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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 August 2016 (04.08.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/122494 A1

(51) International Patent Classification:
G06F 12/02 (2006.01)

(21) International Application Number:
PCT/US2015/013319

(22) International Filing Date:
28 January 2015 (28.01.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).

(72) Inventors: **WANG, Han**; 11445 Compaq Center Drive W., Houston, Texas 77070 (US). **VENUGOPAL, Raghavan**; 11445 Compaq Center Drive W., Houston, Texas 77070 (US). **ELLIOTT, Robert C.**; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).

(74) Agents: **SHOOKMAN, Jeb A.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79 , Fort Collins, CO 80528 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

Published:

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

(54) Title: GARBAGE COLLECTOR

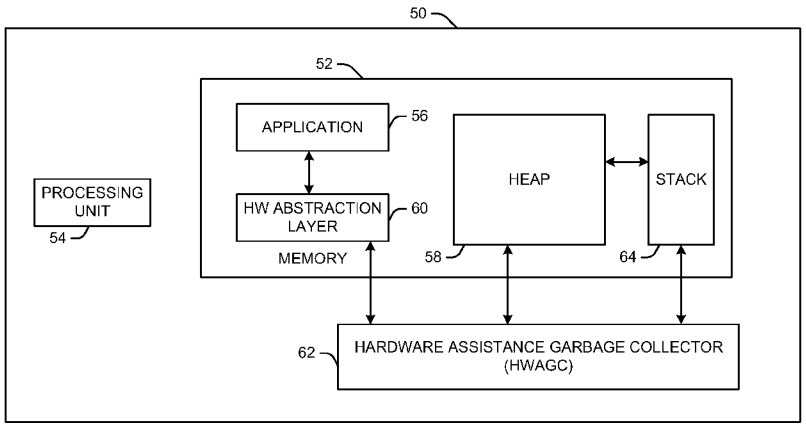

FIG. 1

(57) Abstract: A system can include a memory to store data and machine readable instructions. The memory can include a stack comprising an array of pointers to memory blocks in a heap. The system can also include a hardware assisted garbage collector (HWAGC) to control allocation of the blocks of memory in the heap. The HWAGC can also maintain a copy of the stack and detect a dead object in the heap.

WO 2016/122494 A1

GARBAGE COLLECTOR

BACKGROUND

[0001] In computer science, the task of fulfilling a memory allocation request includes locating a block of unused memory of sufficient size. Memory allocation requests can be satisfied by allocating portions from a large pool of memory referred to as the heap or free store. At any given time, some parts of the heap can be in use, while some can be "free" (unused) and thus available for future allocations.

[0002] Garbage collection is a form of automatic memory management. A garbage collector, or just collector, attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program (e.g., dead objects). Garbage collection is often portrayed as the opposite of manual memory management, which requires the programmer to specify which objects to de-allocate and return to the memory system. However, many systems use a combination of approaches, including other techniques such as stack allocation and region inference.

[0003] Many programming languages require garbage collection, either as part of the language specification (for example, Java, C#, D language, Go and most scripting languages) and/or for practical implementation. These programming languages can be referred to as garbage collecting languages. Other languages were designed for use with manual memory management, but have garbage collected implementations available (for example, C and C++). Some programming languages, such as Ada, Modula-3, and C++/CLI allow both garbage collection and manual memory management to co-exist in the same application by using separate heaps for collected and manually managed objects; others, like D, are garbage collected but allow the user to manually delete objects and also entirely disable garbage collection.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] FIG. 1 illustrates an example of a system to implement hardware assisted garbage collection.

[0005] FIG. 2 illustrates a conceptualize drawing of a heap.

[0006] FIG. 3 illustrates another example of a system to implement hardware assisted garbage collection.

[0007] FIG. 4 illustrates yet another example of a system to implement hardware assisted garbage collection.

[0008] FIG. 5 illustrates still yet another example of a system to implement hardware assisted garbage collection.

[0009] FIG. 6 illustrates still yet another example of a system to implement garbage collection.

[0010] FIG. 7 illustrates an example of a flowchart of a method for implementing garbage collection.

DETAILED DESCRIPTION

[0011] Example systems are described to implement hardware assisted garbage collection. A hardware assisted garbage collector (HWAGC) can allocate and de-allocate memory blocks in a heap. The HWAGC can create a copy of a stack for the heap and monitor the copy of the stack to detect a dead object in the heap. In response to detecting the dead objects, the HWAGC can free the memory blocks allocated to the dead object for reallocation. Additionally, the HWAGC can defragment the heap to increase a number of consecutive free memory blocks. The defragmenting can include moving a given object in the heap from a current memory block (at a current location) to a new memory block (at a new location) to remove "gaps" in the heap. The HWAGC can operate concurrently with (and independently from) operations of a software application that instantiates and employs objects in the heap. In this manner, the operations of the HWAGC do not halt execution of the software application.

[0012] FIG. 1 illustrates an example of a system 50 that could be employed to implement garbage collection in a computing device. The system 50 could be implemented, for example, as a server, a desktop computer, a laptop computer, a mobile device (e.g., a smartphone), etc. The system 50 can include a memory 52 for storing data and machine readable instructions. The memory 52 could be implemented, for example, as non-volatile memory, volatile memory, or a combination thereof. In some examples, the memory 52 could be implemented as a non-volatile dual in-line

memory module (NVDIMM). In other examples, the memory 52 could be implemented as a dynamic read access memory (DRAM) module. The system 50 can also include a processing unit 54 to access the memory 52 and execute machine readable instructions. The processing unit 54 can include one or more processor cores.

[0013] The memory 52 can include an application 56 (e.g., a software application) executing in the memory 52. The application 56 can be, for example, a compiled set of machine instructions. In some examples, the application 56 can be compiled from a program written with automatic and/or manual garbage collection, such as Java, C++, C#, D language, Go, Ada, Modula-3, etc. In some examples, the application 56 can include or be running within a runtime environment, such as the Java Runtime Environment (JRE). The application 56 can include instructions that instantiate and employ an object that can be allocated in a heap 58 of the memory 52 in a manner described herein. The heap 58 is a partition of the memory 52 that can be employed for dynamic memory allocation.

[0014] To instantiate the object, the application 56 can be programmed to provide a memory allocation request to a hardware abstraction layer 60. The hardware abstraction layer 60 can be implemented, for example, as a software interface between the application 56 and a hardware assistance garbage collector (HWAGC) 62. The memory allocation request can include a size of a memory block needed to instantiate the object.

[0015] The hardware abstraction layer 60 can forward the memory allocation request to the HWAGC 62. The HWAGC 62 can be implemented as hardware (circuitry). In some examples, the HWAGC 62 can be implemented as logic in a field programmable gate array (FPGA). In other examples, the HWAGC 62 can be implemented as a microcontroller with instructions embedded therein, an application specific integrated circuit (ASIC) chip, etc. The HWAGC 62 can operate concurrently with and independently of the instructions executed by the processing unit 54. In some examples, the HWAGC 62 can be implemented on a memory module that includes the memory 52 (e.g., a NVDIMM module or a DRAM module). In other examples, the HWAGC 62 can be implemented on a memory controller of the system 50.

