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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
31 December 2008 (31.12.2008)

PCT

(10) International Publication Number
WO 2009/002750 A2

(51) International Patent Classification:
G06F 9/46 (2006.01)

(21) International Application Number:
PCT/US2008/067138

(22) International Filing Date: 16 June 2008 (16.06.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/823,167 27 June 2007 (27.06.2007) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **DUFFY, John Joseph**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **CALLAHAN, David**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **ESSEY, Edward George**; One Microsoft Way, Redmond, Washington 98052-6399 (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, BR, BW, BY, BZ, CA, CH, 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, KM, 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, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, 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, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:
— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*
— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:
— *without international search report and to be republished upon receipt of that report*

(54) **Title:** ORDER PRESERVATION IN DATA PARALLEL OPERATIONS

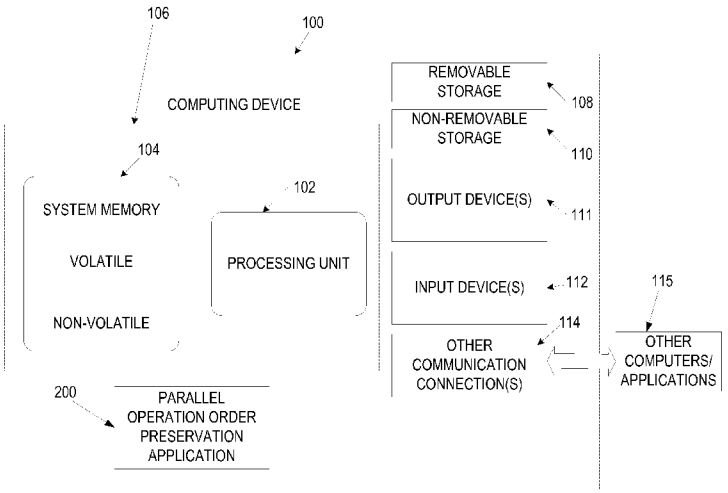

FIG. 1

(57) **Abstract:** Various technologies and techniques are disclosed for preserving input element ordering in data parallel operations. This ordering may be based on element ordinal position in the input or a programmer-specified key-selection routine that generates sortable keys for each input element. Complex data parallel operations are re-written to contain individual data parallel operations that introduce partitioning and merging. Each partition is then processed independently in parallel. The system ensures that down-stream operations remember ordering information established by certain other operations, using techniques that vary depending upon which categories the consumer operations are in. Data is merged back into one output stream using a final merge process that is aware of the ordering established among data elements.

WO 2009/002750 A2

ORDER PRESERVATION IN DATA PARALLEL OPERATIONS

BACKGROUND

[001] Software programs have been written to run sequentially since the beginning days of software development. Steadily over time computers have become much more powerful, with more processing power and memory to handle advanced operations. This trend has recently shifted away from ever-increasing single-processor clock rates and towards an increase in the number of processors available in a single computer, i.e. away from sequential execution and toward parallel execution. Software developers want to take advantage of improvements in computer processing power, enabling their software programs to run faster as new hardware is adopted. With parallel hardware, however, this requires a different approach: developers must arrange for one or more tasks of a particular software program to be executed in parallel (sometimes called "concurrently"), so that the same logical operation can utilize many processors at one time, and deliver better performance as more processors are added to the computers on which such software runs.

[002] When parallelizing previously-written sequential algorithms, it is often desirable to keep as much of the previous sequential program behavior as is possible. However, typical parallel execution of existing sequential logic introduces new behavioral characteristics and presents problems that can introduce challenges into the migration from sequential to parallel algorithms. Moreover, it is also possible that such a problem could represent changes to non-negotiable sequential behavior, prohibiting migration altogether. One category of such problems is that of preserving data ordering, either by ordinal position or keys generated based on programmer specified key-selection logic.

[003] As an illustration, imagine a programmer wrote this program text, which uses a language integrated query comprehension as a way of representing a data parallel computation:

```
int[] A = … generate some interesting input …;
Array.Sort(A); // sort 'A' in place
int[] B = (from x in A select x*x).ToArray();
```

[004] The sequential algorithm preserves relative ordering among elements in 'A' for the output elements in 'B', simply by virtue of the sequential evaluation of the query whose results are assigned to 'B'. If the query comprehension is run in parallel using typical data parallel execution, the relative ordering among elements may become scrambled. As an example, imagine 'A' contains the elements { 0, 1, 2, 3 }; the programmer will likely expect that, after execution, 'B' contains { 0, 1, 4, 9 }. This problem can apply generally to all data parallel operations, not just query comprehensions.

SUMMARY

[005] Various technologies and techniques are disclosed for preserving order in data parallel operations. Complex combinations (e.g. trees) of data parallel operations are re-written to contain data parallel operations that introduce partitioning and merging. Partitioning allows each partition to process a disjoint subset of the input in parallel, and the results are later merged back into one set of output for consumption. The system ensures that operations consuming the output of an order establishing operation, either directly or indirectly, remember necessary ordering information so that the merge operation can preserve the order using techniques that vary depending upon which categories the operations are in.

[006] In one implementation, input element ordering is preserved in data parallel operations by performing various steps. Two kinds of ordering are supported: key- and ordinal-based. Both are modeled by using order keys: in the former, order keys are generated by applying a programmer-specified key-selection function against input elements, while in the latter, order keys are generated by extracting ordinal element position (e.g. with the data source is an array with indices). First, a complex operation, comprised of individual data parallel operations, is analyzed to label each operation in the data source with a respective category. In one implementation, two categories important to the discussion are: physically-reordering, in which an operation may disturb an existing ordering among elements (physically), and logically-reordering, in which an operation subsumes all previous ordering constraints on the elements in favor of a new ordering. Logically-reordering operations are typically also physically-reordering, but are not required

to be. If no constituent operations in the complex operation are in the physically- or logically-reordering category, then the input data elements' order keys are simply remembered, if order matters, so that they may be used during the merge step. If one or more operations are in the physically-reordering category, then order keys must be propagated during the reordering so that they can be recovered during the merge step, and the merge step must perform a sort to reestablish the correct intra-partition ordering. If one or more operations are in the logically-reordering category, then the operation that is closest to the merge is responsible for providing order key information required to reconstruct order for the merge operation. If multiple logically-reordering operations exist, only the one closest to the merge places the aforementioned requirements on physically-reordering operations that consume its output directly or indirectly. The final merge process is then performed using one of various techniques to sort elements by order keys to produce a final ordered set.

[007] This Summary was provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[008] Figure 1 is a diagrammatic view of a computer system of one implementation.

[009] Figure 2 is a diagrammatic view of a parallel operation order preservation application of one implementation operating on the computer system of Figure 1.

[010] Figure 3 is a high-level process flow diagram for one implementation of the system of Figure 1.

[011] Figure 4 is a process flow diagram for one implementation of the system of Figure 1 illustrating the high level stages involved in preserving existing element position in data parallel operations.

[012] Figure 5 is a process flow diagram for one implementation of the system of Figure 1 illustrating the more detailed stages involved in preserving existing element position in data parallel operations.

[013] Figure 6 is a process flow diagram for one implementation of the system of Figure 1 illustrating the stages involved in providing a process that facilitates order preservation.

