



- (51) International Patent Classification:
G06F 9/38 (2006.01) *G06F 9/345* (2006.01)
- (21) International Application Number:
PCT/US2017/042341
- (22) International Filing Date:
17 July 2017 (17.07.2017)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
15/212,863 18 July 2016 (18.07.2016) US
- (71) Applicant: **ADVANCED MICRO DEVICES, INC.**
[US/US]; One AMD Place, Sunnyvale, California 94088
(US).
- (72) Inventor: **JONES, William Evan**; One AMD Place, Sun-
nyvale, California 94088 (US).
- (74) Agent: **DAVIDSON, Ryan S.**; Davidson Sheehan LLP,
8834 North Capital of Texas Hwy., Suite 100, Austin, Texas
78759 (US).
- (81) Designated States (*unless otherwise indicated, for every
kind of national protection available*): AE, AG, AL, AM,
AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ,
CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO,
DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN,

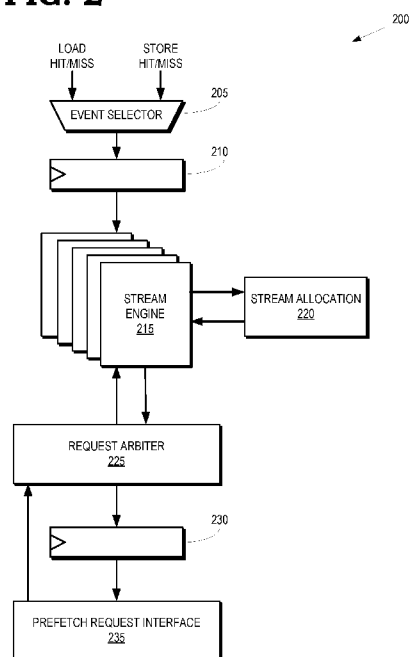
HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP,
KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME,
MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ,
OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA,
SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN,
TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Published:
— with international search report (Art. 21(3))

(54) Title: STRIDE PREFETCHER FOR INCONSISTENT STRIDES

FIG. 2



(57) Abstract: A processing system includes a cache and a prefetcher to prefetch lines from a memory into the cache. The prefetcher receives a memory access request to a first address in the memory and sets a stride length associated with an instruction that issued the memory access request to a length of a line in the cache. The stride length indicates a number of bytes between addresses of lines that are prefetched from the memory into the cache.

STRIDE PREFETCHER FOR INCONSISTENT STRIDES

BACKGROUND

Description of the Related Art

Processing systems typically use speculation to improve performance by prefetching data from a memory into a cache in the expectation that a processor in the processing system will subsequently request the prefetched data from the cache. Stride prefetchers identify patterns in memory addresses that are accessed by instructions being executed by a processor. For example, the stride prefetcher stores the memory address accessed by an instruction and determines a stride length that is equal to the difference between the current memory address and a memory address that was previously accessed by the instruction. The stride prefetcher counts the number of consecutive memory accesses that have the same stride length. The number is typically referred to as the stride confidence. If the stride confidence for a stride length is above a threshold, the stride prefetcher identifies a memory access pattern that indicates that the instruction has accessed a sequence of memory accesses that differ by the stride length. The stride prefetcher then predicts that the instruction will subsequently request information from memory addresses that follow the memory access pattern, *i.e.*, memory addresses that are separated by the stride length. Information is prefetched from the predicted memory addresses into the cache so that the processor executing the instruction can retrieve the information from the cache.

BRIEF DESCRIPTION OF THE DRAWINGS

The present disclosure is better understood, and its numerous features and advantages made apparent to those skilled in the art by referencing the accompanying drawings. The use of the same reference symbols in different drawings indicates similar or identical items.

FIG. 1 is a block diagram of a processing device, according to some embodiments.

FIG. 2 is a block diagram of a prefetcher according to some embodiments.

FIG. 3 is a flow diagram of a method for setting stride lengths based on a length of a line in a cache and prefetching data from the cache based on the stride length according to some embodiments.

DETAILED DESCRIPTION

5 Prefetched information is retrieved from a memory added to a cache in quantized blocks that have a predetermined size, which can be determined based on a bandwidth or structure of an interface between the memory and the cache or the size of a line in the cache. For example, if lines in the cache store 64 bytes, every prefetch request causes 64 bytes of data to be retrieved from the memory and stored
10 in a line of the cache. A sequence of instructions that each request eight bytes of data will produce a memory access pattern with a stride length of eight. However, each prefetch based on the memory access pattern will retrieve 64 bytes of data. Consequently, the stride prefetcher performs as many as seven redundant prefetches to retrieve information that was already retrieved from the memory and stored in the
15 line of the cache in response to the first prefetch request for the first eight bytes of data. In addition to accessing memory in blocks that are smaller than a cache line, some instructions access memory with a stride length that is not constant, *e.g.*, with a sequence of strides equal to 8 bytes, 16 bytes, 16 bytes, 8 bytes, *etc.* A conventional stride prefetcher that receives this sequence would not detect any stride length with a
20 significant stride confidence. Furthermore, in some cases a memory access request is dropped or skipped, which can cause the stride length to change even though the underlying memory access pattern associated with the instruction remains the same. For example, if the stride length for a sequence of instructions is 128 bytes, dropping one memory access request will cause the stride prefetcher to calculate a stride
25 length of 256 bytes, which can cause the stride prefetcher to reduce the stride confidence to zero so that information is not prefetched until a new sequence of memory access requests separated by 128 bytes has been received. Missing one or more prefetch opportunities because of a variable stride length, a skipped memory access, or dropped memory access can reduce the performance of the processing
30 system.

The performance of a stride prefetcher can be improved by setting a stride length associated with an instruction that issues a memory access request equal to a length

of a line in a cache. The stride length is set to the length of the cache line based on a difference between a first address indicated by the memory access request and a second address in a previous memory access request by the instruction. For example, the stride prefetcher sets the stride length associated with the memory access request to the length of the line in the cache if the difference between the first address and the second address is less than or equal to the length of the line in the cache. The stride can be positive or negative. In cases where the stride is negative, the stride length is set to a negative value of the length. For example, the stride length can be set to a negative value of the cache line length if the difference between the first address and the second address is negative and the absolute value of the difference is less than or equal to the length. In some variations, the stride length is set to twice the length of the line in the cache if the difference is greater than the length and less than or equal to twice the length.

