(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 2016/122585 A1

(51) International Patent Classification:
G11C 7/10 (2006.01)

(21) International Application Number:
PCT/US2015/013748

(22) International Filing Date:
30 January 2015 (30.01.2015)

(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: **GERBER, Simon**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **MILOJICIC, Dejan S.**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US).

(74) Agents: **PAGAR, Preetam B.** 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))*

[Continued on next page]

(54) Title: MODIFYING CHARACTERISTICS OF A MEMORY SEGMENT

(57) Abstract: A method of modifying characteristics of an existing memory segment in a virtual address space for a running software application. The existing memory segment includes an identification of a first backing memory region in physical memory for storing contents of the existing memory segment. The method includes tracking memory access statistics of the running software application, identifying new characteristics for the existing memory segment based on the tracked memory access statistics, and generating an updated memory segment that includes the identified new characteristics.

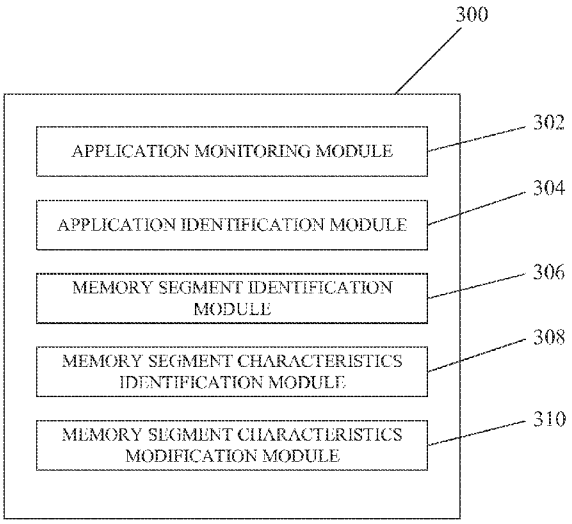

Fig. 3

WO 2016/122585 A1

— *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))*

MODIFYING CHARACTERISTICS OF A MEMORY SEGMENT

Background

[0001] With the introduction of systems with large amounts of physical memory with different characteristics, allocation of memory to applications and the operating system can become complex. Application programs are becoming complex, with large address spaces and with varying access patterns across the process address spaces.

Brief Description of the Drawings

[0002] Figure 1 is a diagram illustrating a computing environment suitable for implementing aspects of a memory segment configuration system according to one example.

[0003] Figure 2 is a diagram illustrating an address space and segment list of an application according to one example.

[0004] Figure 3 is a block diagram illustrating modules of a memory segment configuration system according to one example.

[0005] Figure 4 is a flow diagram illustrating a method of configuring memory segments of a running application according to one example.

[0006] Figure 5 is a flow diagram illustrating a method of configuring memory segments of a running application according to another example.

[0007] Figure 6 is a flow diagram illustrating a method of configuring memory segments of a running application using a database of application types according to one example.

[0008] Figure 7 is a flow diagram of a method for gathering memory access statistics using a PEBS (precise-event-based sampling) infrastructure according to one example.

[0009] Figure 8 is a flow diagram of a method for gathering memory access statistics by monitoring page faults according to one example.

[0010] Figure 9 is a flow diagram illustrating a method of dynamically modifying characteristics of memory segments of a running application according to one example.

[0011] Figure 10 is a flow diagram illustrating a method of dynamically modifying characteristics of memory segments of a running application according to another example.

[0012] Figure 11 is a flow diagram illustrating a method of dynamically modifying characteristics of memory segments of a running application according to yet another example.

[0013] Figure 12 is a flow diagram illustrating a method of dynamically modifying characteristics of memory segments of a running application, which combines aspects of the methods shown in Figures 9-11, according to one example.

Detailed Description

[0014] In the following detailed description, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific examples in which the disclosure may be practiced. It is to be understood that other examples may be utilized and structural or logical changes may be made without departing from the scope of the present disclosure. The following detailed description, therefore, is not to be taken in a limiting sense, and the scope of the present disclosure is defined by the appended claims. It is to be understood that features of the various examples

described herein may be combined, in part or whole, with each other, unless specifically noted otherwise.

[0015] One example is directed to the dynamic adaptation of an operating system address space, and more particularly to dynamically changing an operating system address space with memory segments with different characteristics.

[0016] With the introduction of systems with large amounts of physical memory with different characteristics, allocation of memory to applications and the operating system can become complex. Computer processors and memories are becoming increasingly more complex and heterogeneous, enabling segments of memory with different characteristics (e.g., page size, persistence, access times (access latency), bandwidth, etc.). Application programs are becoming complex, with large address spaces and with varying access patterns across the process address spaces. This results in suboptimal performance of applications because it is difficult to tune parts of the address spaces.

[0017] One specific example is directed to a mechanism to dynamically, at run-time, change individual memory segment configurations by determining an improved configuration of the segments from tracked past behavior, and by changing the configurations through copying segments to newly configured segments either dynamically or on demand. Adapting characteristics of segments of an application's address space improves application performance by having individual segments configured to have improved characteristics.

[0018] Figure 1 is a diagram illustrating a computing environment 10 suitable for implementing aspects of a memory segment configuration system according to one example. In the illustrated example, the computing system or computing device 10 includes one or more processing units 12 and system memory 14. Depending on the exact configuration and type of computing device, memory 14 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, Memristive, phase change, and spin transfer torque memory, etc.), or some combination of the two.

[0019] Computing device 10 may also have additional or different features/functionality and additional or different hardware and software. For

example, computing device 10 may also include additional storage (removable and/or non-removable) including, but not limited to, magnetic or optical disks or tape. Such additional storage is illustrated in Figure 1 by removable storage 16 and non-removable storage 18. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any suitable method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Memory 14, removable storage 16 and non-removable storage 18 are all examples of computer storage media (e.g., non-transitory computer-readable storage media storing computer-executable instructions that when executed by at least one processor cause the at least one processor to perform a method). Computer storage media includes RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices. Any such computer storage media may be part of computing device 10.

[0020] The various elements of computing device 10 are communicatively coupled together via one or more communication links 15. Computing device 10 also includes one or more communication connections 24, such as network connections, that allow computing device 10 to communicate with other computers/applications 26. Computing device 10 may also include input device(s) 22, such as keyboard, pointing device (e.g., mouse), pen, voice input device, touch input device, etc. Computing device 10 may also include output device(s) 20, such as a display, speakers, printer, etc.

[0021] Figure 1 and the above discussion are intended to provide a brief general description of a suitable computing environment in which one or more examples may be implemented. It should be understood, however, that handheld, portable, and other computing devices of all kinds are contemplated for use. Figure 1 thus illustrates an example of a suitable computing system environment 10 in which the examples described herein may be implemented, although as made clear above, the computing system environment 10 is one example of a suitable computing environment and is not intended to suggest

any limitation as to the scope of use or functionality of the examples. Neither should the computing environment 10 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the example operating environment 10.

[0022] As shown in Figure 1, a memory segment configuration system 300 is stored in system memory 14. One example of system 300 creates different memory segments of an application's address space with different characteristics, and changes the characteristics of the memory segments based on memory access patterns observed after their creation. In this disclosure, address space regions of an application are referred to as memory segments in the virtual address space. Each memory segment according to one example has the following properties: (1) an address range that the memory segment covers; (2) a segment page size; and (3) identification of a backing memory region. The segment page size is defined by the hardware the system is running on. The backing memory region is a range of physical memory that is used to back the segment's contents. The backing memory region can have different characteristics based on the type of memory it is in. Memory segment configuration system 300 is described in further detail below with reference to Figures 2-4.

[0023] Figure 2 is a diagram illustrating an address space (AS) 202 and segment list 206 of a software application ("Application A") according to one example. As shown in Figure 2, the address space 202 for Application A includes memory segments 204(1)-204(5) (collectively referred to as memory segments 204), and the segment list 206 includes memory segment characteristics 208(1)-208(5) (collectively referred to as memory segment characteristics 208). In the illustrated example, the memory segment characteristics for each memory segment 204 include a base address, a segment size, a page size, and an identification of a backing memory region. Memory segment 204(1) ("segment #2") is a stack for Application A and includes memory segment characteristics 208(3). Memory segment 204(2) ("segment #4") is one of the larger segments and includes memory segment characteristics 208(5). Memory segment 204(3) ("segment #3") is one of the

larger segments and includes memory segment characteristics 208(4). Memory segment 204(4) ("segment #1") is a data section for Application A and includes memory segment characteristics 208(2). Memory segment 204(5) ("segment #0") is a text section for Application A and includes memory segment characteristics 208(1). Memory segments 204(1), 204(3), 204(4), and 204(5) are backed by system RAM 210 as represented by segments 214(1), 214(3), 214(4), and 214(5), respectively, and memory segment 204(2) is backed by system non-volatile memory (NVM) 212, as represented by segment 214(2), and uses larger pages. One example of the memory segment configuration system 300 shown in Figure 2 creates different memory segments, such as memory segments 204, of an application's address space, such as address space 202, with different characteristics, such as the characteristics 208, and changes the characteristics of the memory segments based on memory access patterns observed after their creation.

[0024] Figure 3 is a block diagram illustrating modules of a memory segment configuration system 300 according to one example. System 300 includes an application monitoring module 302, an application identification module 304, a memory segment identification module 306, a memory segment characteristics identification module 308, and a memory segment characteristics modification module 310. In one example, system 300 is implemented in an operating system, and performs dynamic changes in the characteristics of address space segments based on the past knowledge of the use of the segments, such as the sparsity and contiguity of segments, as well as spatial and temporal access patterns. In one implementation, the functionality of system 300 is exposed to the applications themselves, which allows each application to autonomously modify its address space to account for previously observed access patterns. In another implementation, system 300 is implemented in an OS level service that observes the applications' behavior and then adjusts their address space characteristics based on the observed access patterns. It is noted that the functionality of the modules in system 300 can be combined into a single module, or can be combined or broken apart in any other desired manner. Each

module in system 300 according to one example is a combination of hardware and software executing on that hardware to provide a given functionality.