[014] Figure 7 is a process flow diagram for one implementation of the system of Figure 1 that illustrates the stages involved in providing a final merge process that produces the final ordered set.

[015] Figure 8 is a process flow diagram for one implementation of the system of Figure 1 that illustrates a variation for performing a data parallel sort operation on the input to match the physical ordering with logical ordering.

DETAILED DESCRIPTION

[016] The technologies and techniques herein may be described in the general context as an application that preserves order in data parallel operations, but the technologies and techniques also serve other purposes in addition to these. In one implementation, one or more of the techniques described herein can be implemented as features within a framework program such as MICROSOFT® .NET Framework, or from any other type of program or service that handles data parallel operations in programs.

[017] In one implementation, a system is provided that re-writes parallel operations to contain operations that introduce partitioning and merging. Each partition is processed in parallel, and logical element position is preserved. The data is merged back into one output stream to produce a final ordered set. In one implementation, the system allows ordinal order preservation, regardless of the partitioning strategy chosen, to ensure that the relative ordering of any two elements in the output, e0 and e1, is equivalent to the relative ordering of the corresponding elements in the input (or those elements used to generate e0 and e1, in the case of e.g. a mapping operation).

[018] In one implementation, the system is operable to assist with key-based order preservation. Sometimes an ordering is established among elements that are not

necessarily ordinal-based. A sort operation, for instance, logically and physically reorders the data. One way to preserve proper order when parallelizing such operations is to remember the key information for long enough to delay the merge until the end, where at least one merge operation will be incurred anyway. This is quite similar to ordinal order preservation: in fact, as described below, ordinal order preservation is a special case of key-based order preservation, where the original element indices are modeled as sort keys.

[019] As shown in Figure 1, an exemplary computer system to use for implementing one or more parts of the system includes a computing device, such as computing device 100. In its most basic configuration, computing device 100 typically includes at least one processing unit 102 and memory 104. Depending on the exact configuration and type of computing device, memory 104 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. This most basic configuration is illustrated in Figure 1 by dashed line 106.

[020] Additionally, device 100 may also have additional features/functionality. For example, device 100 may also include additional storage (removable and/or non-removable) including, but not limited to, magnetic or optical disks or tape. Such additional storage is illustrated in Figure 1 by removable storage 108 and non-removable storage 110. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Memory 104, removable storage 108 and non-removable storage 110 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by device 100. Any such computer storage media may be part of device 100.

[021] Computing device 100 includes one or more communication connections 114 that allow computing device 100 to communicate with other computers/applications 115. Device 100 may also have input device(s) 112 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 111 such as a display, speakers, printer, etc. may also be included. These devices are well known in the art and need not be discussed at length here. In one implementation, computing device 100 includes parallel operation order preservation application 200. Parallel operation order preservation application 200 will be described in further detail in Figure 2.

[022] Turning now to Figure 2 with continued reference to Figure 1, a parallel operation order preservation application 200 operating on computing device 100 is illustrated. Parallel operation order preservation application 200 is one of the application programs that reside on computing device 100. However, it will be understood that parallel operation order preservation application 200 can alternatively or additionally be embodied as computer-executable instructions on one or more computers and/or in different variations than shown on Figure 1. Alternatively or additionally, one or more parts of parallel operation order preservation application 200 can be part of system memory 104, on other computers and/or applications 115, or other such variations as would occur to one in the computer software art.

[023] Parallel operation order preservation application 200 includes program logic 204, which is responsible for carrying out some or all of the techniques described herein. Program logic 204 includes logic for re-writing a complex data parallel operation to contain operations that introduce partitioning and merging 206 (as described below with respect to Figure 3); logic for analyzing the complex operation and labeling each constituent operation as physically-reordering, logically-reordering, both, or neither 207 (as described below with respect to Figure 5); logic for processing each partition in parallel 208 (as described below with respect to Figure 3); logic for preserving logical element position in data parallel operations 210 (as described below with respect to Figures 4 and 5); logic for merging data back into one output stream to produce a final ordered set 212 (as

described below with respect to Figures 3 and 5); and other logic for the operating application 220.

[024] Turning now to Figures 3-8 with continued reference to Figures 1-2, the stages for implementing one or more implementations of parallel operation order preservation application 200 are described in further detail. In some implementations, the processes of Figures 3-8 are at least partially implemented in the operating logic of computing device 100. Figure 3 is a high level process flow diagram for parallel operation order preservation application 200. The process begins at start point 240 with re-writing a complex data parallel operation to contain constituent operations that are logically transparent, but which physically introduce partitioning and merging (stage 242). The term "complex operation" as used herein is meant to include the composition of one or more data parallel operations into a more complex operation, such as a tree structure in which nodes are data parallel operations, a singular operation, linked list of operations, graph, etc. In one implementation, a partition operation simply partitions the input space into multiple disjoint partitions (stage 243) using one of a variety of techniques: dynamically, on-demand generating "chunks" of interesting size, partitioning into disjoint size by computing indices for the region's boundaries, striping data in smaller granules, hash-coding based on a certain key-selection function, and so on. Each partition is then processed independently and in parallel (stage 244). The merge operation later takes these partitions and merges the data back into one output stream (stage 246). The process ends at end point 248.

[025] In between the partition and merge steps, operations like projection, filtering, joining, sorting, etc. can be found. Most operations fall into one of three categories:

1. Logical order reordering.
2. Physical order reordering.
3. Both (physical and logical order reordering).
4. Neither (physical and logical order preserving).

[026] A sort is an example of category #1: the elements are logically reordered, in that the ordering established by the sort must be preserved in the final set, which also means that whatever ordering information we had previously been maintaining is now obsolete. A sort may optionally physically rearrange elements, in which case it is in category #3. A hash repartitioning operation is an example of something in category #2: elements might be redistributed to other partitions, based on a key-selection function, causing an unpredictable and nondeterministic interleaving of elements among the partitions. But the repartitioning operation is solely meant to physically distribute elements, and has no impact to the logical ordering established among elements. Most operations fall into category #4: projection, filtering, etc., which neither reorder elements physically nor establish any sort of logical ordering information that must be preserved. These are just a few non-limiting examples, as there are obviously many more examples of operations in each category. Note that an operation in any category may omit input elements from its output in any category, e.g. filters and joins. Some examples of preserving logical element ordering for such operations are described in further detail in Figures 4-8.

[027] Figure 4 illustrates one implementation of the high level stages involved in preserving existing element position in data parallel operations. The process begins at start point 270 with the providing an operation that establishes a logical ordering among elements (stage 272). The establishment of logical ordering might take the form of requiring that element ordinal positions are preserved, or alternatively may take the form of a sort operation with a programmer-specified key-selection routine which, given an element, generates a key used for ordering. The operation requiring order preservation places a restriction on all downstream operations that will process its output (stage 274). The term "downstream operation" as used herein means any operation that directly or indirectly consumes the output of an operation. These downstream operations remember the ordering information using tasks that vary based upon which category the operation is in (stage 276), i.e. physically-reordering, logically-reordering, both, or neither. The process ends at end point 278.