[0016] The HWAGC 62 can be configured to control allocation in the heap 58. The memory 52 can include a stack 64. The stack 64 includes a list an array (e.g., a list) of pointers to memory addresses in the heap 58. The HWAGC 62 can be configured to maintain a copy of the stack. In response to the memory allocation request, the HWAGC 62 can generate a pointer in the stack 64 to point to a block of memory in the heap 58 that has a size sufficient to accommodate the object. Additionally, the generation of the pointer can also be recorded in the copy of the stack that is stored on the HWAGC 62, such that consistency between the stack 64 and the copy of the stack in the HWAGC 62 is maintained. The HWAGC 62 can return the generated pointer to the hardware abstraction layer 60. In response, the hardware abstraction layer 60 can return the pointer to the application 56. The application 56 can then access the block of memory in the heap 58 pointed to by the pointer returned in response to the memory allocation request.

[0017] Additionally, during the course of execution of the application 56, the application may delete a previously instantiated object. To delete an object, the application 56 can provide a delete request to the hardware abstraction layer 60. The delete request can include the pointer that points to a memory address in the heap 58 that stores the data for the object being deleted.

[0018] The hardware abstraction layer 60 can provide the HWAGC 62 with the delete request. The HWAGC 62 can control the stack 64, and cause the pointer in the stack 64 identified in the delete request to be deleted. The HWAGC 62 can also delete the pointer identified in the delete request from the copy of the stack, thereby maintaining consistency between stack 64 and the copy of the stack included in the HWAGC 62.

[0019] Additionally, the HWAGC 62 can include a heap scanner that can scan the copy of the stack stored in the HWAGC 62 to track live objects. Each live object is reachable. Each live object can be a root object or a child of a root object. Objects that are unreachable through root objects are referred to as dead objects.

[0020] FIG. 2 illustrates a conceptual diagram of a heap 100 depicting root objects 102, child objects 104 and dead objects 106. Each root object 102 can be representative of a local or global variable, an active thread a static variable, a

framework reference (e.g., a Java Native Interface (JNI)) employed by the application 56, etc. As is illustrated, each root object 102 includes the child objects 104 that are reachable by the root objects 102. Some of the child objects 104 reference other child objects 104. Both the root objects 102 and the child objects 104 are considered live objects. The dead objects 106 are unreachable. That is, no root object 102 references any of the dead objects 106. In most situations, dead objects 106 were originally live objects that were (previously) referenced by a root object, directly or indirectly. However, at some point during execution of the application 56, the direct or indirect connection between the root object 102 and the dead object 106 was severed. Such a severing can result from deletion of the corresponding root object 102, the deletion of a child object 104 that previously referenced the dead object 106 or the deletion of the reference (at the root object 102 or the child object 104) to the dead object 106.

[0021] Referring back to FIG. 1, by monitoring the copy of the stack at the HWAGC 62, the heap scanner can identify dead objects in the heap 58. Upon detection of a dead object (e.g., garbage), a stack scanner of the HWAGC 62 can delete the pointer to the dead object in the stack 64 and the pointer to the dead object in the copy of the stack in the HWAGC 62. In this manner, the HWAGC 62 can reallocate the block of memory in the heap 58 that was previously allocated to the dead object, such as in response to another memory allocation request. In this manner the HWAGC 62 acts as a garbage collector. Moreover, since the HWAGC 62 can operate concurrently (e.g., in parallel) with operations of the processing unit 54, the stack scanner of the HWAGC 62 can delete the pointer in the stack 64 and the copy of the stack included in the HWAGC 62 without halting execution of the application 56 (or any other application), which halting is sometimes referred to as "stopping the world".

[0022] Additionally or alternatively, the heap scanner of the HWAGC 62 can be configured to defragment the live objects in the heap 58 to ensure that a number consecutive free memory blocks in the heap 58 are available. To defragment the heap 58, the heap scanner copy data of an object stored in a memory block (an old memory block) to a new memory block. Upon completion of the copy, the heap scanner can update the pointer in the stack 64 and the copy of the stack in the HWAGC 62

referencing the old memory block to reference the new memory block, thereby moving the object. This process can be repeated to increase the number of free consecutive memory blocks in the heap 58.

[0023] By employing the system 50, the hardware abstraction layer 60 and HWAGC 62 can operate as a garbage collector that avoid the unwanted "stop the world" operation of pure software garbage collectors. Additionally, by defragmenting the heap 58 to increase the number of free consecutive memory blocks in the heap 58, large objects can be instantiated.

[0024] FIG. 3 illustrates another example of a system 200 that includes a HWAGC 62. The system 200 can be employed, for example, to implement the system 50 illustrated in FIG. 1. The system 200 can be employed, for example, to implement a computing device, such as a server, a desktop computer, a laptop computer or a mobile computing device, such as a tablet computer or a smartphone.

[0025] The system 200 can include memory 202 to store data and machine readable instructions. The memory 202 could be implemented as a memory module. The memory 202 could be implemented, for example, as volatile memory (e.g., DRAM) or non-volatile memory (e.g., NVDIMM) or a combination thereof. The system 200 can also include a processing unit 204 that can access the memory 202 and execute the machine readable instructions. The processing unit 204 can include one or more processor cores. The system 200 can also include a HWAGC 206. The HWAGC 206 can be implemented to include circuity and logic (e.g., on an FPGA, a microcontroller with embedded instructions, an ASIC, etc.). The HWAGC 206 can be implemented, for example, on a memory module or a memory controller.

[0026] The system 200 could be implemented, for example in a computing cloud. In such a situation, features of the system 200, such as the processing unit 204, the memory 202 and the HWAGC 206 could be representative of a single instance of hardware or multiple instances of hardware with applications executing across the multiple of instances (distributed) of hardware (e.g., computers, routers, memory, processors, or a combination thereof). Alternatively, the system 200 could be implemented on a single dedicated server.

[0027] The memory 202 can include a runtime environment (RTE) 208 that can provide a framework for executing software applications. The RTE 208 can be implemented, for example as JRE, the common language runtime, etc. In some examples, the RTE 208 can include a virtual machine, such as the Java Virtual Machine (JVM). The RTE 208 can include a software application 210 that is executing in the RTE 208. The software application 210 can include a compiled set of machine readable instructions (e.g., application software).

[0028] The RTE 208 can include a software address mapping component 212 that can be configured to track addresses of variables and objects employed by the software application. The RTE 208 can also include an address mapping and reference mapping component 214 and a software garbage collector 216 that can interface with a hardware abstraction layer 218 of the memory 202. The hardware abstraction layer 218 can provide an interface between the RTE 208 and the HWAGC 206. The RTE 208 can be configured such that the address mapping and reference mapping component 214 and the software garbage collector 216 are each configured as a pass-through to the hardware abstraction layer 218. In this manner, the hardware abstraction layer 218 and the HWAGC 206 can perform the memory allocation and other processing that is conventionally performed by the address mapping and reference mapping component 214 and the software garbage collector 216.

[0029] The software application 210 can execute code that initiates instantiation of a new object. The new object could be, for example, a root object or a child object. The software address mapping 212 can detect the initiation of the instantiation of the new object and determine an amount of memory needed for the new object. Upon such a determination, the software address mapping 212 can provide a memory allocation request that characterizes the size of the memory block needed for the new object to the address mapping and reference mapping component 214. In response, the address mapping and reference mapping component 214 can forward the memory allocation request to the hardware abstraction layer 218. The hardware abstraction layer 218 can provide the HWAGC 206 with the memory allocation request.

