector Ap1 340 has the same number of bits as there are rows in A1 308, and bitvectors Sp2 342 and Sp3 344 have the same number of bits as there are rows in S1 304. In these tables, rows that satisfy the predicate are assigned a bit value "1" and rows that do not satisfy the predicate are assigned the bit value "0". However, these bit values may be inverted, depending on the implementation.

[0058] After the result bitvectors for each predicate have been generated, the final bitvector may be generated through the combine process described above. In the case of Query 1 the combine process may be implemented as dictated by the logical operators in the WHERE clause. Accordingly, system control 108 first programs combine unit 116 to perform a bitwise OR on bitvectors Sp2 342 and Sp3 344. The result of the OR operation is then used to perform a bitwise AND with bitvector Ap1 340. FIG. 3D shows a block diagram of the system after predicate result bitvectors have been combined to produce the final bitvector used to retrieve a final set of filtered rows.

[0059] Referring to FIG. 3D, bitvector cache 114 stores bitvector Sp2|Sp3 344, which is the resulting bitvector from performing the bitwise OR operation on bitvectors Sp2 342 and Sp3 344. Combine unit 116 then performs a bitwise AND operation using bitvector Sp2|Sp3 344 and bitvector Ap1 340 to generate final bitvector Ap1(Sp2|Sp3) 348. These bitvectors

are shown in Table 12 below. The final bitvector shown in the last column of Table 12 represents rows, within block 1 302, that meet all the predicates of Query 1.

AMOUNT	Ap1	SALESMAN	Sp2	Sp3	Sp2 or Sp3	Ap1 and (Sp1 or Sp2)
400	1	Pedro	1	0	1	1
200	0	Pedro	1	0	1	0
150	0	Alex	0	1	1	0
350	1	Michael	0	0	0	0
600	1	Alex	0	1	1	1
650	1	Alex	0	1	1	1
470	1	Pedro	1	0	1	1

Table 12: Results of the bitvector combine operations, including the final bitvector for block 302

[0060] The final bitvector is sent to project unit 118, which uses this bitvector to project rows from the appropriate columns. In Query 1, the SELECT statement indicates that data should be projected from the SALESMAN, CUSTOMER, and AMOUNT columns of the sales table. Accordingly, S1 304 may be streamed from input cache 110 to project unit 118. Project unit 118 may go through the final bitvector bit by bit and send to output cache 120 the rows of the S1 304 that correspond to "1s" in the final bitvector. Project unit 118 repeats this process for the CUSTOMER and AMOUNT columns using the same final bitvector.

[0061] FIG. 3E is a block diagram of the system at the end of the filter and project operation. Sr1 350 represents the projected SALEMAN column data from block 302, Cr1 352 represents the projected CUSTOMER column data from block 302, and Ar1 354 represents the projected AMOUNT column data from block 302. This data may be sent out as a result while the system begins processing block 310.

[0062] The filtering and project operations described above may then be repeated on the data stored in other blocks such as block 310. For purposes of illustration, it is assumed that the sales table also includes the rows shown below in Table 13.

SALE_ID	SALESMAN	CUSTOMER	AMOUNT
8	Pedro	Gainsley Corp.	100
9	Alex	Lexau's Lexan	370
10	Alex	Lexau's Lexan	500
11	Michael	Lexau's Lexan	120
12	Pedro	Gainsley Corp.	280

Table 13: Additional rows of example sales table

Block 310 stores data for rows 8-12 in column-oriented format. For example, S2 312 may store the following values in contiguous order: Pedro, Alex, Alex, Michael, Pedro. Similarly, C2 314 stores rows 8-12 of the CUSTOMER column, and A2 316 stores rows of the AMOUNT column.

[0063] Table 14 below shows the bitvectors generated after performing the filtering operations on the data stored in block 310.

AMOUNT	Ap1	SALESMAN	Sp2	Sp3	Sp2 or Sp3	Ap1 and (Sp1 or Sp2)
100	0	Pedro	1	0	1	0
370	1	Alex	0	1	1	1
500	1	Alex	0	1	1	1
120	0	Michael	0	0	0	0
280	1	Pedro	1	0	1	1

Table 14: Results of the bitvector combine operations, including the final bitvector for block 310

[0064] The final bitvector shown in the last column of Table 14 identifies the rows within block 310 that satisfy all the query predicates. Project unit 118 parses this final bitvector bit by bit and send to output cache 120 the rows of S2 312 that correspond to "1s" in the final bitvector. Project unit 118 repeats this process for C2 314 and A2 316 using the same final bitvector.

SELECTIVE ROW FILTERING BASED ON ADDRESS GENERATION

[0065] In some embodiments, the rows that are supplied to filter unit 112 during predicate evaluation may be restricted based on results obtained from a previous predicate evaluation. For example, certain rows that do not satisfy a previously evaluated predicate may not need to be considered when evaluating a subsequent predicate. By selectively providing rows to filter unit 112 for processing, filter unit 112 may avoid having to evaluate the entire column for each predicate.

[0066] FIG. 4 is a block diagram illustrating an example system architecture with an address generator for performing filter and project operations, according to an embodiment. System 400 is a variation of system 100 that includes address generator 402. System 400 may use address generator 402 for selectively supplying rows of a column being filtered to filter unit 112.

[0067] In an embodiment, address generator 402 uses the bitvector result of a previous filter to supply a subset of the rows to the filter unit for subsequent filters. The manner in which a bitvector is used to restrict the rows supplied to filter unit 112 depends on the logical operators specified in the query. For example, with Query 1, the first predicate is ANDed with the subsequent predicates. Thus, if a row does not meet the first predicate, then that row does not need to be considered for the next two predicates.

[0068] In the example implementation above, address generator 402 can use bitvector Ap1 340 to only supply the rows that met the first predicate to filter 112 unit when producing bitvectors Sp2 342 and Sp3 344. Specifically, because the bitvector associated with the predicate AMOUNT>200 is 1001111, the second and third rows need not be evaluated against the other predicates. The greater the number of subsequent evaluations that can be skipped, the more efficient the query evaluation. For example, if AMOUNT>200 had produced a bitvector 0000000, then the entire evaluation of the remaining predicates could be skipped.

[0069] Because the next two predicates are ORed together, a row that meets the second predicate does not need to be considered when evaluating the third predicate. That is, the bitvector 1100001 associated with the second predicate indicates that the first, second and seventh rows can be skipped during the evaluation of the third predicate.

[0070] In fact, during the evaluation of the third predicate, address generator 402 may use the result bitvectors of the first and second predicate to determine that only the fourth, fifth and sixth rows need to be evaluated against the third predicate. Specifically, the second and third rows can be skipped because they fail to satisfy the first predicate, and the first, second and seventh rows can be skipped because they do satisfy the second predicate.

ORDER OF EVALUATION

[0071] The order in which predicates are evaluated may vary depending on the implementation. In an embodiment, the predicates may be evaluated in a sequential order. For example, the predicates may be evaluated serially from left to right or right to left as specified in the query.

[0072] In another embodiment, the order of predicate evaluation may be based on the likelihood that the predicate will filter out a large number of rows (i.e. the "selectivity" of the predicate). When predicates that are highly selective are evaluated first, a greater number of rows are filtered out earlier in the filtering process. In system 400, this results in address

generator 402 providing a smaller subset of rows to filter unit 112 during subsequent predicate evaluations. Thus, processing more selective predicates before less selective predicates may reduce processing overhead.

[0073] In an embodiment, system control 108 estimates the selectivity of a predicate based on the operators specified in the query. For example, predicates that are ANDed with other predicates are more likely to be highly selective than predicates that are ORed with other predicates. In another example, the equivalence predicate operator ("=") is more likely to be highly selective than the not equal predicate operator ("<>" or "!="). Based on the estimation, system control 108 programs the filter unit 112 in sequential order from the most selective predicate to the least selective predicate.

[0074] In other embodiments, one or more predicates specified in a query are processed in parallel. For example, filter unit 112 may be programmed with two or more predicates specified in a query. Filter unit 112 may evaluate both predicates concurrently. Techniques for parallelizing the predicate evaluation process are described further below.

PROJECTING FILTERED DATA IN A SYSTEM THAT USES AN ADDRESS GENERATOR

[0075] In another embodiment, address generator 402 may translate the final bitvector into a set of memory addresses for each row that satisfies all query predicates. Address generator 402 may then use the memory addresses to request only these rows from input cache 110 and provide them to project unit 118 for output. This may save processing overhead because the entire column does not need to be streamed through project unit 118.

[0076] For example, at step 214, the final bitvector may be provided to address generator 402 from bitvector cache 114. Address generator 402 then determines the memory addresses for each row that has a corresponding bit value indicating that the row satisfied the set of predicates. Address generator 402 sends memory fetch requests to input cache 110 using these memory addresses. If these rows are already loaded into input cache 110, then they may be streamed directly from input cache 110 to project unit 118, which output the rows to output cache 120. Alternatively, the rows may be sent to address generator 402

MULTIPLE PREDICATES PER COLUMN

[0077] In the examples given above, filter unit 112 applied a single predicate each time column data is passed through. In alternative embodiments, filter unit 112 may be configured to process multiple predicates for each column. For example, sample Query 1 includes the predicates SALESMAN = Pedro or SALESMAN = Alex. Both of these predicates relate to

the same column. Therefore, system control 108 may program filter unit with both predicates such that both predicates may be evaluated with a single pass of the SALESMAN column through filter unit 112.

MULTIPLE COLUMNS PER UNIT

[0078] In the examples given above, filter unit 112 operated on a single column input. In alternative embodiments, filter unit 112 may include a plurality of column inputs. Multiple column inputs may be helpful when evaluating certain predicates. For example, the clause WHERE SALESMAN = CUSTOMER references both the SALESMAN and CUSTOMER columns in the same predicate. If filter unit 112 had two column inputs, then the predicate could be evaluated on a single pass. Even in cases where two columns are not included in a single predicate, having multiple columns inputs may be used to process predicates on multiple columns simultaneously.

[0079] Project unit 118 may also include a plurality of column inputs, depending on the implementation. For example, if project unit 118 had multiple column inputs, then the final bitvector may be applied concurrently to the multiple columns to project the results in parallel.

EXAMPLE ARCHITECTURE FOR PROCESSING GROUPING AND AGGREGATION OPERATIONS

[0080] In an embodiment, specialized hardware may be configured to perform grouping and aggregation operations. FIG. 5 illustrates an example system architecture which may be configured to perform grouping and aggregation database operations, according to an embodiment. System 500 may include all the elements of system 100 or system 400. In addition or as an alternative to project unit 118, system 500 also includes aggregation unit 502.

[0081] System 500 may be combined or otherwise integrated with system 100 or 400 in any suitable manner. Each of the overlapping blocks may be implemented as the same hardware unit or as separate independent units. For example, filter unit 112 may be the same hardware unit that performs predicate filtering in system 100 or system 400. This same unit can also be used to create groups based on column data according to the techniques described below. Alternatively, separate filter units and/or other database units may be used to process grouping operations and filtering operations. In other embodiments, system 500 may be implemented independently of and/or separately from the filtering and projection logic illustrated in system 100 or 400.

GROUPING AND AGGREGATING DATA USING A PREDICATE FILTER

[0082] FIG. 6 illustrates an example process for performing grouping and aggregation operations, according to an embodiment. The process illustrated in FIG. 6 may be implemented on any suitable system, such as system 500 illustrated in FIG. 5.

[0083] Referring to FIG. 6, in step 602, a query is received that includes a request to aggregate data grouped by one or more columns. For example, the query may include any suitable aggregation function including, without limitation, those listed in Table 5. The aggregation function may be used in conjunction with a GROUP BY statement specifying one or more columns for grouping the aggregate result data. However, any suitable syntax may be used to specify the aggregation function and grouping columns.

[0084] In step 604, a row of a first column that is being grouped is sent to filter unit 112. In the case of sample Query 3, for instance, the first row of the SALESMAN column may be sent to filter unit 112. In the case of sample Query 4, the SALESMAN and CUSTOMER column may be combined and sent to filter unit 112 according to techniques described further below. Alternatively, if filter unit 112 has multiple column inputs as described in further detail below, then the first row of both the SALESMAN and CUSTOMER column may be sent to filter unit 112 concurrently.

[0085] In step 606, filter unit 112 identifies an element associated with the first row of the column. In an embodiment, the element is an item of data stored within the first row of the column. For example, referring to the example sales table shown in Table 3, the first element of the SALESMAN column is "Pedro", and the first element of the CUSTOMER column is "Gainsley Corp."

[0086] In step 608, filter unit 112 uses equivalence to the first element identified at step 606 as a predicate to filter out rows that do not belong to the group to which the first element belongs. For example, assuming the first element in the SALESMAN column is "Pedro", filter unit 112 uses the predicate "SALESMAN=Pedro" to filter out all rows that do not belong to the "SALESMAN=Pedro" group. System control 108 may program filter unit 112 with this logic in response to receiving the query at step 602.

[0087] In step 610, the remaining rows (i.e., those rows other than the first row) of the column are sent to filter unit 112. In an embodiment, the remaining rows are streamed in contiguous order from input cache 110 to filter unit 112 in the first pass. In subsequent passes, address generator may feed filter unit 112 only those rows that have not been previously grouped.

[0088] In step 612, filter unit 112 filters out rows that do not satisfy the filter to generate the group of rows to which the first row belongs (e.g. the group of all rows where SALESMAN=Pedro). This step may include generating a bitvector or other data structure that identifies each row that satisfies the equivalence predicate. Similar to the predicate filtering described above, each bit of the bitvector may correspond to a distinct row within the column, where a first bit value indicates that the row satisfies the equivalence predicate and is therefore part of the group, and a second bit value indicates that the row does not satisfy the equivalence predicate and is therefore not part of the group.

[0089] In step 614, the first row and rows that match the first row are grouped and sent to aggregation unit 502. In an example embodiment, filter unit 112 sends the bitvector generated at step 612 to address generator 402, which uses the bitvector to request from memory only those rows that are part of the current group. Once received, address generator 402 sends these rows to aggregation unit 502 for aggregation.

[0090] In step 616, aggregation unit 502 aggregates the values of the grouped rows as dictated by the query. In the case of Queries 3 and 4, for example, aggregation unit 502 would sum the values of the grouped rows stored in the AMOUNT column.

[0091] In step 618, system 500 determines whether there are any remaining rows that have not been grouped yet. If there are, then the process returns to step 604, where the grouping and aggregation operation is repeated for only those rows that have not yet been grouped. Accordingly, filter unit 112 is reprogrammed to use an element of a first ungrouped row as the equivalence predicate to form a new group.

[0092] In the present example from Table 3, rows 1 and 2 would have been grouped in the SALESMAN=Pedro group. Therefore, the third row would be the first not-yet-grouped row. The third row has the value "Alex" in the SALESMAN column, which is the grouping column of the query. Therefore, the second group is determined based on the filter SALESMAN=Alex. The second group would include rows 3, 5 and 6.

[0093] During the third iteration of step 604, row 4 is the first remaining ungrouped row. Row 4 has the value "Michael" in the SALESMAN column. Therefore, the third group is determined based on the filter SALEMAN=Michael. The third group would only include row 4. After the formation of the third group, the process illustrated in FIG. 6 would end, because there would be no ungrouped rows remaining.

[0094] During each iteration, system 500 generates and aggregates a group based on a new predicate. If all rows have been grouped, then the process ends. If the relevant columns are stored over a plurality of RAM blocks, then this process may be repeated for each of the plurality of RAM blocks, as described in further detail below, to produce a final result.