The stride prefetcher is able to detect a memory access pattern for the instruction on the basis of the stride lengths that have been set to the cache line length and, in some cases, the stride prefetcher prefetches information from a memory to the cache based on the memory access pattern. For example, the stride prefetcher can increment a stride confidence if the first address is in a different line than the second address and the stride length associated with the first address is the same as the stride length associated with the second address. The stride confidence is not incremented if the first address is in the same line as the second address. The stride prefetcher then prefetches information from the memory to the cache when the stride confidence exceeds a threshold.

FIG. 1 is a block diagram of a processing device 100, according to some embodiments. The processing device 100 includes a processing unit 105 (or “processor”) that is configured to access instructions or data that are stored in a main memory 110 via an interface 115. In some variations, the processing unit 105 is used to implement a central processing unit (CPU), a graphics processing unit (GPU) an accelerated processing unit (APU), an application-specific integrated circuit (ASIC), a field programmable gate array (FPGA), or other type of processing unit. The processing unit 105 shown in Figure 1 includes four processor cores 111, 112, 113, 114 (collectively referred to herein as “the processor cores 111-114”) that are used to

execute the instructions or manipulate the data. The processor cores 111-114 can also be referred to as compute units.

The processing unit 105 shown in Figure 1 also implements a hierarchical (or multilevel) cache complex that is used to speed access to the instructions or data by storing selected instructions or data in the caches. However, some embodiments of the device 100 implement different configurations of the processing unit 105, such as configurations that use external caches, different numbers of processor cores 111-114, and the like. Moreover, some embodiments associate different numbers or types of caches with the processor cores 111-114.

The cache complex depicted in FIG. 1 includes a level 2 (L2) cache 120 for storing copies of instructions or data that are stored in the main memory 110. Some embodiments of the L2 cache 120 are implemented using an associativity such as 2-way associativity, 8-way associativity, 16-way associativity, direct mapping, fully associative caches, and the like. Relative to the main memory 110, the L2 cache 120 is implemented using faster memory elements. The L2 cache 120 can also be deployed logically or physically closer to the processor cores 111-114 (relative to the main memory 110) so that information can be exchanged between the L2 cache 120 and the processor cores 111-114 more rapidly or with less latency.

The illustrated cache complex also includes L1 caches 121, 122, 123, 124 (collectively referred to as "the L1 caches 121-124") for storing copies of instructions or data that are stored in the main memory 110 or the L2 cache 120. Each of the L1 caches 121-124 is associated with a corresponding processor core 111-114. The L1 caches 121-124 can be implemented in the corresponding processor cores 111-114 or the L1 caches 121-124 can be implemented outside the corresponding processor core 112. Relative to the L2 cache 120, the L1 caches 121-124 are implemented using faster memory elements so that information stored in the lines of the L1 caches 121-124 can be retrieved quickly by the corresponding processor cores 111-114. The L1 caches 121-124 can also be deployed logically or physically closer to the processor cores 111-114 (relative to the main memory 110 or the L2 cache 120) so that information can be exchanged between the processor cores 111-114 and the L1 caches 121-124 more rapidly or with less latency (relative to communication with the main memory 110 or the L2 cache 120).

Some embodiments of the L1 caches 121-124 are separated into caches for storing instructions and data, which are referred to as the L1-I caches 125, 126, 127, 128 (collectively referred to herein as “the L1-I caches 125-128”) and the L1-D caches 131, 132, 133, 134 (collectively referred to herein as “the L1-D caches 131-134”).

5 Separating or partitioning the L1 caches 121-124 into the L1-I caches 125-128 for storing instructions and the L1-D caches 131-134 for storing data allows these caches to be deployed closer to the entities that are likely to request instructions or data, respectively. Consequently, this arrangement can reduce contention, wire delays, and generally decrease latency associated with instructions and data. A
10 replacement policy dictates that the lines in the L1-I caches 125-128 are replaced with instructions from the L2 cache 120 and the lines in the L1-D caches 131-134 are replaced with data from the L2 cache 120. However, some embodiments of the L1 caches 121-124 are partitioned into different numbers or types of caches that operate according to different replacement policies. Furthermore, some programming or
15 configuration techniques allows the L1-I caches 125-128 to store data or the L1-D caches 131-134 to store instructions, at least on a temporary basis.

The L2 cache 120 illustrated in Figure 1 is inclusive so that cache lines resident in the L1 caches 121-124, 125-128, 131-134 are also resident in the L2 cache 120. The L1 caches 121-124 and the L2 cache 120 represent one example of a multi-level
20 hierarchical cache memory system. However, some embodiments of the processing device 100 use different multilevel caches including elements such as L0 caches, L1 caches, L2 caches, L3 caches, and the like, some of which may or may not be inclusive of the others.

Each of the caches 120, 121-124, 125-128, 131-134 includes a plurality of lines for
25 storing copies of the information from the memory 110. The lines have a predetermined length. For example, the length of a cache line can be set to a value of 64 bytes, although the length of the cache line is a matter of design choice. Furthermore, in some variations, the length of the cache lines can differ in the different caches 120, 121-124, 125-128, 131-134. Information is retrieved from the
30 main memory (e.g., in response to a cache miss or a prefetch request) in blocks that have a length that is equal to the length of the cache line. The size of the retrieved block is therefore independent of the amount of information requested in the memory

access request. For example, as discussed herein, a memory access request for eight bytes of data beginning at an address in the memory 110 results in 64 bytes of data (beginning at the address in the memory access request) being retrieved from the memory 110 and stored in a cache line of one or more of the caches 120, 121-124, 125-128, 131-134.

The processing unit 105 includes one or more prefetchers 135 for prefetching instructions or data from the memory 110 into one or more of the caches 120, 121-124, 125-128, 131-134 before one of the processor cores 111-114 has generated a memory access request for the instructions or data. The prefetchers 135 are able to detect patterns in addresses in memory access requests issued by instructions that are executing on the processor cores 111-114. The patterns are detected based on numbers of bytes that separate addresses in successive memory access requests, which are referred to herein as “strides.” The numbers of bytes between the addresses are referred to herein as “stride lengths.” Thus, a memory access pattern can be represented by an initial address of a memory location and a sequence of subsequent addresses that correspond to the initial address plus integer multiples of the stride length of the memory access pattern. The number of integer multiples of the stride length that are used to prefetch information from the memory 110 into one or more of the caches 120, 121-124, 125-128, 131-134 is referred to herein as the “stride distance.” For example, if the stride distance for a memory access pattern is three, the prefetchers 135 can prefetch information indicated by three successive addresses that are separated from each other by a stride length.