[0030] The HWAGC 206 can be configured to allocate and de-allocate blocks of memory of a heap 220 stored in the memory 202. The HWAGC 206 can include control

logic 207 that can control the operation of the HWAGC 206. The memory 202 can include a stack 222 that has N number of pointers 224, where N is an integer greater than or equal to one. Each of the N number of pointers 224 (labeled in FIG. 3 as "P1"…"PN") can store a corresponding address (labeled in FIG. 3 as "A1"…"AN"). Each address of each pointer 224 can identify an address in the heap 220. Moreover, each address identified in a pointer 224 can store data of a corresponding one of N number of objects 226. For purposes of simplification of explanation, in FIG. 3, the first pointer 224 ("P1" of FIG. 3) is illustrated and described as storing the first address ("A1" of FIG. 3) that points to the memory location in the heap 220 of the first object 226 ("OBJECT 1" of FIG. 3). The second to Nth pointers 224, addresses and objects 226 are also illustrated and described in a similar manner. However, in other situations, different orientations and correspondences could be employed.

[0031] The HWAGC 206 can include a stack scanner 228 that can include (or access) a stack copy 330. The stack copy 330 can be a copy of the stack 222. Moreover, changes to the stack 222 are also made to the stack copy 330 and vice versa, such that consistency between the stack 222 and the stack copy 330 is maintained. Accordingly, the stack copy 330 can include N number of pointers 332, wherein each of the N number of pointers 332 in the stack copy 330 corresponds to a pointer 224 of the stack 222. Additionally, in FIG. 3, the first pointer 332 ("P1" of FIG. 3) of the stack copy 330 includes the same address ("A1" of FIG. 3) as the first pointer 224 ("P1" of FIG. 3) of the stack 222. Thus, as illustrated, both the first pointer 224 ("P1" of FIG. 3) of the stack 222 and the first pointer 332 ("P1" of FIG. 3) of the stack copy 330 include an address that points to the first object ("OBJECT 1" of FIG. 3). Moreover, the second to Nth pointers 332 are configured in a similar manner.

[0032] In response to the memory allocation request, the control logic 207 can generate a pointer in the stack 222. For purposes of simplification of explanation, it is assumed that the first pointer ("P1" of FIG. 3) is generated in response to the memory allocation request. Additionally, the control logic 207 also generates a pointer 332 in the stack copy 332. In the present situation, it is presumed that the first pointer 332 ("P1" of FIG. 3) in the stack copy 330 matches the first pointer ("P1" of FIG. 3) of the stack 222. The first pointer 224 of the stack 222 and the first pointer 332 of the stack copy 330

point to the first address of a memory block assigned to the first object 226 ("OBJECT 1" of FIG. 3). The size of the memory block for the first object 226 can be based, for example, on a size identified in the memory allocation request.

[0033] The HWAGC 206 can include a heap scanner 334 that can maintain a list of live objects 336 in the heap 220. Upon generation of the pointer for the first object 226, the control logic 207 can add the pointer to the list of live objects 336 of the heap 220. The list of live objects 336 can include M number of pointers 338, where M is an integer greater than or equal to one. As illustrated and described with respect to FIG. 2, dead objects (e.g., garbage) are unreachable by root objects, and live objects are directly or indirectly reachable through at least one root object.

[0034] The control logic 207 can return the generated pointer ("P1" of FIG. 3) to the hardware abstraction layer 218. The hardware abstraction layer 218 can provide the software address mapping 212, via the address mapping and reference mapping component 214 with the generated pointer. The software address mapping 212 can receive the generated pointer and provide a message to the software application 210 indicating that memory has been allocated for the new object. In response, the software application 210 can complete instantiation of the new object. Such completion can include writing data to the block of memory assigned to the new object. To write the data to the block of memory, the software application 210 can provide the data for the new object to the software address mapping 212. Additionally, the data can be passed to the hardware abstraction layer 218, and the hardware abstraction layer 218 can pass the data to the control logic 207 that can write the data to the memory block of the new object ("OBJECT 1" of FIG. 3). Similarly, at some point, the software application 210 may read data from the new object. In such a situation, the software application can request data from the new object, and the software address mapping 212 can provide a read request that includes the pointer associated with the new object to the hardware abstraction layer 218. The hardware abstraction layer 218 can send the read request to the control logic 207, and the control logic 207 can read the data at the address identified in the pointer and return data read to the hardware abstraction layer 218 to satisfy the read request.

[0035] Similarly, at some point, the software application 210 may provide a request to delete the new object to the software address mapping 212. In response, the software address mapping 212 can provide a delete request to the hardware abstraction layer 218 via the address mapping and reference mapping component 214 of the software garbage collector 216. The delete request can include the pointer to the new object. As noted, in the present situation, it is presumed that the new object is the first object 226 ("OBJECT 1" of FIG. 3) of the heap 220, such that the pointer would be the first pointer 224 ("P1") in the stack 222. The delete request can be provided to the HWAGC 206. In response, the control logic 207 can delete the first pointer 224 from the stack 222. Additionally, the control logic 207 can delete the first pointer 332 from the stack copy 330 to maintain consistency between the stack 222 and the stack copy 330. Further, the control logic 207 can remove the pointer 338 ("P1" of FIG. 3) for the new object from the list of live objects 336.

[0036] As the software application 210 continues to execute, numerous objects can be generated in a similar manner as described for generation of the first object 226. In some (but not necessarily all) instances, deletion request for the objects 226 can be processed in a similar manner. Through such a generation and deletion or possibly due to a lack of deleting, some of the N number of objects 226 may become dead objects (e.g., garbage). As illustrated and described with respect to FIG. 2, dead objects are objects that have not been deleted but are unreachable directly or indirectly by root objects. Through implementing garbage collection procedures, the memory blocks assigned to dead objects are freed for reallocation by the HWAGC 206 in the following manner.

[0037] To remove dead objects such that the associated memory block in the heap 220 can be reallocated, the heap scanner 334 can monitor the list of live objects 336 of the heap 220 and cooperate with the stack scanner 228 to compare the list with the pointers 332 in the stack copy 330. In this manner, a dead object can be an object that has a pointer in the stack copy 330, but is absent from the list of live objects 336. In one example (hereinafter, "the given example"), it is presumed that the second pointer ("P2") of the stack copy 330 is not included in the list of live objects 336, as highlighted by the shaded pattern in FIG. 3. In the given example, upon detection

that the second object ("OBJECT 2" of FIG. 3) is dead, the stack scanner 228 can delete the second pointer 332 associated with the dead object from the stack copy 330. Additionally, the stack scanner 228 can delete the second pointer 224 from the stack 222 to maintain consistency with the stack copy 330 and thereby freeing the memory block associated with the second object 226 for reallocation. Moreover, the operations of the HWAGC 206 can be executed independently from and concurrently with the operations of the RTE 208, including, but not limited to the operations of the software application 210. In this manner there is no need to halt execution ("stop the world") of the software application 210 in order to perform garbage collection procedures.