[0025] Figure 4 is a flow diagram illustrating a method 400 of configuring memory segments of a running application according to one example. In one example, system 300 (Figure 3) performs method 400. At 402 in method 400, application monitoring module 302 monitors running applications and gathers memory statistics. At 404, application identification module 304 identifies one or more of the running applications for adjustment of memory segment characteristics. In one example, to identify which applications out of multiple running applications to consider for address space characteristics adjustment, a fitness number is calculated for each of the running applications. One example fitness number for an application, p, can be obtained using the following Equation I:

[0026] Equation I

[0027] $$f(p)=c_0(1-(m_{tlb}(p)/a_{all}(p))) + c_1(1-(l_{nvm}(p)/a_{nvm}(p))) + c_2(1-(s_{mem}(p)/s_{all}(p)))$$

[0028] Where:

[0029] $m_{tlb}(p)$ = Number of TLB misses;

[0030] $a_{all}(p)$ = Number of memory references;

[0031] $l_{nvm}(p)$ = Number of NVM reads;

[0032] $a_{nvm}(p)$ = Number of NVM references;

[0033] $s_{mem}(p)$ = Number of stall cycles on memory references;

[0034] $s_{all}(p)$ = Number of stall cycles; and

[0035] c_0, c_1, c_2 = scaling coefficients.

[0036] As shown in Equation I, the fitness equation includes three terms. The first term (multiplied by coefficient c_0) is the TLB (translation lookaside buffer) miss ratio of application, p, and represents the fitness of the current selection of page sizes, as bad page size selections will lead to higher miss ratios. The second term (multiplied by coefficient c_1) is the NVM read ratio, and represents the fitness of NVM usage. The NVM read ratio will be high if there are mostly NVM reads, which is an indication to consider caching any read-heavy NVM-backed buffers in DRAM. The third term (multiplied by coefficient c_2) is the ratio of CPU stalls caused by memory references, and represents the fitness of the

current memory locality of the memory segments of application, *p*. A high stall ratio indicates that one or more segments of application, *p*, are not local to the computation on those segments.

[0037] The application(s) with the lowest fitness number(s) is/are then selected at 404 in method 400 by application identification module 304 for further consideration. At 406 in method 400, for each application selected at 404, memory segment identification module 306 performs a detailed observation of the memory references for each memory segment of the selected application, and identifies memory segments of the application to be adjusted. In one example, module 306 uses Equation I on the observed per-segment data to generate per-segment fitness numbers, and identifies memory segments at 406 based on the per-segment fitness numbers (e.g., memory segments that have bad fitness numbers). At 408 in method 400, memory segment characteristics identification module 308 identifies improved segment characteristics for the segments identified at 406. In another example, subsequent iterations are performed over the candidate sections only rather than over all applications, and then a periodic switch is performed between the two approaches.

[0038] At 410 in method 400, memory segment characteristics modification module 310 adjusts segment characteristics of memory segments identified at 406 using the improved segment characteristics identified at 408. Example memory segment characteristics that may be adjusted at 410 include page size, Non-Uniform Memory Access (NUMA) region, memory type (e.g. RAM vs. NVM), and NVM locality.

[0039] Examples of dynamic changes in the characteristics of address space segments that are made at 410 in method 400 based on observed access patterns include, but are not limited to, the following: (1) For sparsely populated segments (i.e., most of the segment is not used), one example adjusts the characteristics of the segment to use smaller pages, which allows the segment to be populated in a more fine-grained fashion and improves the performance of the actual populating; (2) For large segments whose access pattern predominantly includes repeated accesses to decently-sized buffers (e.g., accessing elements of a matrix repeatedly), one example adjusts the

characteristics of the segment to use larger pages to improve performance (e.g., increases in speed of 1.4 – 4x for matrix multiplication have been measured); (3) For applications that read from segments that are backed by non-volatile memory repeatedly, one example adjusts the characteristics of the segments to be backed by RAM rather than non-volatile memory to increase performance; and (4) For multi-threaded applications that execute across multiple NUMA domains, one example adjusts the characteristics of the segments such that the data for each thread is located in the local NUMA domain.

[0040] In one example, the selection of improved memory segment characteristics for arbitrary applications is enabled by benchmarking a set of applications using different sets of memory segment characteristics to discover and create a database including a set of application types. The database can be made available to any application in the system using various techniques (e.g., providing the database as a system service or a shared library).

[0041] Figure 5 is a flow diagram illustrating a method 500 of configuring memory segments of a running application according to another example. Methods 400 and 500 can be used to adjust address space characteristics on-line (i.e. while the application is running). At 502, memory access statistics for a running application are gathered for a user-defined amount of time. At 504, an application type for the running application is identified based on the statistics gathered at 502. At 506, improved memory segment characteristics for the identified application type are identified. At 508, the identified improved characteristics are applied to one or more memory segments of the running application. At 510, the method 500 waits for a user-defined timeout period, and then returns to 502.

[0042] Method 500 (due to the fact that it is executed periodically) can deal with applications whose type changes in phases without any modifications as well as adjusting the characteristics of individual memory segments without having to change the characteristics of the whole address space. Another example for applying the improved characteristics is to employ a system where applications provide metadata that includes their application type as described above, or a

list of memory segments and characteristics for selected segments which are applied when the application is launched.

[0043] Figure 6 is a flow diagram illustrating a method 600 of configuring memory segments of a running application using a database of application types according to one example. The method 600 starts at 602. At 604, the method 600 determines whether an auto mapping method for adjusting memory segment characteristics is enabled for the running application. If it is determined at 604 that the auto mapping method is not enabled, the method 600 returns to 604 to repeat the determination. If it is determined at 604 that the auto mapping method is enabled, the method 600 moves to 606 to gather memory access statistics for the running application. At 608, a determination is made whether enough data has been gathered at 606. If it is determined at 608 that enough data has not been gathered, the method 600 returns to 606 to gather additional memory access statistics. If it is determined at 608 that enough data has been gathered, the method 600 moves to 610 to identify an application type for the running application based on the statistics gathered at 606.

[0044] At 614 in method 600, using database 612, memory segment characteristics are selected based on the application type identified at 610. Database 612 according to one example is a database of application types with associated best characteristics for each application type. Database 612 according to one example includes data gathered from benchmarking a representative set of applications covering all application types which are extractable from memory access statistics.

[0045] At 616 in method 600, a conversion method is executed to convert one or more memory segments of the running application from their original configuration with an original set of segment characteristics to a modified configuration with the new segment characteristics selected at 614. At 618, it is determined whether a recheck threshold has been reached. If it is determined at 618 that the recheck threshold has not been reached, the method 600 returns to 618 to repeat the determination. If it is determined at 618 that the recheck

threshold has been reached, the method 600 returns to 606 to repeat the process described above.

[0046] One aspect of some examples described herein involves observing the behavior of a running application to produce statistics regarding the application's access patterns and use of different memory segments. One example implementation of this is to employ hardware performance counters to gather statistics of an application's access patterns to its address space. An example of hardware that supports this type of hardware performance counters is Intel®'s 4th generation Core™ processors, which have a facility called PEBS (precise-event-based sampling) Data Address profiling. The PEBS Data Address profiling enhances the performance counters that count memory accesses by providing a snapshot of the processor state at the point of the memory access. This snapshot includes the accessed address for each recorded memory access.

[0047] Figure 7 is a flow diagram of a method 700 for gathering memory access statistics using a PEBS infrastructure according to one example. In method 700, page size is the attribute that is being improved. At 702 in method 700, a buffer that can be used by the PEBS infrastructure to save the records containing the processor snapshot state is allocated. At 704, two of the programmable performance counters are configured to use PEBS with data address profiling. At 706, the PEBS infrastructure is configured to use the buffer allocated at 702. At 708, a first one of the performance counters is configured to count events of the type MEM_UOPS_RETIRED.ALL_LOADS, and a second one of the performance counters is configured to count events of the type MEM_UOPS_RETIRED.ALL_STORES. At 710, program execution is started (or resumed) for a short, fixed amount of time. At 712, upon expiration of the fixed amount of time, the performance counters are disabled. At 714, all data addresses from PEBS records are collected. At 716, all of the collected data addresses are sorted into buckets according to the 4KiB page the addresses are in. The example method 700 is specific to Intel®'s x86 architecture, and, as mentioned above, is based on a scenario where the page size of segments is being improved. Thus, at 716, all data accesses are sorted according to the

smallest possible pages in the system (x86 64bit architecture allows pages of 4KiB, 2MiB and 1GiB).

[0048] As mentioned above, data accesses are sorted into buckets of the size of the smallest hardware page size in the system (e.g., 4KiB for x86). Technically this is possible because PEBS is reporting "linear addresses" rather than hardware pages for the sampled memory accesses. Thus, accesses on 2MiB pages are artificially divided into 4KiB buckets to see whether a benefit may be obtained from splitting 2MiB pages into 4KiB pages (e.g., if different hot spots are seen in a single 2MiB page).

[0049] At 718, the number of addresses in each bucket is counted, and a map of 4KiB pages to number of accesses of that page is created. Leveraging the information generated by method 700, access statistics for each memory segment of an application's address space can be generated by looking up all pages corresponding to a memory segment in the map created at 718.

[0050] For systems that do not have the facilities to measure performance in hardware, another example uses the methods used in NUMA computer systems to keep the data in memory that is local to the compute. These methods typically involve forcibly unmapping all of an application's pages, and then keeping track of the page faults for each page. Figure 8 is a flow diagram of a method 800 for gathering memory access statistics by monitoring page faults according to one example. At 802, all page table entries are marked as invalid (valid = 0). At 804, the method 800 keeps track of each handled page fault. At 806, the fault handler sets valid = 1 for faulting pages. At 808, after a predetermined (e.g., user-defined) time period, the method 800 returns to 802. Method 800 allows the creation of statistics on segment access patterns without the need for specialized hardware at the cost of some performance. The performance cost stems from the fact that page faults are being artificially introduced into the system to count accesses to different regions of the address space.