[028] Figure 5 illustrates one implementation of the more detailed stages involved in preserving existing element position in data parallel operations. The process begins at start point 290 with generating a complex data parallel operation out of individual data parallel operations and binding it to a data source (stage 292). The system then analyzes the complex operation to label each constituent operation with its category (stage 294). If there are any operations in the "physical and logical order reordering" category (decision point 296), then the operation in that category closest to the merge is used to mark the individual operation's location within the complex operation, such that all downstream operations must remember the information that is required to reconstruct order during the merge (stage 298). All operations that consume output either directly or indirectly from such an operation of that category perform the process that facilitates order preservation (stage 300). The final merge process uses data from the prior 2 steps to produce the final ordered set (stage 302) and the process ends at end point 310.

[029] If, however, there are not any operations in the "physical and logical order reordering" category (decision point 296), and if the system does care about the order (decision point 304), then the system will remember the necessary element ordering information among the elements (stage 308), e.g. indices for ordinal preservation or keys for sorts, and will then proceed to stage 300 and 302 with preserving order preservation and performing the final merge process as previously described. If the system does not care about order, then nothing special is done (stage 306) and the process ends at end point 310.

[030] Figure 6 illustrates one implementation of the stages involved in providing a process that facilitates order preservation. The process begins at start point 320 with each operation in the "physical order reordering" category being notified that ordering needs preserved (stage 322). An analysis process applies heuristics to determine if there is alternative algorithm that would be more efficient than propagating and remembering for the logical element ordering information (stage 324). If the analysis process reveals there is a better alternative (decision point 326), then the system proceeds with the alternative operation (stage 329). If there is not a better alternative (decision point 326), then the system correlates and

remembers the ordering information, as previously described (stage 328). The process ends at end point 330.

[031] Figure 7 illustrates one implementation of the stages involved in providing a final merge process that produces the final ordered set. The process begins at start point 340 with performing an incremental sort on the elements if all constituent operations in the complex operation have been careful to preserve physical ordering (stage 342). Since each of the partitions are already sorted, there is no need to independently resort each partition. For cases in which the logical-to-physical ordering relationship has not been preserved during execution of the complex operation, the final merge process must first reconstruct the ordering among each partition, as described in Figure 8. It does this by performing independent sort operations on each partition in parallel. The merge process then proceeds to incrementally merges the independently sorted partitions, by using one of many techniques (stage 344), e.g. a heap data structure to remember previous comparison results, a merge sort, etc., to produce the final ordered set (stage 346). The process ends at end point 348.

[032] Figure 8 illustrates one implementation of the stages involved in performing a data parallel sort operation on the input, in which the logical ordering among elements is changed but the physical reordering (i.e. the sort) is deferred to the final merge. If multiple sorts appear in a complex operation, this avoids needing to perform multiple sorts when only the effects of the last will remain in the final merged output anyhow. The process of Figure 8 is also used when all operations have not been careful to preserve physical ordering, as mentioned above. The process begins at start point 370 altering the sort operation so it logically generates keys and establishes order preservation property (stage 372). The system then delegates responsibility for physically sorting the elements to the final merge operation (stage 374). The process ends at end point 376.

[033] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described

above are disclosed as example forms of implementing the claims. All equivalents, changes, and modifications that come within the spirit of the implementations as described herein and/or by the following claims are desired to be protected.

[034] For example, a person of ordinary skill in the computer software art will recognize that the examples discussed herein could be organized differently on one or more computers to include fewer or additional options or features than as portrayed in the examples.

What is claimed is:

1. A computer-readable medium having computer-executable instructions for causing a computer to perform steps comprising:

re-writing a complex data parallel operation to contain data parallel operations that introduce partitioning and merging (206);

processing each resulting partition in parallel (208);

establishing logical order information during the execution of data parallel operations (207); and

merging data back into one output stream using a final merge process that is operable to preserve a desired logical ordering (212).

2. The computer-readable medium of claim 1, wherein the logical order information preserved is derived from a position of an element in an input to the data parallel operations (272).

3. The computer-readable medium of claim 1, wherein the final merge process is operable to produce a final ordered set from the logical order information (302).

4. The computer-readable medium of claim 1, wherein the final merge process is operable to perform an incremental sort on elements if all operations have been careful to preserve physical ordering (302).

5. The computer-readable medium of claim 4, wherein the one output stream contains ordered results of the final merge process (302).

6. The computer-readable medium of claim 1, wherein the merge operation is responsible for sorting and merging if all operations are not careful to preserve physical ordering (246).

7. The computer-readable medium of claim 6, wherein sorts with established ordering are altered to generate keys that establish a logical order preservation property (372).

8. The computer-readable medium of claim 1, wherein the order information preserved is derived from a key-selection function which extracts a sort key from each element in an input to the data parallel operations (308).

9. A method for preserving the relative input element positions in the output of the execution of a complex data parallel operation comprising the steps of:

accessing an operation that establishes a logical ordering among elements and requires order preservation (272); and

ensuring that downstream operations remember ordering information using techniques that vary depending upon which of a plurality of categories the operations are in (276).

10. The method of claim 9, wherein a complex operation is analyzed to label each constituent operation with a respective category, the respective category being selected from the group consisting of a physical order reordering category, a logical order reordering category, a physical and logical order reordering category, and a none of the above category (207) .

11. The method of claim 9, wherein if any operations are in a logical order reordering category, or both, then the operation in the logical order reordering category that is closest to the merge operation remembers information required to subsequently reconstruct order (298).

12. The method of claim 11, wherein all operations in a physical reordering category that consume output from the closest operation to the merge remember information required to reconstruct order during the final merge (300).

13. The method of claim 12, wherein the order preservation process comprises:

notifying each operation in a physical order reordering category that ordering needs preserved (322); and

if there is an alternative algorithm that is determined to be more efficient than remembering changes for ordering (326), which is not in the physical order reordering category, then proceeding with the alternative operation (329).

14. The method of claim 13, further comprising:

if there is not the alternative algorithm that is determined to be more efficient than remembering changes for ordering, then proceeding with correlating and remembering changes (328).

15. The method of claim 13, wherein heuristics are used for the determination about the alternative algorithm (328).

16. A computer-readable medium having computer-executable instructions for causing a computer to perform the steps recited in claim 9 (200).

17. A method for preserving existing ordinal element position in data parallel operations comprising the steps of:

retrieving a data source (292);

analyzing a complex data parallel operation to label each operation in the data source with a respective category (294);

if any operations are in a logical order reordering category (296), then remembering information required to reconstruct order in an operation in the logical order reordering category that is closest to a merge (298);

performing an order preservation process that facilitates order preservation in all operations that consume output from the logical order reordering operation closest to the merge (300); and

performing a final merge process to produce a final ordered set (302).

18. The method of claim 17, wherein if no operations are in the physical and logical order reordering category, and if ordering is not important, then nothing special is done (306).

19. The method of claim 17, wherein if no operations are in the physical and logical order reordering category, and if ordering is important, then input data relative ordering among elements is preserved during the final merge process (308).

20. A computer-readable medium having computer-executable instructions for causing a computer to perform the steps recited in claim 17 (200).