[0038] Additionally, the heap scanner 334 can scan the addresses of the list of live objects 336 to identify "gaps" in the heap 220. The gaps can be removed in a defragmenting or "compacting" process. For instance, continuing with the given example, upon deletion of the second object 226 ("OBJECT 2" of FIG. 3), the heap scanner 334 can be configured to move each of the third to Nth objects 226 to new locations to remove the "gap" in the heap 220 that results from deleting the second object 226. Alternatively, to remove the gap, in some examples, only the first object ("OBJECT 1" of FIG. 3) could be moved to the position of the second object 226. It is noted that these are conceptual examples only, and that actual moving of the objects 226 may include moving the objects 226 across separate physical memory modules. Moreover, elimination and/or reduction of the gaps can often require moving many objects since the objects may have a great variety of sizes. However, for purposes of simplification of explanation, in the given example, it is presumed that the gap can be removed by moving the first object 226 to the location in the heap 220 that was occupied by the second object 226 prior to deletion.

[0039] In the given example, to move the first object 226 ("OBJECT 1" of FIG. 3) to the memory block previously occupied by the second object 226 ("OBJECT 2" of FIG. 3), the heap scanner 334 can provide the stack scanner 228 with a move request that identifies the pointer of the first object 226 and a memory address of the block for the second object 226. In response, the stack scanner 228 can add a reference in the first pointer 332 in the stack copy 330 that references an address of the memory block

previously occupied by the second object 226. The additional reference is labeled as "A2" in FIG. 3, such that the first pointer 332 ("P1" of FIG. 3) of the stack copy 330 has two different addresses, namely "A1 & A2". Upon adding the new reference to the first pointer 332 in the stack copy 330, the heap scanner 334 can be configured to copy the contents of the first object 226 to the memory address identified in the new reference, which is the memory block previously occupied by the second object 226 in the given example. During the copying process, the first pointer 224 in the stack 222 can remain unchanged. The copying process can be executed at low priority and can be interrupted in response to a read or write request for the first object 226.

[0040] As noted, if the HWAGC 206 receives a read request for the first object 226 prior to completion of the copy process, the copy process can be interrupted. During interruption, the control logic 207 can process the read request. In particular, the control logic 207 can access the first pointer 224 of the stack 222, which references the "old" address of the first object ("OBJECT 1" of FIG. 3). The contents of the first object 226 (at the old address) can be provided via the hardware abstraction layer 218 to the software address mapping 212 to satisfy the read request for the first object 226. Additionally, in some examples, the control logic 207 can write the content provided to the software address mapping 212 to the new address for the first object 226 (namely, the pervious location of the second object 226) to reduce the work load remaining on the copy process. Moreover, upon satisfying the read request, the copy process can be resumed by the HWAGC 206. Additionally, it is noted that the copy process can be completed without halting the execution of the software application 210 or any other application, since the HWAGC 206 can operate independently and concurrently with other programs.

[0041] Additionally, as noted, in the given example, if the HWAGC 206 receives a write request for the first object 226 prior to completion of the copy process, the copy process can be interrupted. During the interruption, the control logic 207 can write data included in the write request to both the old location for the first object 226 and the new location for the first object 226. Upon satisfying the write request, the copy process can be resumed by the HWAGC 206.

[0042] In the given example, upon completion of the copy process, the stack scanner 228 can update the first pointer 224 of the stack 222 to reference the new address (the memory block previously occupied by the second object 226). Additionally, the stack scanner 228 can modify the first pointer of the stack copy 330 to reference only the new address for the first object 226 and remove reference to the old address of the first object 226. Similarly, the heap scanner 334 can update the list of live objects 336 to reflect the moving of the first object 226. In this manner, upon completion of the copy process, consistency between the stack 222 and the stack copy 230 is restored and the first object 226 is effectively moved from the old address to the new address, and the memory block at the old address can be reallocated in a manner described herein.

[0043] The moving process of objects 226 may be executed many times by the HWAGC 206 to complete a defragmenting process. Upon completion of the defragmenting process, the heap 220 can increase the amount of consecutive memory blocks available in the heap 220. Increasing the number of consecutive memory blocks can increase the maximum size that can be allocated for a single object.

[0044] FIG. 4 illustrates another example of a system 250 that can implement garbage collection. The system 200 can be implemented in manner similar to the system 50 of FIG. 1. The system 250 can include a memory 252 to store data and machine readable instructions. The memory 252 includes a stack 254 comprising an array of pointers to memory blocks in a heap 256. The system 250 can also include a HWAGC 258 to control allocation of the blocks of memory in the heap 256. The HWAGC 258 can also maintain a copy of the stack 260 and detect a dead object in the heap 256.

[0045] FIG. 5 illustrates yet another example of a system 300 that can implement garbage collection. The system 300 can be implemented in a manner similar to the system 50 of FIG. 1. The system 300 can include a memory 302 to store data and machine readable instructions. The system 300 can also include a HWAGC 304. The HWAGC 304 can include a control module 305 to control allocation of a heap 306 of the memory 302 and to free memory blocks assigned to dead objects in the heap 306. The memory 302 can include a stack 307 that can include an array of pointers to memory

blocks in the heap 306. The HWAGC 304 can also include a maintain module 308 to maintain a copy of the stack 307 at the HWAGC 304. The system 300 can further include a processing unit 309 to access the memory 302 and execute the machine readable instructions. The machine readable instructions can include a software application 310 to instantiate objects stored in the heap 306 of the memory 302. The machine readable instructions can further include a hardware abstraction layer 312 to operate as an interface between the software application 310 and the HWAGC 304. The HWAGC 304 can operate concurrently with operations of the software application 310.

[0046] In some examples, the HWAGC 304 can include a defragment module 314 to defragment the heap 306. Defragmentation of the heap 306 can increase an amount of unallocated consecutive memory blocks in the heap 306.

[0047] FIG. 6 illustrates still yet another example of a system 350 that can implement garbage collection. The system 350 can be implemented in a manner similar to the system 50 of FIG. 1. The system 350 can include a memory 352 to store data and machine readable instructions. The system 350 can also include a HWAGC 354. The HWAGC 354 can include a control module 355 to control allocation of a heap 356 of the memory 352 and to free memory blocks assigned to dead objects in the heap 356. The system 350 can further include a processing unit 358 to access the memory 352 and execute the machine readable instructions. The machine readable instructions can include a software application 360 to instantiate objects stored in the heap 356 of the memory 352. The machine readable instructions can further include a hardware abstraction layer 362 to operate as an interface between the software application 350 and the HWAGC 354. The HWAGC 354 can operate concurrently with operations of the software application 360.

[0048] In view of the foregoing structural and functional features described above, example methods will be better appreciated with reference to FIG. 7. While, for purposes of simplicity of explanation, the example methods of FIG. 7 is shown and described as executing serially, it is to be understood and appreciated that the present examples are not limited by the illustrated order, as some actions could in other examples occur in different orders, multiple times and/or concurrently from that shown

and described herein. Moreover, it is not necessary that all described actions be performed to implement a method.

[0049] FIG. 7 illustrates an example of a flowchart of a method 400 for implementing garbage collection. The method 400 can be implemented, for example, by a HWAGC, such as the HWAGC 62 of FIG. 1. At 410, the HWAGC can control allocation and de-allocation of blocks of memory in a heap of memory. The memory can include a stack. The stack can include an array of pointers that each point to a memory block in the heap. At 420, a copy of the stack can be maintained at the HWAGC. At 430, a pointer to a dead object in the heap can be deleted by the HWAGC.