[0051] One aspect of some examples described herein involves dynamically converting memory segments of address spaces from one page type to another

page type. A page type as used herein refers to all characteristics of the segment such as page size, persistence, NUMA region and others.

[0052] Figure 9 is a flow diagram illustrating a method 900 of dynamically modifying characteristics of memory segments of a running application according to one example. At 902 in method 900, a segment to be converted is made to be read-only. At 904, the data stored in the segment's initial backing memory region is copied to a new backing memory region with the desired characteristics. At 906, the segment is updated to point to the new backing memory region created at 904. At 908, the method 900 frees up the initial backing memory region.

[0053] During 904 and 906 of method 900, write accesses to the segment may fail and the application may be stalled until the conversion is completed. Method 900 can be modified by recognizing conversions that do not involve the data being copied to a different backing region (e.g., changing the page size of a segment while keeping the other characteristics), and applying a conversion procedure for such segments. The conversion procedure (shown in Figure 10) may have further preconditions that are dictated by the underlying hardware (e.g., how the address space segment and the backing memory region are aligned for commonly available paging memory management units) and may fall back on method 900 if those preconditions are not met.

[0054] Figure 10 is a flow diagram illustrating a method 1000 of dynamically modifying characteristics of memory segments of a running application according to another example. At 1002, preconditions are checked to determine if the method 1000 can be applied. If the preconditions are not satisfied, method 900 (Figure 9) is used. At 1004, the method 1000 creates new segment metadata parallel to application execution. At 1006, the application's segment metadata for the segment under conversion is atomically switched to the segment created at 1004. An advantage of method 1000 is that it should not stall the application even if it writes to the segment during the conversion because the data is not copied.

[0055] Another example of dynamically modifying characteristics of memory segments of a running application employs copy-on-reference mechanisms

when changing segment characteristics of sparse segments (Figure 11). Figure 11 is a flow diagram illustrating a method 1100 of dynamically modifying characteristics of memory segments of a running application according to yet another example. At 1102, segment density of a segment to be modified is checked to determine if the segment density is above a given threshold. If the segment density is above the threshold, method 900 (Figure 9) is used. If the segment density is not above the threshold, the method 1100 continues. At 1104, a new backing region for the segment is created with the desired characteristics. At 1106, copy-on-reference mappings from old to new backing regions are created using a copy-on-reference method. At 1108, the method 1100 atomically updates the segment to use the new mappings. At 1110, after all data has been copied from the old backing region to the new backing region, the old backing region is freed. Method 1100 could also employ prefetching techniques to copy the data in the segment in parallel to application execution.

[0056] Figure 12 is a flow diagram illustrating a method 1200 of dynamically modifying characteristics of memory segments of a running application, which combines aspects of methods 900, 1000, and 1100, according to one example. The method 1200 starts at 1202. At 1204, preconditions for no-copy modification are checked. A “no-copy” modification is a modification that does not involve the data in a segment being copied to a different backing memory region (e.g., splitting a 2MiB page into 512 4KiB pages). If the preconditions checked at 1204 are met, the method 1200 moves to 1208 where metadata with new parameters are created at 1208. At 1210, the segment is atomically updated to use the new metadata, and the method 1200 moves to 1230, which indicates the end of the method 1200. If the preconditions checked at 1204 are not met, the method 1200 moves to 1206, where the density of the segment is checked (e.g., the segment is checked to determine if it is dense or sparse).

[0057] If it is determined at 1206 that the segment is dense, the method 1200 moves to 1212 where the segment is made to be read-only. At 1214, a backing region with new parameters is created. At 1216, data from the old backing region is copied to the new backing region. At 1218, the segment is atomically updated to point to the new backing region. At 1220, the old backing region is

freed, and the method 1200 moves to 1230, which indicates the end of the method 1200.

[0058] If it is determined at 1206 that the segment is sparse, the method 1200 moves to 1222 where a new backing region with the desired properties is created. At 1224, copy-on-reference mappings between the old backing region and the new backing region are created. At 1226, the segment is atomically updated to use the copy-on-reference mappings created at 1224. At 1228, it is determined whether all of the data has been copied from the old backing region to the new backing region. If it is determined at 1228 that all of the data has not been copied, the method 1200 repeats the check at 1228. If it is determined at 1228 that all of the data has been copied, the method 1200 moves to 1220 to free the old backing region, and the method 1200 ends at 1230.

[0059] One example of the present disclosure is directed to a method of modifying characteristics of an existing memory segment in a virtual address space for a running software application. The existing memory segment includes an identification of a first backing memory region in physical memory for storing contents of the existing memory segment. The method includes tracking memory access statistics of the running software application, identifying new characteristics for the existing memory segment based on the tracked memory access statistics, and generating an updated memory segment that includes the identified new characteristics.

[0060] Another example of the present disclosure is directed to a memory segment configuration system. The system includes an application monitoring module to track memory access statistics of a running software application, wherein the running software application comprises a virtual address space with a first memory segment. The system also includes a memory segment characteristics identification module to identify new characteristics for the first memory segment based on the tracked memory access statistics, and a memory segment characteristics modification module to generate an updated memory segment that includes the identified new characteristics.

[0061] Yet another example of the present disclosure is directed to a non-transitory computer-readable storage medium storing computer-executable

instructions that when executed by at least one processor cause the at least one processor to perform a method. The method includes tracking memory access statistics of the running software application, wherein the running software application comprises a virtual address space with a first memory segment, and wherein the first memory segment includes an initial set of characteristics including an initial page size and an initial identification of a backing memory region. The method includes modifying at least one of the initial page size and the initial identification of a backing memory region of the first memory segment based on the tracked memory access statistics.

[0062] Examples described herein provide the following: (1) improved flexibility in choosing address space characteristics to best match application behavior; (2) the ability to choose new address space characteristics during application runtime based on previous memory access patterns; and (3) improved support for choosing emerging memory characteristics. Features of some examples described herein include the following: (1) the ability to statically and dynamically decide on the characteristics of each segment based on the past knowledge of the use of the segment, such as sparsity/contiguity of segments, and spatial and temporal access patterns; (2) mechanisms to effect changes in segment configuration while an application is running; (3) the ability to change segment characteristics at runtime without application restart; and (4) performance counter hardware to observe application behavior during normal execution to improve performance of on-line statistics on memory accesses.

[0063] Although specific examples have been illustrated and described herein, a variety of alternate and/or equivalent implementations may be substituted for the specific examples shown and described without departing from the scope of the present disclosure. This application is intended to cover any adaptations or variations of the specific examples discussed herein. Therefore, it is intended that this disclosure be limited only by the claims and the equivalents thereof.

CLAIMS

1. A method of modifying characteristics of an existing memory segment in a virtual address space for a running software application, wherein the existing memory segment includes an identification of a first backing memory region in physical memory for storing contents of the existing memory segment, the method comprising:

tracking memory access statistics of the running software application;

identifying new characteristics for the existing memory segment based on the tracked memory access statistics; and

generating an updated memory segment that includes the identified new characteristics.

2. The method of claim 1, wherein generating an updated memory segment comprises:

configuring a new memory segment to have the identified new characteristics; and

copying contents of the existing memory segment to the new memory segment.

3. The method of claim 1, wherein generating an updated memory segment comprises:

copying segment contents from the first backing memory region to a second backing memory region that includes the identified new characteristics; and

updating the existing memory segment to include an identification of the second backing memory region.

4. The method of claim 1, wherein the identified new characteristics include at least one of page size, persistence, Non-Uniform Memory Access (NUMA) region, memory type, and non-volatile memory (NVM) locality.

5. The method of claim 1, wherein the memory access statistics include at least one of sparsity, contiguity, spatial access patterns, and temporal access patterns.

6. The method of claim 1, and further comprising:

tracking memory access statistics of a plurality of running software applications;

calculating a fitness number for each of the running software applications based on the tracked memory access statistics; and

identifying at least one of the running software applications for memory segment adjustment based on the calculated fitness numbers.

7. The method of claim 6, wherein each of the fitness numbers is a weighted metric of user-specified weights multiplied by results from the tracked memory statistics.

8. The method of claim 7, wherein each of the fitness numbers includes a first term comprising a translation lookaside buffer (TLB) miss ratio, a second term comprising a non-volatile memory read ratio, and a third term comprising a ratio of CPU stalls caused by memory references.

9. The method of claim 6, and further comprising:

calculating per-segment fitness numbers for a plurality of memory segments in the identified at least one of the running software applications; and

identifying at least one of the plurality of memory segments for adjustment based on the per-segment fitness numbers.

10. The method of claim 1, and further comprising:

providing a database of application types and a respective set of memory segment characteristics for each of the application types;

identifying an application type for the running software application based on the tracked memory access statistics; and

identifying the new characteristics from the database based on the identified application type.

11. A memory segment configuration system, comprising:

an application monitoring module to track memory access statistics of a running software application, wherein the running software application comprises a virtual address space with a first memory segment;

a memory segment characteristics identification module to identify new characteristics for the first memory segment based on the tracked memory access statistics; and

a memory segment characteristics modification module to generate an updated memory segment that includes the identified new characteristics.

12. The system of claim 11, and further comprising:

an application identification module to calculate a fitness number for each of a plurality of running software applications based on tracked memory access statistics for each of the running software applications, and identify at least one of the running software applications for memory segment adjustment based on the calculated fitness numbers.

13. The system of claim 12, wherein each of the fitness numbers is a weighted metric of user-specified weights multiplied by results from the tracked memory statistics.

