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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 August 2017 (24.08.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/142525 A1

(51) International Patent Classification:
*G06F 15/173* (2006.01) *G06F 12/02* (2006.01)

(21) International Application Number:
PCT/US2016/018227

(22) International Filing Date:
17 February 2016 (17.02.2016)

(25) Filing Language: English

(26) Publication Language: English

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

(72) Inventors: **VOLOS, Haris**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). **LI, Jun**; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US).

(74) Agents: **KIRCHEV, Ivan T.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road Mail Stop 79, Fort Collins, Colorado 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: ALLOCATING A ZONE OF A SHARED MEMORY REGION

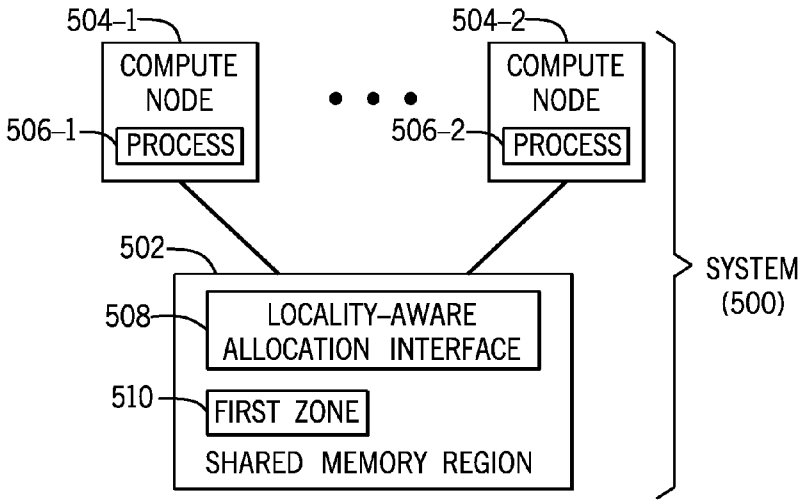


FIG. 5

(57) Abstract: In some examples, a system includes a shared memory region, a plurality of compute nodes, and a plurality of processes executable in the plurality of compute nodes. A first process of the plurality of processes is to use an allocation interface to perform locality-aware allocation of a first zone of the shared memory region, the first zone owned by the first process and accessible by the first process and a second process of the plurality of processes. The locality-aware allocation of the first zone to allocate the first zone that has a specified proximity to a given compute node of the plurality of compute nodes.

WO 2017/142525 A1

# ALLOCATING A ZONE OF A SHARED MEMORY REGION

## Background

[0001] A distributed system can include a number of compute nodes that can execute program code. Each instance of a program code executed on a compute node can be referred to as a process, such that multiple processes are executed in the compute nodes. As part of the execution of the processes, data generated by one process can be shared with another process.

## Brief Description Of The Drawings

[0002] Some implementations are described with respect to the following figures.

[0003] Fig. 1 is a block diagram of an example distributed system that includes a distributed arrangement of compute nodes and a shared memory region, in accordance with some implementations.

[0004] Fig. 2 is a flow diagram of an example flow according to some implementations.

[0005] Figs. 3A-3C illustrate an example of opening a shared memory heap and allocating blocks from the shared memory heap, according to some implementations.

[0006] Fig. 4 is a flow diagram of an example flow according to further implementations.

[0007] Fig. 5 is a block diagram of an example system according to some implementations.

[0008] Fig. 6 is a block diagram of an example storage medium storing instructions according to some implementations.

Detailed Description

[0009] Processes executing on compute nodes can share data with one another. In some cases, the sharing of data can be performed using network communications, inter-process communications (IPC), or a shared file. Such sharing techniques involve copying data from private memory associated with each process to a shared medium, such as a network, a shared buffer, or a shared file. Such copying of the data from private memory to the shared medium is associated with processing and communications overhead that can reduce overall system performance.

[0010] In accordance with some implementations of the present disclosure, the sharing of data between processes executing on compute nodes can be accomplished using a shared memory region. A shared memory region can refer to a memory region that is implemented in memory media that is associated with respective compute nodes. The memory media can be implemented with memory devices or portions of memory devices, which can include non-volatile memory devices (*e.g.* memristor memory devices, phase-change memory devices, and so forth) and/or volatile memory devices such as dynamic random access memory (DRAM) devices.

[0011] A memory region of the storage space provided by the memory media can refer to at least a portion of the overall storage space that is useable by processes running in a system. Note that the portion of the overall storage space can be made up of a single contiguous region or multiple regions in the overall storage space. In some examples, a shared memory region can be referred to as a shared memory heap. A memory heap (or more simply, heap) refers to a pool of memory that can be allocated for use, where some parts of the heap are free and available for future allocation to processes, while other parts of the heap are currently allocated to respective processes.

[0012] A "process" can refer to an instance of program code (machine-readable instructions such as software or firmware) that can be executed on a compute node. A compute node can include a computer, multiple computers, a processor, multiple

processors, a core (or multiple cores) of a multi-core processor, or any other hardware processing circuit. Multiple processes can execute on a compute node.

[0013] Fig. 1 shows a block diagram of an example distributed system 100 that includes multiple compute nodes 102-1, 102-2, 102-3, and 102-4. Although four compute nodes are shown in the example of Fig. 1, it is noted that in other examples, different numbers of compute nodes can be provided. Multiple processes can be executed on the respective multiple compute nodes 102-1, 102-2, 102-3, and 102-4. For example, processes 104-1 and 104-2 can execute on the compute node 102-1, process 104-3 can execute on the compute node 102-2, a process 104-4 can execute on the compute node 102-3, and processes 104-5, 104-6, and 104-7 can execute on the compute node 102-4.

[0014] The distributed system 100 also includes a shared memory region 106 (e.g. a shared memory heap), which is accessible by each compute node 102-1 to 102-4. In some implementations, the shared memory region 106 is directly accessible by each compute node 102-1 to 102-4, where directly accessible can refer to a compute node being able to issue a memory command over a memory bus to access a location in the shared memory region 106.

[0015] The shared memory region 106 can be partitioned or divided into multiple zones 110-1, 110-2, 110-3, and 110-4. Although four zones are shown in Fig. 1, it is noted that in further examples, the shared memory region 106 can be divided into a different number of zones. Each zone can refer to a contiguous region of the shared memory region 106. Each zone of the shared memory region 106 can be physically allocated from a respective memory medium associated with a corresponding compute node. Note that the shared memory region 106 can be implemented with multiple memory media that can be associated with corresponding compute nodes 102-1 to 102-4.

[0016] The memory media can be provided at different locations within the distributed system 100, with each memory medium being more closely associated with one of the compute nodes than another of the compute nodes. For example, a

first memory medium can be more closely associated with a first compute node than to other compute nodes.