[0050] In view of the foregoing structural and functional description, those skilled in the art will appreciate that portions of the systems and method disclosed herein may be embodied as a method, data processing system, or computer program product such as a non-transitory computer readable medium. Accordingly, these portions of the approach disclosed herein may take the form of an entirely hardware embodiment, an entirely software embodiment (e.g., in a non-transitory machine readable medium), or an embodiment combining software and hardware. Furthermore, portions of the systems and method disclosed herein may be a computer program product on a computer-usable storage medium having computer readable program code on the medium. Any suitable computer-readable medium may be utilized including, but not limited to, static and dynamic storage devices, hard disks, optical storage devices, and magnetic storage devices.

[0051] Certain embodiments have also been described herein with reference to block illustrations of methods, systems, and computer program products. It will be understood that blocks of the illustrations, and combinations of blocks in the illustrations, can be implemented by computer-executable instructions. These computer-executable instructions may be provided to one or more processors of a general purpose computer, special purpose computer, or other programmable data processing apparatus (or a combination of devices and circuits) to produce a machine, such that the instructions, which execute via the one or more processors, implement the functions specified in the block or blocks.

[0052] These computer-executable instructions may also be stored in computer-readable memory that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer-readable memory result in an article of manufacture including instructions which implement the function specified in the flowchart block or blocks. The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide steps for implementing the functions specified in the flowchart block or blocks.

[0053] Implementations of the subject matter described in this specification can be implemented in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the subject matter described is this specification, or any combination of one or more such back-end, middleware, or front-end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

[0054] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

[0055] What have been described above are examples. It is, of course, not possible to describe every conceivable combination of structures, components, or methods, but one of ordinary skill in the art will recognize that many further combinations and permutations are possible. Accordingly, the invention is intended to embrace all such alterations, modifications, and variations that fall within the scope of

this application, including the appended claims. Where the disclosure or claims recite "a," "an," "a first," or "another" element, or the equivalent thereof, it should be interpreted to include one or more than one such element, neither requiring nor excluding two or more such elements. As used herein, the term "includes" means includes but not limited to, and the term "including" means including but not limited to. The term "based on" means based at least in part on.

CLAIMS

What is claimed is:

1. A system comprising:

a memory to store data and machine readable instructions, wherein the memory includes a stack comprising an array of pointers to memory blocks in a heap;

a hardware assisted garbage collector (HWAGC) to:

control allocation of the blocks of memory in the heap;

maintain a copy of the stack; and

detect a dead object in the heap.

2. The system of claim 1, wherein the HWAGC is further to delete a pointer to the dead object, thereby facilitating reallocation of a memory block associated with the dead object.

3. The system of claim 2, wherein the dead object is detected in the absence of halting execution of a program that instantiates objects stored in the heap.

4. The system of claim 1, wherein the HWAGC comprises a live objects list that includes a list of live objects in the heap, wherein each live object is either a root object or reachable by a root object.

5. The system of claim 1, wherein the HWAGC is further to defragment the heap.

6. The system of claim 5, wherein the defragmenting includes moving a given object in the heap to increase an amount of unallocated consecutive memory blocks.

7. The system of claim 6, wherein moving the given object comprises:

adding a reference to a new memory block in a pointer for the given object that is stored in the copy of the stack; and

copying data stored in a current memory block of the heap assigned to the given object to the new memory block of the heap.

8. The system of claim 7, wherein the copying is interrupted in response to a request for read or write access to the given object.

9. The system of claim 9, wherein moving the given object further comprises changing a reference of a pointer for the given object that is stored in the stack to the new memory block.

10. The system of claim 1, wherein the memory and the HWAGC are implemented on a non-volatile dual inline memory module.

11. The system of claim 1, wherein the HWAGC is implemented on a memory controller.

12. A system comprising:

a memory to store data and machine readable instructions;

a hardware assistance garbage collector (HWAGC) comprising:

a control module to:

control allocation of a heap of the memory; and

free memory blocks assigned to dead objects in the heap; and

a processing unit to access the memory and execute the machine readable instructions, the machine readable instructions comprising:

a software application to instantiate objects stored in the heap of the memory; and

a hardware abstraction layer to operate as an interface between the software application and the HWAGC;

wherein the HWAGC is further to operate concurrently with operations of the software application.

13. The system of claim 12, wherein the HWAGC further comprises a defragment module to defragment the heap to increase an amount of unallocated consecutive memory blocks in the heap.

14. The system of claim 12, wherein the memory comprises a stack comprising an array of pointers to memory blocks in the heap and the HWAGC further comprises a maintain module to maintain a copy of the stack.

15. A method comprising:

controlling, at a hardware assisted garbage collector (HWAGC), allocation and de-allocation of blocks of memory in a heap of memory, wherein the memory comprises a stack comprising an array of pointers that each point to a memory block in the heap;

maintaining a copy of the stack at the HWAGC; and

deleting, at the HWAGC, a pointer to a dead object in the heap.

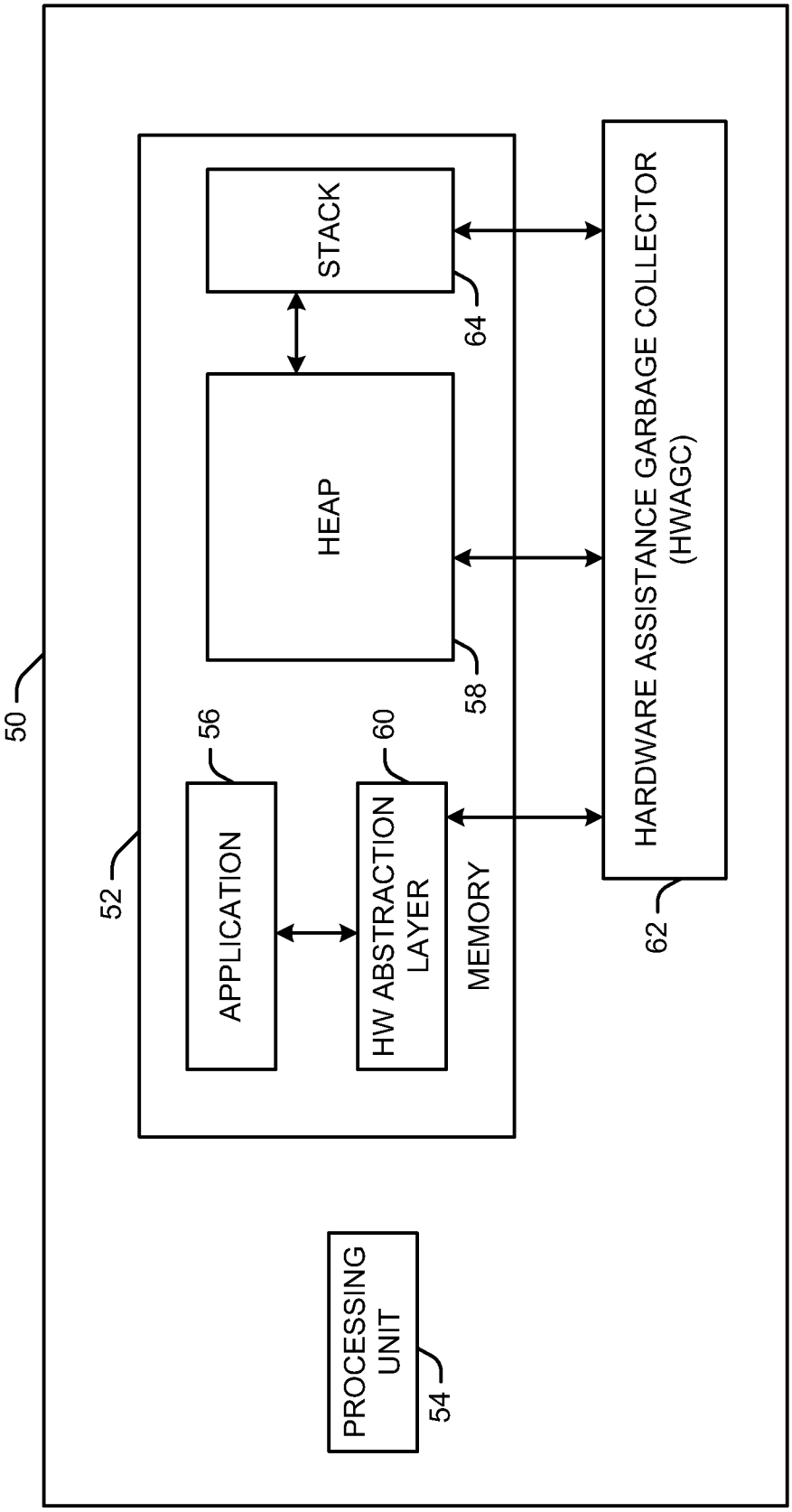
50
52
STACK
64
HEAP
58
56
APPLICATION
60
HW ABSTRACTION LAYER
MEMORY
HARDWARE ASSISTANCE GARBAGE COLLECTOR (HWAGC)
62
PROCESSING UNIT
54