14. A non-transitory computer-readable storage medium storing computer-executable instructions that when executed by at least one processor cause the at least one processor to perform a method, comprising:

tracking memory access statistics of the running software application, wherein the running software application comprises a virtual address space with a first memory segment, and wherein the first memory segment includes an

initial set of characteristics including an initial page size and an initial identification of a backing memory region; and

modifying at least one of the initial page size and the initial identification of a backing memory region of the first memory segment based on the tracked memory access statistics.

15. The computer-readable storage medium of claim 14, wherein the memory access statistics include at least one of sparsity, contiguity, spatial access patterns, and temporal access patterns.

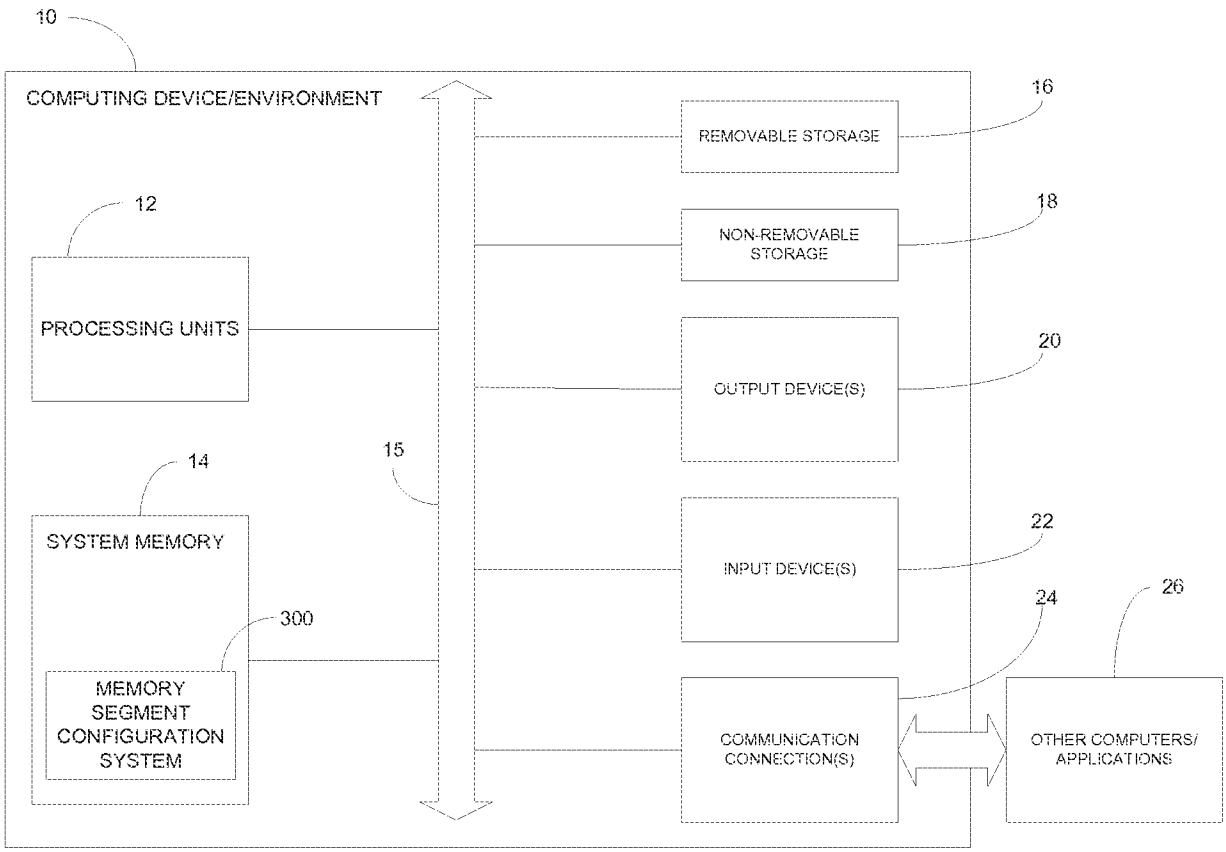
10
COMPUTING DEVICE/ENVIRONMENT
16
REMOVABLE STORAGE
12
18
NON-REMOVABLE STORAGE
PROCESSING UNITS
20
OUTPUT DEVICE(S)
15
14
SYSTEM MEMORY
22
INPUT DEVICE(S)
24
26
300
MEMORY SEGMENT CONFIGURATION SYSTEM
COMMUNICATION CONNECTION(S)
OTHER COMPUTERS/ APPLICATIONS

Fig. 1

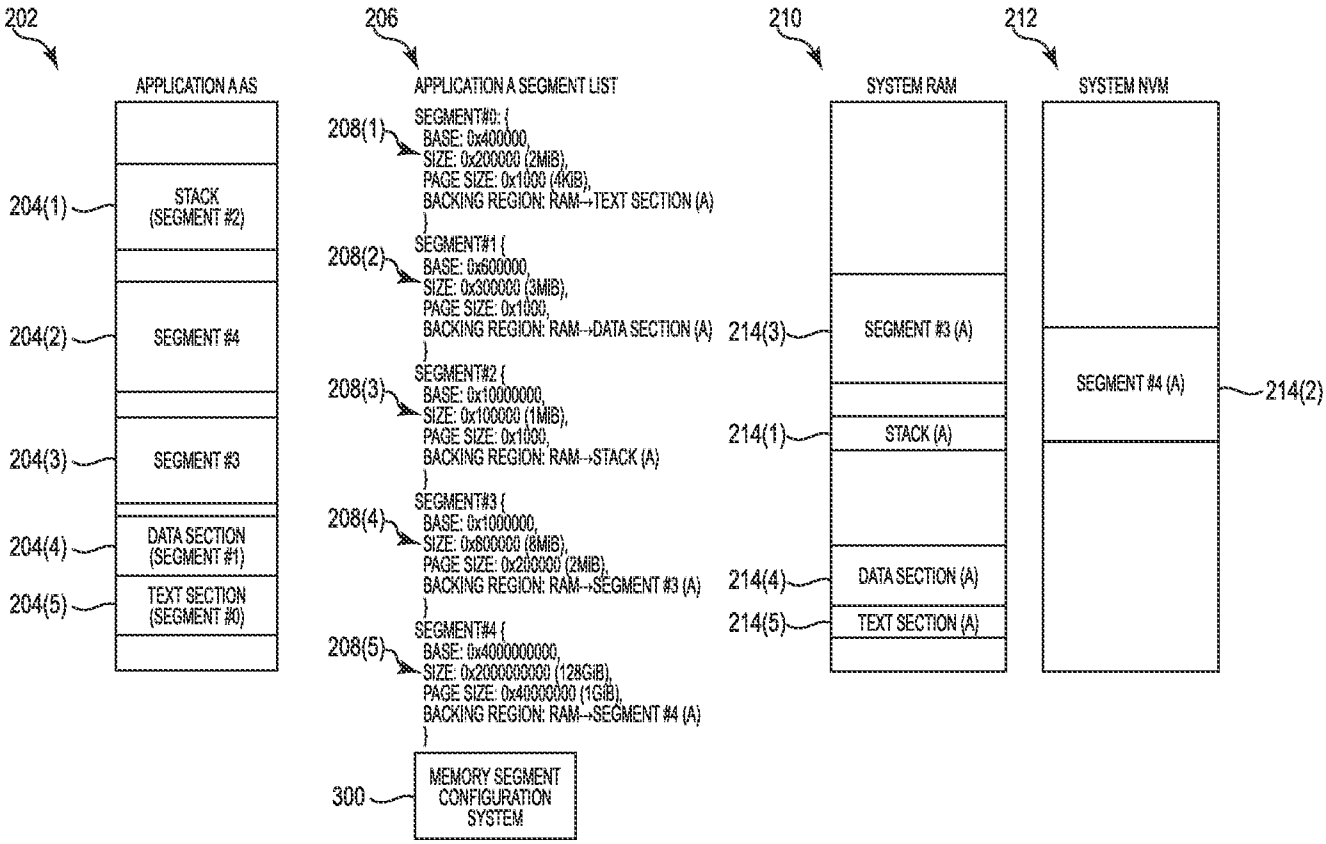
202
APPLICATION A AS
204(1)
STACK (SEGMENT #2)
204(2)
SEGMENT #4
204(3)
SEGMENT #3
204(4)
DATA SECTION (SEGMENT #1)
204(5)
TEXT SECTION (SEGMENT #0)
206
APPLICATION A SEGMENT LIST
208(1)
SEGMENT#0: {
BASE: 0x400000,
SIZE: 0x200000 (2MiB),
PAGE SIZE: 0x1000 (4KiB),
BACKING REGION: RAM→TEXT SECTION (A)
}
208(2)
SEGMENT#1 {
BASE: 0x600000,
SIZE: 0x300000 (3MiB),
PAGE SIZE: 0x1000,
BACKING REGION: RAM→DATA SECTION (A)
}
208(3)
SEGMENT#2 {
BASE: 0x10000000,
SIZE: 0x100000 (1MiB),
PAGE SIZE: 0x1000,
BACKING REGION: RAM→STACK (A)
}
208(4)
SEGMENT#3 {
BASE: 0x1000000,
SIZE: 0x800000 (8MiB),
PAGE SIZE: 0x200000 (2MiB),
BACKING REGION: RAM→SEGMENT #3 (A)
}
208(5)
SEGMENT#4 {
BASE: 0x4000000000,
SIZE: 0x2000000000 (128GiB),
PAGE SIZE: 0x40000000 (1GiB),
BACKING REGION: RAM→SEGMENT #4 (A)
}
300
MEMORY SEGMENT CONFIGURATION SYSTEM
210
SYSTEM RAM
214(3)
SEGMENT #3 (A)
214(1)
STACK (A)
214(4)
DATA SECTION (A)
214(5)
TEXT SECTION (A)
212
SYSTEM NVM
SEGMENT #4 (A)
214(2)