DETERMINING SUBSEQUENT GROUPINGS BASED ON PREVIOUSLY GENERATED BITVECTORS

[0095] In an embodiment, the bitvectors generated at step 612 used to identify row groupings (referred to herein as "group-membership bitvectors") may also be used to selectively provide rows to filter unit 112 for subsequent groupings. Specifically, each bit in a group-membership bitvector corresponds to a particular row. When the bit is set to a first bit value such as a "1" within a group-membership bitvector, this indicates that a group for the particular row has been identified. If none of the group-membership bitvectors that have already been generated have set the bit for a particular row to the first bit value, then the particular row has not yet been associated with a group. Therefore, that particular row may be provided to filter unit 112 for subsequent processing to determine to which group the particular row belongs.

[0096] Combine unit 116 may perform bitwise operations on one or more of the group-membership bitvectors generated at step 612 to generate a bitmask identifying ungrouped rows. In one embodiment, after the first bitvector is generated, a bitwise NOT operation may be performed on the bitvector. This results in a bitmask where the first bit value identifies rows that have not been previously grouped. Accordingly, address generator 402 may operate in the same manner to translate the bitmask into memory addresses for these rows. Address generator 402 may then request retrieval of these rows, and only these rows, such that only rows that are not already assigned to groups are provided to filter unit 112 for subsequent grouping and aggregation operations.

[0097] After a second group-membership bitvector has been generated, the NOT operation alone will not work to identify previously ungrouped rows, because there are now multiple bitvectors in the set. Therefore, to generate the bitmask identifying previously ungrouped rows, combine unit 116 may perform a bitwise exclusive or (XOR) operation between the group-membership bitvector for the current group and the previously generated bitmask. This process is illustrated in the example implementation below.

BLOCK-BY-BLOCK GROUPING AND AGGREGATION

[0098] In one embodiment, the grouping and aggregation operations described above may be performed on a block-by-block basis. For example, table data may be stored in a hybrid-columnar format in a plurality of RAM blocks. The process of FIG. 6 may be implemented on a first block to generate a first result set identifying the groups and the aggregate values within the first block. This process may repeat in the same fashion for each

of the remaining RAM blocks. Accordingly, a result set identifying the groups and aggregate results is generated for each remaining block.

[0099] Depending on the particular implementation, output cache 120 may fill with data before processing on the plurality of blocks has completed. For example, output cache 120 may not have sufficient storage to store the result sets for every block if the relevant data is spread over many blocks. Storage in output cache 120 may also be consumed more quickly if there are a large number of groups or the column data elements are large.

[0100] To free up storage space in output cache 120 or to generate a final result, a plurality of result sets for different blocks may be grouped and aggregated according to the process describe in FIG. 6. For example, after the output cache fills or the amount of available storage space is otherwise less than a threshold, the data in output cache 120 may be sent to the input cache 110. A group and aggregate operation is then run on the input cache data in the same fashion as described previously. Thus, the results records of the different result sets are grouped and aggregated, which may free up more storage by consolidating the result sets. If this process does not free up space in output cache 120 or the output cache 120 reaches a state of such high occupancy that performance seriously degrades, then the output cache contents may be passed to another unit such as a general purpose processor for larger-scale aggregation.

EXAMPLE SYSTEM IMPLEMENTATION OF GROUPING/AGGREGATION OPERATIONS

[0101] FIGS. 7A to 7E are a series of block diagrams illustrating different states of a system in the process of performing grouping and aggregation operations, according to an embodiment. In particular, these figures illustrate system 500 processing the sample Query 3 shown in Table 6 above.

[0102] FIG. 7A shows a system diagram after processing a first group for Query 3. As illustrated, the SALESMAN and the AMOUNT column are divided between a plurality of blocks. A first portion of the SALESMAN column, S1 704, and a first portion of the AMOUNT column, A1 706, are stored in block 702, and a second portion of the SALESMAN column, S2 712, and a second portion of the AMOUNT column, A2 714, are stored in block 710. In an embodiment, the data is stored within these blocks in a hybrid-columnar format.

[0103] When Query 3 is received, system control 108 causes a first portion of the SALESMAN column, S1 704, to be loaded into input cache 110. The SALESMAN column may be streamed from input cache 110 to filter unit 112 to group rows based on the

SALESMAN column. In order to compute the first group, system control 108 programs filter unit 112 to use equivalence to the first element of the SALESMAN as a filtering predicate. This is represented by predicate 720 of FIG. 7A. In the present example, the first element of the SALESMAN column is "Pedro", so the first group is the group of rows where SALESMAN=Pedro.

[0104] Filter unit 112 may operate in the same manner described above for the filtering and project operations with additional logic for determining the first element of a column for use in predicate evaluation. Once programmed with predicate 720, the remaining rows of the SALESMAN column are streamed from input cache 110 to filter unit 112 for evaluation. Given the sample sales table shown in Table 3 and the filter SALESMAN=Pedro, filter unit 112 would generate the group-membership bitvector shown in Table 15 below.

SALESMAN	Bitvector1
Pedro	1
Pedro	1
Alex	0
Michael	0
Alex	0
Alex	0
Pedro	1

Table 15: Filter unit output from predicate SALESMAN = Pedro

[0105] Group-membership bitvector1 corresponds to the predicate results of predicate 720. Each bit in the group-membership bitvector with a value of "1" identifies a row in the SALESMAN column with a value of "Pedro." Thus, all rows that belong to a first group are identified by the same bit value. Conversely, each bit that has a bit value of "0" identifies a row that does not have a value of "Pedro" and, therefore, does not belong to the first group.

[0106] After filter unit 112 has generated the group-membership bitvector shown in Table 15, the group-membership bitvector may then be sent to aggregation unit 502. Aggregation unit 502 operates on the rows of the AMOUNT column indicated by the bitvector as satisfying predicate 720. In one embodiment, address generator 402 translates the group-membership bitvector into memory addresses of the rows in A1 706 for which the corresponding bit value equals "1". These memory addresses are used to retrieve from RAM 102, into input cache 110, the values from A1 706 of the rows that belong to the group. These values are then provided to aggregation unit 502 for aggregation.

[0107] In addition to the group-membership bitvector, aggregation unit 502 also receives an indication of the type of aggregation operation it is to perform. For example, system control 108 may program aggregation unit 502 to perform an aggregation function specified in the query. Example aggregation functions include without limitation the aggregation functions shown in Table 5. In the case of Query 3, aggregation unit 502 sums the data stored in the rows of the AMOUNT column indicated by group-membership Bitvector1. Aggregation unit 502 may generate two outputs for the result set: the name of the group and the result of the aggregation operation for the group. The output of a first group when processing Query 3 is shown in Table 16 below. Aggregation unit 502 stores this output (aggregate result 750) in output cache 120.

GROUP	SUM
Pedro	1070

Table 16: First output of aggregate unit

Some aggregation operations may involve more than two outputs. For example, the AVG function may save GROUP, RUNNING AVERAGE, and TOTAL ELEMENTS in order to calculate the average for subsequent occurences of the group.

[0108] FIG. 7B is a system diagram after processing a second group. Concurrently with aggregation unit 502 operating on the first group-membership bitvector to generate aggregate results for the first group, filter unit 112 may begin producing the next group-membership bitvector for a second group. This time, filter unit 112 is only fed the rows that produced a "0" the first time they were passed through the filter unit. In an example embodiment, combine unit 116 performs bitwise NOT operation 740 on Bitvector1 to generate a bitvector mask ~Bitvector1 730 and stores this bitvector mask in bitvector cache 114. The bitvector mask ~Bitvector1 730 identifies all rows that have not yet been grouped.

[0109] Once generated, this bitvector mask may then be sent to address generator 402, which translates ~Bitvector1 730 into memory addresses for the rows that have not been previously grouped and causes only these rows to be delivered to filter unit 112. Filter unit 112 may then operate on these ungrouped rows by taking the first element from the previously ungrouped rows provided by address generator 402 and using equivalence to this first element as the new predicate 722. In the present example, equivalence to the value "Alex" is used as the new predicate 722. Filter unit 112 evaluates the remaining rows that have not been previously grouped using predicate 722 to generate a group-membership bitvector for the second group, Bitvector2 732 and stores this group-membership bitvector in

bitvector cache 114. Each bit in Bitvector2 732 with a bit value "1" corresponds to a row belonging to the second group. Aggregation unit 502 uses Bitvector2 to generate the aggregate result 752, which sums the AMOUNT column for those rows in the second group. Result 752 is stored in output cache 120.

[0110] FIG. 7C is a system diagram after processing a third group. Processing the third group may proceed in a similar fashion to processing the second group, except that a different bitvector mask is used to indicate which rows should be sent to filter unit 112. To determine which rows should be provided to filter unit 112, combine unit 116 may perform an exclusive or (XOR 742) bitwise operation with Bitvector2 and the previous bitvector mask ~Bitvector1 to generate the new bitvector mask BV mask 736. Each bit of BV mask 736 with a bit value of "1" corresponds to a row that has not yet been grouped. Address generator 402 uses BV mask 736 to provide filter unit 112 with rows that have not yet been grouped. Filter unit 112 uses equivalence to the value "Michael" as the new predicate 724 to generate Bivector3 734. Aggregation unit 502 uses Bitvector3 734 to generate the aggregate result 754, which sums the AMOUNT column for those rows in the third group. Result 754 is stored in output cache 120. Table 17 below shows the group-membership bitvectors generated in these steps. If there are additional groups in a block, their processing would be analogous to that of the third group.

SALESMAN	Bitvector1	Bitvector2	Bitvector3
Pedro	1	0	0
Pedro	1	0	0
Alex	0	1	0
Michael	0	0	1
Alex	0	1	0
Alex	0	1	0
Pedro	1	0	0

Table 17: Bitvectors produced through the third pass of the filter unit

At the end of aggregation of data in the first block, the table stored in output cache 120 may be represented by the table shown in Table 18 below.

GROUP	SUM
Pedro	1070
Alex	1400

Michael	350

Table 18: Aggregate output after a single block

[0111] FIG. 7D is a system diagram after two blocks in RAM have been processed. In particular, the same processes that were applied to block 702 may be applied to block 710 to group and aggregate data stored in the second block. Block 710 stores a second portion of the SALESMAN column, S2 712, and a second portion of the AMOUNT column, A2 714. For purposes of simplicity, it is assumed that, in block 710, the sales for Pedro, Alex, and Michael sum to 1000, 2000, and 3000, respectively. Thus, the grouping and aggregation operations generate result set 756, which is stored in output cache 120 along with the result set generated when processing the first block. Thus, output cache 120 stores combined result set 758. Table 19 below shows a representation of the output after two blocks have been processed.

GROUP	SUM
Pedro	1070
Alex	1400
Michael	350
Pedro	1000
Alex	2000
Michael	3000

Table 19: Aggregate output after two blocks

[0112] FIG. 7E is a system diagram after results in the output cache has been processed. If available storage in output cache 120 is less than a threshold or a final aggregate result for each of the blocks is ready to be computed, result set 758 may be sent to input cache 110 for further processing and consolidation. A grouping and aggregation operation is then run on the input cache data in the same fashion as described previously. After this aggregate pass, the data that ends up in the result cache is shown in Table 20 below. This data is stored as result 760 in output cache 120.

GROUP	SUM
Pedro	2070
Alex	3400
Michael	3350

Table 20: Result cache data after aggregate on a full result cache

[0113] Thus, the separate result sets generated for each block are grouped and aggregated to generate a single result set for both blocks.

[0114] If there are additional RAM blocks remaining, this process continues until all blocks are aggregated. If the result cache completely fills or reaches a state of such high occupancy that performance seriously degrades, then the result cache contents may be passed to another unit such as a general purpose processor for larger-scale aggregation.

CACHE SIZING

[0115] The sizes of input cache 110, bitvector cache 114, and output cache 120 may vary depending on the particular implementation. Small caches have several benefits, including faster operation and smaller area consumption. However, larger cache sizing may reduce the number of RAM accesses during the filter and project operations. Therefore, the optimal cache size of the various caches may vary depending on the implementation.

[0116] In one embodiment, the size of input cache 110, bitvector cache 114 and/or output cache 120 is selected to approximate the size of one or more blocks of RAM, such as blocks 104A to 104N. With cache sizes that approximate block sizes, the system may efficiently perform filter and project operations on a block-by-block or multi-block basis with limited RAM accesses.

[0117] In another embodiment, the caches are sized such that the input cache has sufficient storage for the column being grouped as well as the column storing the aggregate data. Both of these columns may be processed multiple times if the group cardinality is greater than one in a block, so having a cache large enough to store both columns may save the time and power of reading them from RAM repeatedly.

[0118] In another embodiment, the output cache may be sized depending on the expected cardinality of the overall dataset as well as the cardinality within blocks. The higher the cardinality of the number of groups within a block, the quicker the output cache will fill causing the result data to be aggregated more often during the grouping and aggregation operations described above. Likewise, if the overall cardinality of the group data is high, the output cache may not be able to hold the results for every group no matter how many extra aggregates are performed. If the output cache is full even after an aggregate has been run on its contents, a higher-level processing node may be used to complete the aggregate, which may cause degradation in performance. Therefore, an output cache that is large enough to hold at least the final result set may improve performance.

[0119] The size of the caches may also be selected based on the relative speeds of the memory interface for RAM 102 and database units such as filter unit 112, combine unit 116, and project unit 118. If the memory is fast compared to the database units, then a smaller cache may be preferable because the cost associated with frequent loads may be small. Conversely, if the memory is slow compared to the processing speed of the database units, then a larger cache size may be more efficient.

[0120] In another embodiment, the size of input cache 110, bitvector cache 114 and/or output cache 120 is selected based on the nature of the database workload. In some instances, column data within a block may be processed by the database units multiple times such as when many predicates are applied to the same column. If such a scenario is common in a particular implementation, then larger cache sizes may improve performance by allowing column data to reside in the caches for a greater period of time.

MULTI-COLUMN GROUPING

[0121] The above grouping and aggregation example describes processing Query 3 from Table 6, but the same system may process Query 4, which groups data based on multiple columns instead of a single column. In one embodiment, the columns involved in the GROUP BY may be combined together. For example, in the case of Query4, the SALESMAN and CUSTOMER columns may be combined, such as by concatenation. The combined column may then be sent to filter unit 112 in step 602. The two columns would then be separated at some point, such as before being written into the output cache. System 500 may include an additional database unit for combining the columns before they are sent to filter unit 112 and an additional database unit for separating the columns before the output cache 120.

[0122] In another embodiment, a filter unit that is capable of operating on multiple columns, such as described above, is used to process multi-column groupings. For example, a filter unit with multiple column inputs would be able to implement and evaluate a predicate such as SALESMAN = first element of SALESMAN column AND CUSTOMER = first element of CUSTOMER column in a single pass. For query3, a filter unit capable of handling a two-column input would be sufficient. If the GROUP BY groups data by more than two columns, the filter unit may be configured to accept more columns or the system may iterate through the combinations of columns.

MULTIPLE GROUPS PER PASS

[0123] In an embodiment, filter unit 112 may be configured to process multiple streams such that multiple groups may be determined in a single pass. For example, as the CUSTOMER column is processed by filter unit 112, one stream may apply the predicate SALESMAN = first element of SALESMAN, while another stream waits for the first case of SALESMAN != first element of SALESMAN and uses that SALESMAN value as the equivalence predicate. This approach is similar to the techniques for performing multi-column grouping described in the preceding section, but the same column is sent to all streams and the predicates are based on the results of previous predicates.

[0124] According to this process, filter unit 112 generates a group-membership bitvector for each group that it processes. If the filter unit is capable of producing *N* groups per pass, where *N* represents a positive integer value, then bitvector cache 114 may be configured to store *N*+2 group-membership bitvectors: *N* bitvectors that are the output of filter unit 112, one bitvector that ORs the *N* bitvectors together, and the bitvector mask that operates on the ORed bitvectors as described in the example implementation above.