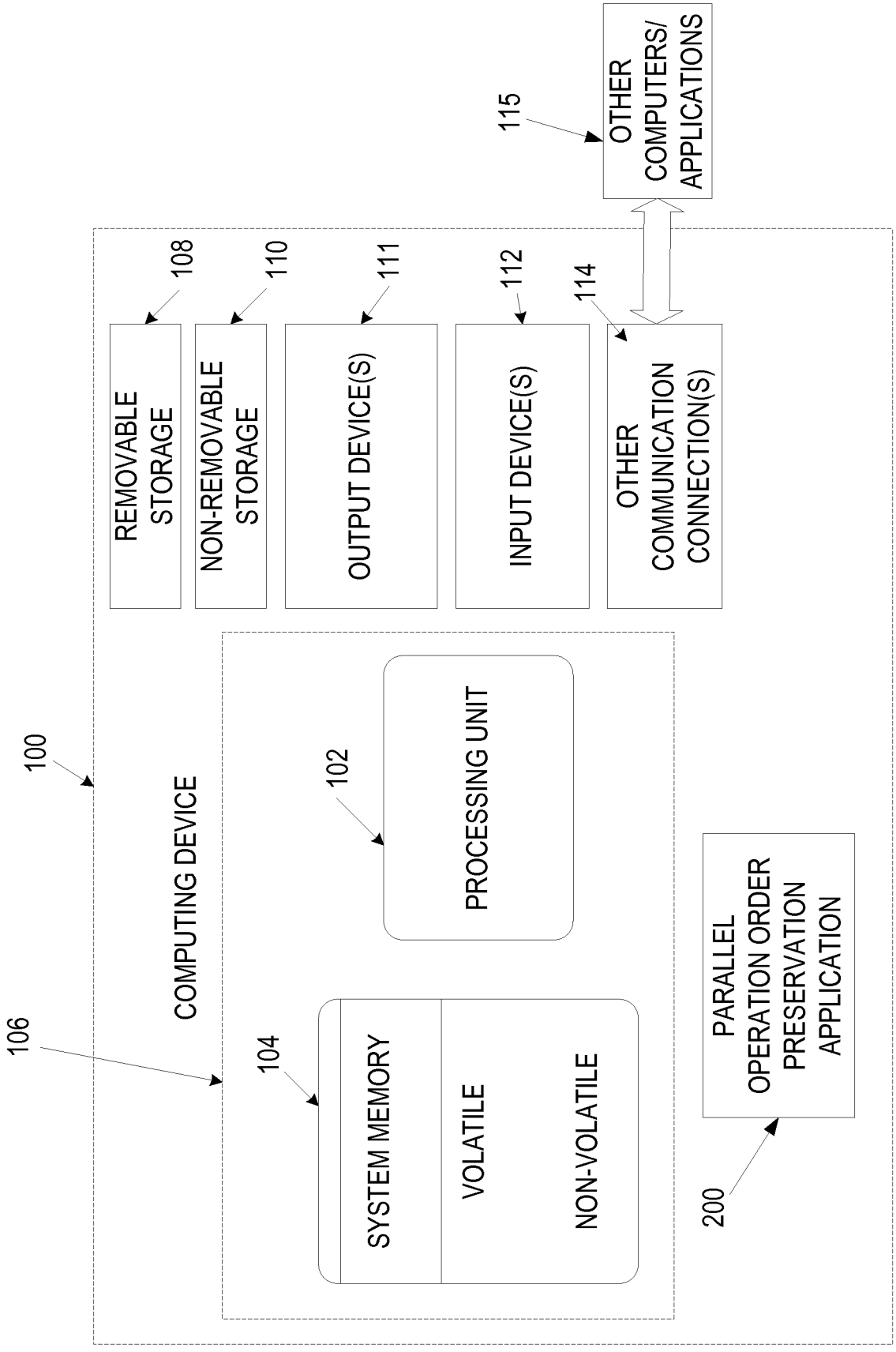
100
COMPUTING DEVICE
106
102
PROCESSING UNIT
104
SYSTEM MEMORY
VOLATILE
NON-VOLATILE
200
PARALLEL OPERATION ORDER PRESERVATION APPLICATION
108
REMOVABLE STORAGE
110
NON-REMOVABLE STORAGE
111
OUTPUT DEVICE(S)
112
INPUT DEVICE(S)
114
OTHER COMMUNICATION CONNECTION(S)
115
OTHER COMPUTERS/ APPLICATIONS

FIG. 1

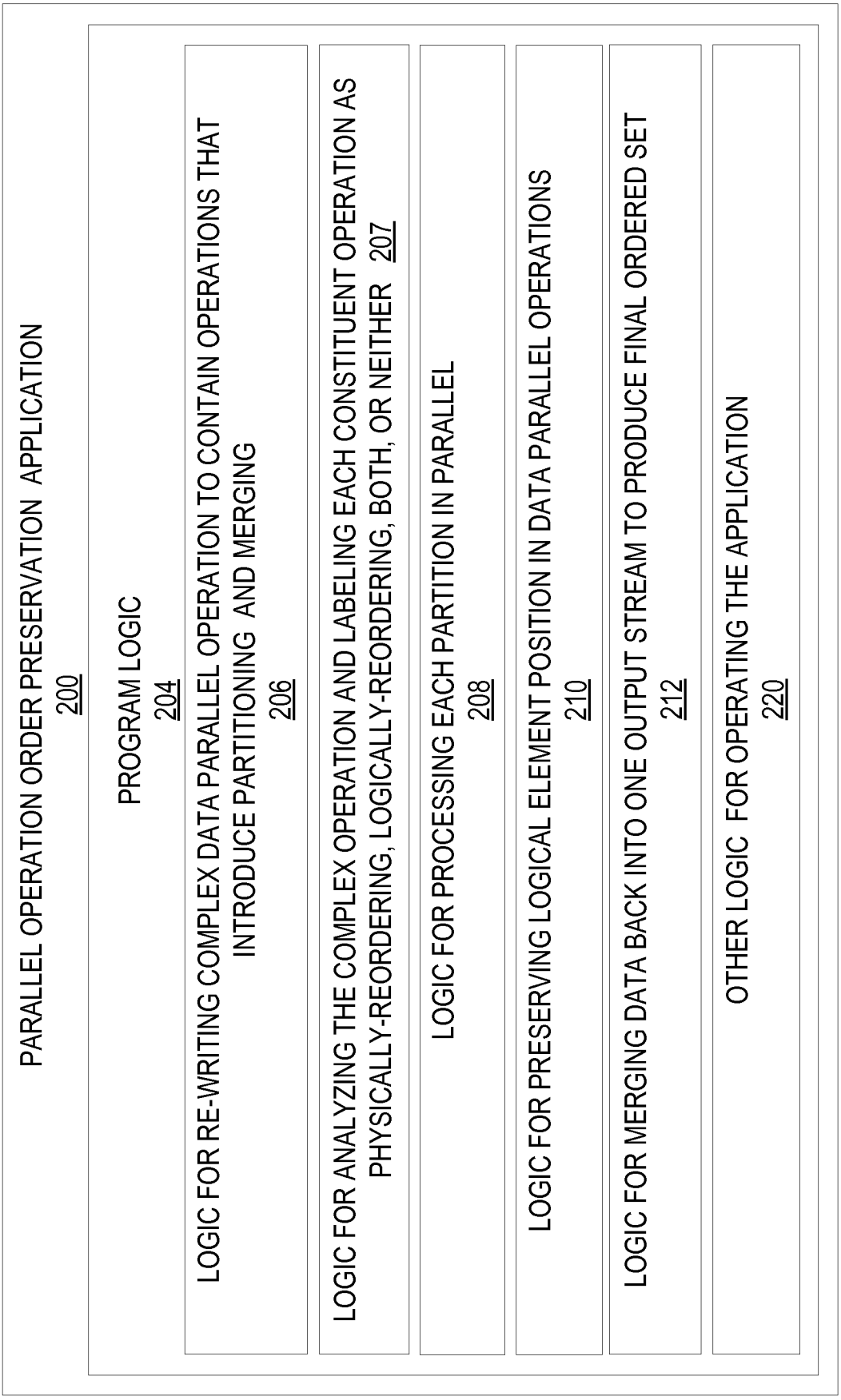
PARALLEL OPERATION ORDER PRESERVATION APPLICATION
200
PROGRAM LOGIC
204
LOGIC FOR RE-WRITING COMPLEX DATA PARALLEL OPERATION TO CONTAIN OPERATIONS THAT INTRODUCE PARTITIONING AND MERGING
206
LOGIC FOR ANALYZING THE COMPLEX OPERATION AND LABELING EACH CONSTITUENT OPERATION AS PHYSICALLY-REORDERING, LOGICALLY-REORDERING, BOTH, OR NEITHER 207
LOGIC FOR PROCESSING EACH PARTITION IN PARALLEL
208
LOGIC FOR PRESERVING LOGICAL ELEMENT POSITION IN DATA PARALLEL OPERATIONS
210
LOGIC FOR MERGING DATA BACK INTO ONE OUTPUT STREAM TO PRODUCE FINAL ORDERED SET
212
OTHER LOGIC FOR OPERATING THE APPLICATION
220