FIG. 1

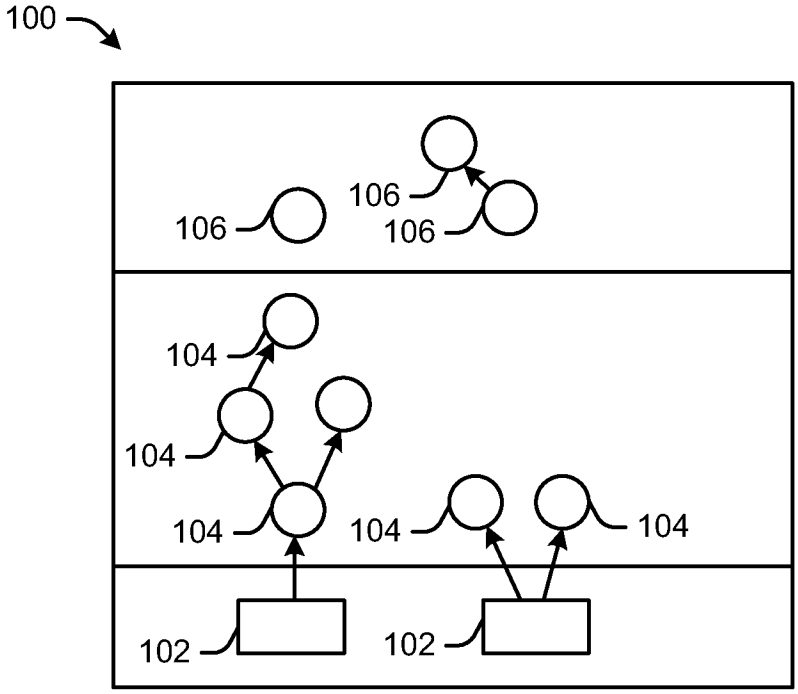

FIG. 2

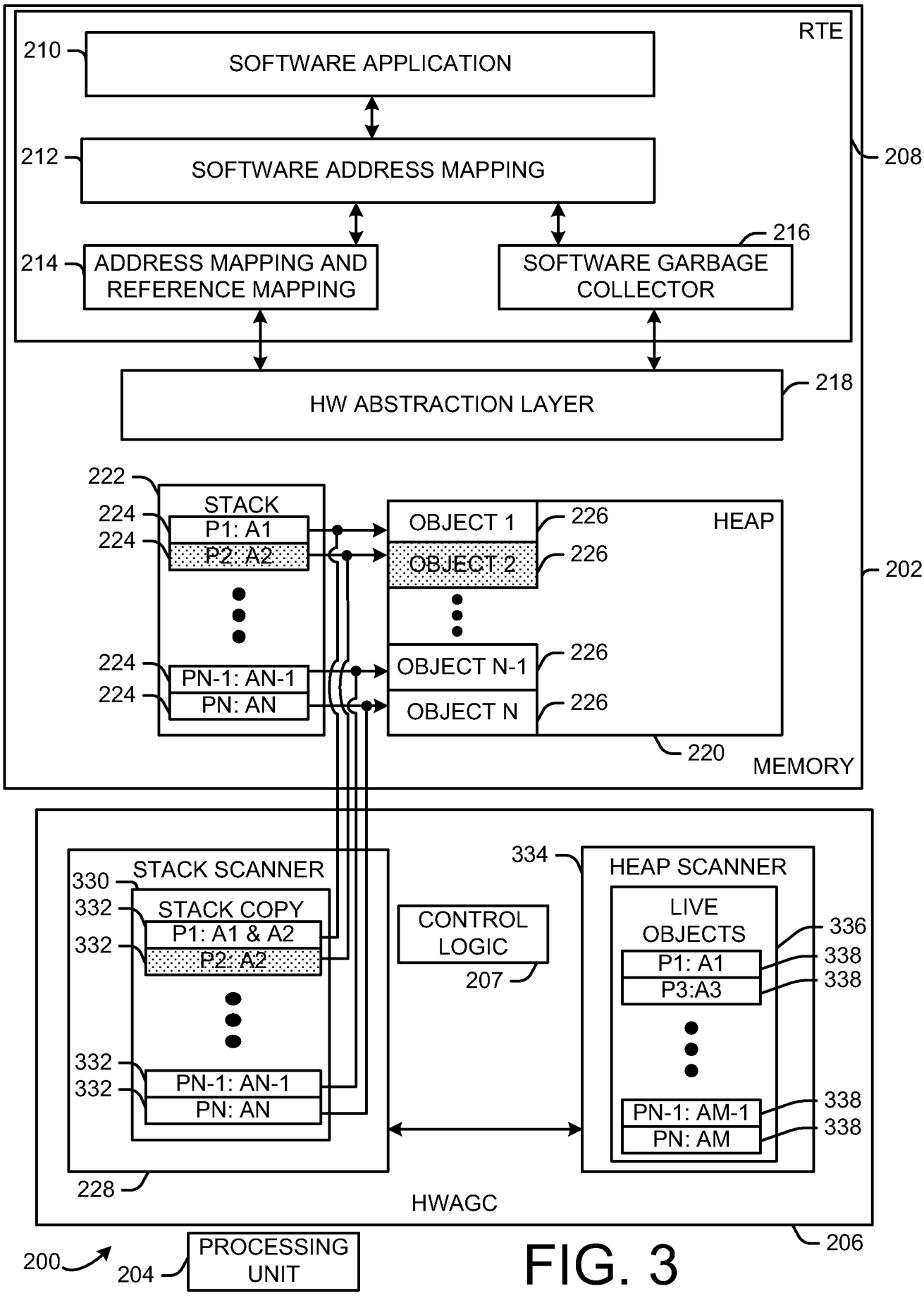
RTE
210
SOFTWARE APPLICATION
212
SOFTWARE ADDRESS MAPPING
214
ADDRESS MAPPING AND REFERENCE MAPPING
216
SOFTWARE GARBAGE COLLECTOR
208
218
HW ABSTRACTION LAYER
222
STACK
224
P1: A1
224
P2: A2
224
PN-1: AN-1
224
PN: AN
OBJECT 1
226
OBJECT 2
226
OBJECT N-1
226
OBJECT N
226
HEAP
220
MEMORY
202
334
HEAP SCANNER
LIVE OBJECTS
336
P1: A1
338
P3:A3
338
PN-1: AM-1
338
PN: AM
338
CONTROL LOGIC
207
330
STACK SCANNER
332
STACK COPY
P1: A1 & A2
332
P2: A2
332
PN-1: AN-1
332
PN: AN
228
HWAGC
206
200
204
PROCESSING UNIT