[0125] In alternative embodiments, bitvector cache 114 may be configured to store *N* or *N*+1 bitvectors, depending on the implementation. For example, some of the *N* bitvectors can be overwritten by the OR combination or the bitvector mask.

[0126] When aggregating data, aggregation unit 502 may operate on the *N* bitvectors sequentially. Alternatively, if aggregation unit 502 is also configured to process multiple streams, aggregation unit 502 may operate on these bitvectors simultaneously.

ADDITIONAL PARALLELISM

[0127] The techniques described above illustrate a sequential processing of the steps presented. However, some of these steps may be performed in parallel, depending on the implementation. For example, combine unit 116 could process bitvectors as the bits became available from filter unit 112 rather than waiting for filter unit 112 to complete the predicate evaluation. As another example, while project unit 118 is processing a column for output cache 120, filter unit 112 may begin processing the next column to be filtered. Other steps such as cache loading and access could be performed in parallel with other filtering and projection steps as well. In yet another example, while aggregation unit 502 is processing a column for output cache 120, filter unit 112 may start processing the next group. Other operations such as cache loading and access could also operate in parallel.

HARDWARE OVERVIEW

[0128] According to one embodiment, the techniques described herein are implemented by one or more special-purpose computing devices. The special-purpose computing devices may be hard-wired to perform the techniques, or may include digital electronic devices such as one or more application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs) that are persistently programmed to perform the techniques, or may include one or more general purpose hardware processors programmed to perform the techniques pursuant to program instructions in firmware, memory, other storage, or a combination. Such special-purpose computing devices may also combine custom hard-wired logic, ASICs, or FPGAs with custom programming to accomplish the techniques. The special-purpose computing devices may be desktop computer systems, portable computer systems, handheld devices, networking devices or any other device that incorporates hard-wired and/or program logic to implement the techniques.

[0129] For example, FIG. 8 is a block diagram that illustrates a computer system 800 upon which an embodiment of the invention may be implemented. Computer system 800 includes a bus 802 or other communication mechanism for communicating information, and a hardware processor 804 coupled with bus 802 for processing information. Hardware processor 804 may be, for example, a general purpose microprocessor.

[0130] Computer system 800 also includes a main memory 806, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 802 for storing information and instructions to be executed by processor 804. For example, RAM 102 may be implemented in main memory 806. Main memory 806 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 804. Such instructions, when stored in non-transitory storage media accessible to processor 804, render computer system 800 into a special-purpose machine that is customized to perform the operations specified in the instructions.

[0131] Computer system 800 further includes a read only memory (ROM) 808 or other static storage device coupled to bus 802 for storing static information and instructions for processor 804. A storage device 810, such as a magnetic disk or optical disk, is provided and coupled to bus 802 for storing information and instructions.

[0132] Computer system 800 may be coupled via bus 802 to a display 812, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 814, including alphanumeric and other keys, is coupled to bus 802 for communicating information and command selections to processor 804. Another type of user input device is cursor control 816, such as a mouse, a trackball, or cursor direction keys for communicating direction

information and command selections to processor 804 and for controlling cursor movement on display 812. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0133] Computer system 800 may also include query processing logic 832 for performing filter, project, grouping, and/or aggregation operations. Query processing logic 832 may be implemented using one or more elements illustrated in system 100, system 400, or system 500.

[0134] Computer system 800 may implement the techniques described herein using customized hard-wired logic, one or more ASICs or FPGAs, firmware and/or program logic which in combination with the computer system causes or programs computer system 800 to be a special-purpose machine. According to one embodiment, the techniques herein are performed by computer system 800 in response to processor 804 executing one or more sequences of one or more instructions contained in main memory 806. Such instructions may be read into main memory 806 from another storage medium, such as storage device 810. Execution of the sequences of instructions contained in main memory 806 causes processor 804 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions.

[0135] The term "storage media" as used herein refers to any non-transitory media that store data and/or instructions that cause a machine to operate in a specific fashion. Such storage media may comprise non-volatile media and/or volatile media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 810. Volatile media includes dynamic memory, such as main memory 806. Common forms of storage media include, for example, a floppy disk, a flexible disk, hard disk, solid state drive, magnetic tape, or any other magnetic data storage medium, a CD-ROM, any other optical data storage medium, any physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, NVRAM, any other memory chip or cartridge.

[0136] Storage media is distinct from but may be used in conjunction with transmission media. Transmission media participates in transferring information between storage media. For example, transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 802. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0137] Various forms of media may be involved in carrying one or more sequences of one or more instructions to processor 804 for execution. For example, the instructions may initially be carried on a magnetic disk or solid state drive of a remote computer. The remote

computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 800 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 802. Bus 802 carries the data to main memory 806, from which processor 804 retrieves and executes the instructions. The instructions received by main memory 806 may optionally be stored on storage device 810 either before or after execution by processor 804.

[0138] Computer system 800 also includes a communication interface 818 coupled to bus 802. Communication interface 818 provides a two-way data communication coupling to a network link 820 that is connected to a local network 822. For example, communication interface 818 may be an integrated services digital network (ISDN) card, cable modem, satellite modem, or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 818 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 818 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0139] Network link 820 typically provides data communication through one or more networks to other data devices. For example, network link 820 may provide a connection through local network 822 to a host computer 824 or to data equipment operated by an Internet Service Provider (ISP) 826. ISP 826 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 828. Local network 822 and Internet 828 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 820 and through communication interface 818, which carry the digital data to and from computer system 800, are example forms of transmission media.

[0140] Computer system 800 can send messages and receive data, including program code, through the network(s), network link 820 and communication interface 818. In the Internet example, a server 830 might transmit a requested code for an application program through Internet 828, ISP 826, local network 822 and communication interface 818.

[0141] The received code may be executed by processor 804 as it is received, and/or stored in storage device 810, or other non-volatile storage for later execution.

EXTENSIONS AND ALTERNATIVES

[0142] In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. The sole and exclusive indicator of the scope of the invention, and what is intended by the applicants to be the scope of the invention, is the literal and equivalent scope of the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction.

CLAIMS

What is claimed is:

1. A method comprising:

programming a filtering unit with a predicate that specifies criteria for filtering results of a query that targets a table;

wherein the predicate specifies a condition for a particular column of the table;

generating a predicate result by loading values from the particular column into an input cache and causing the filtering unit to apply the predicate to the values;

wherein the predicate result identifies rows of the table that have values, within the particular column, that satisfy the condition specified by the predicate;

selecting rows to return, as results of the query, based at least in part on the predicate result;

returning the selected rows as results to the query;

wherein the method is performed by one or more computing devices.

2. The method of Claim 1 wherein the predicate result is a bitvector and each bit of the bitvector corresponds to a particular row and identifies whether the particular row satisfies the condition specified by the first predicate.

3. The method of Claim 1, wherein:

said predicate is a first predicate of a set of predicates associated with the query;

said predicate result is a first predicate result;

the method further comprises:

programming the filtering unit with a second predicate, of the set of predicates, that specifies second criteria for filtering results of the query;

wherein the second predicate specifies a second condition for a second column of the table;

generating a second predicate result by loading values from the second column into the input cache and causing the filtering unit to apply the second predicate to the values;

wherein the second predicate result identifies rows of the table that have values, within the second column, that satisfy the second condition specified by the second predicate;

wherein selecting rows to return as results based at least in part on the predicate result comprises:

combining the first predicate result with the second predicate result to generate a combined result;

using the combined result to identify rows that satisfy both the first predicate and the second predicate;

selecting rows identified by the combined result as satisfying both the first predicate and the second predicate.

4. The method of Claim 1 further comprising:

for each predicate of the set of predicates:

programming the filtering unit with said each predicate, of the set of predicates, that specifies particular criteria for filtering results of the query;

generating a predicate result by loading values from at least one portion of a column into the filtering unit and causing the filtering unit to apply said each predicate to the values;

wherein selecting rows to return as results comprises:

combining each predicate result to generate a final result that identifies rows that satisfy all criteria specified by the set of predicates;

selecting the rows that are identified by the final result as satisfying the criteria specified by the set of predicates.

5. The method of Claim 4,

wherein the predicate result generated for said each predicate is used to determine which row values to provide to the filtering unit for evaluation of a next predicate;

wherein the set of predicates are evaluated in a serial order;

wherein the serial order is based on a likelihood that a predicate will filter out a large number of rows;

wherein predicates that are more likely to filter out a large number of rows are evaluated before predicates that are less likely to filter out a large number of rows.

6. The method of Claim 4, wherein selecting rows as results further comprises:

translating the final result into a set of memory addresses;

wherein each memory address in the set of memory addresses identifies a memory location of a row that satisfies the criteria specified by the set of predicates.

7. The method of Claim 1, wherein causing the filtering unit to apply the predicate to the values comprises:

generating a first bit value for a corresponding row of the particular column if a value of the corresponding row satisfies the condition;

generating a second bit value, different than the first bit value, for the corresponding row if the value of the corresponding row does not satisfy the condition.

8. The method of Claim 1, wherein generating the predicate result by loading values from the particular column into an input cache and causing the filtering unit to apply the predicate to the values comprises:

loading a first set of values from the particular column stored in a first block of memory:

causing the filtering unit to apply the predicate to the first set of values to generate a first bitvector that identifies rows of the table that have values stored within the first block of memory that satisfy the condition specified by the predicate;

loading a second set of values from the particular column stored in a second block of memory:

causing the filtering unit to apply the predicate to the second set of values to generate a second bitvector that identifies rows of the table that have values stored within the second block of memory that satisfy the condition specified by the predicate.

9. The method of Claim 1, wherein causing the filtering unit to apply the predicate to the values comprises causing the filtering unit to concurrently apply the predicate to a plurality of values from the particular column.

10. One or more non-transitory computer-readable media storing instructions, which, when executed by one or more processors, cause one or more computing devices to perform operations comprising:

programming a filtering unit with a predicate that specifies criteria for filtering results of a query that targets a table;

wherein the predicate specifies a condition for a particular column of the table;

generating a predicate result by loading values from the particular column into an input cache and causing the filtering unit to apply the predicate to the values;

wherein the predicate result identifies rows of the table that have values, within the particular column, that satisfy the condition specified by the predicate;

selecting rows to return, as results of the query, based at least in part on the predicate result;

returning the selected rows as results to the query;

wherein the method is performed by one or more computing devices.

11. The one or more non-transitory computer-readable media of Claim 10, wherein the predicate result is a bitvector and each bit of the bitvector corresponds to a particular row and identifies whether the particular row satisfies the condition specified by the first predicate.

12. The one or more non-transitory computer-readable media of Claim 10, wherein:

said predicate is a first predicate of a set of predicates associated with the query;

said predicate result is a first predicate result;

the one or more non-transitory computer-readable media further storing instructions causing the one or more computing devices to perform operations comprising:

programming the filtering unit with a second predicate, of the set of predicates, that specifies second criteria for filtering results of the query;

wherein the second predicate specifies a second condition for a second column of the table;

generating a second predicate result by loading values from the second column into the input cache and causing the filtering unit to apply the second predicate to the values;

wherein the second predicate result identifies rows of the table that have values, within the second column, that satisfy the second condition specified by the second predicate;

wherein selecting rows to return as results based at least in part on the predicate result comprises:

combining the first predicate result with the second predicate result to generate a combined result;

using the combined result to identify rows that satisfy both the first predicate and the second predicate;

selecting rows identified by the combined result as satisfying both the first predicate and the second predicate.

13. The one or more non-transitory computer-readable media of Claim 10 further storing instructions that cause the one or more computing devices to perform operations comprising:
for each predicate of the set of predicates:
programming the filtering unit with said each predicate, of the set of predicates, that specifies particular criteria for filtering results of the query;
generating a predicate result by loading values from at least one portion of a column into the filtering unit and causing the filtering unit to apply said each predicate to the values;
wherein instructions for selecting rows to return as results comprise instructions for:
combining each predicate result to generate a final result that identifies rows that satisfy all criteria specified by the set of predicates;
selecting the rows that are identified by the final result as satisfying the criteria specified by the set of predicates.

14. The one or more non-transitory computer-readable media of Claim 13,
wherein the predicate result generated for said each predicate is used to determine which row values to provide to the filtering unit for evaluation of a next predicate;
wherein the set of predicates are evaluated in a serial order;
wherein the serial order is based on a likelihood that a predicate will filter out a large number of rows;
wherein predicates that are more likely to filter out a large number of rows are evaluated before predicates that are less likely to filter out a large number of rows.

15. The one or more non-transitory computer-readable media of Claim 13, wherein instructions for selecting rows as results comprises instructions for:
translating the final result into a set of memory addresses;
wherein each memory address in the set of memory addresses identifies a memory location of a row that satisfies the criteria specified by the set of predicates.

16. The one or more non-transitory computer-readable media of Claim 10, wherein instructions for causing the filtering unit to apply the predicate to the values comprise instructions for:
generating a first bit value for a corresponding row of the particular column if a value of the corresponding row satisfies the condition;

generating a second bit value, different than the first bit value, for the corresponding row if the value of the corresponding row does not satisfy the condition.

17. The one or more non-transitory computer-readable media of Claim 10, wherein instructions for generating the predicate result by loading values from the particular column into an input cache and causing the filtering unit to apply the predicate to the values comprise instructions for:

loading a first set of values from the particular column stored in a first block of memory:

causing the filtering unit to apply the predicate to the first set of values to generate a first bitvector that identifies rows of the table that have values stored within the first block of memory that satisfy the condition specified by the predicate;

loading a second set of values from the particular column stored in a second block of memory:

causing the filtering unit to apply the predicate to the second set of values to generate a second bitvector that identifies rows of the table that have values stored within the second block of memory that satisfy the condition specified by the predicate.

18. The one or more non-transitory computer-readable media of Claim 10, wherein instructions for causing the filtering unit to apply the predicate to the values comprises instructions for causing the filtering unit to concurrently apply the predicate to a plurality of values from the particular column.

19. A system for performing filtering and projection operations comprising:

a control unit configured to:

receive a set of predicates that specify criteria for filtering results to a query that targets a table;

program a filter unit to perform a apply a particular predicate, of the set of predicates, that specifies a condition for a particular column of the table;

a filter unit configured to:

receive values from the particular column and apply the particular predicate to the values;

generate a bitvector that identifies rows of the table that have values, within the particular column, that satisfy the condition specified by the particular predicate;

a project unit configured to:

select rows to return, as results of the query, based at least in part on the bitvector;

return the selected rows as results to the query.

20. The system of Claim 19 wherein:

said predicate is a first predicate of a set of predicates associated with the query;

the control unit is further configured to:

program the filter unit with a second predicate, of the second predicate, that specifies a second condition for a second column of the table

the filter unit is further configured to:

receive values from the second column and apply the second predicate to the values;

generate a second bitvector that identifies rows of the table that have values, within the second column, that satisfy the second condition specified by the second predicate;

the system further comprises a combine unit configured to:

combine the first bitvector with the second bitvector by applying a bitwise operator to generate a third bitvector;

wherein the project unit is configured to

use the third bitvector to identify rows that satisfy both the first predicate and the second predicate;

select rows identified by the third bitvector as satisfying both the first predicate and the second predicate

AMENDED CLAIMS
received by the International Bureau on 20 December 2013(20.12.2013)

1. A method comprising:

programming a circuit into reconfigurable hardware of a filtering unit based on a predicate that specifies criteria for filtering results of a query that targets a table;

wherein the predicate specifies a condition for a particular column of the table;

generating a predicate result by loading values from the particular column into an input cache and causing the filtering unit to apply the predicate to the values;

wherein the predicate result identifies rows of the table that have values, within the particular column, that satisfy the condition specified by the predicate;

selecting rows to return, as results of the query, based at least in part on the predicate result;

returning the selected rows as results to the query;

wherein the method is performed by one or more computing devices.