[0017] A memory medium is more closely associated with a first compute node than a second compute node if the first compute node can access the memory medium with lower latency than the second compute node (in other words, the access time involved in the first compute node accessing the memory medium is less than the access time involved when the second compute node accesses the memory medium). In some examples, a given memory medium that is more closely associated with a given compute node can be provided in the same computing platform, on a common integrated circuit die, or otherwise is interconnected by a relatively lower latency communications link to the given compute node.

[0018] A zone allocated from a memory medium that is more closely associated to a given compute node can be accessed by the given compute node with lower latency than by another compute node. Stated differently, different zones of the shared memory region 106 can be accessed with different latencies by a particular compute node. The arrangement of the shared memory region 106 is a non-uniform memory access (NUMA) arrangement, in which different compute nodes may have different access latencies to different memory portions of the shared memory region 106.

[0019] Each zone is owned by a single process, i.e. the process that requested allocation of the zone. However, even though the zone is owned by a single process, the zone is accessible concurrently by multiple processes that execute on the corresponding compute nodes 102-1 to 102-4, such that data stored in the zone can be shared by the multiple processes. The multiple processes are able to perform both read access and write access of a zone owned by a particular process.

[0020] The zones 110-1 to 110-4 are thus shared zones that allow the processes104-1 to 104-7 to share data with each other using the zones. By using the shared zones, a process would not have to copy data from a private memory region (e.g. private heap) of the process to a shared medium. Instead, each process

can write data to a shared zone using a direct memory write access, and another process can read the data written to the shared zone using a direct memory read access.

[0021] A zone is owned by a process if the process requested allocation of the zone and that has the ability to de-allocate (free) the zone, such that the freed zone can be allocated to another process.

[0022] In some examples, each zone is a fixed-size contiguous region of the shared memory region 106, such that the different zones that are allocated for respective processes all have the same fixed size. In other examples, different zones can have different sizes.

[0023] In some implementations, a zone can further be split into smaller blocks. The blocks split from a zone can have arbitrary sizes, such that each zone can be split into arbitrary sized blocks.

[0024] The shared memory region 106 is also associated with a memory manager 108 that can perform a number of different tasks. The memory manager 108 can be implemented as a combination of hardware processing circuitry and program code that can provide respective tasks. The hardware processing circuit can include a microprocessor, a microcontroller, a core of a microprocessor, a programmable gate array, a programmable integrated circuit, or any other hardware processing circuit.

[0025] The memory manager 108 can manage read and/or write access of the shared memory region 106.

[0026] In addition, in accordance with some implementations of the present disclosure, the memory manager 108 includes a locality-aware allocation interface 112 that is accessible by processes executing on the compute nodes 102-1 to 102-4 to allocate a portion (e.g. a block) of the shared memory region 106. In some examples, the locality-aware allocation interface 112 can also be used by a process to de-allocate (free) a memory portion (e.g. a block) of the shared memory region

106. An "allocation interface" can refer to a component that is accessible by a process to request that an action be performed with respect to the shared memory region 106.

[0027] A locality-aware allocation of a block of the shared memory region 106 can refer to an allocation in which the block that is allocated has a specified proximity to a given compute node of the compute nodes 102-1 to 102-4 (e.g. the block is allocated from a zone that can be accessed by a first compute node with lower latency than another zone). For example, the specified proximity can be indicated by a locality indication provided with a memory allocation request, such as a call of a malloc() routine. The locality indication can be in the form of a hint or other information that can specify or suggest that the block is to be allocated from a zone that is more closely associated with a given compute node. For example, the locality indication can be in the form of an identifier of a compute node, an identifier of a process, location information associated with a compute node, or any other information that can be used to derive a target locality for the block that is to be allocated.

[0028] In alternative examples, a locality-aware allocation of a block can allocate the block from a zone that is more closely associated with a compute node on which the process that requested the allocation is executed. For example, if the process requesting the allocation of the block is running on compute node 102-1, then the locality-aware allocation can allocate the block from the zone 110-1 that is more closely associated with the compute node 102-1.

[0029] In some examples, the locality-aware allocation interface 112 can be part of a library of methods (a "method" can refer to machine-readable instructions) that can be called by processes to allocate and de-allocate portions of the memory region 106. An example of such a library is discussed in connection with Figs. 3A-3C further below. A method to allocate a block of the shared memory region 106 can be a malloc() method, while a method to de-allocate a block of the shared memory region 106 can be a free() method. The malloc() method can allocate a block of a specified size. In response to invocation of the malloc() method, the requested

memory block of the specified size is allocated for the requesting process, and a pointer that refers to the allocated memory block can be returned to the process for use by the process in accessing the allocated memory block. This pointer can also be communicated from the allocating process to a second process, for the second process to use the pointer in accessing the allocated memory block.

[0030] The free() method, when invoked by a process, frees a memory block referred to by a specified pointer. In further examples, another method can be used to perform bulk de-allocation of blocks (discussed further below).

[0031] In other examples, instead of using the malloc() method and free() method, the locality-aware allocation interface 112 can provide other routines or functions that can be invoked by a process to perform allocation of a memory portion or de-allocation of a memory portion, among other tasks.

[0032] Fig. 2 is a flow diagram of an example flow 200 that can be performed in accordance with some implementations. In some examples, the flow 200 can be performed by the distributed system 100 that includes the compute nodes 102-1 to 102-4 and the memory manager 108, or by a different system that includes a computer or an arrangement of multiple computers. The flow 200 includes executing (at 202) multiple processes on multiple compute nodes, such as the compute nodes 102-1 to 102-4 of Fig. 1.

[0033] The flow 200 further includes allocating (at 204), by a first process of the multiple processes, a first zone in a shared memory region (*e.g.* the shared memory region 106), where the shared memory region is partitionable into multiple zones, and where the first zone is owned by the first process and is accessible by the multiple processes. For example, the allocating of the first zone can be performed in response to the first process calling the locality-aware allocation interface 112 to perform an allocation from the shared memory region 106. The allocating of the first zone includes selecting (at 206) the first zone from the multiple zones based on the specified proximity of the first zone to a given compute node of the compute nodes.

[0034] Figs. 3A-3C illustrate an example relating to use of a shared memory region (such as the shared memory region 106 of Fig. 1). In the example according to Figs. 3A-3C, the shared memory region is referred to as a shared heap. Two user processes 302-A and 302-B are depicted, where each user process corresponds to an executed instance of application code 304-A and 304-B, respectively. Although Figs. 3A-3C show an example in which the processes that can allocate and use portions of a shared memory region are user processes, it is noted that in other examples, other types of processes can allocate and use portions of a shared memory region.

[0035] A user process refers to a process generated due to execution of an application or other program code in the user space of a system. The user space of a system is contrasted to a kernel space of an operating system, where the kernel space is reserved for running code of the kernel of an operating system.

