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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 December 2016 (01.12.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/190892 A1

(51) International Patent Classification:
G06F 9/455 (2006.01)

(21) International Application Number:
PCT/US2015/042983

(22) International Filing Date:
30 July 2015 (30.07.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
2690/CHE/2015 28 May 2015 (28.05.2015) IN

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

(72) Inventors: **KOLHE, Jitendra Onkar**; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, Karnataka 560048 (IN). **PUCHIMANDA RAMACHA, Vrashi Ponnappa**; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, Karnataka 560048 (IN). **ABRAHAM, Santosh**; Sy.No.192, Whitefield Road, Mahadevapura Post, Bangalore, Karnataka 560048 (IN).

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

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH,

[Continued on next page]

(54) Title: IMPROVING PERFORMANCE OF VIRTUAL MACHINES

(57) Abstract: Some examples describe improving performance of a virtual machine. In an example, information related to guest address space of a guest operating system may be shared with a hypervisor supporting the guest operating system in a paravirtualization environment, wherein the guest operating system is configured to share the information related to guest address space with the hypervisor, and the hypervisor is configured to receive the information related to guest address space shared by the guest operating system. The hypervisor may use the information related to guest address space during performance of a task for the guest operating system.

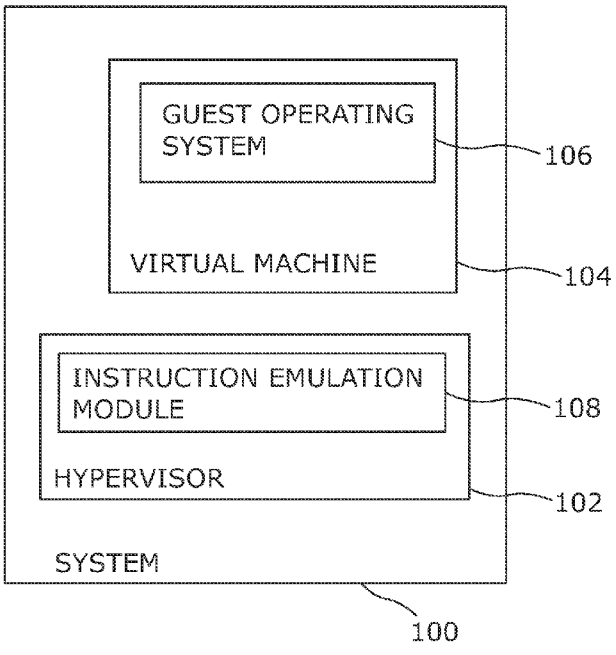

Fig. 1

WO 2016/190892 A1

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))*

IMPROVING PERFORMANCE OF VIRTUAL MACHINES

Background

[001] Paravirtualization is a virtualization technique in which a guest operating system is modified prior to installation inside a virtual machine. In other words, paravirtualization allows for the presentation of a software interface to virtual machines that may be similar, but not identical to an underlying hardware. In an instance, in the modified interface, non-virtualizable instructions may be replaced with hypercalls to a virtual machine monitor (VMM) layer or hypervisor.

Brief Description of the Drawings

[002] For a better understanding of the solution, embodiments will now be described, purely by way of example, with reference to the accompanying drawings, in which:

[003] FIG. 1 is a block diagram of an example computing system for improving performance of virtual machines;

[004] FIG. 2 is a flowchart of an example method of improving performance of virtual machines; and

[005] FIG. 3 is a block diagram of an example system for improving performance of virtual machines.

Detailed Description

[006] Virtualization allows creation of a virtual version of a resource, such as an operating system, a hardware platform, storage resource etc. which could be shared, for instance, among different clients. Multiple virtual machines

(VM) may be created on a host device by a hypervisor or virtual machine monitor (VMM).

[007] Hypervisors play a key role in scalability and performance of virtual machines. In software based CPU virtualization, the operating system running inside the virtual machine (also referred as guest OS) may run in non-privilege mode on the host. In such case, the hypervisors may be required to intercept all privileged instructions being executed by guest operating systems and emulate them to provide the functionality expected by the guest operating system. Binary translation is one mechanism used by software base CPU virtualizations or hypervisors to intercept privileged instructions. A considerable amount of physical CPU bandwidth may be consumed by hypervisors in emulating privileged instructions, leaving less physical CPU bandwidth to run useful workloads. This factor may have a direct impact on throughputs and response time of workloads and may be considered as a cause of concern for customers planning to switch their workloads to a virtualized environment.

[008] There may be instances even when a hypervisor is running on hardware that supports virtualization may require to use binary translation technique for certain set of guest executable pages. For example, during a process context switch in a guest kernel, a set of control register needs may need to be updated. In such case, instead of faulting on every instruction for updating control registers, the context switch code may be binary translated to fault only once into a specific hypervisor handler which may update all control registers in a single instance.