FIG. 3

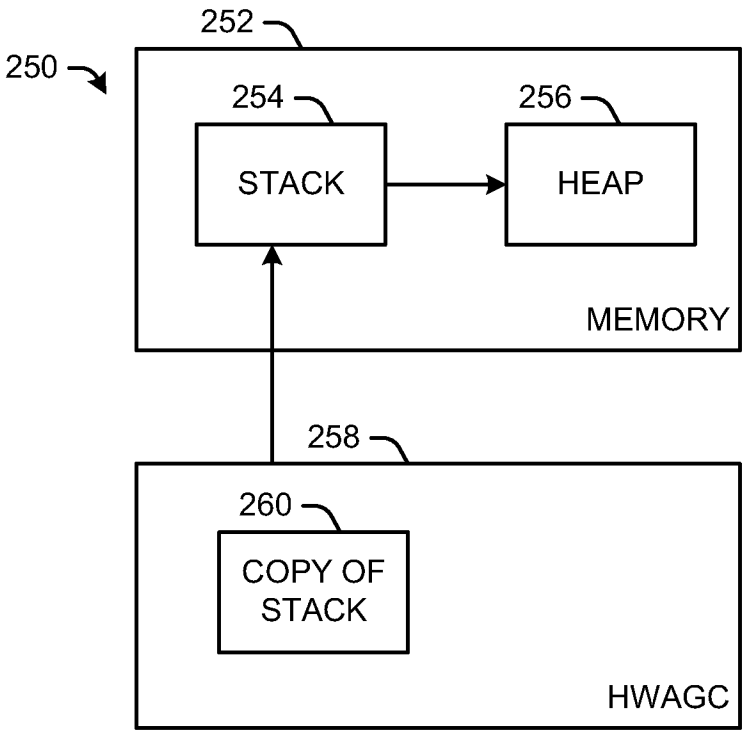

FIG. 4

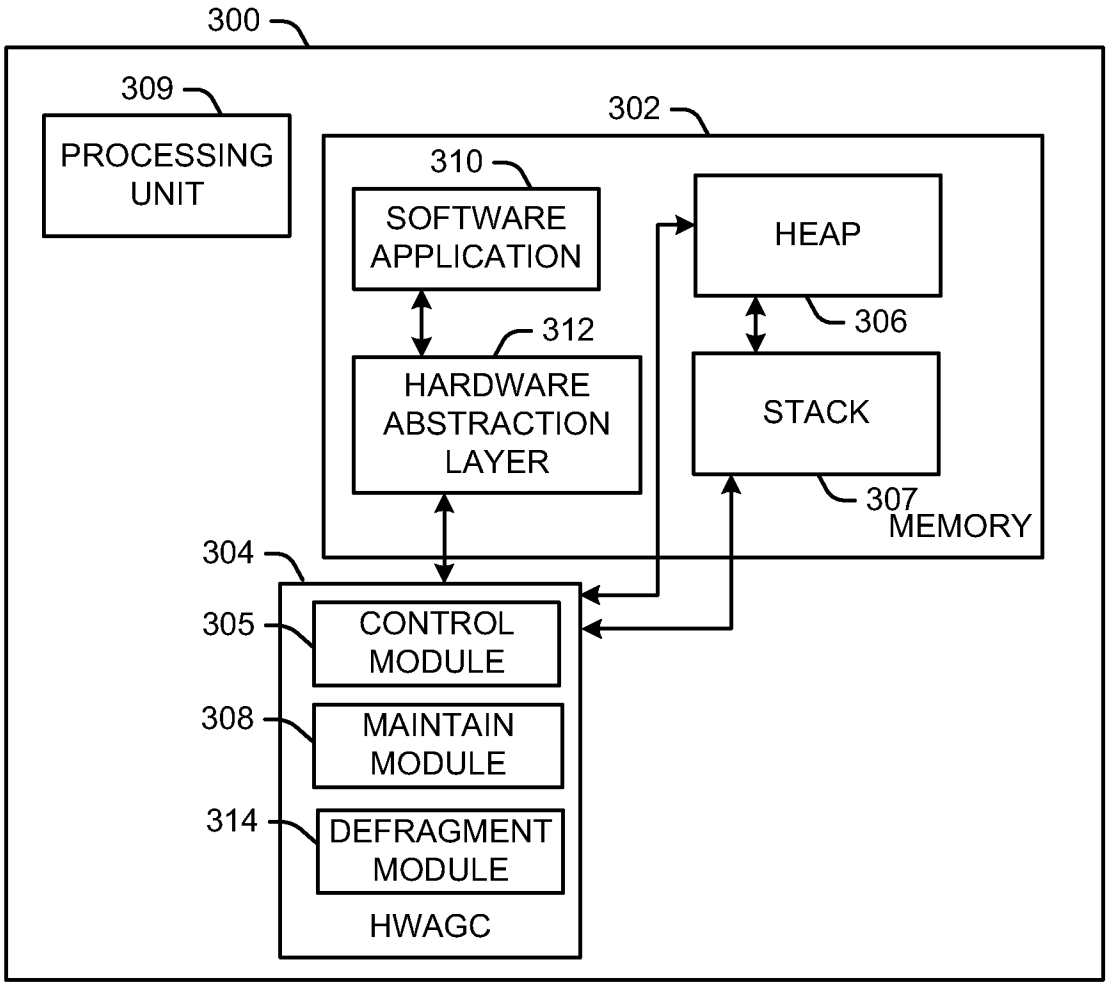
300
309
PROCESSING UNIT
302
310
SOFTWARE APPLICATION
HEAP
306
312
HARDWARE ABSTRACTION LAYER
STACK
307
MEMORY
304
305
CONTROL MODULE
308
MAINTAIN MODULE
314
DEFRAGMENT MODULE
HWAGC