[0036] In some examples, a shared heap is backed by (i.e. stored in) a shared heap file 306 that is divided into multiple zones (zone 0, zone 1, zone 2, and zone 3 shown in Fig. 3A). In other examples, the shared heap can be implemented with multiple shared heap files (e.g. one shared heap file per zone or each shared heap file including multiple zones). The shared heap file 306 is stored in a shared file system, where the shared file system is stored in memory. The shared heap includes (1) metadata describing which regions of the heap are allocated and which are free (for allocation), and (2) data stored by the user in the allocated memory regions.

[0037] A user process can open a shared heap by making a call 308 to a heap library 308-A. The heap library 308-A is an instance of a heap library that is part of the user process 302-A. Multiple instances of the heap library can be invoked in respective different user processes.

[0038] The heap library can include various methods that can be called by a user process. In some examples, one method is a HEAP_OPEN method, which is invoked by the call 308. The call 308 of the HEAP_OPEN method causes the

shared heap file 306 to be memory mapped (310) as a shared heap 312-A into a virtual address space of the user process 302-A. The shared heap 312-A in the virtual address space of the user process 302-A is an instance of the shared heap file 306.

[0039] In the example of Fig. 3A, the user process 302-B has not yet opened a shared heap in the virtual address space of the user process 302-B.

[0040] Memory-mapping the shared heap file 306 refers to creating (such as by an operating system) a segment (e.g. the shared heap 312-A) in the virtual address space of the caller (in this case the user process 302-A), and then assigning each memory byte of the segment to a byte of the shared heap file 306. The assignment can be done in a direct byte-to-byte fashion, meaning that byte 0 of the segment corresponds to byte 0 of the shared heap file 306, byte 1 of the segment corresponds to byte 1 of the shared heap file 306, and so on. Because the assignment is direct, accessing a memory byte in the segment (e.g. shared heap 312-A) results in directly accessing the corresponding file byte in the shared heap file 306. When multiple user processes open the shared heap, then each of the multiple user processes gets its own segment in its own virtual address space. Thus, different users get different mappings between the segment and the underlying shared heap file 306. Because the shared heap file 306 is shared, the changes made to by different user processes are visible to each other.

[0041] As part of opening the shared heap by the user process 302-A, a handle that identifies the shared heap 312-A is created, where the handle can be used by the user process 302-A to allocate or de-allocate memory from the shared heap 312-A. A handle can refer to any value that can be used to identify the shared heap 312-A.

[0042] In some examples, the handle can carry a generation indication that provides an indication of a respective instance (generation) of the shared heap. In some examples, the generation indication is in the form of a generation identifier (e.g. a generation number such as a global integer) to identify the specified

generation of the shared heap. More specifically, in some examples, a generation includes a collection of zones and blocks allocated during the instantiation of the shared heap. Such allocations and data stored in the allocated zones and blocks survive between heap generations. Thus, blocks allocated and data stored in these blocks during the lifetime of a previous generation can be accessed by later generations.

[0043] Note that each opening of a shared heap by a user process causes a new generation of the shared heap to be created, and thus a new generation identifier is assigned. As an example, the shared heap 312-A that is opened by the user process 302-A can have a generation identifier of 1. If the user process 302-A exits and then re-opens the shared heap, then the generation identifier is incremented to identify a next generation of the shared heap. As a further example, if the user process 302-B later opens the shared heap, the generation identifier is incremented again for the shared heap opened by the user process 302-B.

[0044] To acquire ownership of a zone, the HEAP_OPEN method locks the zone by atomically writing the unique generation identifier. The unique generation identifier can be written into the metadata of the zone.

[0045] After the shared memory 312-A has been opened by the user process 302-A, the user process 302-A can make a call 314 (as shown in Fig. 3B) to allocate a block from the shared heap 312-A. The call 314 to allocate the block from the shared heap 312-A can be a call to a malloc() method in the heap library 308-A. An example of the malloc() method is set forth below:

void* malloc(size_t size, int numa_node_hint).

[0046] The called malloc() method passes in two input parameters: size_t (which specifies the desired size of the block to be allocated), and numa_node_hint (which is a hint that provides an indication of a target locality of the block to be allocated).

[0047] In response to the call 314 to allocate a block, it is determined that zone 0 satisfies the locality indicate by numa_node_hint, and thus zone 0 of the shared

heap 312-A is allocated to the user process 302-A, which has an identifier of 5 in the example. Zone 0 is associated with metadata that indicates that zone 0 is owned by a user process with identifier 5. In addition, the block of size size_t is allocated in zone 0 in response to the call 314. Once the block is allocated, a pointer to the block can be returned to the user process 302-A, for use in accessing data in the block.

[0048] In some examples, the pointer can include a relocatable base-relative smart pointer for logically addressing (naming) an allocated block. In contrast to a virtual address, the base-relative smart pointer our pointers is relocatable, which means that the shared heap does not have to be memory mapped into the same virtual address range in each process. Instead, a logical address represents an offset relative to the base of the virtual address region that maps the shared heap so that the heap can be memory mapped to different virtual address regions in the processes.

[0049] Although not shown, the user process 302-A can invoke additional calls to allocate further blocks from the shared heap 312-A. For each such additional call, if there is sufficient space in zone 0 to allocate the further block of memory, then the block can be allocated from zone 0. However, if there is insufficient space in zone 0 to satisfy the request to allocate the further block, then a new zone can be allocated to the user process 302-A, and this new zone can be associated with owner 5.

[0050] In a different example, an additional call to allocate a further block can include a different numa_node_hint value, which can cause a different zone to be allocated to the user process 302-A even though zone 0 (already allocated to the user process 302-A) may have sufficient space for the further block.

[0051] Fig. 3C shows an example where the user process 302-B has issued a call 315 to open the shared heap (represented as 312-B). In response to the call 315 to open the shared heap, a memory mapping 316 is performed between the shared heap file 306 and the shared heap 312-B in the virtual address space of the user process 302-B.

[0052] In response to the call 315 to open the shared heap 312-B, the generation identifier associated with the shared heap file 306 can be incremented to identify a new generation of the shared heap. In the example of Fig. 3C, zone 2 is allocated to the user process 302-B. Zone 2 is associated with metadata that indicates that zone 2 is owned by a user process with identifier 6, which is the identifier of the user process 302-B in the example.

[0053] Ownership of a zone is transient in that the heap library relinquishes ownership of a zone as soon as the owning user process exits.

[0054] Fig. 4 is a flow diagram of an example flow 400 according to further implementations. In some examples, the flow 400 can be performed by the distributed system 100 of Fig. 1, or by a different system that includes a computer or an arrangement of multiple computers. The flow 400 includes concurrently allocating (at 402), by multiple processes executing on multiple compute nodes, respective zones of a shared memory region, where a first zone of the zones allocated by a first process is owned by the first zone and is accessible by the multiple processes, and a second zone allocated by a second process is owned by the second zone and is accessible by the multiple processes. The allocating of the respective zones can be performed in response to calls by the processes to the locality-aware allocation interface 112, for example.