2. The method of Claim 1 wherein the predicate result is a bitvector and each bit of the bitvector corresponds to a particular row and identifies whether the particular row satisfies the condition specified by the first predicate.

3. The method of Claim 1, wherein:

said predicate is a first predicate of a set of predicates associated with the query;

said predicate result is a first predicate result;

the method further comprises:

programming a second circuit into the reconfigurable hardware of the filtering unit based on a second predicate, of the set of predicates, that specifies second criteria for filtering results of the query;

wherein the second predicate specifies a second condition for a second column of the table;

generating a second predicate result by loading values from the second column into the input cache and causing the filtering unit to apply the second predicate to the values;

wherein the second predicate result identifies rows of the table that have values, within the second column, that satisfy the second condition specified by the second predicate;

wherein selecting rows to return as results based at least in part on the predicate result comprises:

combining the first predicate result with the second predicate result to generate a combined result;

using the combined result to identify rows that satisfy both the first predicate and the second predicate;

selecting rows identified by the combined result as satisfying both the first predicate and the second predicate.

4. The method of Claim 1 further comprising:

for each predicate of the set of predicates:

programming a particular circuit into the reconfigurable hardware of the filtering unit based on said each predicate, of the set of predicates, that specifies particular criteria for filtering results of the query;

generating a predicate result by loading values from at least one portion of a column into the filtering unit and causing the filtering unit to apply said each predicate to the values;

wherein selecting rows to return as results comprises:

combining each predicate result to generate a final result that identifies rows that satisfy all criteria specified by the set of predicates;

selecting the rows that are identified by the final result as satisfying the criteria specified by the set of predicates.

5. The method of Claim 4,

wherein the predicate result generated for said each predicate is used to determine which row values to provide to the filtering unit for evaluation of a next predicate;

wherein the set of predicates are evaluated in a serial order;

wherein the serial order is based on a likelihood that a predicate will filter out a large number of rows;

wherein predicates that are more likely to filter out a large number of rows are evaluated before predicates that are less likely to filter out a large number of rows.

6. The method of Claim 4, wherein selecting rows as results further comprises:
translating the final result into a set of memory addresses;
wherein each memory address in the set of memory addresses identifies a memory location of a row that satisfies the criteria specified by the set of predicates.

7. The method of Claim 1, wherein causing the filtering unit to apply the predicate to the values comprises:
generating a first bit value for a corresponding row of the particular column if a value of the corresponding row satisfies the condition;
generating a second bit value, different than the first bit value, for the corresponding row if the value of the corresponding row does not satisfy the condition.

8. The method of Claim 1, wherein generating the predicate result by loading values from the particular column into an input cache and causing the filtering unit to apply the predicate to the values comprises:
loading a first set of values from the particular column stored in a first block of memory:
causing the filtering unit to apply the predicate to the first set of values to generate a first bitvector that identifies rows of the table that have values stored within the first block of memory that satisfy the condition specified by the predicate;
loading a second set of values from the particular column stored in a second block of memory:
causing the filtering unit to apply the predicate to the second set of values to generate a second bitvector that identifies rows of the table that have values stored within the second block of memory that satisfy the condition specified by the predicate.

9. The method of Claim 1, wherein causing the filtering unit to apply the predicate to the values comprises causing the filtering unit to concurrently apply the predicate to a plurality of values from the particular column.

10. One or more non-transitory computer-readable media storing instructions, which, when executed by one or more processors, cause one or more computing devices to perform operations comprising:

programming a circuit into reconfigurable hardware of a filtering unit with a predicate that specifies criteria for filtering results of a query that targets a table;

wherein the predicate specifies a condition for a particular column of the table;

generating a predicate result by loading values from the particular column into an input cache and causing the filtering unit to apply the predicate to the values;

wherein the predicate result identifies rows of the table that have values, within the particular column, that satisfy the condition specified by the predicate;

selecting rows to return, as results of the query, based at least in part on the predicate result;

returning the selected rows as results to the query.

11. The one or more non-transitory computer-readable media of Claim 10, wherein the predicate result is a bitvector and each bit of the bitvector corresponds to a particular row and identifies whether the particular row satisfies the condition specified by the first predicate.

12. The one or more non-transitory computer-readable media of Claim 10, wherein:

said predicate is a first predicate of a set of predicates associated with the query;

said predicate result is a first predicate result;

the one or more non-transitory computer-readable media further storing instructions causing the one or more computing devices to perform operations comprising:

programming a second circuit into the reconfigurable hardware of the filtering unit based on a second predicate, of the set of predicates, that specifies second criteria for filtering results of the query;

wherein the second predicate specifies a second condition for a second column of the table;

generating a second predicate result by loading values from the second column into the input cache and causing the filtering unit to apply the second predicate to the values;

wherein the second predicate result identifies rows of the table that have values, within the second column, that satisfy the second condition specified by the second predicate;

wherein selecting rows to return as results based at least in part on the predicate result comprises:

combining the first predicate result with the second predicate result to generate a combined result;

using the combined result to identify rows that satisfy both the first predicate and the second predicate;

selecting rows identified by the combined result as satisfying both the first predicate and the second predicate.

13. The one or more non-transitory computer-readable media of Claim 10 further storing instructions that cause the one or more computing devices to perform operations comprising:

for each predicate of the set of predicates:

programming a particular circuit into the reconfigurable hardware of the filtering unit based on said each predicate, of the set of predicates, that specifies particular criteria for filtering results of the query;

generating a predicate result by loading values from at least one portion of a column into the filtering unit and causing the filtering unit to apply said each predicate to the values;

wherein instructions for selecting rows to return as results comprise instructions for:

combining each predicate result to generate a final result that identifies rows that satisfy all criteria specified by the set of predicates;

selecting the rows that are identified by the final result as satisfying the criteria specified by the set of predicates.

14. The one or more non-transitory computer-readable media of Claim 13,

wherein the predicate result generated for said each predicate is used to determine which row values to provide to the filtering unit for evaluation of a next predicate;

wherein the set of predicates are evaluated in a serial order;

wherein the serial order is based on a likelihood that a predicate will filter out a large number of rows;

wherein predicates that are more likely to filter out a large number of rows are evaluated before predicates that are less likely to filter out a large number of rows.

15. The one or more non-transitory computer-readable media of Claim 13, wherein instructions for selecting rows as results comprises instructions for:
translating the final result into a set of memory addresses;
wherein each memory address in the set of memory addresses identifies a memory location of a row that satisfies the criteria specified by the set of predicates.

16. The one or more non-transitory computer-readable media of Claim 10, wherein instructions for causing the filtering unit to apply the predicate to the values comprise instructions for:

generating a first bit value for a corresponding row of the particular column if a value of the corresponding row satisfies the condition;

generating a second bit value, different than the first bit value, for the corresponding row if the value of the corresponding row does not satisfy the condition.

17. The one or more non-transitory computer-readable media of Claim 10, wherein instructions for generating the predicate result by loading values from the particular column into an input cache and causing the filtering unit to apply the predicate to the values comprise instructions for:

loading a first set of values from the particular column stored in a first block of memory:

causing the filtering unit to apply the predicate to the first set of values to generate a first bitvector that identifies rows of the table that have values stored within the first block of memory that satisfy the condition specified by the predicate;

loading a second set of values from the particular column stored in a second block of memory:

causing the filtering unit to apply the predicate to the second set of values to generate a second bitvector that identifies rows of the table that have values stored within the second block of memory that satisfy the condition specified by the predicate.

18. The one or more non-transitory computer-readable media of Claim 10, wherein instructions for causing the filtering unit to apply the predicate to the values comprises instructions for causing the filtering unit to concurrently apply the predicate to a plurality of values from the particular column.

19. A system for performing filtering and projection operations comprising:

a control unit configured to:

receive a set of predicates that specify criteria for filtering results to a query that targets a table;

program a circuit into reconfigurable hardware of a filter unit to apply a particular predicate, of the set of predicates, that specifies a condition for a particular column of the table;

a filter unit configured to:

receive values from the particular column and apply the particular predicate to the values;

generate a bitvector that identifies rows of the table that have values, within the particular column, that satisfy the condition specified by the particular predicate;

a project unit configured to:

select rows to return, as results of the query, based at least in part on the bitvector;

return the selected rows as results to the query.

20. The system of Claim 19 wherein:

said predicate is a first predicate of a set of predicates associated with the query;

the control unit is further configured to:

program a second circuit into reconfigurable hardware of the filter unit with a second predicate, of the second predicate, that specifies a second condition for a second column of the table

the filter unit is further configured to:

receive values from the second column and apply the second predicate to the values;

generate a second bitvector that identifies rows of the table that have values, within the second column, that satisfy the second condition specified by the second predicate;

the system further comprises a combine unit configured to:

combine the first bitvector with the second bitvector by applying a bitwise operator to generate a third bitvector;

wherein the project unit is configured to

use the third bitvector to identify rows that satisfy both the first predicate and the second predicate;

select rows identified by the third bitvector as satisfying both the first predicate and the second predicate

FIG. 1

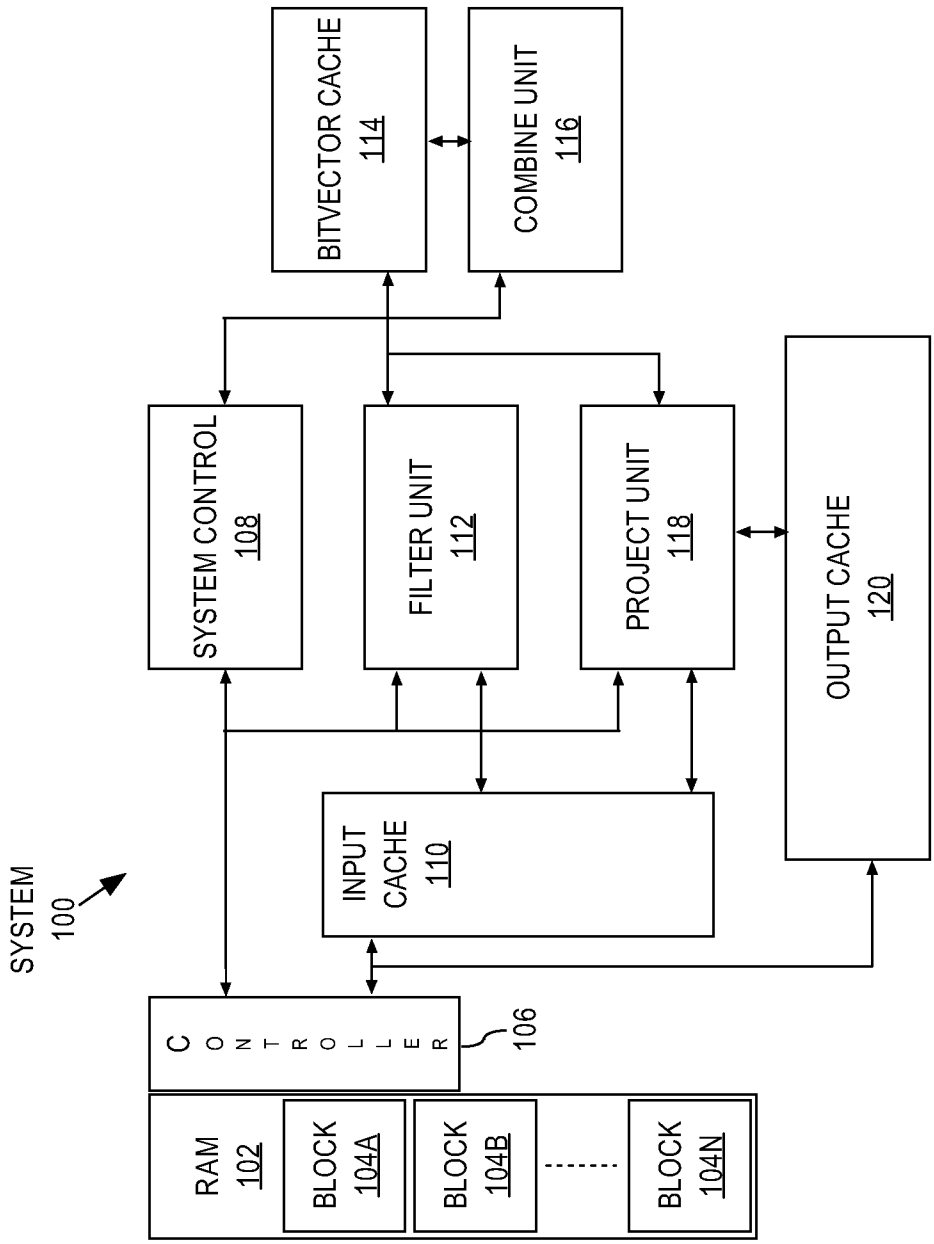
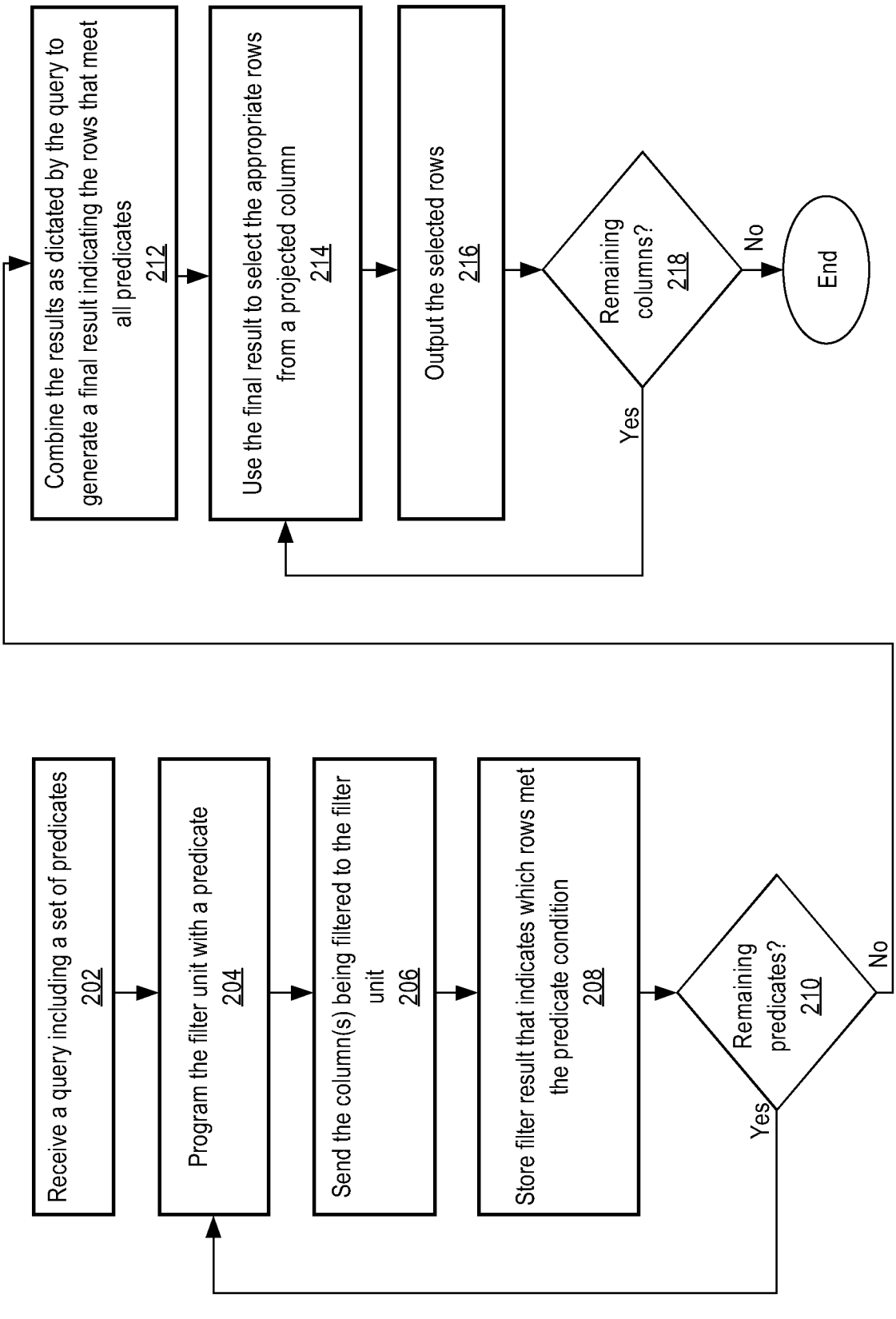