FIG. 2

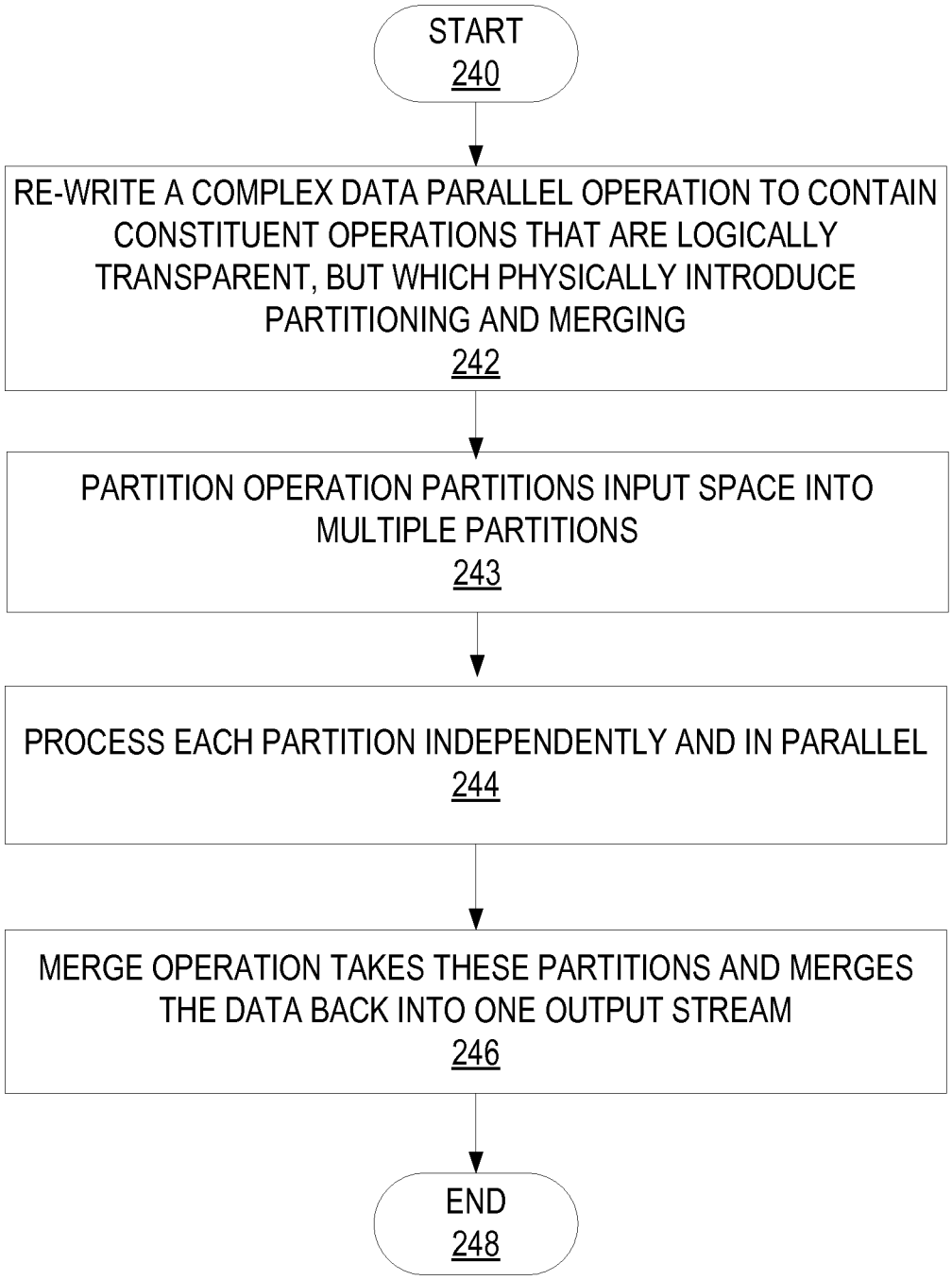
START
240
RE-WRITE A COMPLEX DATA PARALLEL OPERATION TO CONTAIN CONSTITUENT OPERATIONS THAT ARE LOGICALLY TRANSPARENT, BUT WHICH PHYSICALLY INTRODUCE PARTITIONING AND MERGING
242
PARTITION OPERATION PARTITIONS INPUT SPACE INTO MULTIPLE PARTITIONS
243
PROCESS EACH PARTITION INDEPENDENTLY AND IN PARALLEL
244
MERGE OPERATION TAKES THESE PARTITIONS AND MERGES THE DATA BACK INTO ONE OUTPUT STREAM
246
END
248