[0055] The flow 400 further includes performing (at 404), by a memory manager (*e.g.* memory manager 108 in Fig. 1), bulk de-allocation of blocks of the shared memory region that are associated with a given generation of the shared memory region. Bulk de-allocation can be performed when a user process invokes a call to a bulk-free() method that is part of the heap library (*e.g.* 308-A or 308-B illustrated in Fig. 3A). Performing bulk de-allocation can be more efficient when attempting to free multiple blocks, as compared to having to make multiple calls to free individual blocks.

[0056] Bulk de-allocation involves the de-allocation of multiple blocks that are associated with a given generation, as identified by a generation identifier. Bulk de-

allocation can be performed when a shared memory region is opened by a process, which can be performed when a process is started (e.g. initially started or restarted due to a previous crash of the process). In examples where the bulk de-allocation is invoked when re-opening a shared memory region after a crash has occurred, the bulk de-allocation can be part of a recovery procedure to recover from the crash. In other examples, the bulk de-allocation can be performed at other times or in response to other events.

[0057] The memory manager can bulk-free blocks by zeroing all zone metadata associated with zones associated with a given generation identifier, and then the ownership of such zones can be released by zeroing the zone blocks. Zeroing zone metadata refers to deleting or otherwise rendering the zone metadata invalid such that the zone metadata can be over-written. Zeroing a block refers to deleting the content of the block or otherwise rendering the block content invalid so that it can be over-written with other data.

[0058] In other examples, instead of performing bulk de-allocation, a call to free a specific block can be made by a process, such as by using a free() method in the heap library. To free a block, the free() method can locate the zone containing the block, and if the zone is owned by the process that invoked the request to free the block, then the block can be freed by the memory manager. Otherwise, de-allocation fails, and the user process would have to direct the call to free the block to the user process owning the zone.

[0059] Because multiple users share the heap, it is possible that a first process allocates a block, passes a pointer to the allocated block to a second process, and then the second process attempts to free (or de-allocate) the allocated block after the second process is done with processing the content of the allocated block. The second process does not own this block, so any attempt by the second process to de-allocate the block will fail. Instead the second process has to ask, such as by using an inter-process communication or other communication mechanism, the first (owner) process to de-allocate the block.

[0060] Fig. 5 is a block diagram of an example system according to further implementations. The system includes a shared memory region 502, which can be similar to the shared memory region 106 of Fig. 1. The system 500 further includes multiple compute nodes 504-1 and 504-2, and respective processes 506-1 and 506-2 are executed by the respective compute nodes 504-1 and 504-2. A first process of the multiple processes (506-1 and 506-2) can use a locality-aware allocation interface 508 to perform locality-aware allocation of a first zone of the shared memory region 502, where the first zone is owned by the first process and is accessible by the first process 506-1 and the second process 506-2. The locality-aware allocation of the first zone is to allocate the first zone that has a specified proximity to a given compute node of the compute nodes 504-1 and 504-2.

[0061] Fig. 6 is a block diagram of a non-transitory machine-readable or computer-readable storage medium 600 that stores instructions that upon execution cause a system to perform various tasks. The instructions include process execution instruction 602 to execute multiple processes on multiple compute nodes. The instructions further include zone allocation instructions 604 to allocate, by a first process of the multiple processes, a first zone in a shared memory region that is partitioned into multiple zones, where the first zone is owned by the first process and is accessible by the multiple processes. The zone allocation instructions can select the first zone from multiple zones based on a specified proximity of the first zone to a given compute node of the compute nodes.

[0062] The storage medium 600 can include one or multiple different forms of memory including semiconductor memory devices such as dynamic or static random access memories (DRAMs or SRAMs), erasable and programmable read-only memories (EPROMs), electrically erasable and programmable read-only memories (EEPROMs) and flash memories; magnetic disks such as fixed, floppy and removable disks; other magnetic media including tape; optical media such as compact disks (CDs) or digital video disks (DVDs); or other types of storage devices. Note that the instructions discussed above can be provided on one computer-readable or machine-readable storage medium, or alternatively, can be provided on

multiple computer-readable or machine-readable storage media distributed in a large system having possibly plural nodes. Such computer-readable or machine-readable storage medium or media is (are) considered to be part of an article (or article of manufacture). An article or article of manufacture can refer to any manufactured single component or multiple components. The storage medium or media can be located either in the machine running the machine-readable instructions, or located at a remote site from which machine-readable instructions can be downloaded over a network for execution.

[0063] In the foregoing description, numerous details are set forth to provide an understanding of the subject disclosed herein. However, implementations may be practiced without some of these details. Other implementations may include modifications and variations from the details discussed above. It is intended that the appended claims cover such modifications and variations.

What is claimed is:

1. A system comprising:
a shared memory region;
a plurality of compute nodes;
a plurality of processes executable in the plurality of compute nodes, a first process of the plurality of processes to use an allocation interface to perform locality-aware allocation of a first zone of the shared memory region, the first zone owned by the first process and accessible by the first process and a second process of the plurality of processes,
the locality-aware allocation of the first zone to allocate the first zone that has a specified proximity to a given compute node of the plurality of compute nodes.

2. The system of claim 1, wherein the shared memory region is accessible by the plurality of processes according to a non-uniform memory access (NUMA) arrangement.

3. The system of claim 1, wherein the first process is to provide an indication of a target locality with respect to the plurality of compute nodes, and the locality-aware allocation of the first zone is to allocate the first zone that has the specified proximity, based on the indication, to the given compute node.

4. The system of claim 1, wherein the locality-aware allocation of the first zone is to allocate the first zone that has the specified proximity to the given compute node that is more closely associated with a compute node in which the first process is executed than another compute node.

5. The system of claim 1, wherein another process of the plurality of processes is to use the allocation interface to perform locality-aware allocation of a second zone concurrently with the locality-aware allocation of the first zone by the first process.

6. The system of claim 5, wherein the first zone and the second zone have a common size.

7. The system of claim 1, wherein the shared memory region is responsive to a request to perform bulk de-allocation of blocks in the shared memory by de-allocating the blocks that share a given generation, at least a subset of the blocks being part of the first zone.

8. The system of claim 7, wherein an indication of the given generation is associated with the blocks responsive to opening an instance of the shared memory region, and wherein successive openings of respective instances of the shared memory cause different generation indications to be associated with blocks of the respective instances of the shared memory region.

9. A method comprising:

concurrently allocating, by a plurality of processes executing on a plurality of compute nodes, respective zones of a shared memory region, wherein a first zone of the zones allocated by a first process of the plurality of processes is owned by the first process and is accessible by the plurality of processes, and a second zone of the zones allocated by a second process of the plurality of processes is owned by the second process and is accessible by the plurality of processes; and

performing, by a memory manager, bulk de-allocation of blocks of the shared memory region that are associated with a given generation of the shared memory region.

10. The method of claim 9, wherein the bulk de-allocation of blocks is performed in response to opening of the shared memory region.