[009] As part of a binary translation, a hypervisor may be required to scan guest text pages (or instruction pages) and identify if the page needs to be binary translated. If the guest text page requires a binary translation, the hypervisor may need to maintain both original and translated copies of the same page. This is because the hypervisor tries to give an illusion to the

guest OS that it is running unmodified. When the guest OS tries to execute an instruction from the translated page, the hypervisor ensures that the instructions from the binary translated page are executed. And when the guest OS tries to read or write the binary translated page, hypervisor may return original instructions. This may be carried out by setting appropriate access bits in page translation. Thus, the guest OS never comes to know that the hypervisor may have modified its text.

[0010] Binary translation of certain text pages may have a direct impact on the performance of a hypervisor. For example, when a guest OS tries to access a page for a read or write operation, the hypervisor may be expected to first provide the page translation. Since hypervisor while emulating insert translation instruction may not be aware whether the guest page being accessed is either text or data, the hypervisor may need to check if any binary translated text may be present for the address being referred to. This may be to ensure that the read and write access occurs on only original page and not on binary translated page. Similarly, when a guest OS purges a kernel translation, a hypervisor may be required to cleanup any binary translation book keeping structures and also sync with all other virtual CPUs to avoid any accidental purge of translated text, in the event a remote virtual CPU may be executing the same translated text. Since the check for a binary translated page occurs in a critical path (emulation of insert, purge translation instructions etc.), the overall workload performance of a virtual machine may be impacted.

[0011] Another example of a scenario that may impact performance of a virtual machine may include the extra overhead incurred in guest page table lookups during execution of a guest operating system. To elaborate, in an instance, hypervisors typically try to give an illusion to virtual machines that they are booted with zero base memory (i.e. the guest operating system presumes that its physical address range starts from 0th address). Thus, the operating system inside the virtual machine may create all its address

translation with zero base memory. Hypervisors may convert a guest physical address to a host physical address while inserting address translation into a hardware translation lookaside buffer (TLB). This may also include converting a guest physical address of a guest's page table to a host physical address.

[0012] In a virtualized environment, for a typical address translation miss in a TLB, the hypervisor may first do a lookup in its own page tables. If the address translation is not found there, hypervisor may do a lookup in the guest operating system's page tables. However, to do a lookup in the guest page tables, first the translation for a guest page table may need to be resolved before it is pushed into the TLB. This may add an overhead during resolution of address translation upon TLB miss faults which in turn may add to the overall virtualization overhead.

[0013] To address these issues, the present disclosure describes a solution for improving performance of a virtual machine. In an example, information related to guest address space of a guest operating system may be shared with a hypervisor supporting the guest operating system in a paravirtualization environment, wherein the guest operating system is configured to share the information related to guest address space with the hypervisor, and the hypervisor is configured to receive the information related to guest address space shared by the guest operating system. The hypervisor may use the information related to guest address space during performance of a task for the guest operating system. In an instance, using the information may improve the hypervisor's performance during emulation of a privileged instruction for the guest operating system. Privileged instruction may be defined to include an instruction that may only be executed when the processor is running in a special privileged mode. Some non-limiting examples of privileged instructions may include operations such as I/O, interrupt handling, and memory management.

[0014] FIG. 1 is a block diagram of an example computing system 100 for improving performance of a virtual machine. Computing system 100 may represent any type of computing device capable of reading machine-executable instructions. Examples of computing system 100 may include, without limitation, a server, a desktop computer, a notebook computer, a tablet computer, a thin client, a mobile device, a personal digital assistant (PDA), a phablet, and the like. In an example, computing system 100 may include a hypervisor 102 and a virtual machine 104. Although only hypervisor 102 and one virtual machine 104 are shown in FIG. 1, other examples of this disclosure may include more than one hypervisor and more than one virtual machine.

[0015] In an example, computing system 100 may include hardware, such as for example, one or more processors (not shown) and a machine-readable storage medium (not shown) communicatively coupled through a system bus. The processor may be any type of Central Processing Unit (CPU), microprocessor, or processing logic that interprets and executes machine-readable instructions stored in the machine-readable storage medium. The machine-readable storage medium may be a random access memory (RAM) or another type of dynamic storage device that may store information and machine-readable instructions that may be executed by the processor. For example, the machine-readable storage medium may be Synchronous DRAM (SDRAM), Double Data Rate (DDR), Rambus DRAM (RDRAM), Rambus RAM, etc. or storage memory media such as a floppy disk, a hard disk, a CD-ROM, a DVD, a pen drive, and the like. In an example, the machine-readable storage medium may be a non-transitory machine-readable medium.