Some embodiments of the prefetchers 135 can track strides on subgroups of addresses. For example, the data address streams generated by the processor cores 111-114 can be partitioned based on an instruction pointer (IP), a program counter (PC), a physical page that includes the address, or other criteria. Each prefetcher 135 can then track addresses in the data stream associated with one or more of the partitions. Tracking strides on subgroups of addresses can improve the accuracy of the tracking algorithm.

As discussed herein, some instructions issue memory access requests to access the memory 110 in blocks that are smaller than a cache line and some instructions access memory with a stride length that is not constant, e.g., with a sequence of

strides that have stride lengths equal to 8 bytes, 16 bytes, 16 bytes, 8 bytes, *etc.*

Furthermore, in some cases a memory access request is dropped or skipped, which can cause the stride length detected by the prefetcher 135 to change even though the underlying memory access pattern associated with the instruction remains the

5 same. The prefetchers 135 are therefore configured to set a stride length associated with an instruction that issued a memory access request to a length of a line in the caches 120, 121-124, 125-128, 131-134. In some variations, the prefetchers 135 determine a difference between a first address referenced by a current memory access request issued by an instruction and a second address in a previous memory access request issued by the instruction. The prefetchers 135 can then set the stride
10 length to the cache line length such that an absolute value of the difference is less than the cache line length. For example, if the difference between the first and second addresses is eight bytes and the cache line length is 64 bytes, the stride length is set to 64 bytes. For another example, if the difference between the first and
15 second addresses is negative because the second address is smaller than the first address, the stride length is set to -64 bytes if the absolute value of the difference is less than 64 bytes. Setting a stride length to the cache line length can be referred to as “snapping” the stride length to the cache line length.

The prefetchers 135 detect memory access patterns and prefetch information into the
20 caches 120, 121-124, 125-128, 131-134 based on the stride lengths that have been set or “snapped” to corresponding cache line lengths, as discussed herein. For example, the prefetchers 135 can prefetch a line from the memory 110 if a current value of the stride length associated with an instruction is the same as a previous value of the stride length, which indicates that the memory access requests are
25 continuing to follow the memory access pattern. The prefetchers 135 also check whether the address in the current memory access request is in the same line as the address of the previous memory access request. If so, the prefetchers 135 have already prefetched the data that would have been prefetched in response to the current memory access request and so the prefetchers 135 bypass prefetching any
30 data in response to the current memory access request. In some variations, the prefetchers 135 can implement other features such as snapping the stride length to twice the cache line length, prefetching information based on a stride confidence of

the stride length, issuing more than one prefetch request in response to a memory access request, and the like, as discussed herein.

FIG. 2 is a block diagram of a prefetcher 200 according to some embodiments. The prefetcher 200 is used to implement some embodiments of the prefetchers 135

5 shown in FIG. 1. The prefetcher 200 receives signals indicating events related to memory access requests such as hits or misses associated with a load instruction, hits or misses associated with a store instruction, and the like. Miss address buffer (MAB) events, such as hit or miss events for loads or stores, are received or accessed by an event selector block 205, which is used to select events that are to
10 be passed to other stages of the prefetcher 200. For example, the highest priority event can be stored in the registers 210 until they are passed to one or more stream engines 215 and a stream allocation unit 220, e.g., during a subsequent clock cycle. The priority of events can be determined using a hierarchy such as giving the highest priority to load misses and then assigning successively lower priorities to store
15 misses, load hits, and store hits.

The prefetcher 200 includes one or more stream engines 215 that can be used to manage separate prefetch streams. The stream engines 215 provide a signal to the stream allocation unit 220 to indicate that the current event either hit or missed the stream managed by the stream engine 215. If none of the existing streams indicates
20 a hit for the MAB miss event, then the stream allocation unit 220 can allocate a new stream to a different stream engine 215 using the current event information. When a stream is first allocated, the stream engine 215 sets a page address and an offset value to the current event cache line address. The stream engine 215 can then monitor further MAB events to detect events at addresses adjacent to the current
25 event cache line address in either direction.

As discussed herein, the prefetcher 200 can be configured to set a stride length associated with an instruction that issued a memory access request to a cache line length of the cache that receives the prefetched cache lines. For example, if the current event cache line address is set to A, then the stream engine 215 compares
30 addresses of events to the current event cache line address to determine a difference between the addresses, e.g., addresses A+8 bytes or A-8 bytes would have differences of +8 bytes and -8 bytes, respectively. The stream engine 215

determines a stride length for the event by snapping the differences to a cache line length. For example, a difference of +8 bytes is snapped to a stride length of 64 bytes and a difference of -8 bytes is snapped to a stride length of -64 bytes. For another example, a difference of +80 bytes is snapped to a stride length of 128 bytes and a difference of -80 bytes is snapped to a stride length of -128 bytes.

The snapped stride lengths are used to train the prefetch streams. If the stream engine 215 sees a stream of addresses associated with an instruction that have snapped stride length of 64 bytes, the stream engine 215 defines a stream in the appropriate direction (positive for A+64 bytes and negative for A-64 bytes) and trains a new prefetch stream. Some embodiments of the stream engine 215 can also implement additional features, as discussed herein.

The prefetcher 200 includes a request arbiter 225 that is used to arbitrate prefetch requests from the stream engines 215. The request arbiter 225 can be a rotating priority arbiter, but other types of request arbiter 225 can alternatively be implemented in the prefetcher 200. Requests are transferred from the request arbiter 225 to a register 230 so that the request information can be provided to a prefetch request interface 235, e.g., during a subsequent clock cycle. The prefetch request interface 235 provides feedback to the request arbiter 225, which can be used to select or arbitrate between pending requests from the stream engines 215.

FIG. 3 is a flow diagram of a method 300 for setting stride lengths based on a length of a line in a cache and prefetching data from the cache based on the stride length according to some embodiments. The method 300 is implemented by some embodiments of the prefetchers 135 shown in FIG. 1 or the prefetcher 200 shown in FIG. 2.

At block 305, a memory access request is received. The memory access request references a memory address. For example, the memory access request is generated by an instruction such as a read instruction or a write instruction. In those cases, the memory address indicates a memory location that is to be read or written by the instruction.

At block 310, the prefetcher determines a difference between the current memory address referenced by the memory access request and a previous memory address accessed by the previous memory access. As discussed herein, the difference indicates a number of bytes between the current memory address and the previous
5 memory address, which is either positive or negative depending on the relative positions of the current and previous memory addresses in the memory.