11. The method of claim 9, further comprising performing, by the memory manager, de-allocation of an individual block of the blocks in response to a request from a process of the plurality of processes.

12. The method of claim 9, further comprising performing, by the plurality of processes, non-uniform memory access (NUMA) of the shared memory region.

13. The method of claim 12, wherein the allocating of the respective zones comprises locality-aware allocating of the respective zones, wherein a locality-aware allocating of a first zone of the respective zones comprises allocating the first zone having a specified proximity to a given compute node of the plurality of compute nodes.

14. A non-transitory machine-readable storage medium storing instructions that upon execution cause a system to:

execute a plurality of processes on a plurality of compute nodes;

allocate, by a first process of the plurality of processes, a first zone in a shared memory region that is partitioned into a plurality of zones, the first zone owned by the first process and accessible by the plurality of processes,

wherein the allocating of the first zone comprises selecting the first zone from the plurality of zones based on a specified proximity of the first zone to a given compute node of the compute nodes.

15. The non-transitory machine-readable storage medium of claim 14, wherein the selecting is responsive to information provided with a request to allocate a block of the shared memory region

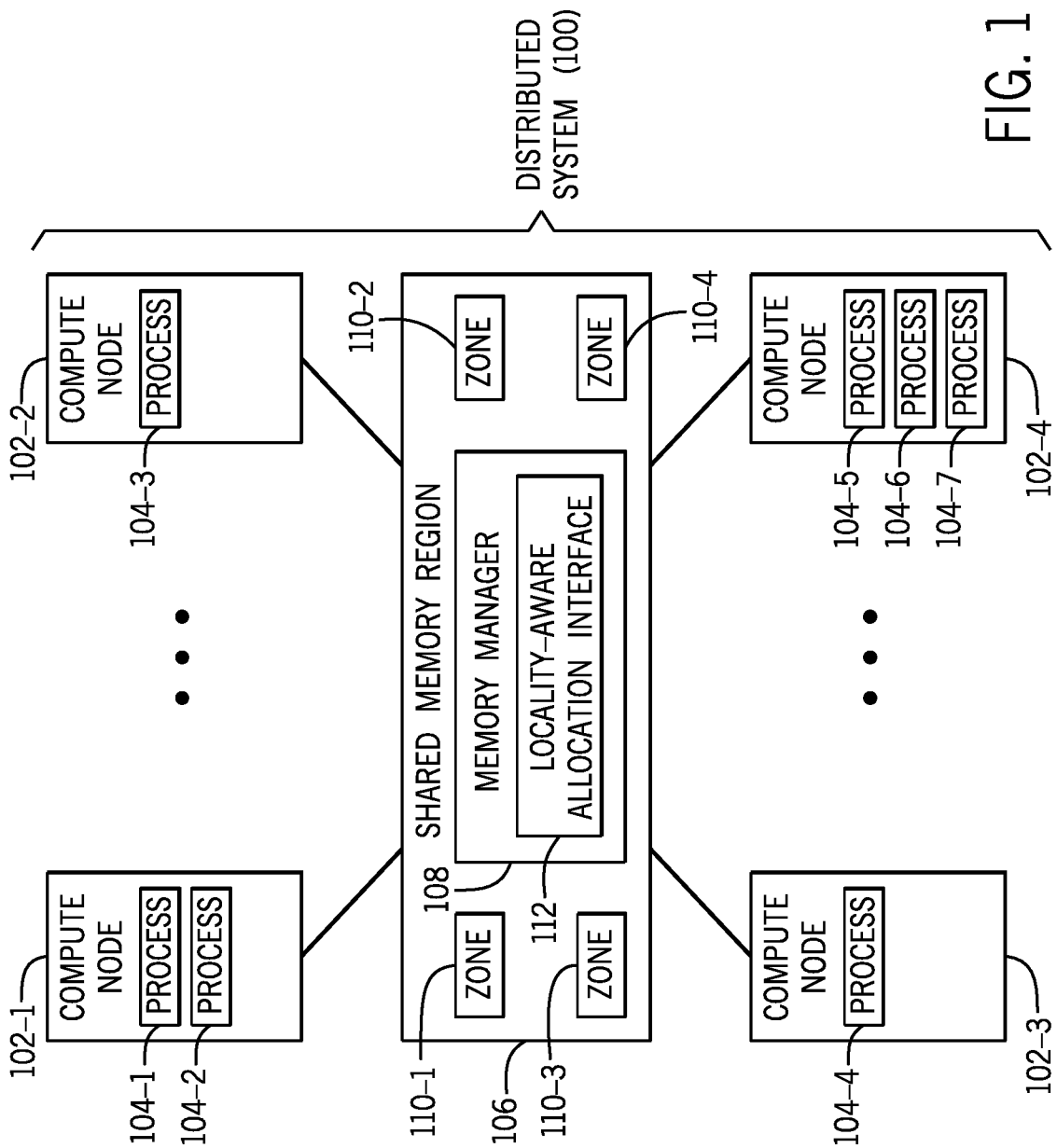
DISTRIBUTED SYSTEM (100)
SHARED MEMORY REGION
MEMORY MANAGER
LOCALITY-AWARE ALLOCATION INTERFACE
ZONE
ZONE
ZONE
ZONE
COMPUTE NODE
PROCESS
102-1
102-2
102-3
102-4
104-1
104-2
104-3
104-4
104-5
104-6
104-7
106
108
110-1
110-2
110-3
110-4
112