FIG. 3

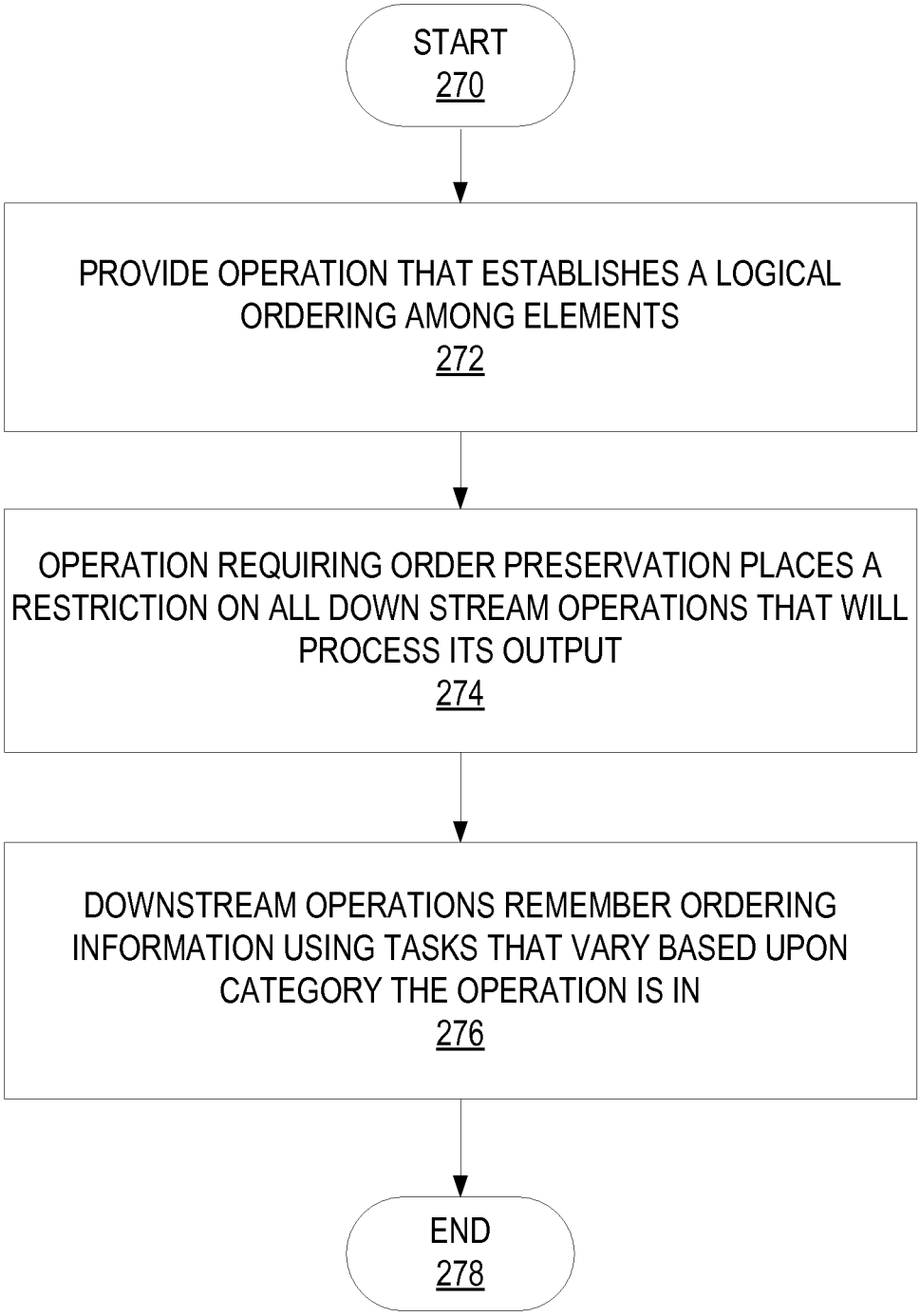
START
270
PROVIDE OPERATION THAT ESTABLISHES A LOGICAL ORDERING AMONG ELEMENTS
272
OPERATION REQUIRING ORDER PRESERVATION PLACES A RESTRICTION ON ALL DOWN STREAM OPERATIONS THAT WILL PROCESS ITS OUTPUT
274
DOWNSTREAM OPERATIONS REMEMBER ORDERING INFORMATION USING TASKS THAT VARY BASED UPON CATEGORY THE OPERATION IS IN
276
END
278

FIG. 4

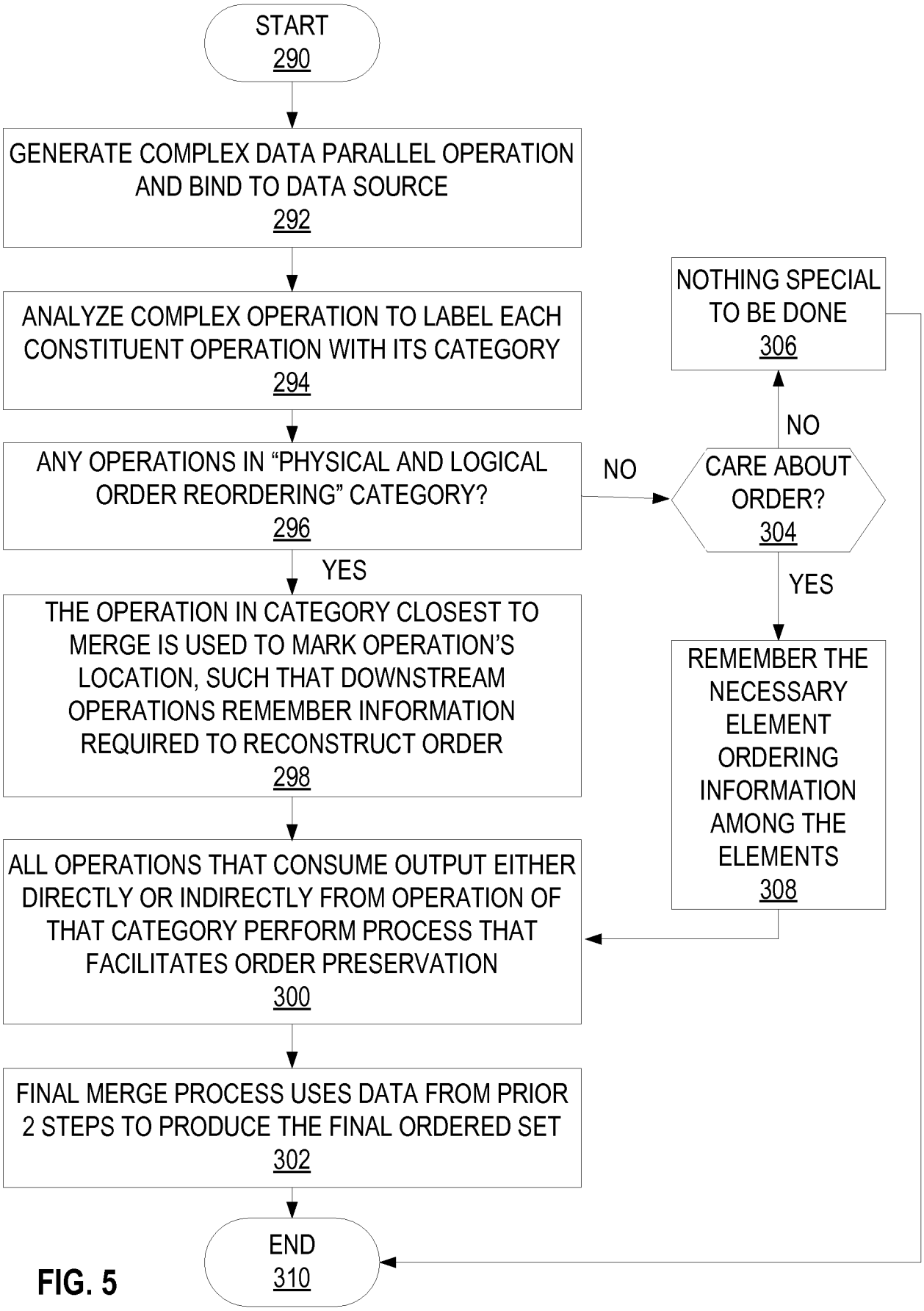
START
290
GENERATE COMPLEX DATA PARALLEL OPERATION AND BIND TO DATA SOURCE
292
ANALYZE COMPLEX OPERATION TO LABEL EACH CONSTITUENT OPERATION WITH ITS CATEGORY
294
ANY OPERATIONS IN "PHYSICAL AND LOGICAL ORDER REORDERING" CATEGORY?
296
NO
CARE ABOUT ORDER?
304
NO
NOTHING SPECIAL TO BE DONE
306
YES
YES
THE OPERATION IN CATEGORY CLOSEST TO MERGE IS USED TO MARK OPERATION'S LOCATION, SUCH THAT DOWNSTREAM OPERATIONS REMEMBER INFORMATION REQUIRED TO RECONSTRUCT ORDER
298
REMEMBER THE NECESSARY ELEMENT ORDERING INFORMATION AMONG THE ELEMENTS
308
ALL OPERATIONS THAT CONSUME OUTPUT EITHER DIRECTLY OR INDIRECTLY FROM OPERATION OF THAT CATEGORY PERFORM PROCESS THAT FACILITATES ORDER PRESERVATION
300
FINAL MERGE PROCESS USES DATA FROM PRIOR 2 STEPS TO PRODUCE THE FINAL ORDERED SET
302
END
310