SYSTEM
100
BITVECTOR CACHE
114
COMBINE UNIT
116
SYSTEM CONTROL
108
FILTER UNIT
112
PROJECT UNIT
118
OUTPUT CACHE
120
INPUT
CACHE
110
CONTROLLER
106
RAM
102
BLOCK
104A
BLOCK
104B
BLOCK
104N

FIG. 2


Receive a query including a set of predicates
202
Program the filter unit with a predicate
204
Send the column(s) being filtered to the filter unit
206
Store filter result that indicates which rows met the predicate condition
208
Remaining predicates?
210
Yes
No
Combine the results as dictated by the query to generate a final result indicating the rows that meet all predicates
212
Use the final result to select the appropriate rows from a projected column
214
Output the selected rows
216
Remaining columns?
218
Yes
No
End

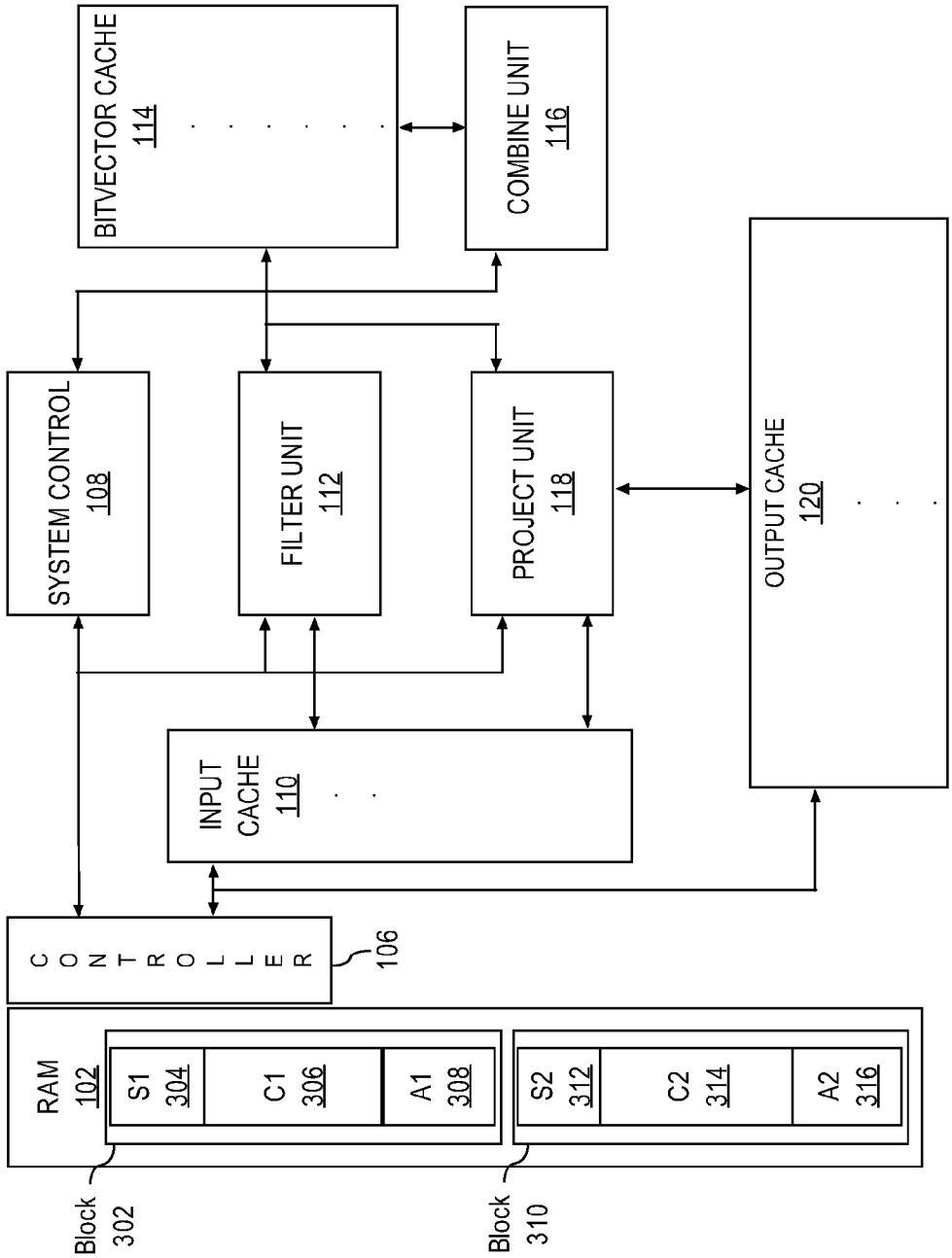

BITVECTOR CACHE
114
COMBINE UNIT
116
SYSTEM CONTROL
108
FILTER UNIT
112
PROJECT UNIT
118
OUTPUT CACHE
120
INPUT
CACHE
110
CONTROLLER
106
RAM
102
S1
304
C1
306
A1
308
S2
312
C2
314
A2
316
Block
302
Block
310

FIG. 3A

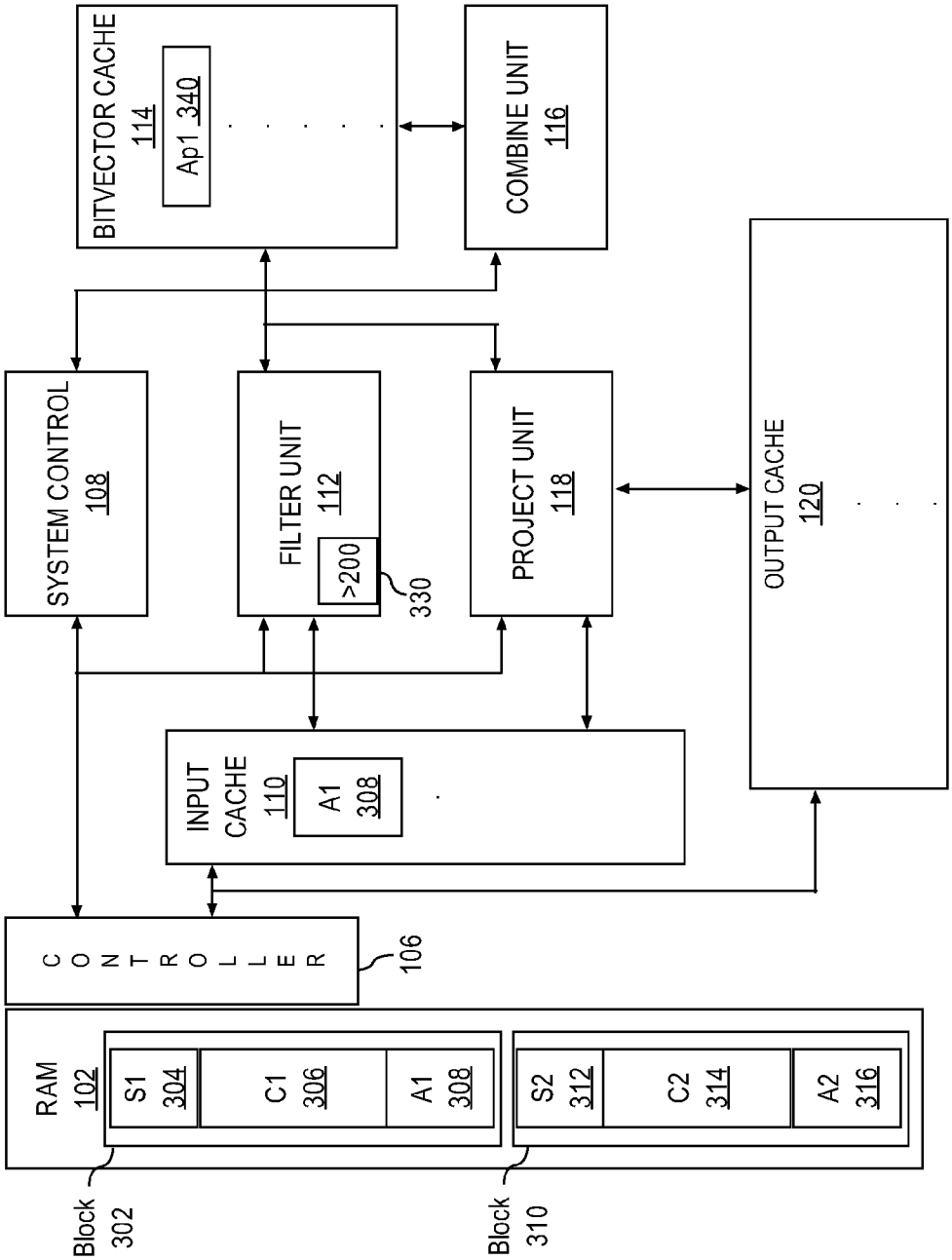

BITVECTOR CACHE
114
Ap1 340
COMBINE UNIT
116
SYSTEM CONTROL
108
FILTER UNIT
112
>200
330
PROJECT UNIT
118
OUTPUT CACHE
120
INPUT
CACHE
110
A1
308
CONTROLLER
106
RAM
102
S1
304
C1
306
A1
308
S2
312
C2
314
A2
316
Block
302
Block
310

FIG. 3B

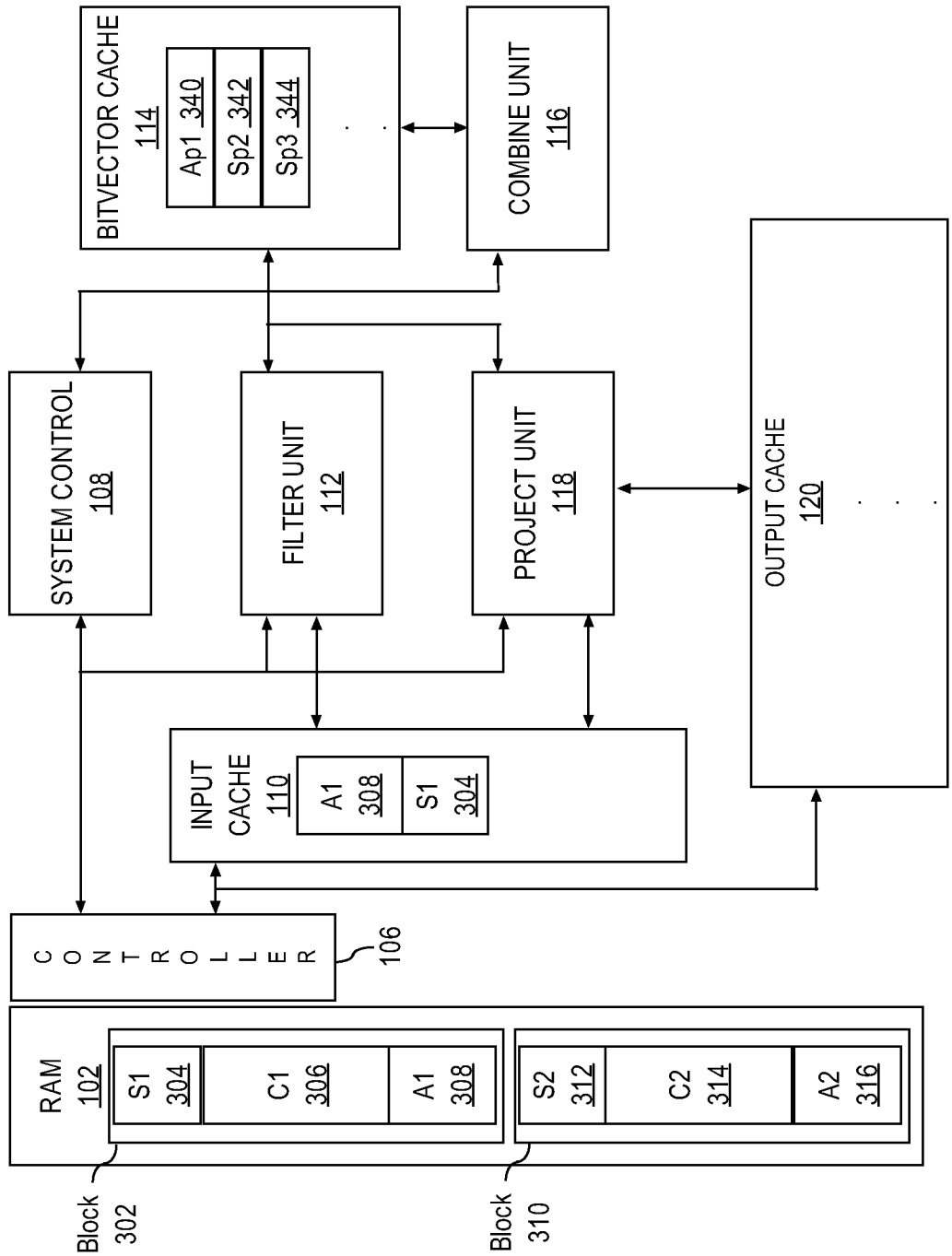

BITVECTOR CACHE
114
Ap1 340
Sp2 342
Sp3 344
COMBINE UNIT
116
SYSTEM CONTROL
108
FILTER UNIT
112
PROJECT UNIT
118
OUTPUT CACHE
120
INPUT CACHE
110
A1
308
S1
304
CONTROLLER
106
RAM
102
S1
304
C1
306
A1
308
S2
312
C2
314
A2
316
Block
302
Block
310