FIG. 1

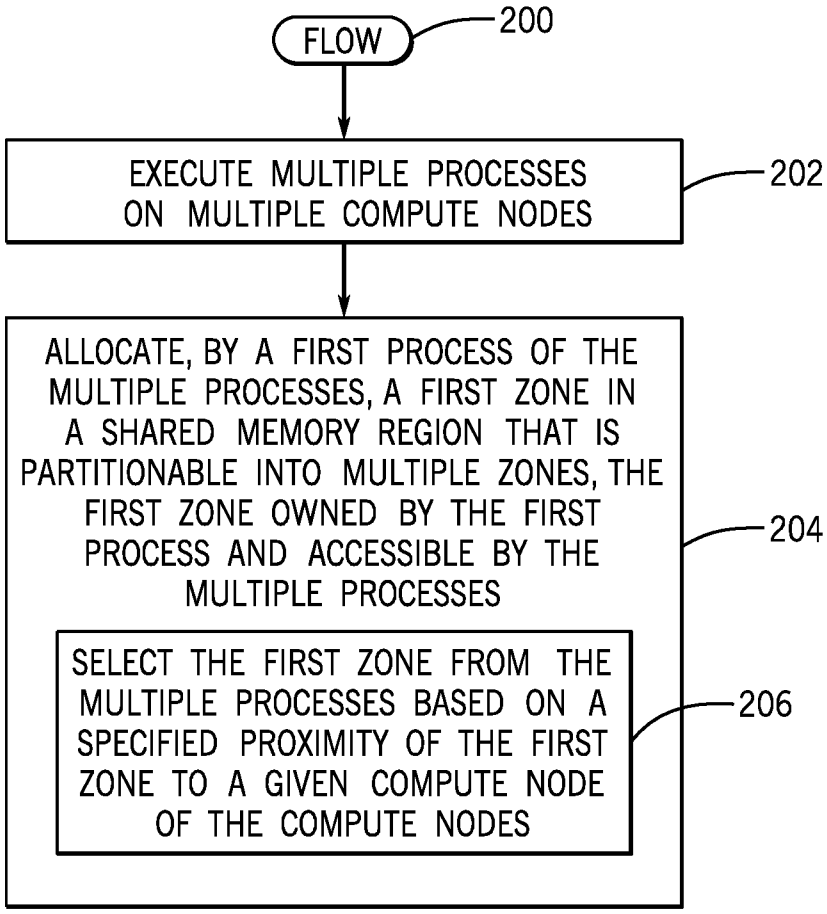
FLOW
200
EXECUTE MULTIPLE PROCESSES ON MULTIPLE COMPUTE NODES
202
ALLOCATE, BY A FIRST PROCESS OF THE MULTIPLE PROCESSES, A FIRST ZONE IN A SHARED MEMORY REGION THAT IS PARTITIONABLE INTO MULTIPLE ZONES, THE FIRST ZONE OWNED BY THE FIRST PROCESS AND ACCESSIBLE BY THE MULTIPLE PROCESSES
204
SELECT THE FIRST ZONE FROM THE MULTIPLE PROCESSES BASED ON A SPECIFIED PROXIMITY OF THE FIRST ZONE TO A GIVEN COMPUTE NODE OF THE COMPUTE NODES
206

FIG. 2

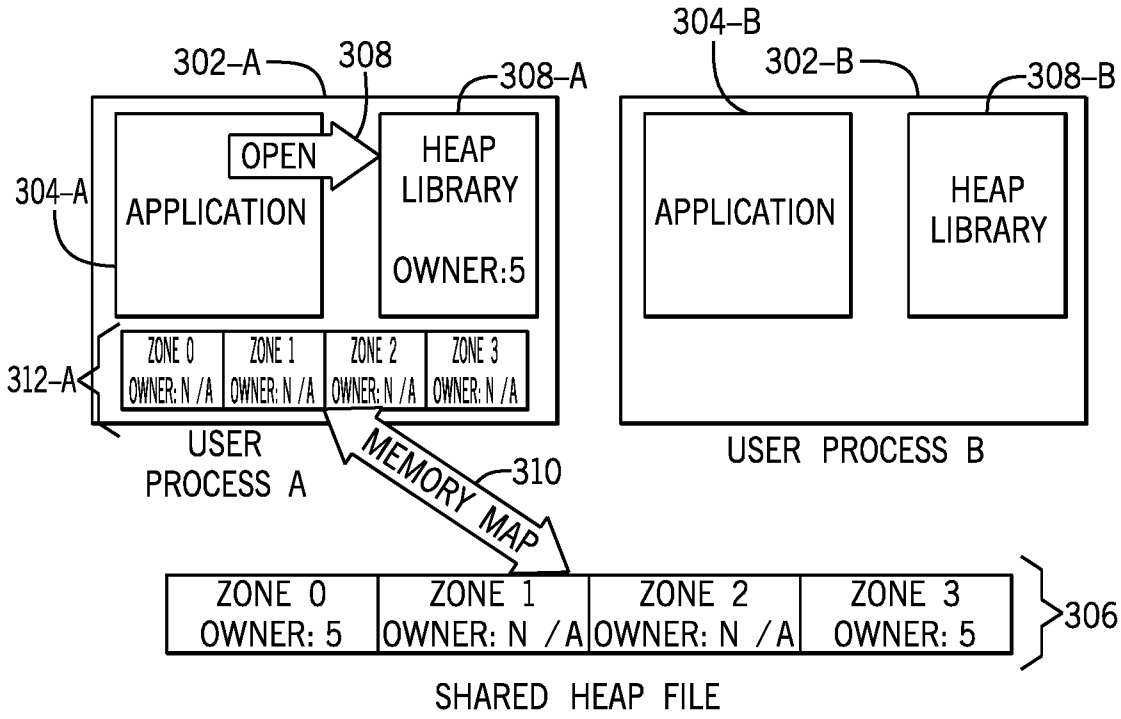
302-A
308
308-A
304-B
302-B
308-B
OPEN
HEAP LIBRARY
OWNER:5
304-A
APPLICATION
APPLICATION
HEAP LIBRARY
ZONE 0 OWNER: N /A
ZONE 1 OWNER: N /A
ZONE 2 OWNER: N /A
ZONE 3 OWNER: N /A
312-A
USER PROCESS A
USER PROCESS B
MEMORY MAP
310
ZONE 0 OWNER: 5
ZONE 1 OWNER: N /A
ZONE 2 OWNER: N /A
ZONE 3 OWNER: 5
306
SHARED HEAP FILE

FIG. 3A

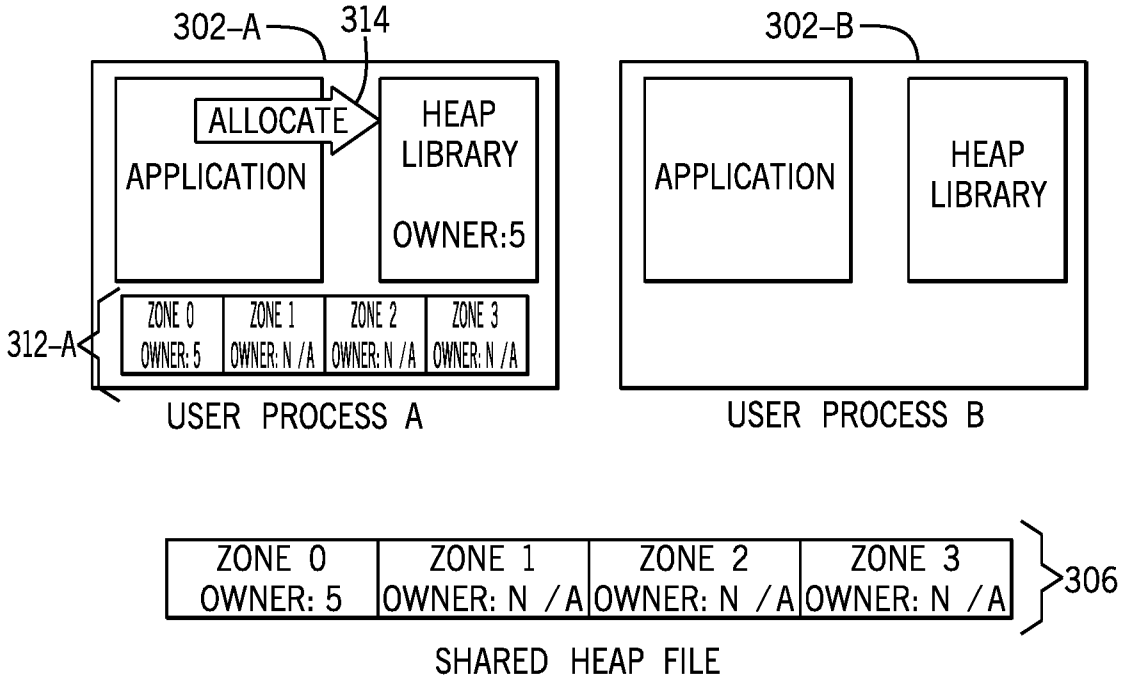
302-A
314
302-B
ALLOCATE
HEAP LIBRARY
APPLICATION
OWNER:5
APPLICATION
HEAP LIBRARY
ZONE 0 OWNER: 5
ZONE 1 OWNER: N /A
ZONE 2 OWNER: N /A
ZONE 3 OWNER: N /A
312-A
USER PROCESS A
USER PROCESS B
ZONE 0 OWNER: 5
ZONE 1 OWNER: N /A
ZONE 2 OWNER: N /A
ZONE 3 OWNER: N /A
306
SHARED HEAP FILE