FIG. 5

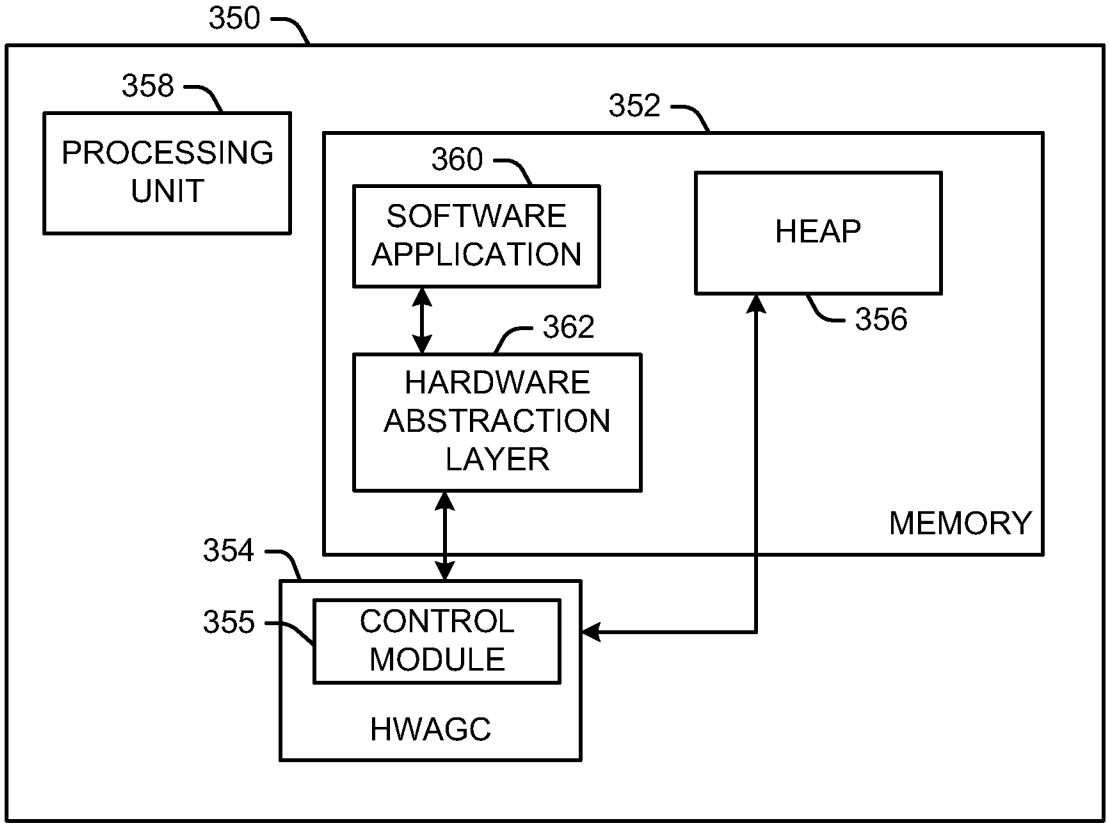
350
358
PROCESSING UNIT
352
360
SOFTWARE APPLICATION
HEAP
356
362
HARDWARE ABSTRACTION LAYER
MEMORY
354
355
CONTROL MODULE
HWAGC

FIG. 6

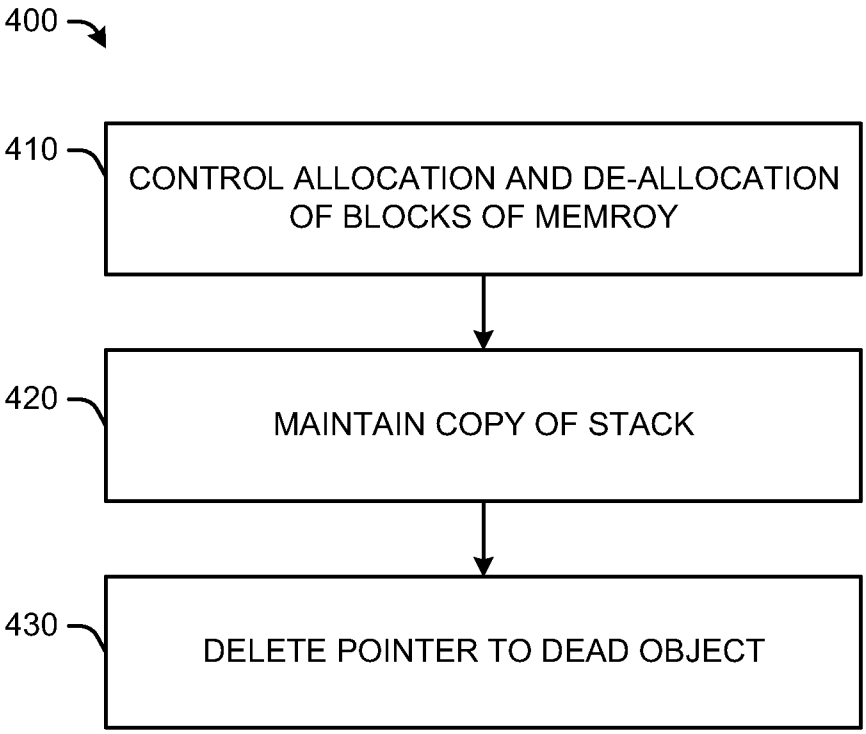
400
410
CONTROL ALLOCATION AND DE-ALLOCATION OF BLOCKS OF MEMROY
420
MAINTAIN COPY OF STACK
430
DELETE POINTER TO DEAD OBJECT

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/013319

A. CLASSIFICATION OF SUBJECT MATTER

G06F 12/02(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 12/02; G06F 12/16; G06F 17/00; G06F 17/30; G06F 12/00; G06F 9/45

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: memory, hardware, assisted, garbage, collector, heap, stack

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2013-0318290 A1 (DAVID F. BACON et al.) 28 November 2013 See paragraphs [0004], [0013]-[0019], [0044], [0047], [0053]; and figures 1, 6.	1-15
Y	US 2013-0185337 A1 (CLOUDERA, INC.) 18 July 2013 See paragraphs [0026]-[0033], [0044]-[0048], [0069]; and figure 1.	1-15
A	US 2002-0166116 A1 (ERIK L. EIDT) 07 November 2002 See paragraphs [0009], [0018]-[0019]; and figure 6.	1-15
A	US 2011-0145304 A1 (JAN GRAY et al.) 16 June 2011 See paragraphs [0015], [0072]; and figure 1A.	1-15
A	WO 2013-110189 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 01 August 2013 See paragraphs [0006]-[0008], [0032]; and figure 1.	1-15

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 October 2015 (13.10.2015)	**16 October 2015 (16.10.2015)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea
Facsimile No. +82-42-472-7140

Authorized officer
BYUN, Sung Cheal
Telephone No. +82-42-481-8262

Form PCT/ISA/210 (second sheet) (January 2015)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2015/013319

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2013-0318290 A1	28/11/2013	US 08856491 B2	07/10/2014
		US 2013-318315 A1	28/11/2013
US 2013-0185337 A1	18/07/2013	None	
US 2002-0166116 A1	07/11/2002	US 06295640 B1	25/09/2001
		US 06453466 B1	17/09/2002
		US 06684392 B2	27/01/2004
US 2011-0145304 A1	16/06/2011	US 08402218 B2	19/03/2013
		US 2013-238579 A1	12/09/2013
WO 2013-110189 A1	01/08/2013	CN 104137093 A	05/11/2014
		DE 112013000650 T5	06/11/2014
		GB 2515414 A	24/12/2014
		JP 2015-505623 A	23/02/2015
		US 08972680 B2	03/03/2015
		US 2013-0191610 A1	25/07/2013