FIG. 3C

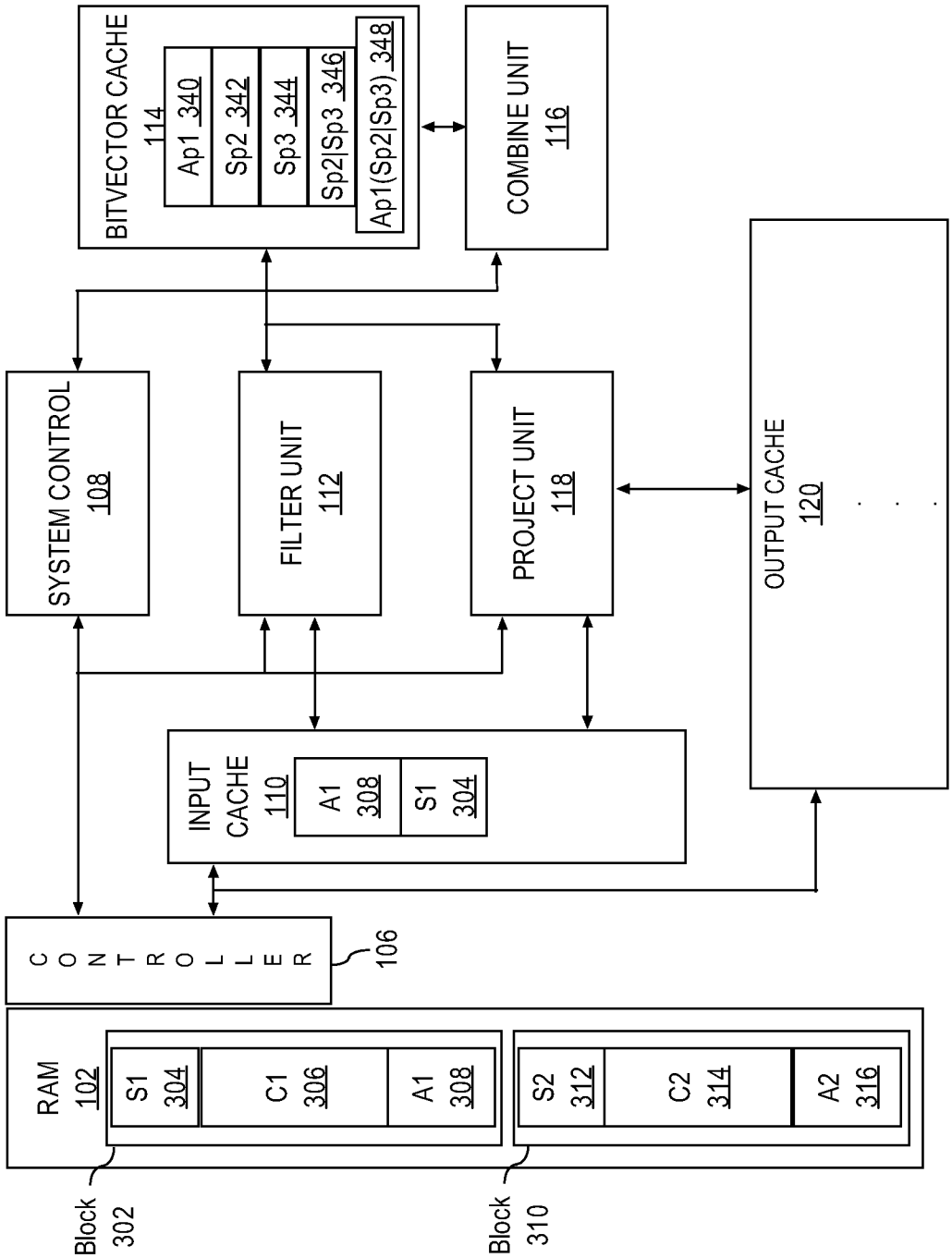

BITVECTOR CACHE
114
Ap1 340
Sp2 342
Sp3 344
Sp2|Sp3 346
Ap1(Sp2|Sp3) 348
COMBINE UNIT
116
SYSTEM CONTROL
108
FILTER UNIT
112
PROJECT UNIT
118
OUTPUT CACHE
120
INPUT CACHE
110
A1
308
S1
304
CONTROLLER
106
RAM
102
S1
304
C1
306
A1
308
S2
312
C2
314
A2
316
Block
302
Block
310

FIG. 3D

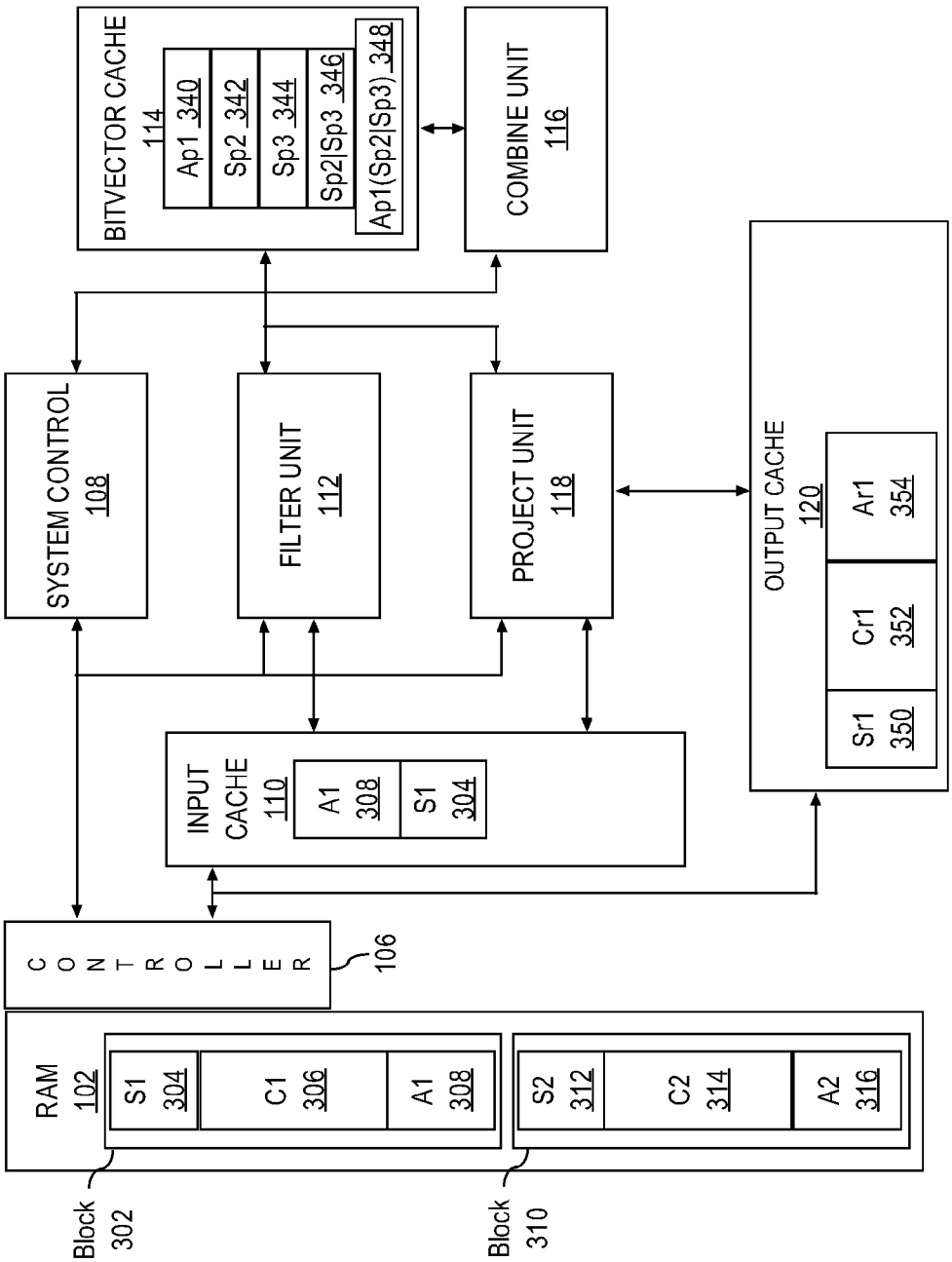

BITVECTOR CACHE 114
Ap1 340
Sp2 342
Sp3 344
Sp2|Sp3 346
Ap1(Sp2|Sp3) 348
COMBINE UNIT 116
SYSTEM CONTROL 108
FILTER UNIT 112
PROJECT UNIT 118
OUTPUT CACHE 120
Ar1 354
Cr1 352
Sr1 350
INPUT CACHE 110
A1 308
S1 304
CONTROLLER 106
RAM 102
S1 304
C1 306
A1 308
S2 312
C2 314
A2 316
Block 302
Block 310

FIG. 3E

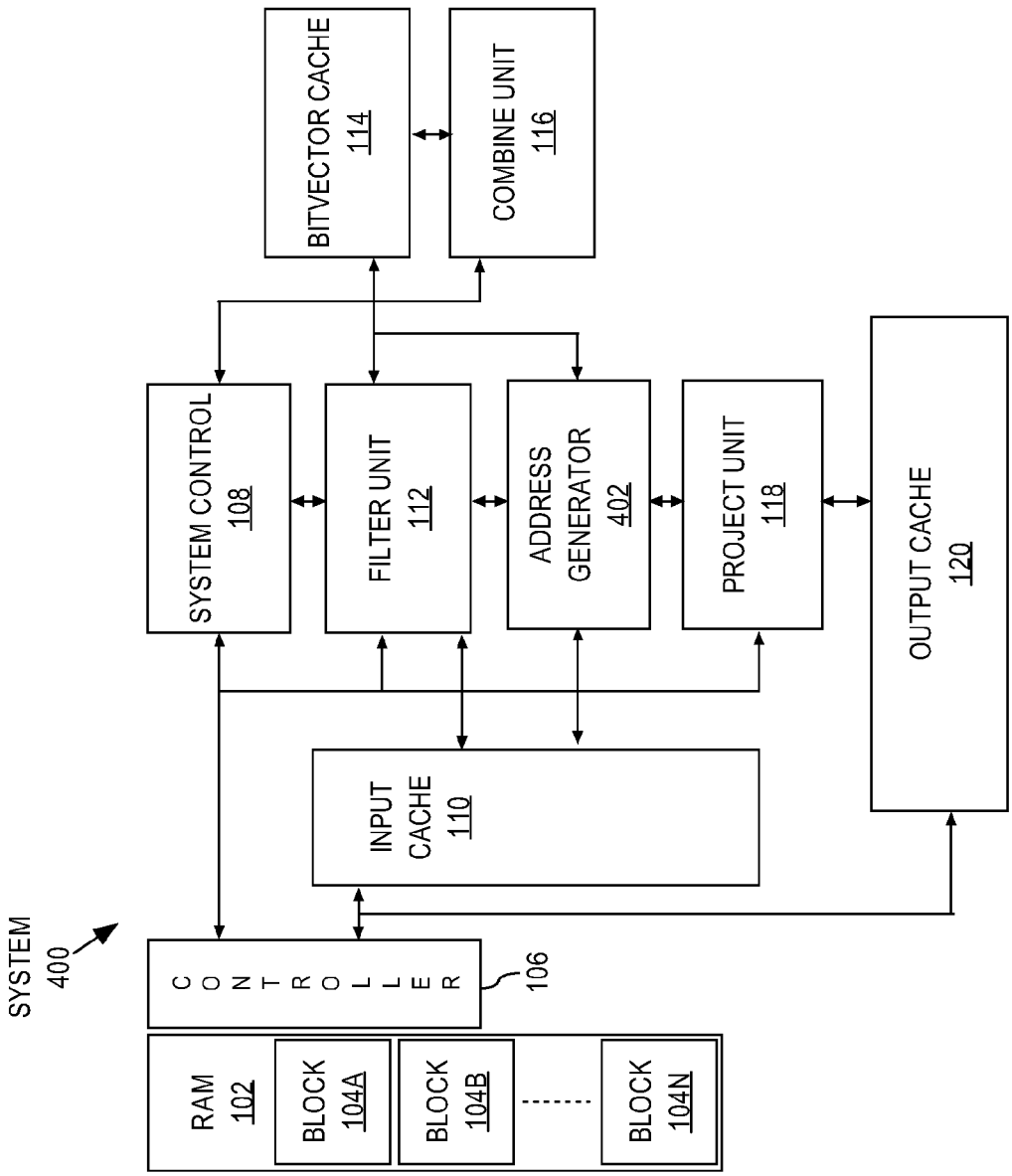

SYSTEM
400
BITVECTOR CACHE
114
COMBINE UNIT
116
SYSTEM CONTROL
108
FILTER UNIT
112
ADDRESS
GENERATOR
402
PROJECT UNIT
118
OUTPUT CACHE
120
INPUT
CACHE
110
CONTROLLER
106
RAM
102
BLOCK
104A
BLOCK
104B
BLOCK
104N

FIG. 4

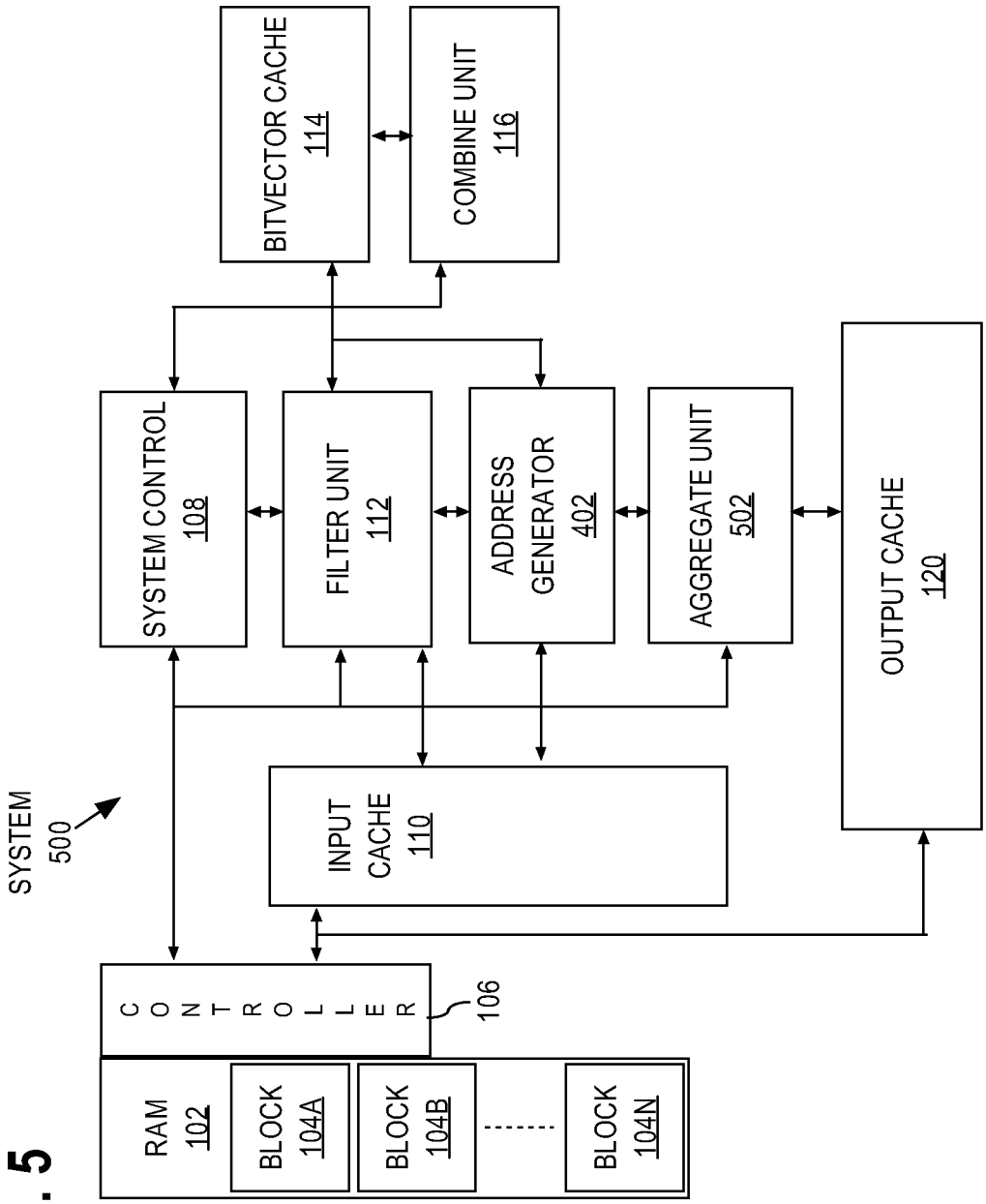

SYSTEM
500
BITVECTOR CACHE
114
COMBINE UNIT
116
SYSTEM CONTROL
108
FILTER UNIT
112
ADDRESS
GENERATOR
402
AGGREGATE UNIT
502
OUTPUT CACHE
120
INPUT
CACHE
110
CONTROLLER
106
RAM
102
BLOCK
104A
BLOCK
104B
BLOCK
104N