FIG. 3B

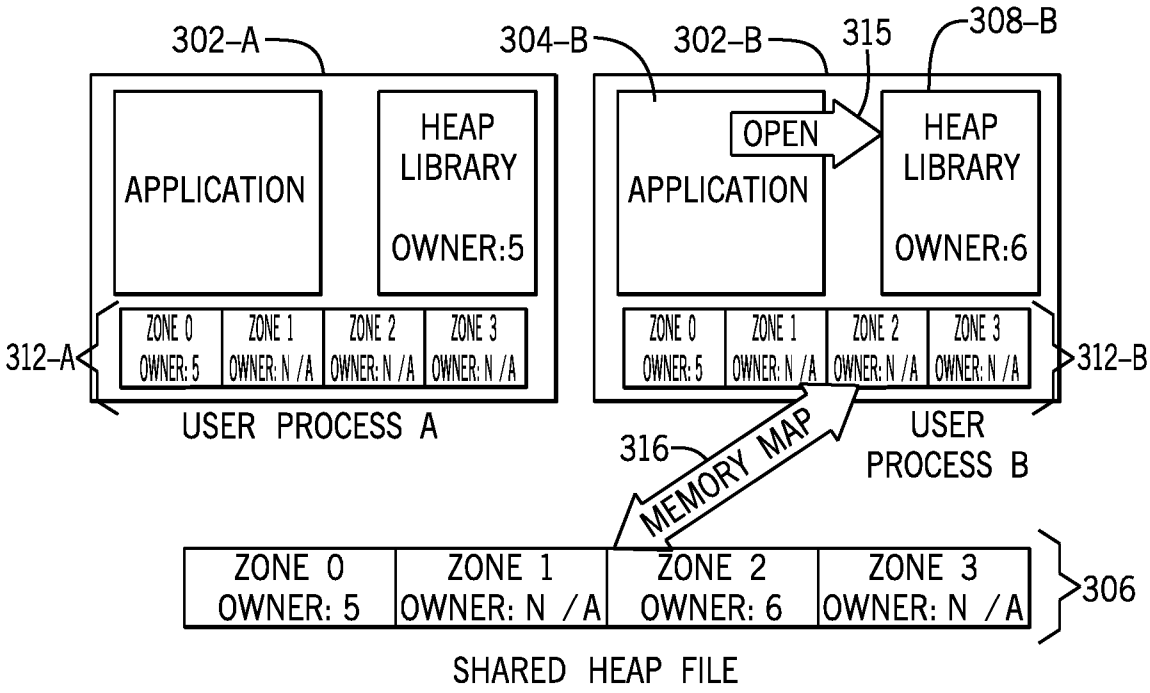
302-A
304-B
302-B
315
308-B
APPLICATION
HEAP LIBRARY
OWNER:5
OPEN
APPLICATION
HEAP LIBRARY
OWNER:6
ZONE 0 OWNER: 5
ZONE 1 OWNER: N /A
ZONE 2 OWNER: N /A
ZONE 3 OWNER: N /A
312-A
312-B
USER PROCESS A
USER PROCESS B
316
MEMORY MAP
ZONE 0 OWNER: 5
ZONE 1 OWNER: N /A
ZONE 2 OWNER: 6
ZONE 3 OWNER: N /A
306
SHARED HEAP FILE

FIG. 3C

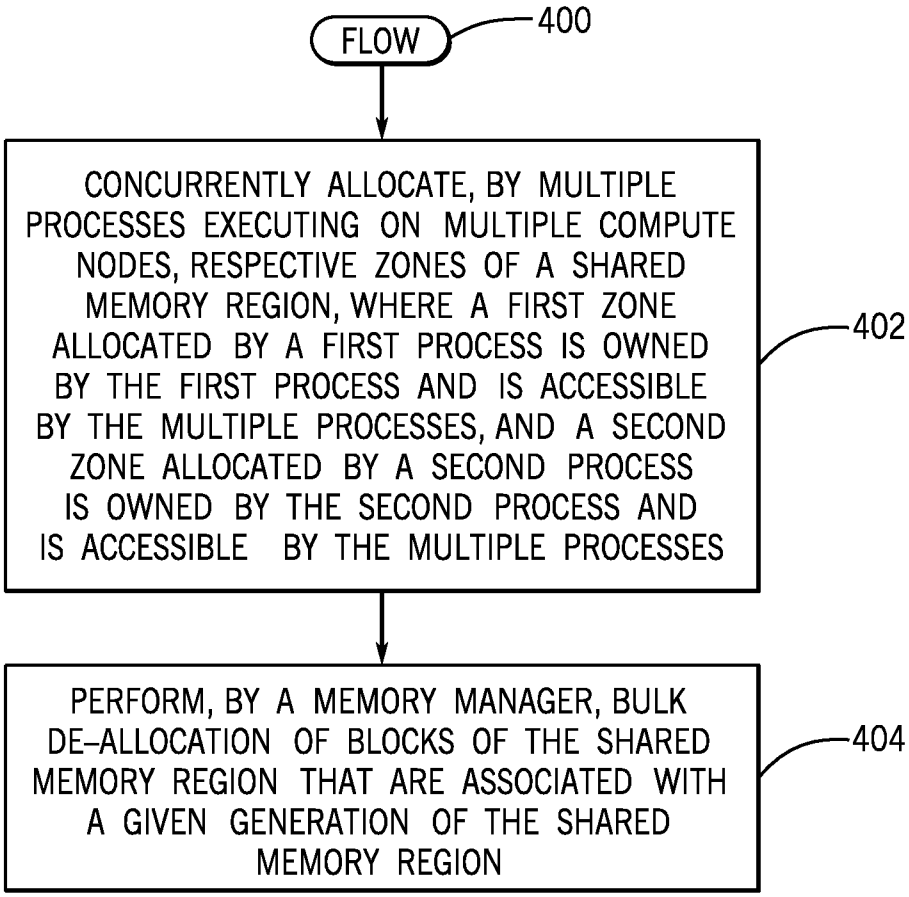
FLOW
400
CONCURRENTLY ALLOCATE, BY MULTIPLE PROCESSES EXECUTING ON MULTIPLE COMPUTE NODES, RESPECTIVE ZONES OF A SHARED MEMORY REGION, WHERE A FIRST ZONE ALLOCATED BY A FIRST PROCESS IS OWNED BY THE FIRST PROCESS AND IS ACCESSIBLE BY THE MULTIPLE PROCESSES, AND A SECOND ZONE ALLOCATED BY A SECOND PROCESS IS OWNED BY THE SECOND PROCESS AND IS ACCESSIBLE BY THE MULTIPLE PROCESSES
402
PERFORM, BY A MEMORY MANAGER, BULK DE-ALLOCATION OF BLOCKS OF THE SHARED MEMORY REGION THAT ARE ASSOCIATED WITH A GIVEN GENERATION OF THE SHARED MEMORY REGION
404