At block 315, the prefetcher sets (or snaps) the absolute value of a stride length associated with the memory access request to a length of a line in a cache that receives the prefetched information. For example, if the cache line length is 64 bytes,
10 absolute values of address differences of ± 8 bytes (or any other differences with absolute values that are less than or equal to 64 bytes) are snapped to 64 bytes. The stride length associated with the memory access request is then determined by attaching appropriate signs to the absolute values depending on the sign of the address difference. For example, the stride length for an address difference of 8
15 bytes is snapped to 64 bytes and the stride length for an address difference of -8 bytes is snapped to -64 bytes. In some variations, larger differences are snapped to a value equal to twice the cache line length. For example, the stride length for an address difference of 96 bytes is snapped to 128 bytes (*i.e.*, twice the cache line length) and the stride length for an address difference of -96 bytes is snapped to -128
20 bytes.

At decision block 320, the prefetcher determines whether to prefetch one or more lines from the memory into the cache based on a comparison of current and previous stride lengths. For example, the prefetcher determines whether the current stride length is equal to the previous stride length, which indicates whether the memory
25 access requests are continuing to follow the same access pattern. The prefetcher also determines whether to prefetch the one or more lines from the memory into the cache based on a comparison of current and previous memory addresses. For example, the prefetcher determines whether the current address is in the same line or a different line than the previous address, which indicates whether information in
30 the line that would be prefetched in response to the current memory access request has already been prefetched into the cache in response to the previous memory access request. If the current stride length is equal to the previous stride length and

the current address is in a different line than the previous address, the method 300 flows to block 325. If the current stride length is not equal to the previous stride length or the current address is in the same line as the previous address, the method flows to block 330.

- 5 At block 325, the prefetcher prefetches at least one line from an address indicated by a stride length and a stride distance associated with the memory access request. For example, if the stride length is 64 bytes and the stride distance is three, the prefetcher prefetches a line from an address in the memory that is separated from the current address by a distance (measured in bytes) equal to a product of the stride
10 length and the stride distance (*i.e.*, 192 bytes). As discussed herein, in some variations the prefetcher prefetches one or more additional lines that are one or more stride lengths from the previously prefetched line.

- At block 330, the prefetcher bypasses prefetching a line because at least a subset of the information in the line that would be prefetched at this stage has already been
15 prefetched into the cache in response to the previous memory access request.

Some variations of prefetcher such as the prefetchers 135 shown in FIG. 1 or the prefetcher 200 shown in FIG. 2 implement one or more additional features to modify the prefetching algorithm, as discussed herein. These features can further enhance the effectiveness of the prefetching algorithm.

- 20 The following pseudocode is an example of a stride prefetcher algorithm that does not snap stride lengths to a length of a line in a cache that receives the prefetched information. In the pseudocode, the memory address referenced by the current memory access request is indicated by *strideAddr*, the memory address referenced by the previous memory access request is indicated by *lastAddr*, the current and
25 previous stride length are indicated by *strideLength* and *last strideLength*, respectively, and the stride distance is indicated by *strideDistance*. Instructions are indicated by an instruction pointer (IP) and values of the stride parameters associated with each instruction are stored in a stride table.

For each instruction which references memory:

if (the IP of the current instruction matches one of the IPs stored
in a stride table) {

 update strideAddr

5 if (current strideLength is the same as last strideLength -and-
currentAddr is to a different line than lastAddr) {

 increment strideConfidence (if it is below some maximum
confidence)

 if (strideConfidence is greater than some threshold) {

10 issue prefetch to currentAddr + (strideDistance *
strideLength)

 if (strideDistance is less than some maximum
distance) {

 issue prefetch to currentAddr +
15 ((strideDistance+1) * strideLength)
 increment strideDistance

 }

 }

 } else if (strideConfidence is non-zero) {

20 decrement strideConfidence

 if (current strideLength is double last strideLength) {

 decrement strideDistance

 issue prefetch to currentAddr + (strideDistance *
strideLength)

25 } else {

 set strideDistance to zero

 }

 } else {

 update strideLength

30 set strideDistance to zero

 }

else if (the current memory reference missed in the local cache) {

 replace a stride prefetcher entry:

 update IP & address with the values from the current

35 memory reference

 set strideConfidence and strideDistance to zero)

}

Prefetchers that implement the preceding pseudocode maintain a value of a stride confidence (*strideConfidence*) to indicate the likelihood that the current instruction is issuing memory access requests that follow an access pattern indicated by the stride parameters. Thus, the prefetcher only prefetches information if the stride confidence is above a threshold value. The stride confidence is incremented each time the same stride length is repeated and the current access is in a different line than the previous access. In some variations, the stride confidence is only incremented up to a maximum stride confidence. Otherwise, the stride confidence is decremented until the stride confidence reaches zero.

- Table 1 is an example of a stride table that includes information indicating a current address for a current memory access request (CurrentAddr), a previous (or last) address for a previous memory access request (LastAddr), a previous stride length (LastStride), a previous stride confidence (LastConf), an indication of whether the current stride length matches the previous stride length (StrideMatch), the current stride length (NewStride), and a current stride confidence (NewConf).

Table 1

CurrentAddr	LastAddr	LastStride	Last Conf	StrideMatch?	NewStride	New Conf
0x0	N/A	N/A	0	No	N/A	0
0x20	0x0	N/A	0	No	0x20	0
0x50	0x20	0x20	0	No	0x30	0
0x70	0x50	0x30	0	No	0x20	0
0xa0	0x70	0x20	0	No	0x30	0
0xc0	0xa0	0x30	0	No	0x20	0
0xf0	0xc0	0x20	0	No	0x30	0
0x110	0xf0	0x30	0	No	0x20	0
0x140	0x110	0x20	0	No	0x30	0
0x160	0x140	0x30	0	No	0x20	0
0x190	0x160	0x20	0	No	0x30	0
0x1b0	0x190	0x30	0	No	0x20	0
0x1e0	0x1b0	0x20	0	No	0x30	0

As indicated in the above table, the stride lengths never match and so the stride confidence does not increase. Thus, the prefetcher does not issue any prefetches, even though the stride length of following an alternating pattern of 0x20, 0x30, 0x20, 0x30, etc.