FIG. 5

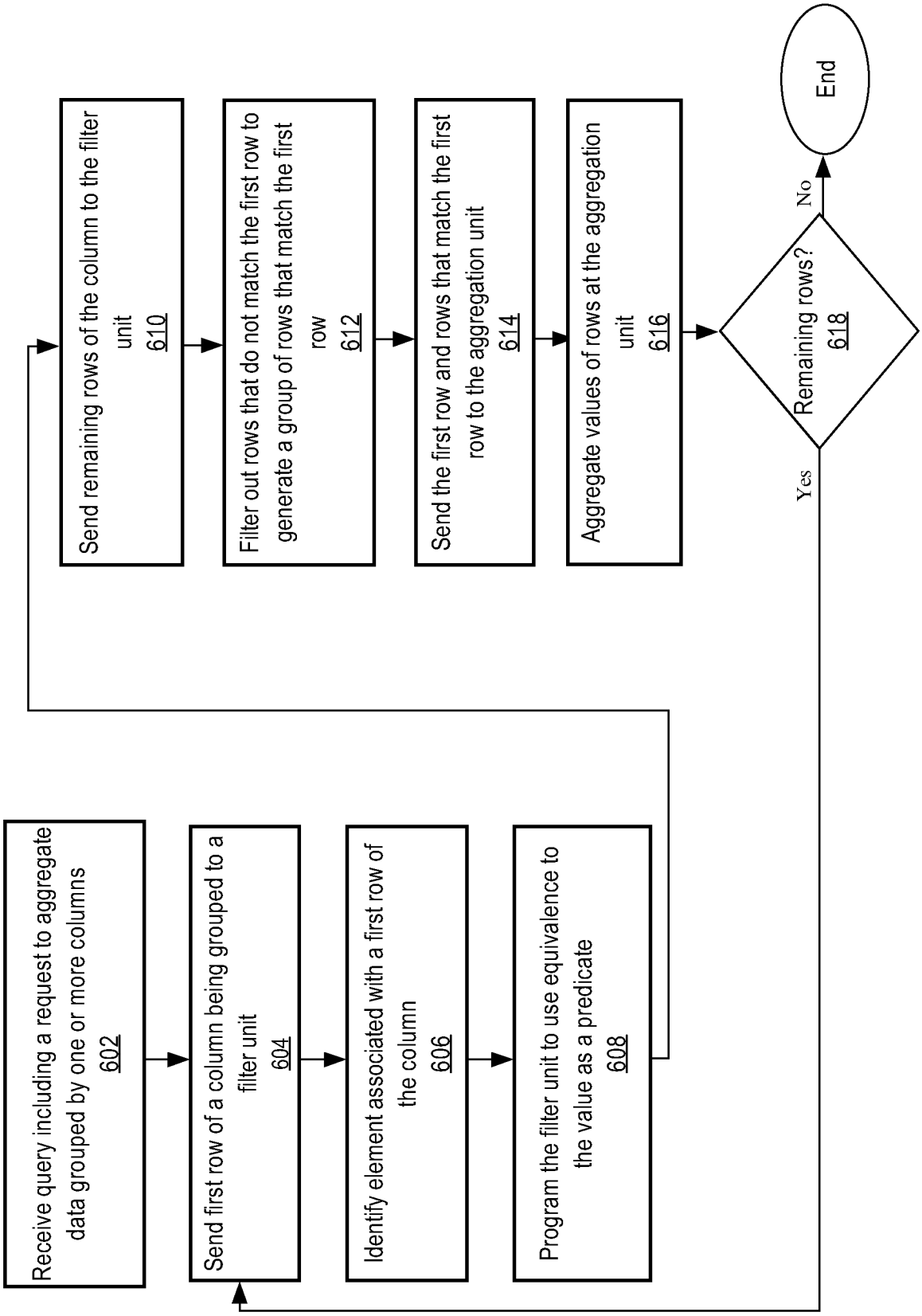

Receive query including a request to aggregate data grouped by one or more columns
602
Send first row of a column being grouped to a filter unit
604
Identify element associated with a first row of the column
606
Program the filter unit to use equivalence to the value as a predicate
608
Send remaining rows of the column to the filter unit
610
Filter out rows that do not match the first row to generate a group of rows that match the first row
612
Send the first row and rows that match the first row to the aggregation unit
614
Aggregate values of rows at the aggregation unit
616
Remaining rows?
618
Yes
No
End

FIG. 6

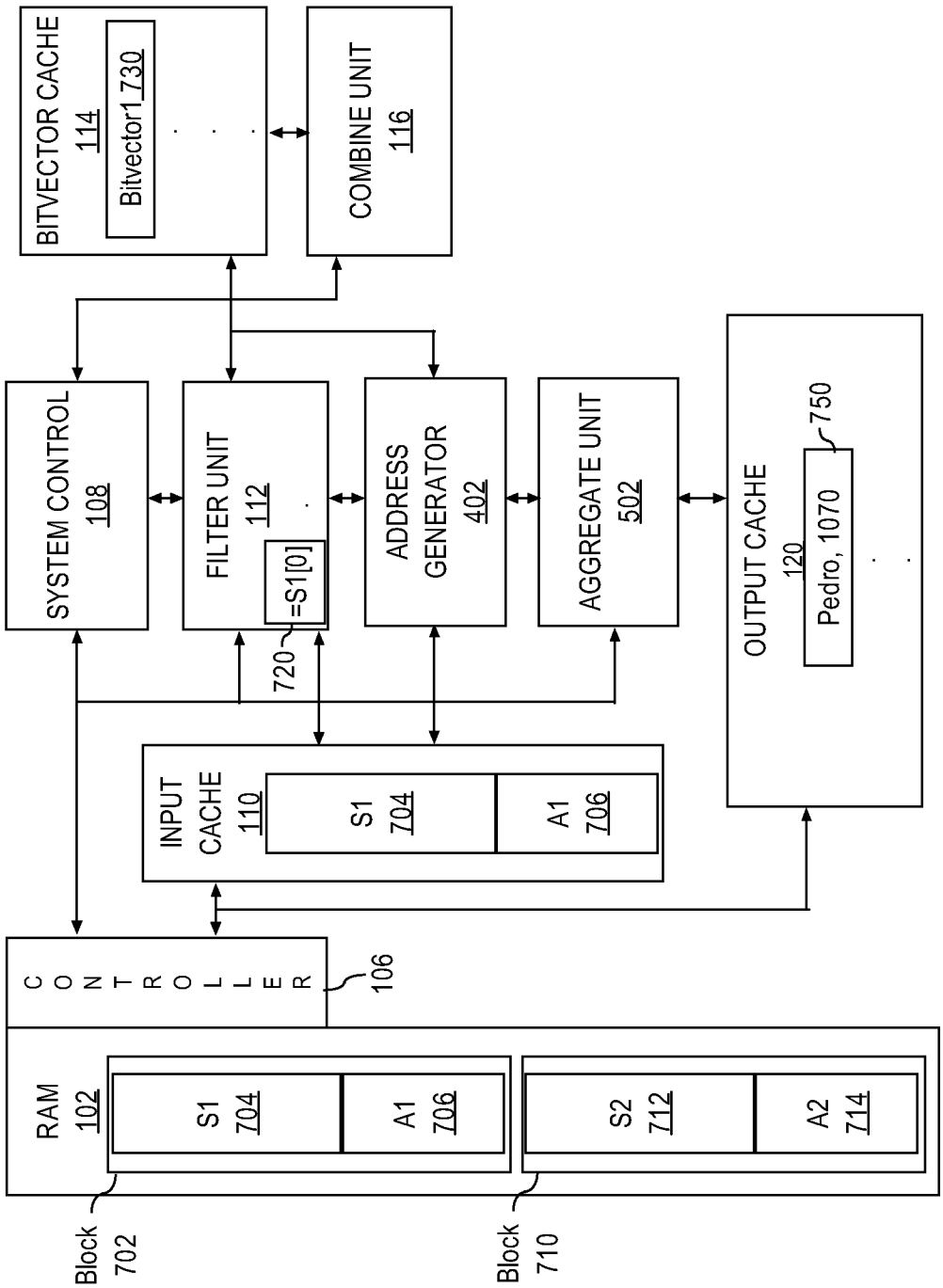

BITVECTOR CACHE
114
Bitvector1 730
COMBINE UNIT
116
SYSTEM CONTROL
108
FILTER UNIT
112
=S1[0]
720
ADDRESS
GENERATOR
402
AGGREGATE UNIT
502
OUTPUT CACHE
120
Pedro, 1070
750
INPUT
CACHE
110
S1
704
A1
706
CONTROLLER
106
RAM
102
S1
704
A1
706
S2
712
A2
714
Block
702
Block
710

FIG. 7A

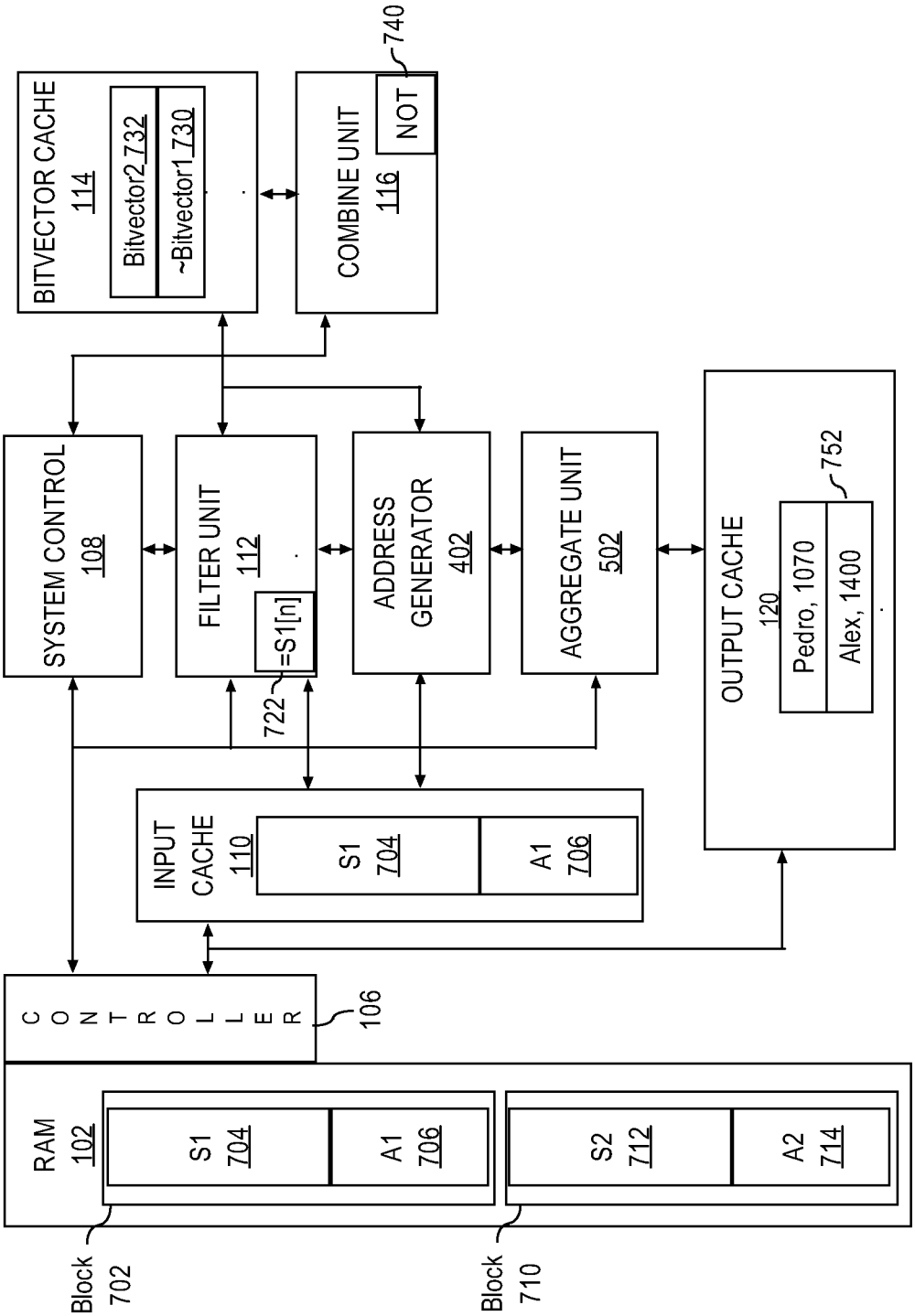

BITVECTOR CACHE
114
Bitvector2 732
~Bitvector1 730
COMBINE UNIT
116
NOT
740
SYSTEM CONTROL
108
FILTER UNIT
112
=S1[n]
722
ADDRESS GENERATOR
402
AGGREGATE UNIT
502
OUTPUT CACHE
120
Pedro, 1070
Alex, 1400
752
INPUT CACHE
110
S1
704
A1
706
CONTROLLER
106
RAM
102
S1
704
A1
706
S2
712
A2
714
Block
702
Block
710

FIG. 7B

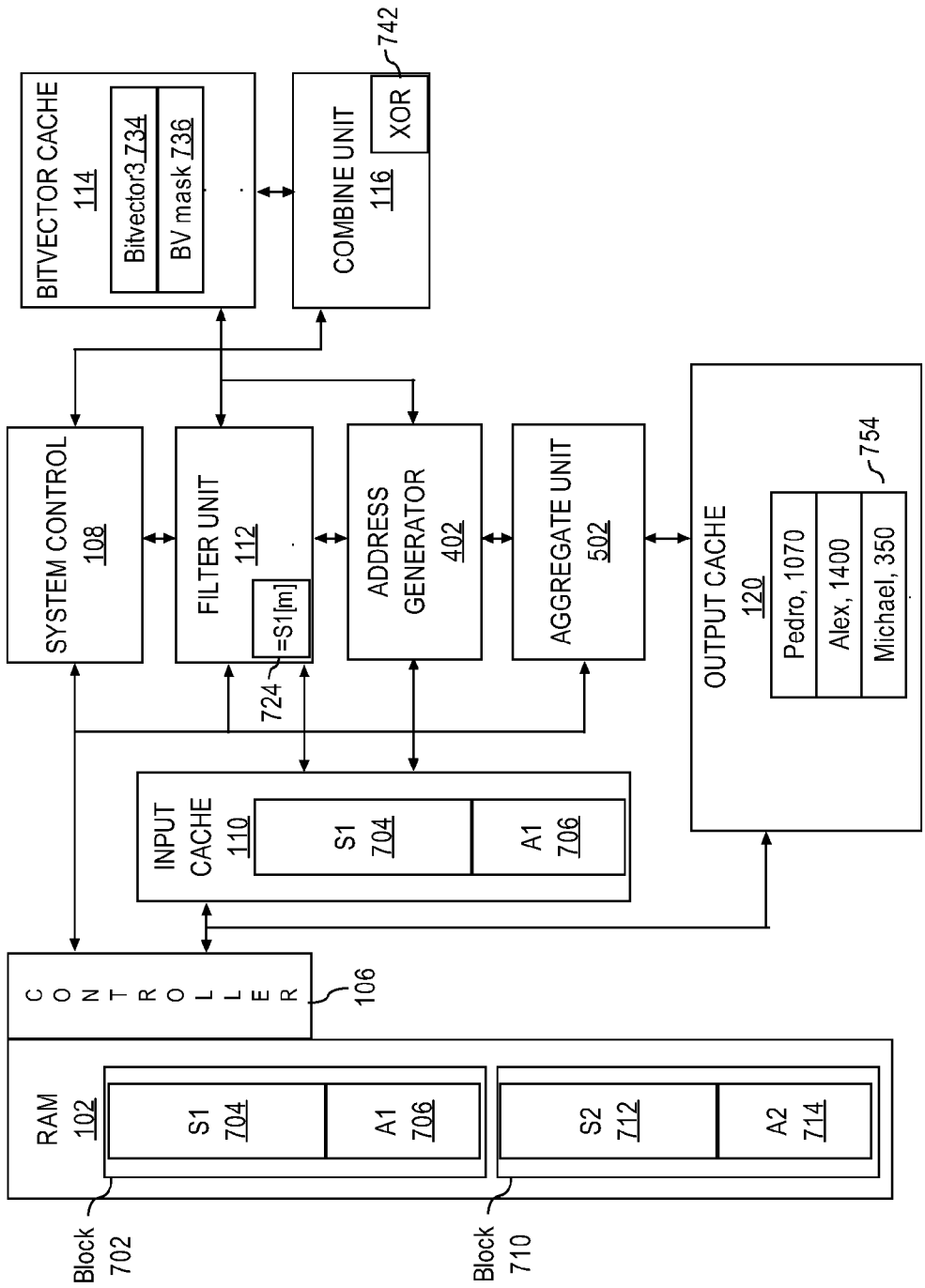

BITVECTOR CACHE 114
Bitvector3 734
BV mask 736
COMBINE UNIT 116
XOR
742
SYSTEM CONTROL 108
FILTER UNIT 112
=S1[m]
724
ADDRESS GENERATOR 402
AGGREGATE UNIT 502
OUTPUT CACHE 120
Pedro, 1070
Alex, 1400
Michael, 350
754
INPUT CACHE 110
S1 704
A1 706
CONTROLLER 106
RAM 102
S1 704
A1 706
S2 712
A2 714
Block 702
Block 710