Fig. 2

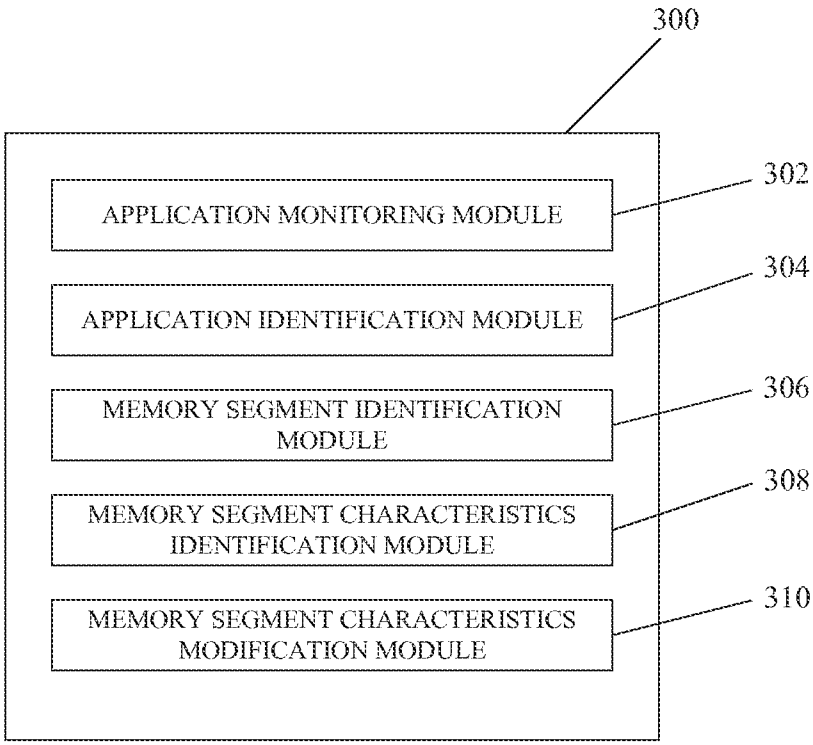
300
APPLICATION MONITORING MODULE
302
APPLICATION IDENTIFICATION MODULE
304
MEMORY SEGMENT IDENTIFICATION MODULE
306
MEMORY SEGMENT CHARACTERISTICS IDENTIFICATION MODULE
308
MEMORY SEGMENT CHARACTERISTICS MODIFICATION MODULE
310

Fig. 3

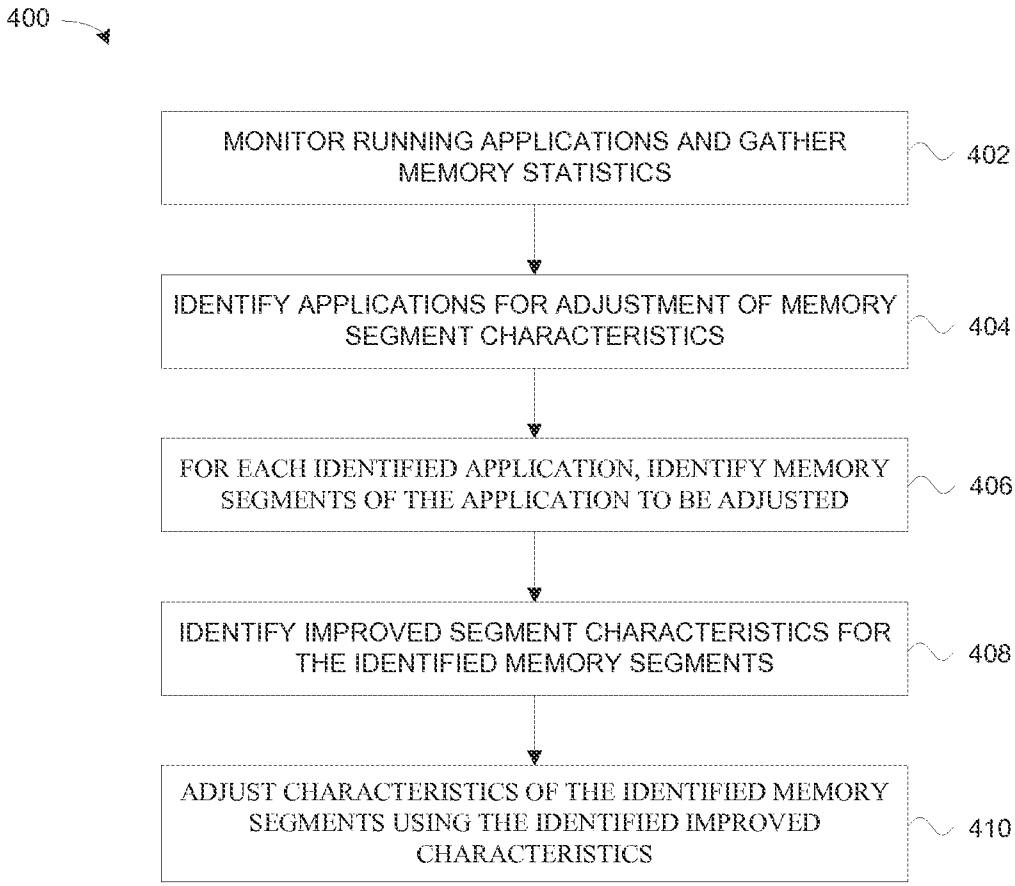
400
MONITOR RUNNING APPLICATIONS AND GATHER MEMORY STATISTICS
402
IDENTIFY APPLICATIONS FOR ADJUSTMENT OF MEMORY SEGMENT CHARACTERISTICS
404
FOR EACH IDENTIFIED APPLICATION, IDENTIFY MEMORY SEGMENTS OF THE APPLICATION TO BE ADJUSTED
406
IDENTIFY IMPROVED SEGMENT CHARACTERISTICS FOR THE IDENTIFIED MEMORY SEGMENTS
408
ADJUST CHARACTERISTICS OF THE IDENTIFIED MEMORY SEGMENTS USING THE IDENTIFIED IMPROVED CHARACTERISTICS
410

Fig. 4

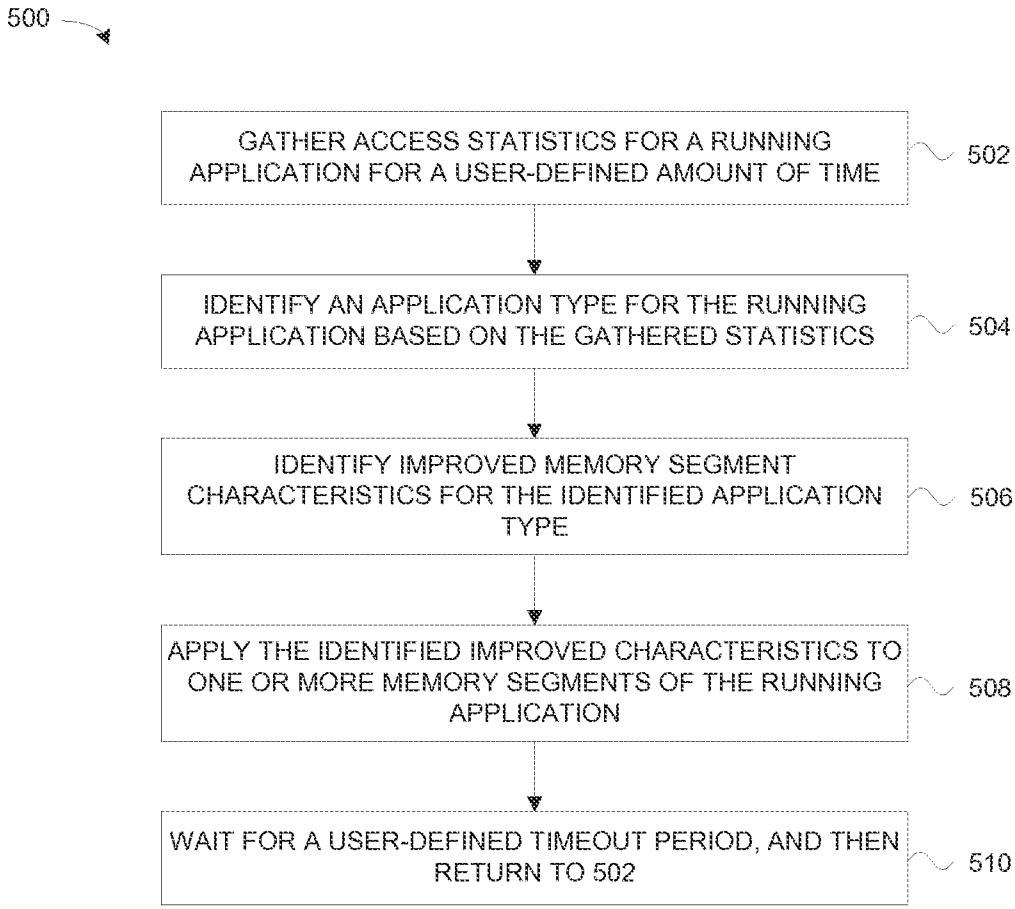
500
GATHER ACCESS STATISTICS FOR A RUNNING APPLICATION FOR A USER-DEFINED AMOUNT OF TIME
502
IDENTIFY AN APPLICATION TYPE FOR THE RUNNING APPLICATION BASED ON THE GATHERED STATISTICS
504
IDENTIFY IMPROVED MEMORY SEGMENT CHARACTERISTICS FOR THE IDENTIFIED APPLICATION TYPE
506
APPLY THE IDENTIFIED IMPROVED CHARACTERISTICS TO ONE OR MORE MEMORY SEGMENTS OF THE RUNNING APPLICATION
508
WAIT FOR A USER-DEFINED TIMEOUT PERIOD, AND THEN RETURN TO 502
510