The following pseudocode is an example of a stride prefetcher algorithm that snaps stride lengths to a length of a line in a cache that receives the prefetched information, as discussed herein. The pseudocode also snaps the stride length to a value equal to twice the length of the line the cache if the stride length is between the length of a cache line and twice the length of the cache line. The cache line length is indicated by *lineSize* in the pseudocode.

For each instruction which references memory:

if (the IP of the current instruction matches one of the IPs stored in the stride table) {

```

10     update strideAddr
        if (current strideLength is the "same" (after snapping address
            differences to -lineSize or lineSize) as last strideLength -
            and- currentAddr is to a different line than lastAddr) {
            increment strideConfidence (if it is below some maximum
15         confidence)
            if (strideConfidence is greater than some threshold) {
                issue prefetch to currentAddr + (strideDistance *
                    strideLength)
                if (strideDistance is less than some maximum
20         distance) {
                    issue prefetch to currentAddr +
                        ((strideDistance+1) * strideLength)
                    increment strideDistance
                }
            }
25         }
    } else if (strideConfidence is non-zero) {
        decrement strideConfidence
        if (current strideLength is "double" (for subline
            strides, between lineSize and 2*lineSize or between -
30     lineSize and -2*lineSize) last strideLength) {
            decrement strideDistance
            issue prefetch to currentAddr + (strideDistance *
                strideLength)
        } else {
35         set strideDistance to zero

```

```
        }  
    } else {  
        update strideLength (snapping subline strides to -  
        lineSize or lineSize)  
5        set strideDistance to zero  
    }  
else if (the current memory reference missed in the local cache) {  
    replace a stride prefetcher entry:  
        update IP & address with the values from the current  
10        memory reference  
        set strideConfidence and strideDistance to zero)  
}
```

In the above example pseudocode, the stride distance can be decremented in response to the current stride length being different than the previous stride length.

15 The stride distance can also be decremented in response to the current address being in the same line as the previous address. These features allow prefetchers that implement the pseudocode to adjust to strides that are either dropped or missing in the memory access pattern. Decrementing or reducing the stride prefetch distance in this manner can avoid gaps in the prefetch pattern introduced by the dropped or
20 missed strides.

Table 2 is an example of a stride table that is produced by applying the above example pseudocode to snap memory address differences to a cache line length before detecting memory access patterns or deciding whether to prefetch cache lines from memory to a cache, as discussed herein. The sequence of memory addresses
25 indicated in the memory access requests shown in Table 2 is the same as the sequence of memory addresses shown in Table 1.

Table 2

CurrentAddr	LastAddr	LastStride	Last Conf	StrideMatch?	NewStride	NewConf
0x0	N/A	N/A	0	No	N/A	0
0x20	0x0	N/A	0	Same line	No update	No update
0x50	0x20	N/A	0	No	<=64B	0
0x70	0x50	<=64B	0	Same line	No update	No update
0xa0	0x70	<=64B	0	Yes	<=64B	1
0xc0	0xa0	<=64B	1	Yes	<=64B	2
0xf0	0xc0	<=64B	2	Same line	No update	No update
0x110	0xf0	<=64B	2	Yes	<=64B	3*
0x140	0x110	<=64B	3	Yes	<=64B	4*
0x160	0x140	<=64B	4	Same line	No update	No update
0x190	0x160	<=64B	4	Yes	<=64B	5*
0x1b0	0x190	<=64B	5	Same line	No update	No update
0x1e0	0x1b0	<=64B	5	Yes	<=64B	6*

The address differences between the addresses in the sequence shown in Table 2 are all less than or equal to 64 bytes and so they are snapped to a value of 64 bytes, as indicated by "<=64B" in the LastStride column. The StrideMatch column indicates that the current stride matches the previous stride beginning with the memory access request to the address 0x70. However, the current address 0x70 of this memory access request is in the same line as the previous address 0x50, so the stride length is not updated and the stride confidence is not updated. The stride length associated with the memory access request to the subsequent address 0xa0 matches the stride length of the previous memory access request and is not in the same line as the address 0x50 of the previous memory access request. Consequently, the stride length is updated to "<=64B" as indicated in the NewStride column and the stride confidence is incremented as indicated in the NewConf column. Successive memory access requests lead to increases in the stride confidence until the stride confidence exceeds the threshold value (e.g., a threshold value of two) following the memory access request to the memory address 0x110. A prefetch request is issued in response to this memory access request, as well as the subsequent memory access requests, as indicated by the asterisk (*) on the stride confidence values.

In some embodiments, the apparatus and techniques described above are implemented in a system including one or more integrated circuit (IC) devices (also referred to as integrated circuit packages or microchips), such as the processing system described above with reference to FIGs. 1-3. Electronic design automation (EDA) and computer aided design (CAD) software tools are used in the design and fabrication of these IC devices. These design tools typically are represented as one or more software programs. The one or more software programs include code executable by a computer system to manipulate the computer system to operate on code representative of circuitry of one or more IC devices so as to perform at least a portion of a process to design or adapt a manufacturing system to fabricate the circuitry. This code can include instructions, data, or a combination of instructions and data. The software instructions representing a design tool or fabrication tool typically are stored in a computer readable storage medium accessible to the computing system. Likewise, the code representative of one or more phases of the design or fabrication of an IC device can be stored in and accessed from the same computer readable storage medium or a different computer readable storage medium.

A computer readable storage medium can include any non-transitory storage medium, or combination of non-transitory storage media, accessible by a computer system during use to provide instructions and/or data to the computer system. Such storage media can include, but is not limited to, optical media (e.g., compact disc (CD), digital versatile disc (DVD), Blu-Ray disc), magnetic media (e.g., floppy disc, magnetic tape, or magnetic hard drive), volatile memory (e.g., random access memory (RAM) or cache), non-volatile memory (e.g., read-only memory (ROM) or Flash memory), or microelectromechanical systems (MEMS)-based storage media.

The computer readable storage medium can be embedded in the computing system (e.g., system RAM or ROM), fixedly attached to the computing system (e.g., a magnetic hard drive), removably attached to the computing system (e.g., an optical disc or Universal Serial Bus (USB)-based Flash memory), or coupled to the computer system via a wired or wireless network (e.g., network accessible storage (NAS)).