FIG. 7C

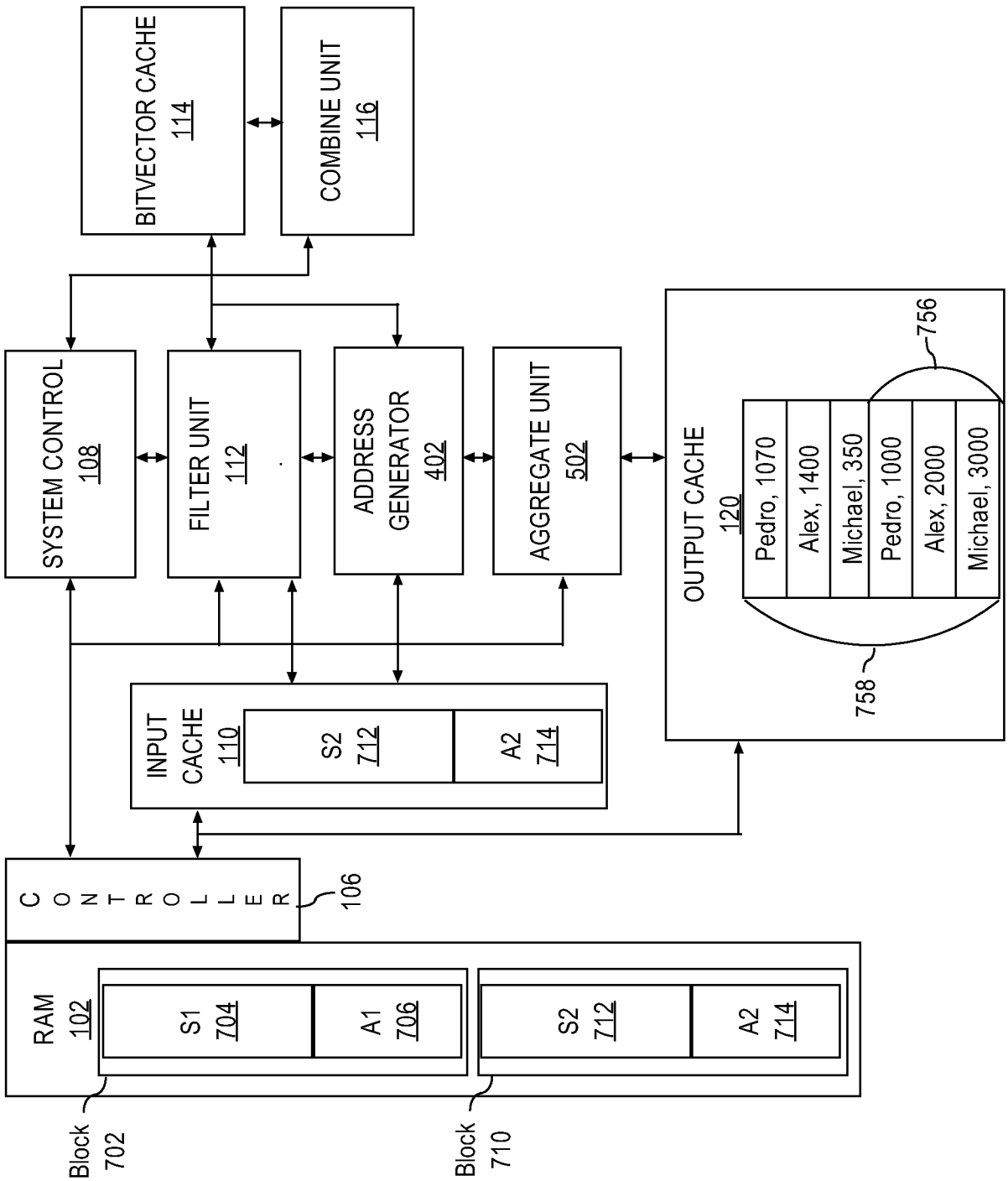

BITVECTOR CACHE 114
COMBINE UNIT 116
SYSTEM CONTROL 108
FILTER UNIT 112
ADDRESS GENERATOR 402
AGGREGATE UNIT 502
OUTPUT CACHE 120
Pedro, 1070
Alex, 1400
Michael, 350
Pedro, 1000
Alex, 2000
Michael, 3000
756
758
INPUT CACHE 110
S2 Z12
A2 Z14
CONTROLLER 106
RAM 102
S1 Z04
A1 Z06
S2 Z12
A2 Z14
Block 702
Block 710

FIG. 7D

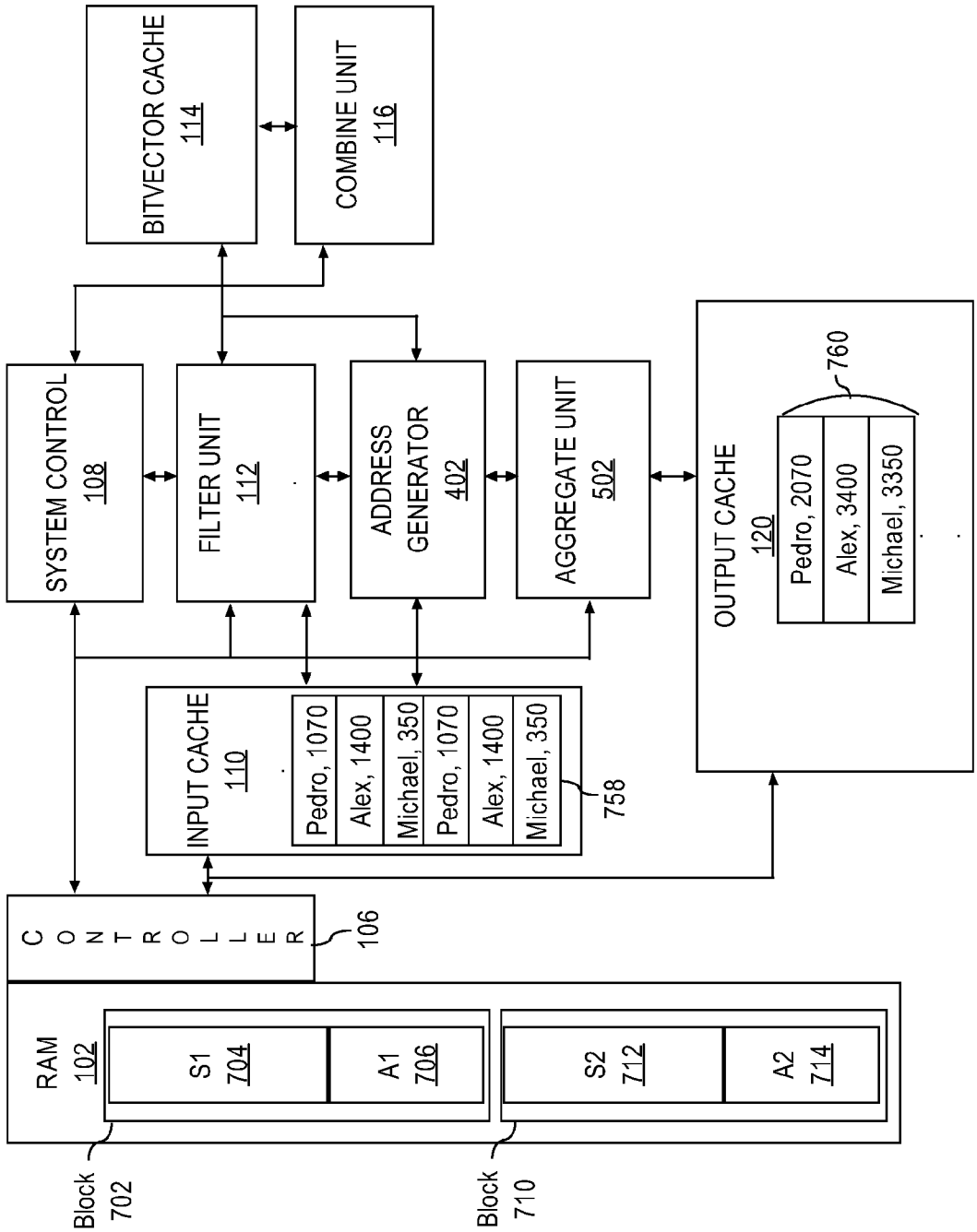

BITVECTOR CACHE 114
COMBINE UNIT 116
SYSTEM CONTROL 108
FILTER UNIT 112
ADDRESS GENERATOR 402
AGGREGATE UNIT 502
OUTPUT CACHE 120
760
Pedro, 2070
Alex, 3400
Michael, 3350
INPUT CACHE 110
Pedro, 1070
Alex, 1400
Michael, 350
Pedro, 1070
Alex, 1400
Michael, 350
758
CONTROLLER 106
RAM 102
S1 704
A1 706
S2 712
A2 714
Block 702
Block 710

FIG. 7E

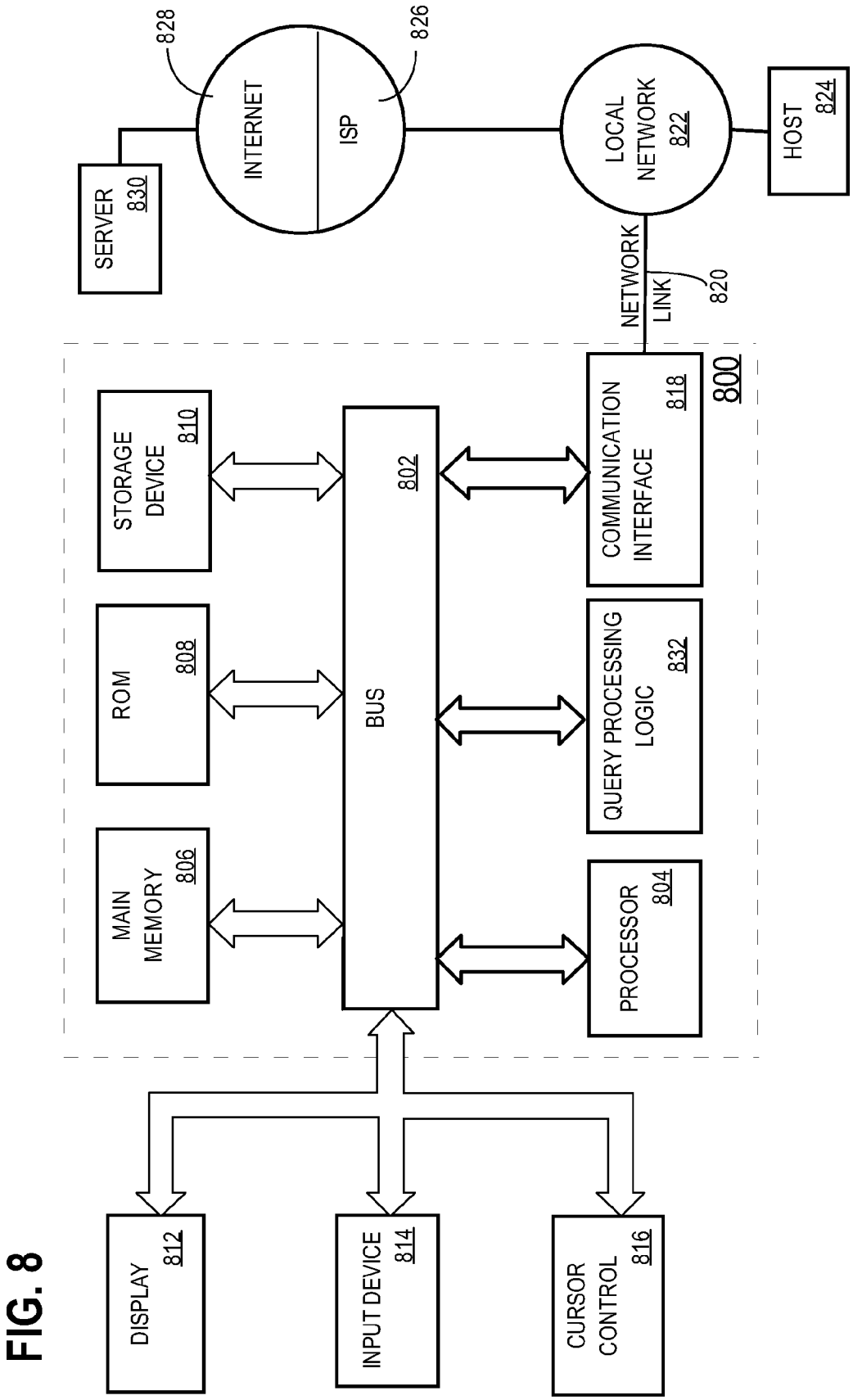

828
826
INTERNET
ISP
SERVER 830
LOCAL NETWORK 822
HOST 824
NETWORK LINK 820
800
STORAGE DEVICE 810
ROM 808
MAIN MEMORY 806
BUS 802
COMMUNICATION INTERFACE 818
QUERY PROCESSING LOGIC 832
PROCESSOR 804
DISPLAY 812
INPUT DEVICE 814
CURSOR CONTROL 816

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/054808

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F17/30
ADD.

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

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2012/054225 A1 (MARWAH VINEET [US] ET AL) 1 March 2012 (2012-03-01) abstract paragraph [0005] - paragraph [0013] paragraph [0022] - paragraph [0028] paragraph [0033] - paragraph [0038] paragraph [0042] paragraph [0047] - paragraph [0068] paragraph [0073] claim 4 ----- -/--	1-20

[X] Further documents are listed in the continuation of Box C.

[X] 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	Date of mailing of the international search report
14 October 2013	18/10/2013

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Boyadzhiev, Yavor

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/054808

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2010/293135 A1 (CANDEA GEORGE [CH] ET AL) 18 November 2010 (2010-11-18) abstract paragraph [0004] paragraph [0009] - paragraph [0012] paragraph [0049] - paragraph [0054] paragraph [0062] - paragraph [0063] paragraph [0070] - paragraph [0071] paragraph [0075] - paragraph [0079] paragraph [0082] - paragraph [0092] paragraph [0104] - paragraph [0106] paragraph [0109] - paragraph [0112] paragraph [0126] -----	1-20
A	US 2009/030874 A1 (DAS DINESH [US] ET AL) 29 January 2009 (2009-01-29) abstract paragraph [0001] - paragraph [0007] paragraph [0013] - paragraph [0020] paragraph [0023] - paragraph [0032] paragraph [0037] - paragraph [0055] paragraph [0059] -----	1-20

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2013/054808

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2012054225 A1	01-03-2012	NONE	
US 2010293135 A1	18-11-2010	NONE	
US 2009030874 A1	29-01-2009	US 2009030874 A1	29-01-2009
		US 2009030883 A1	29-01-2009