Fig. 5

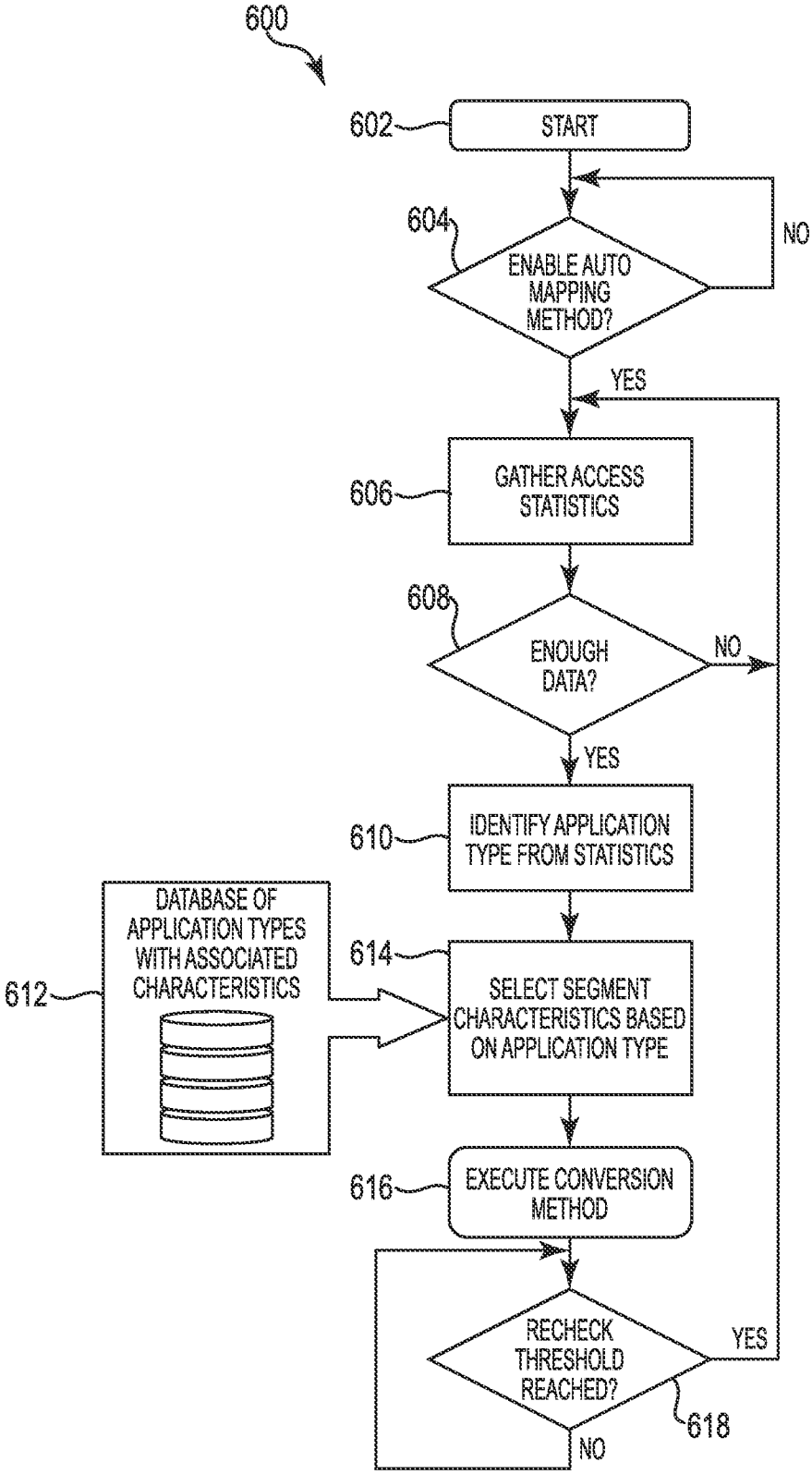
600
602
START
604
ENABLE AUTO MAPPING METHOD?
NO
YES
606
GATHER ACCESS STATISTICS
608
ENOUGH DATA?
NO
YES
610
IDENTIFY APPLICATION TYPE FROM STATISTICS
612
DATABASE OF APPLICATION TYPES WITH ASSOCIATED CHARACTERISTICS
614
SELECT SEGMENT CHARACTERISTICS BASED ON APPLICATION TYPE
616
EXECUTE CONVERSION METHOD
RECHECK THRESHOLD REACHED?
YES
NO
618

Fig. 6

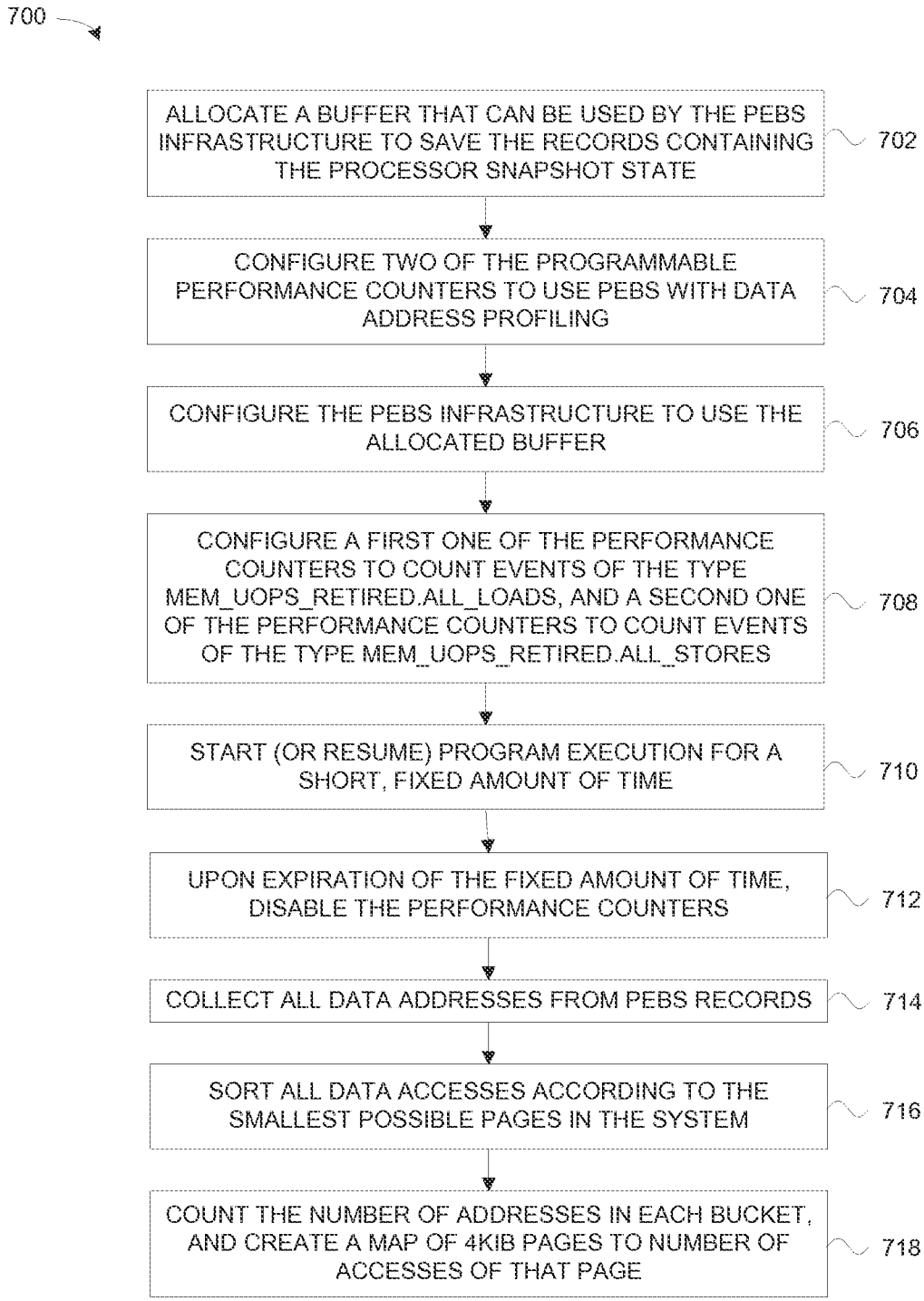
700
ALLOCATE A BUFFER THAT CAN BE USED BY THE PEBS INFRASTRUCTURE TO SAVE THE RECORDS CONTAINING THE PROCESSOR SNAPSHOT STATE
702
CONFIGURE TWO OF THE PROGRAMMABLE PERFORMANCE COUNTERS TO USE PEBS WITH DATA ADDRESS PROFILING
704
CONFIGURE THE PEBS INFRASTRUCTURE TO USE THE ALLOCATED BUFFER
706
CONFIGURE A FIRST ONE OF THE PERFORMANCE COUNTERS TO COUNT EVENTS OF THE TYPE MEM_UOPS_RETIRED.ALL_LOADS, AND A SECOND ONE OF THE PERFORMANCE COUNTERS TO COUNT EVENTS OF THE TYPE MEM_UOPS_RETIRED.ALL_STORES
708
START (OR RESUME) PROGRAM EXECUTION FOR A SHORT, FIXED AMOUNT OF TIME
710
UPON EXPIRATION OF THE FIXED AMOUNT OF TIME, DISABLE THE PERFORMANCE COUNTERS
712
COLLECT ALL DATA ADDRESSES FROM PEBS RECORDS
714
SORT ALL DATA ACCESSES ACCORDING TO THE SMALLEST POSSIBLE PAGES IN THE SYSTEM
716
COUNT THE NUMBER OF ADDRESSES IN EACH BUCKET, AND CREATE A MAP OF 4KIB PAGES TO NUMBER OF ACCESSES OF THAT PAGE
718