In some embodiments, certain aspects of the techniques described above can be implemented by one or more processors of a processing system executing software. The software comprises one or more sets of executable instructions stored

or otherwise tangibly embodied on a non-transitory computer readable storage medium. The software can include the instructions and certain data that, when executed by the one or more processors, manipulate the one or more processors to perform one or more aspects of the techniques described above. The non-transitory computer readable storage medium can include, for example, a magnetic or optical disk storage device, solid state storage devices such as Flash memory, a cache, random access memory (RAM) or other non-volatile memory device or devices, and the like. The executable instructions stored on the non-transitory computer readable storage medium can be in source code, assembly language code, object code, or other instruction format that is interpreted or otherwise executable by one or more processors.

Note that not all of the activities or elements described above in the general description are required, that a portion of a specific activity or device may not be required, and that one or more further activities may be performed, or elements included, in addition to those described. Still further, the order in which activities are listed are not necessarily the order in which they are performed. Also, the concepts have been described with reference to specific embodiments. However, one of ordinary skill in the art appreciates that various modifications and changes can be made without departing from the scope of the present disclosure as set forth in the claims below. Accordingly, the specification and figures are to be regarded in an illustrative rather than a restrictive sense, and all such modifications are intended to be included within the scope of the present disclosure.

Benefits, other advantages, and solutions to problems have been described above with regard to specific embodiments. However, the benefits, advantages, solutions to problems, and any feature(s) that may cause any benefit, advantage, or solution to occur or become more pronounced are not to be construed as a critical, required, or essential feature of any or all the claims. Moreover, the particular embodiments disclosed above are illustrative only, as the disclosed subject matter may be modified and practiced in different but equivalent manners apparent to those skilled in the art having the benefit of the teachings herein. No limitations are intended to the details of construction or design herein shown, other than as described in the claims below. It is therefore evident that the particular embodiments disclosed above may be

altered or modified and all such variations are considered within the scope of the disclosed subject matter. Accordingly, the protection sought herein is as set forth in the claims below.

WHAT IS CLAIMED IS:

1. A method comprising:

receiving, at a processor, a memory access request to a first memory address
issued during execution of an instruction at the processor; and

setting, at the processor, a stride length associated with the instruction to a

5 length of a line in a cache, wherein the stride length indicates a number
of bytes between addresses of lines that are prefetched from the
memory into the cache.

2. The method of claim 1, wherein setting the stride length comprises:

determining a difference between the first memory address and a second

10 memory address of a previous memory request issued by the
instruction; and

setting the stride length to the length of the line such that an absolute value of
the difference is less than an absolute value of the length of the line.

3. The method of claim 2, wherein setting the stride length further comprises:

15 setting the stride length equal to the length of the line in response to the
difference being less than or equal to the length of the line.

4. The method of claim 2, wherein setting the stride length further comprises:

setting the stride length equal to a negative value of the length of the line in
response to the difference being negative and the absolute value of the
20 difference being less than or equal to the length of the line.

5. The method of claim 2, wherein setting the stride length further comprises:

setting the stride length equal to twice the length of the line in response to the
difference being greater than the length and less than or equal to twice
the length; and

25 setting the stride length equal to a negative value of twice the length of the line
in response to the difference being negative and the absolute value of
the difference being greater than the length and less than or equal to
twice the length.

6. The method according to any of the preceding claims, further comprising:

prefetching a line into the cache from a prefetch address in the memory,
wherein the prefetch address is equal to the first memory address plus
a product of the stride length and a stride distance that indicates a
number of strides that are prefetched ahead of the current address.

7. The method of claim 6, wherein the prefetching is performed in response to a
stride confidence associated with the stride length being greater than a threshold.

8. The method of claim 7, further comprising:

determining a previous stride length in response to a previous memory access
request being directed to a line with a memory address different than
the first memory address; and
incrementing the stride confidence in response to the stride length being equal
to the previous stride length.

9. An apparatus comprising:

a cache; and
a prefetcher to prefetch lines from a memory into the cache, wherein the
prefetcher is to receive a memory access request to a first memory
address issued during execution of an instruction and set a stride length
associated with an instruction that issued the memory access request to
a length of a line in the cache, wherein the stride length indicates a
number of bytes between addresses of lines that are prefetched from
the memory into the cache.

10. The apparatus of claim 9, wherein the prefetcher is to determine a difference
between the first memory address and a second memory address in a previous
memory request issued by the instruction and set the stride length to the length such
that an absolute value of the difference is less than an absolute value of the length.

11. The apparatus of claim 10, wherein the prefetcher is to set the stride length
equal to the length of the line in response to the difference being less than or equal to
the length of the line.

12. The apparatus of claim 10, wherein the prefetcher is to set the stride length equal to a negative value of the length of the line in response to the difference being negative and the absolute value of the difference being less than or equal to the length of the line.

5 13. The apparatus according to any of the preceding claims, wherein the prefetcher is to prefetch a line into the cache from a prefetch address in the memory, wherein the prefetch address is equal to the first memory address plus a product of the stride length and a stride distance that indicates a number of strides that are prefetched ahead of the current address.

10 14. The apparatus of claim 13, wherein the prefetcher is to prefetch the line into the cache in response to a stride confidence associated with the stride length being greater than a threshold.

15 15. The apparatus of claim 14, wherein the prefetcher is to increment the stride confidence in response to the stride length being equal to a previous stride length determined in response to a previous memory access request and the first memory address being to a different line than a previous address in the previous memory access request.