FIG. 5

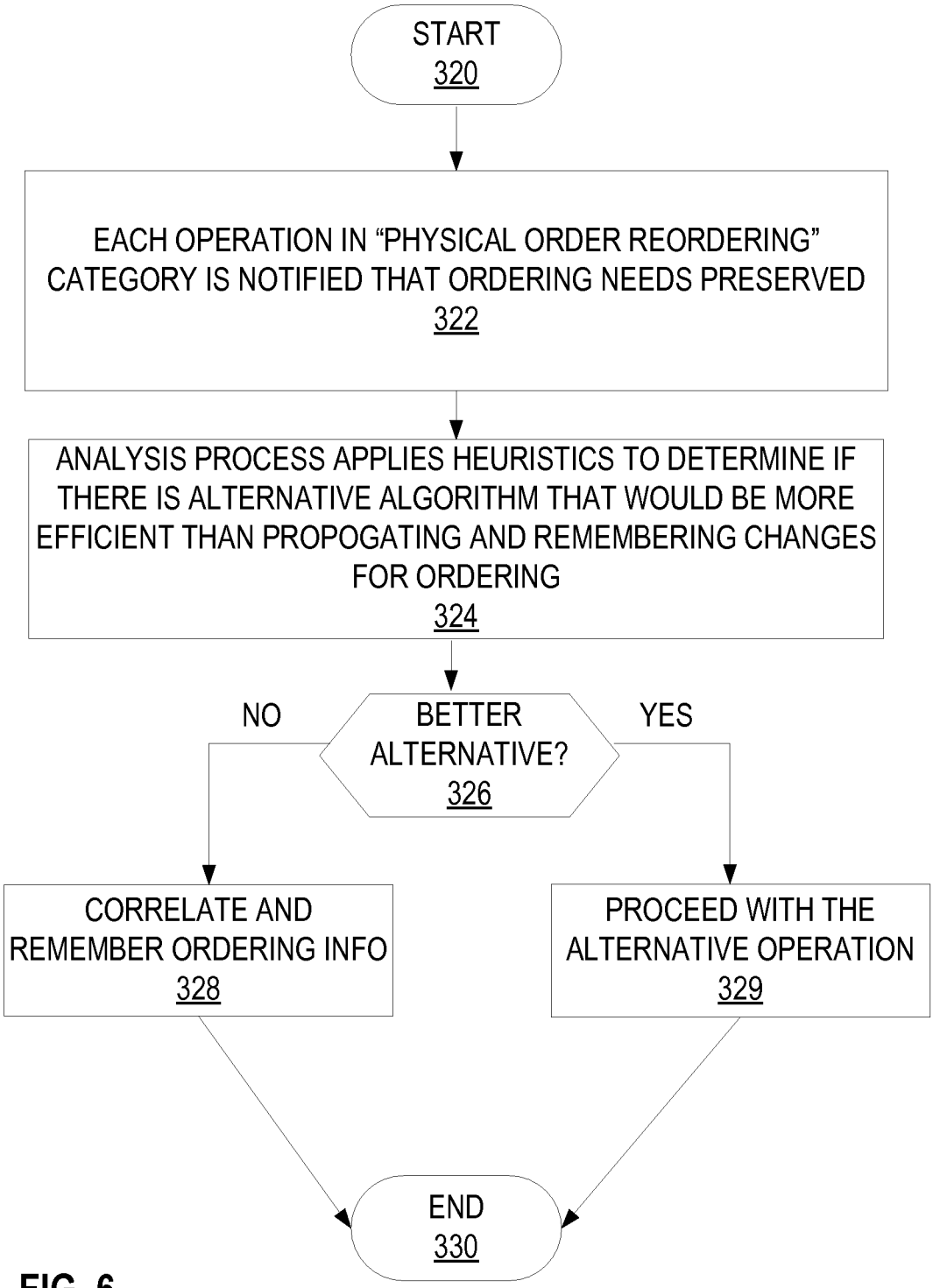
START
320
EACH OPERATION IN "PHYSICAL ORDER REORDERING" CATEGORY IS NOTIFIED THAT ORDERING NEEDS PRESERVED
322
ANALYSIS PROCESS APPLIES HEURISTICS TO DETERMINE IF THERE IS ALTERNATIVE ALGORITHM THAT WOULD BE MORE EFFICIENT THAN PROPOGATING AND REMEMBERING CHANGES FOR ORDERING
324
BETTER ALTERNATIVE?
326
NO
YES
CORRELATE AND REMEMBER ORDERING INFO
328
PROCEED WITH THE ALTERNATIVE OPERATION
329
END
330