Fig. 7

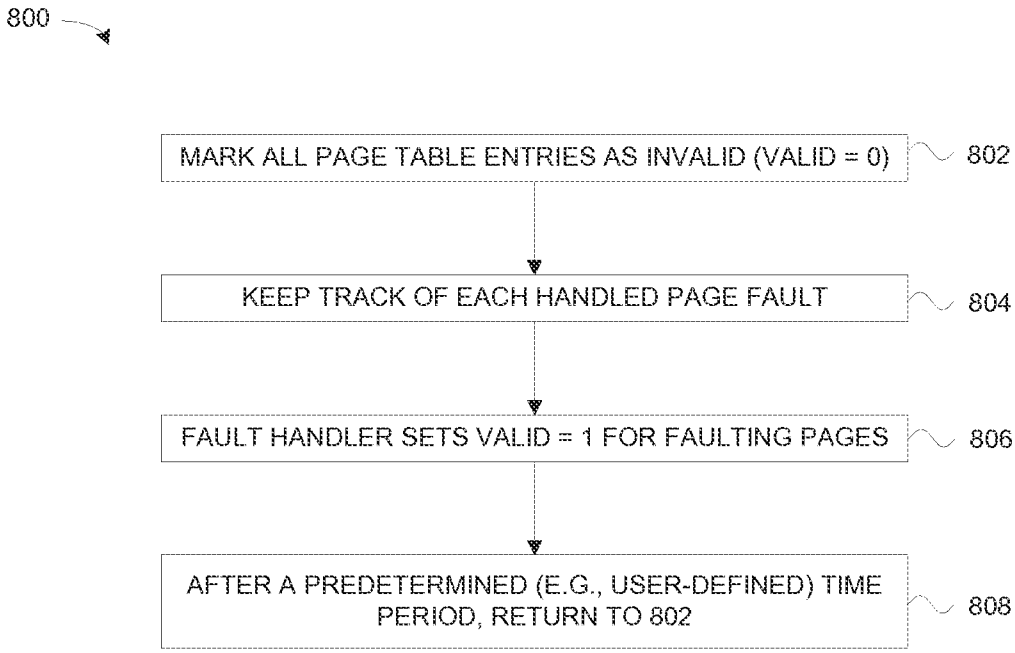
800
MARK ALL PAGE TABLE ENTRIES AS INVALID (VALID = 0)
802
KEEP TRACK OF EACH HANDLED PAGE FAULT
804
FAULT HANDLER SETS VALID = 1 FOR FAULTING PAGES
806
AFTER A PREDETERMINED (E.G., USER-DEFINED) TIME PERIOD, RETURN TO 802
808

Fig. 8

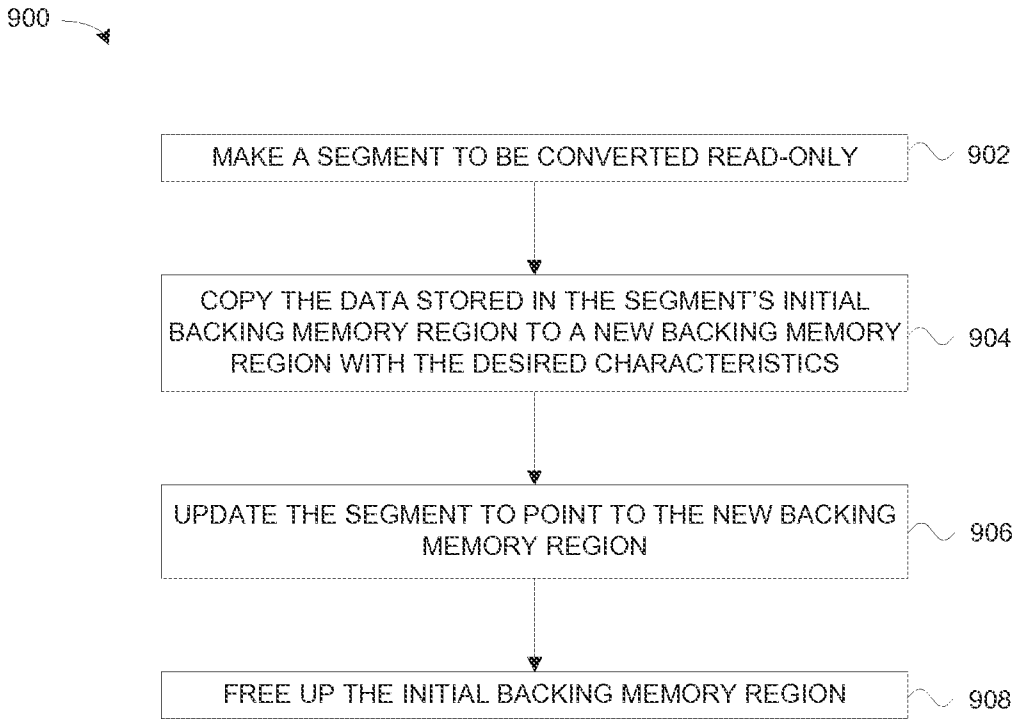
900
MAKE A SEGMENT TO BE CONVERTED READ-ONLY
902
COPY THE DATA STORED IN THE SEGMENT'S INITIAL BACKING MEMORY REGION TO A NEW BACKING MEMORY REGION WITH THE DESIRED CHARACTERISTICS
904
UPDATE THE SEGMENT TO POINT TO THE NEW BACKING MEMORY REGION
906
FREE UP THE INITIAL BACKING MEMORY REGION
908

Fig. 9

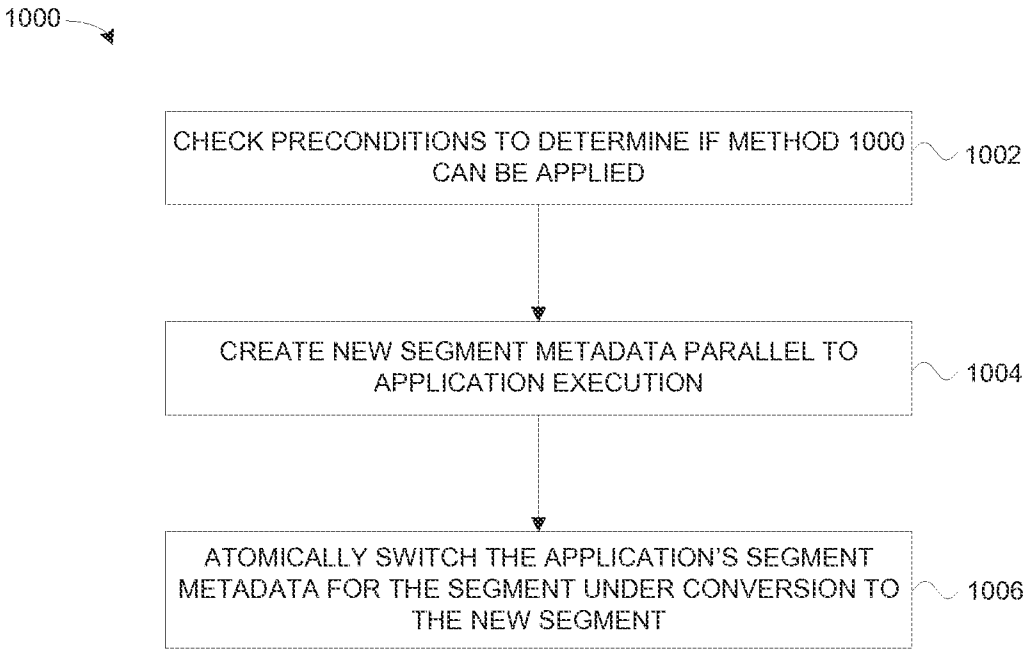
1000
CHECK PRECONDITIONS TO DETERMINE IF METHOD 1000 CAN BE APPLIED
1002
CREATE NEW SEGMENT METADATA PARALLEL TO APPLICATION EXECUTION
1004
ATOMICALLY SWITCH THE APPLICATION'S SEGMENT METADATA FOR THE SEGMENT UNDER CONVERSION TO THE NEW SEGMENT
1006

Fig. 10

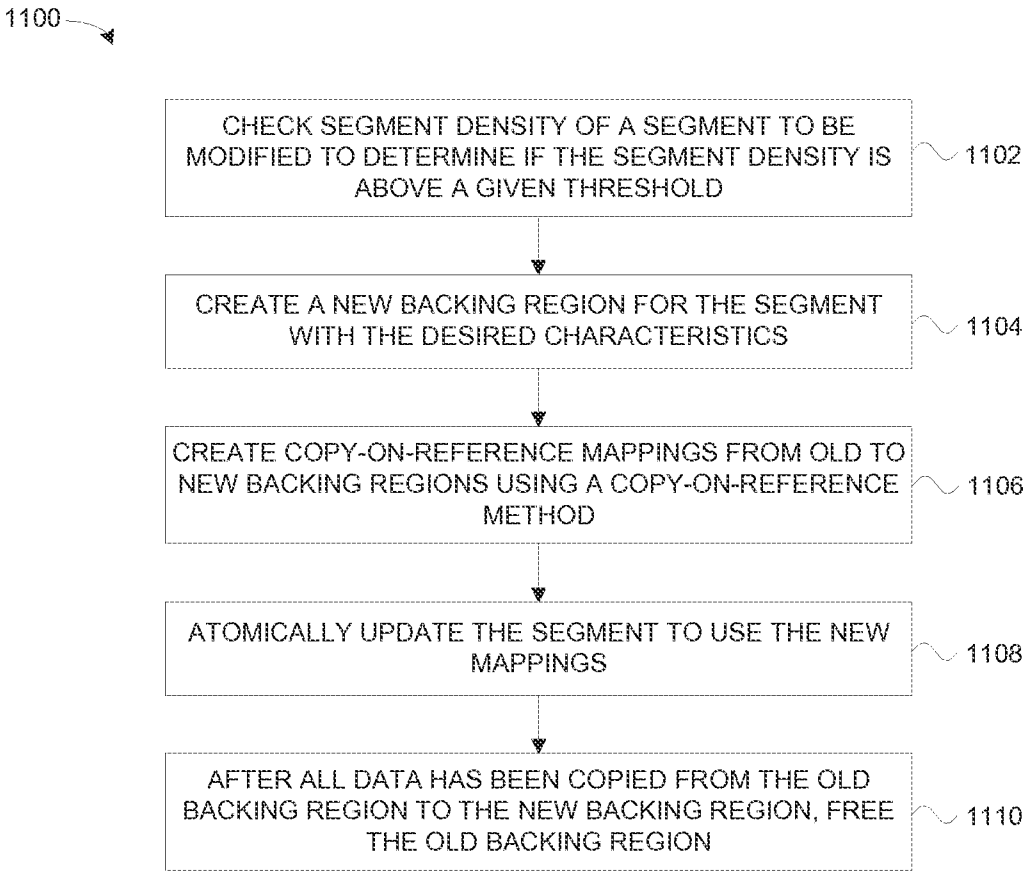
1100
CHECK SEGMENT DENSITY OF A SEGMENT TO BE MODIFIED TO DETERMINE IF THE SEGMENT DENSITY IS ABOVE A GIVEN THRESHOLD
1102
CREATE A NEW BACKING REGION FOR THE SEGMENT WITH THE DESIRED CHARACTERISTICS
1104
CREATE COPY-ON-REFERENCE MAPPINGS FROM OLD TO NEW BACKING REGIONS USING A COPY-ON-REFERENCE METHOD
1106
ATOMICALLY UPDATE THE SEGMENT TO USE THE NEW MAPPINGS
1108
AFTER ALL DATA HAS BEEN COPIED FROM THE OLD BACKING REGION TO THE NEW BACKING REGION, FREE THE OLD BACKING REGION
1110