100

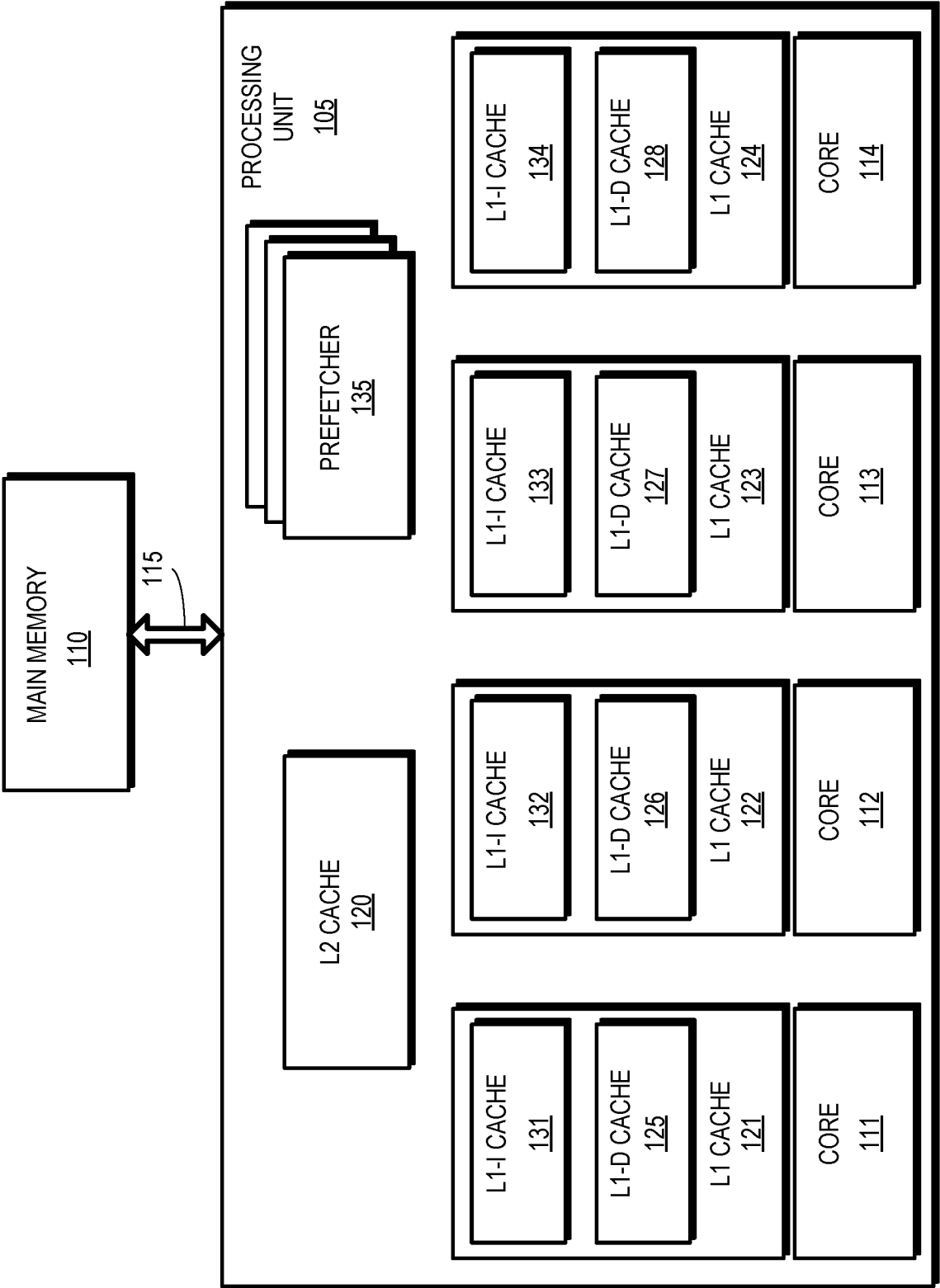
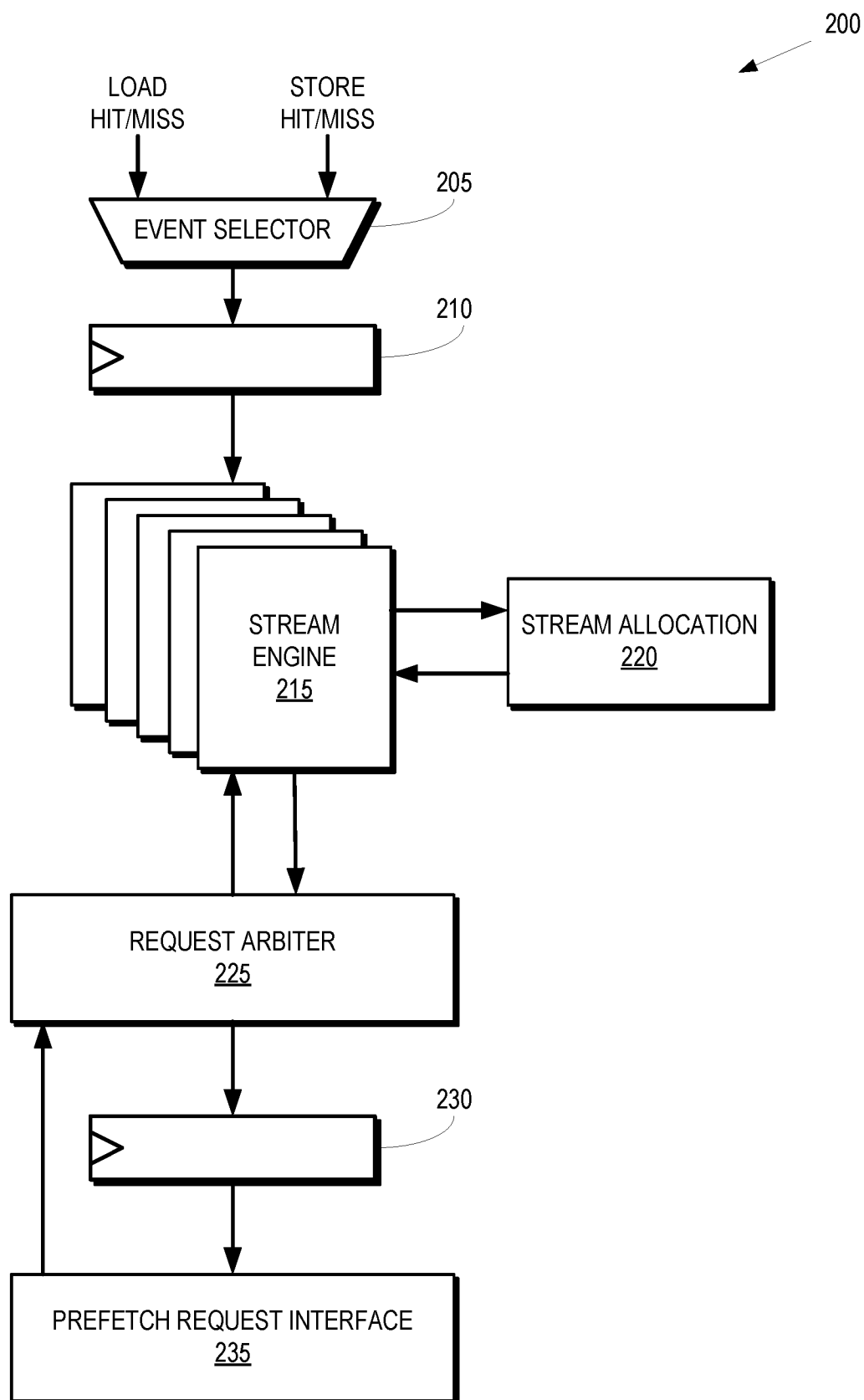
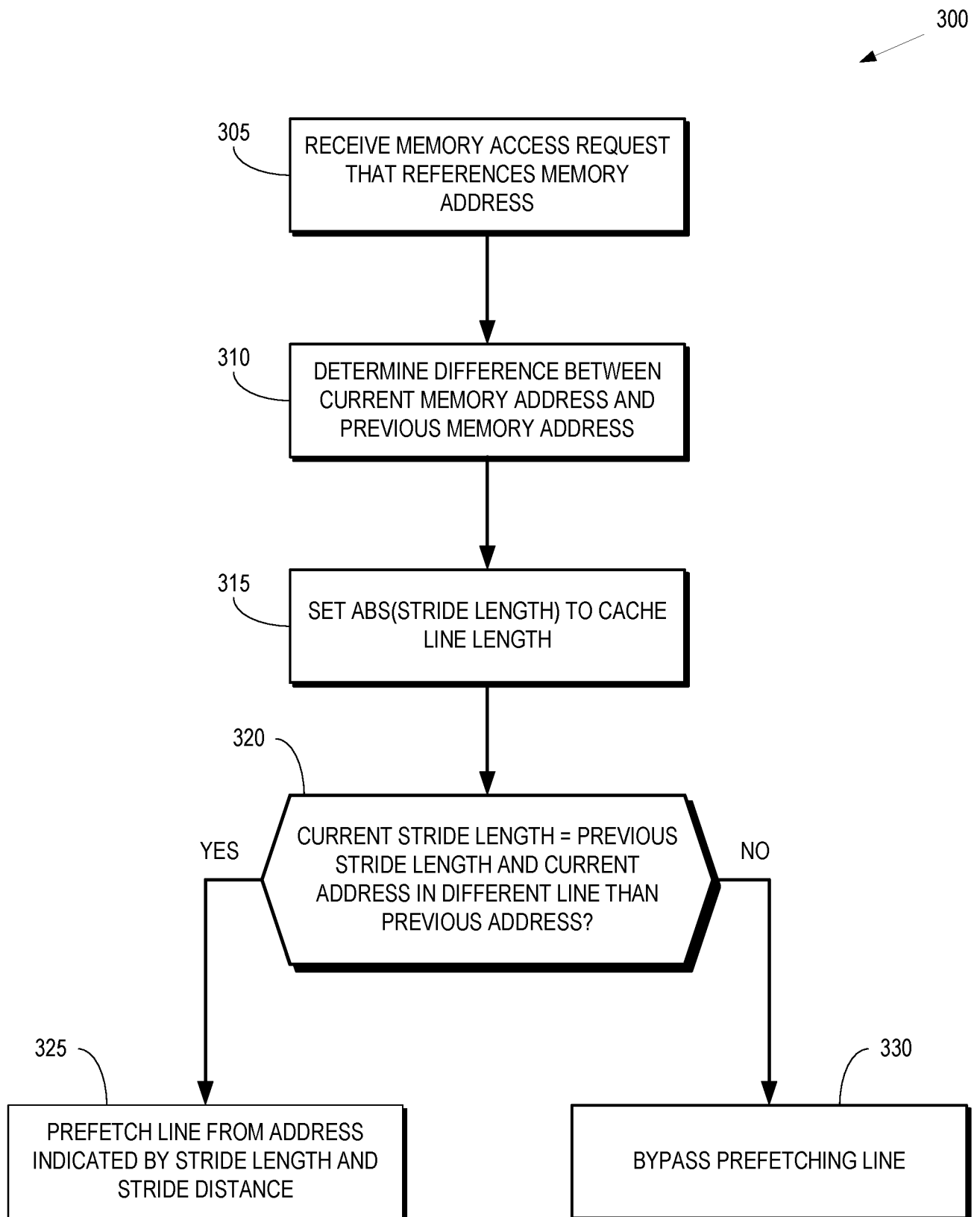