FIG. 6

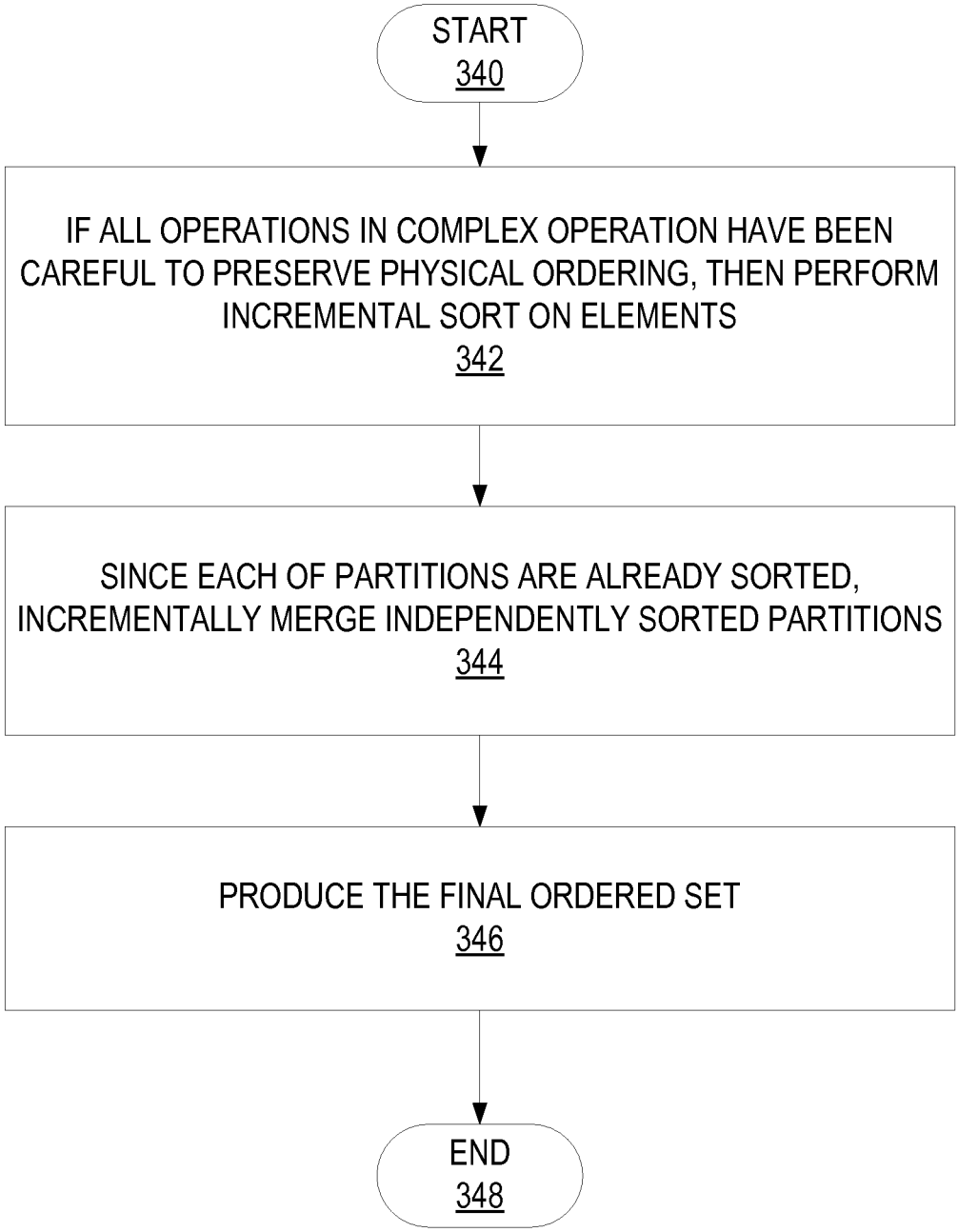
START
340
IF ALL OPERATIONS IN COMPLEX OPERATION HAVE BEEN CAREFUL TO PRESERVE PHYSICAL ORDERING, THEN PERFORM INCREMENTAL SORT ON ELEMENTS
342
SINCE EACH OF PARTITIONS ARE ALREADY SORTED, INCREMENTALLY MERGE INDEPENDENTLY SORTED PARTITIONS
344
PRODUCE THE FINAL ORDERED SET
346
END
348

FIG. 7

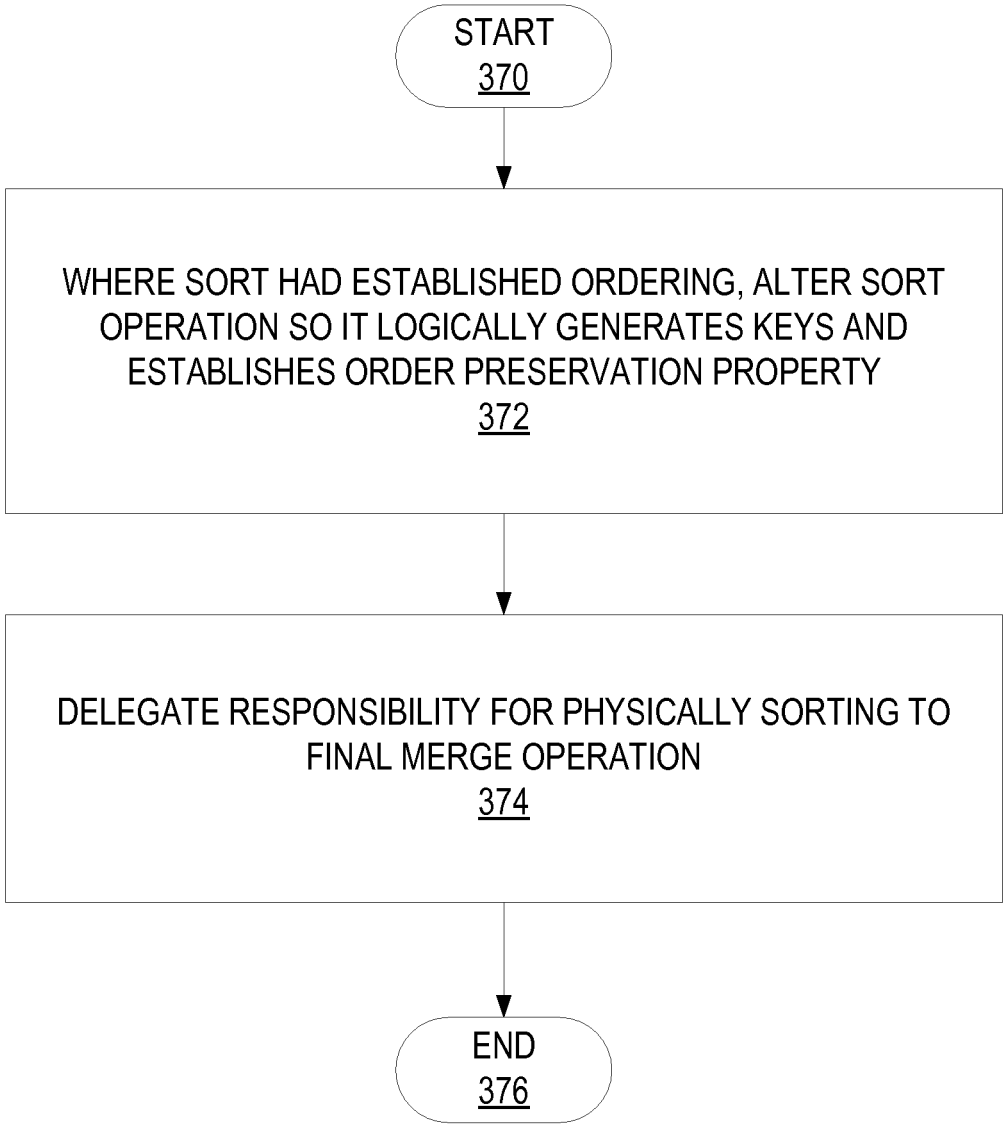
START
370
WHERE SORT HAD ESTABLISHED ORDERING, ALTER SORT OPERATION SO IT LOGICALLY GENERATES KEYS AND ESTABLISHES ORDER PRESERVATION PROPERTY
372
DELEGATE RESPONSIBILITY FOR PHYSICALLY SORTING TO FINAL MERGE OPERATION
374
END
376

FIG. 8