Fig. 11

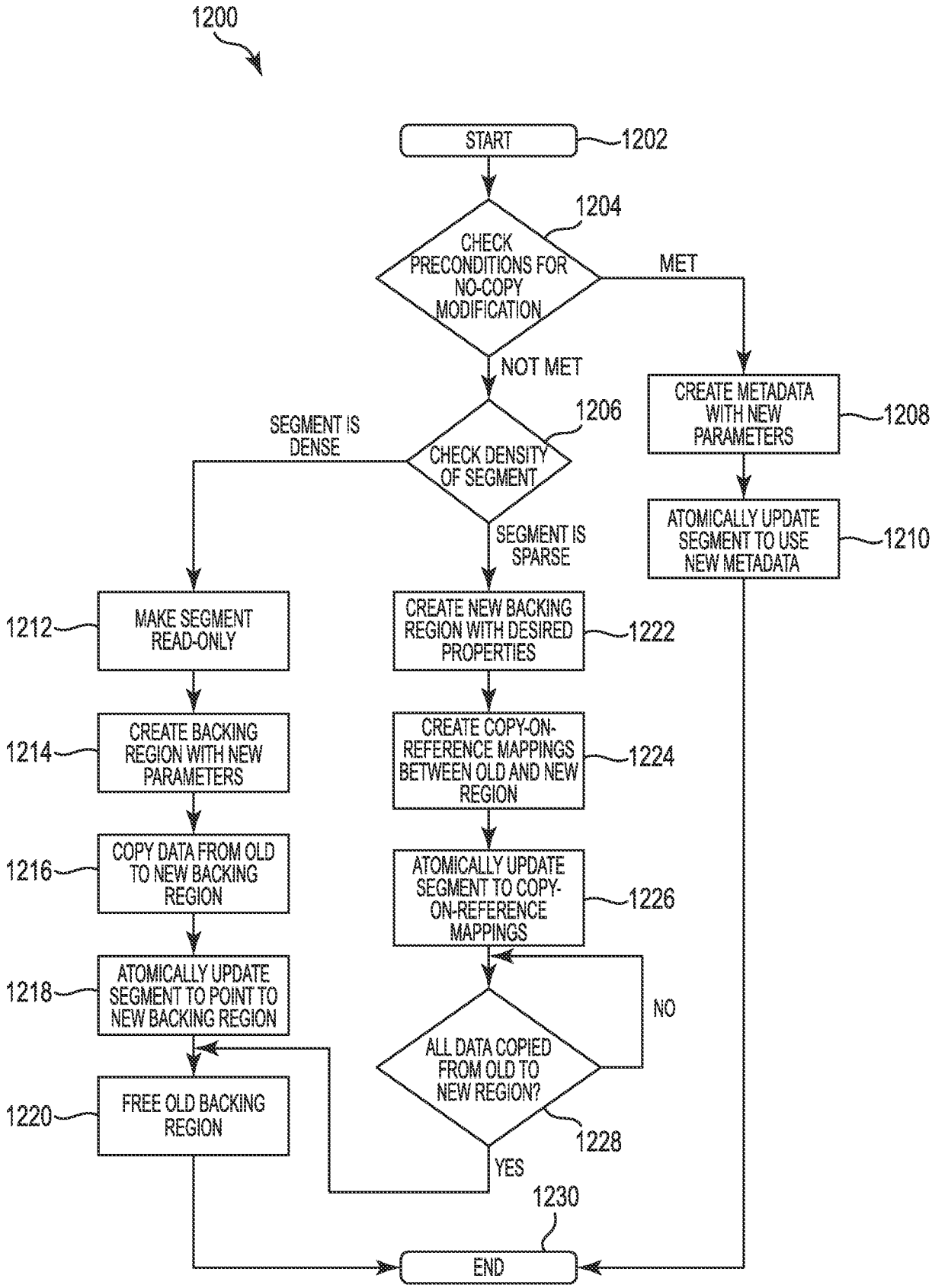
1200
START
1202
CHECK PRECONDITIONS FOR NO-COPY MODIFICATION
1204
MET
NOT MET
CREATE METADATA WITH NEW PARAMETERS
1208
ATOMICALLY UPDATE SEGMENT TO USE NEW METADATA
1210
SEGMENT IS DENSE
CHECK DENSITY OF SEGMENT
1206
SEGMENT IS SPARSE
MAKE SEGMENT READ-ONLY
1212
CREATE BACKING REGION WITH NEW PARAMETERS
1214
COPY DATA FROM OLD TO NEW BACKING REGION
1216
ATOMICALLY UPDATE SEGMENT TO POINT TO NEW BACKING REGION
1218
FREE OLD BACKING REGION
1220
CREATE NEW BACKING REGION WITH DESIRED PROPERTIES
1222
CREATE COPY-ON-REFERENCE MAPPINGS BETWEEN OLD AND NEW REGION
1224
ATOMICALLY UPDATE SEGMENT TO COPY-ON-REFERENCE MAPPINGS
1226
ALL DATA COPIED FROM OLD TO NEW REGION?
1228
NO
YES
1230
END

Fig. 12

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/013748

A. CLASSIFICATION OF SUBJECT MATTER

G11C 7/10(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)
G11C 7/10; G06F 12/10; G06F 12/00; G06F 12/14; G06F 12/08; G06F 9/50

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 segment, virtual address space, characteristic, backing memory region, software application

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2003-0126354 A1 (OPHER D. KAHN et al.) 03 July 2003 See paragraphs [0021], [0030]; claims 1, 11; and figure 1.	1-15
A	US 6442666 B1 (HENRY STRACOVSKY) 27 August 2002 See column 3, lines 30-46; and figure 4.	1-15
A	US 2012-0159103 A1 (MARCUS PEINADO et al.) 21 June 2012 See paragraph [0038]; and figure 4.	1-15
A	WO 2012-074850 A2 (MICROSOFT CORPORATION) 07 June 2012 See paragraph [0073]; and figures 4(a)-4(j).	1-15
A	US 2014-0223442 A1 (RED HAT ISRAEL, LTD.) 07 August 2014 See paragraphs [0021], [0027]; 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
15 October 2015 (15.10.2015)	**27 October 2015 (27.10.2015)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 35208, Republic of Korea
Facsimile No. +82-42-472-7140

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

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2015/013748

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2003-0126354 A1	03/07/2003	AT 431590 T	15/05/2009
		AU 2002-359868 A1	24/07/2003
		CN 1613064 A	04/05/2005
		CN 1613064 C	08/11/2006
		EP 1461706 A1	29/09/2004
		EP 1461706 B1	13/05/2009
		KR 10-0626770 B1	25/09/2006
		KR 10-2004-0064742 A	19/07/2004
		TW 200305803 A	01/11/2003
		TW I284261 B	21/07/2007
		US 6799241 B2	28/09/2004
		WO 2003-058456 A1	17/07/2003
US 6442666 B1	27/08/2002	AU 3352800 A	18/08/2000
		AU 3693900 A	18/08/2000
		CN 1158607 C	21/07/2004
		CN 1160631 C	04/08/2004
		CN 1347526 A	01/05/2002
		CN 1352771 A	05/06/2002
		CN 1352772 A	05/06/2002
		EP 1157335 A1	28/11/2001
		EP 1157335 A4	26/05/2004
		EP 1181644 A1	27/02/2002
		EP 1181644 A4	19/05/2004
		EP 1196850 A1	17/04/2002
		EP 1196850 A4	26/05/2004
		JP 2002-536715 A	29/10/2002
		JP 2002-536716 A	29/10/2002
		JP 2002-536717 A	29/10/2002
		WO 00-45267 A1	03/08/2000
		WO 00-45270 A2	03/08/2000
		WO 00-45270 A8	15/03/2001
		WO 00-45271 A1	03/08/2000
		WO 00-45271 A9	04/10/2001
US 2012-0159103 A1	21/06/2012	US 2015-067272 A1	05/03/2015
		US 8996814 B2	31/03/2015
WO 2012-074850 A2	07/06/2012	CN 102567224 A	11/07/2012
		EP 2646921 A2	09/10/2013
		EP 2646921 A4	26/02/2014
		US 2012-0144092 A1	07/06/2012
		US 8775737 B2	08/07/2014
		WO 2012-074850 A3	16/08/2012
US 2014-0223442 A1	07/08/2014	US 2011-247000 A1	06/10/2011
		US 8656397 B2	18/02/2014