FIG. 1

2/3

**FIG. 2**

3/3

**FIG. 3**

A. CLASSIFICATION OF SUBJECT MATTER**G06F 9/38(2006.01)i, G06F 9/345(2006.01)i**

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 9/38; G06F 9/30; G06F 12/00; G06F 9/312; G06F 12/08; G06F 9/345

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

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

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

eKOMPASS(KIPO internal) & keywords: prefetch, cache line length, set, stride length, difference, memory address

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2008-0091921 A1 (DIAB ABUAIADH et al.) 17 April 2008	1,6,9,13
Y	See paragraphs [0044]-[0054], [0082]; claims 1-2, 7; and figures 2-3.	2-5, 7-8, 10-12, 14-15
Y	US 2003-0204673 A1 (SURESH VENKUMAHANTI et al.) 30 October 2003	2-5, 10-12
	See paragraphs [0015]-[0020]; claim 20; and figures 2-4.	
Y	US 2009-0216956 A1 (KATTAMURI EKANADHAM et al.) 27 August 2009	7-8, 14-15
	See paragraphs [0023]-[0024]; claim 8; and figures 4-5.	
A	US 2008-0301375 A1 (CHRISTOPHER MICHAEL DONAWA et al.) 04 December 2008	1-15
	See paragraphs [0052]-[0055]; claims 1, 7; and figures 5A, 5D.	
A	US 2009-0198965 A1 (RAVI K. ARIMILLI et al.) 06 August 2009	1-15
	See paragraphs [0040]-[0043]; claims 1, 3, 5; and figure 5B.	



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents:

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search

24 October 2017 (24.10.2017)

Date of mailing of the international search report

24 October 2017 (24.10.2017)

Name and mailing address of the ISA/KR

International Application Division

Korean Intellectual Property Office

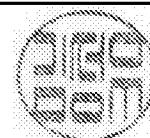
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea

Facsimile No. +82-42-481-8578

Authorized officer

KIM, Seong Woo

Telephone No. +82-42-481-3348



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2017/042341

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2008-0091921 A1	17/04/2008	None	
US 2003-0204673 A1	30/10/2003	US 6785772 B2	31/08/2004
US 2009-0216956 A1	27/08/2009	US 8285941 B2	09/10/2012
US 2008-0301375 A1	04/12/2008	US 2006-0253656 A1	09/11/2006
		US 7421540 B2	02/09/2008
		US 7761667 B2	20/07/2010
US 2009-0198965 A1	06/08/2009	US 8250307 B2	21/08/2012