FIG. 4

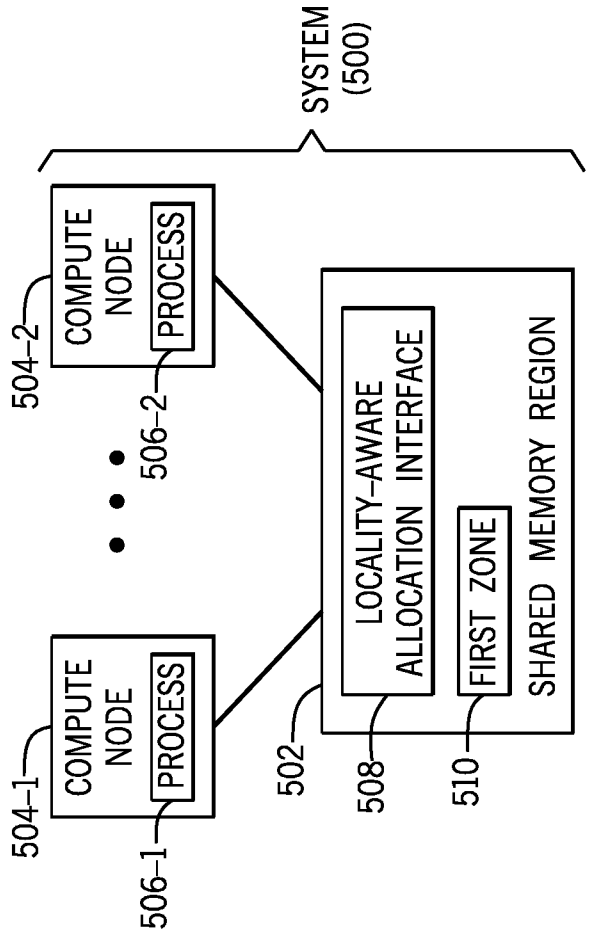
SYSTEM
(500)
504-2
COMPUTE NODE
PROCESS
506-2
504-1
COMPUTE NODE
PROCESS
506-1
502
LOCALITY-AWARE ALLOCATION INTERFACE
508
FIRST ZONE
510
SHARED MEMORY REGION

FIG. 5

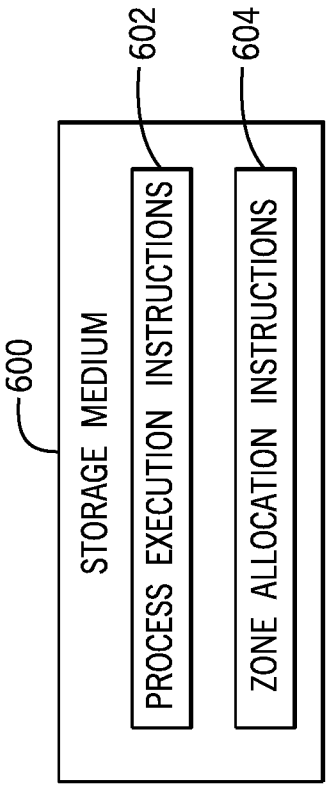
600
STORAGE MEDIUM
602
PROCESS EXECUTION INSTRUCTIONS
604
ZONE ALLOCATION INSTRUCTIONS

FIG. 6

**INTERNATIONAL SEARCH REPORT**

International application No.
**PCT/US2016/018227**

**A. CLASSIFICATION OF SUBJECT MATTER**

**G06F 15/173(2006.01)i, 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 15/173; G06F 12/02; G06F 12/00; G06F 15/76; G06F 15/167; G06F 12/08

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: shared memory, region, zone, proximity, allocation, NUMA (non-uniform memory access), instance, and similar terms.

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | US 2003-0093628 A1 (EUGENE P. MATTER et al.) 15 May 2003<br>See paragraphs [0014]-[0029]; claims 18, 22, 30; and figure 1. | 9-11 |
| Y | | 1-8,12-15 |
| Y | US 2008-0195719 A1 (YUGUANG WU et al.) 14 August 2008<br>See paragraphs [0032], [0050]; claim 2; and figure 4. | 1-8,12-15 |
| A | US 2016-0034398 A1 (DIRK WENDEL et al.) 04 February 2016<br>See paragraphs [0032]-[0033]; claim 1; and figures 1-2. | 1-15 |
| A | US 2010-0095089 A1 (JIN-HYOUNG KWON) 15 April 2010<br>See paragraphs [0039]-[0048]; claim 1; and figure 3. | 1-15 |
| A | US 2010-0161929 A1 (GEORGE NATION et al.) 24 June 2010<br>See paragraphs [0050]-[0055]; claim 1; and figures 3-4. | 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 |
|---|---|
| 09 November 2016 (09.11.2016) | **10 November 2016 (10.11.2016)** |


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
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/US2016/018227**

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| US 2003-0093628 A1 | 15/05/2003 | AU 2002-352689 A1 | 26/05/2003 |
| | | CN 1732444 A | 08/02/2006 |
| | | EP 1446721 A2 | 18/08/2004 |
| | | MY 136470 A | 31/10/2008 |
| | | US 7380085 B2 | 27/05/2008 |
| | | WO 03-042834 A2 | 22/05/2003 |
| US 2008-0195719 A1 | 14/08/2008 | US 9185160 B2 | 10/11/2015 |
| US 2016-0034398 A1 | 04/02/2016 | None | |
| US 2010-0095089 A1 | 15/04/2010 | KR 10-2010-0041309 A | 22/04/2010 |
| US 2010-0161929 A1 | 24/06/2010 | None | |