[0016] In an example, virtual machine 104 may be managed by hypervisor or virtual machine manager (VMM) 102. A hypervisor 102 may be defined as a computer program, firmware or hardware that may create and run one or more virtual machines. A computer system on which a hypervisor is running

a virtual machine may be called as a host machine (for example, computing system 100). Each virtual machine may be called a guest machine. Hypervisor 102 may monitor the hardware resources (for example, one or more processor, memory, peripheral devices, etc.) of computing system and present resources from the physical machine to the virtual machine. In other words, hypervisor 102 may export virtual resources (for example, virtual processors, virtual memory, etc.) rather than the underlying hardware for use of a guest operating system. Hypervisor 102 may present a guest operating system with a virtual operating platform and manage execution of instructions by a guest operating system. In the event computing system hosts more than one virtual machine, hypervisor 102 may isolate virtual machines in a manner that they may be unaware of other virtual machines running on the physical machine. Hypervisor 102 may control the access of a virtual machine 104 to a host's hardware (for example, computing system 100).

[0017] A virtual machine (VM) 104 may be a software implementation of a machine that executes programs like a physical machine. Virtualization may allow creation of one or virtual machines (VM) on a host physical computing system (for example, 100). Virtual machines may be used for a variety of tasks, for example, to run multiple operating systems at the same time and to test a new application on multiple platforms. In an example, virtual machine 104 may host a guest operating system 106, which may support one or more applications. Virtualization allows the guest operating systems to run unmodified on isolated virtual machines.

[0018] In an example, computing system 100 may support virtual memory. In general, virtual memory techniques may provide a mechanism that may allow a computing system to map a virtual address space to a physical address space. Thus, when a processor reads or writes to a memory location, it may use a virtual address. A virtual address space or address space may refer to a range of virtual addresses that an operating system

may make available to a process. As part of the read or write operation, the processor may translate the virtual address to a physical address.

[0019] In an example, hypervisor 102 may provide a set of physical memory ranges to the virtual machine. Hypervisor 102 may provide a separate set of physical memory ranges for each guest operating system on the computing system 100. The hypervisor 102 may maintain a metadata of these sets of physical memory ranges, which allows the hypervisor to run multiple virtual machines simultaneously while protecting the memory of each virtual machine from being accessed by other virtual machines. The range of virtual addresses that may be used by guest operating system 106 may be called as guest address space. This virtual address space in the guest operating system along with hypervisor's metadata may access a set of physical memory ranges assigned to the virtual machine. The guest virtual address space that may be used by a user-mode process may be called as guest user address space. The guest virtual address space that may be used by a kernel-mode process may be called as guest kernel address space.

[0020] The guest kernel may be divided into pages. Some guest kernel pages may contain kernel data, and other pages may contain kernel text or instructions. Each page may be associated with a guest virtual address. A page table may be maintained by the guest operating system to translate a guest virtual address to a guest physical address. Another page table may be maintained by the hypervisor to translate a guest virtual address to a hardware physical address.

[0021] In an example, computing system 100 may include a translation lookaside buffer (TLB). A TLB may be a memory cache that may store recent translations of virtual memory address to physical memory addresses to improve virtual address translation speed. In an example, the hardware that handles this specific translation may be called as the memory management unit. When a memory management unit receives a virtual

address for translation, it may determine if there's an entry in TLB that provides a physical address corresponding to the virtual address. If the requested address is present in the TLB, the retrieved physical address may be used to access memory. If the requested address is not found in the TLB (i.e. a TLB miss), the translation may occur by looking up the page table (i.e. a page walk). After the physical address is determined by the page walk, the virtual address to physical address mapping may be entered into the TLB.

[0022] In an example, computing system 100 may include a Dynamically Loadable Kernel Module (DLKM). A DLKM may be defined as a modularly-packaged module with the capability to be dynamically loaded into a running kernel. In an example, a DLKM module may use uses its own master and system files and contain the module wrapper code and additional data structures that provide dynamic loading and unloading ability. A DLKM module may be dynamically loaded into or unloaded from an operating system kernel without having to re-link the entire kernel or reboot the system.

[0023] Hypervisor 102 may support paravirtualization techniques, hardware supported virtualization or hardware assist. In an example, hypervisor may support a paravirtualization environment wherein both hypervisor and a guest operating system(s) supported by the hypervisor may be modified. In an instance, a guest operating system may be modified to share information related to guest address space of the operating system with the hypervisor. A modified guest operating system may provide entry points into the hypervisor to share guest address ranges. For example, a modified guest operating system may share information related to guest user address space and guest kernel address space with the hypervisor. More specifically, a modified guest operating system may share information related to guest kernel text address ranges and guest kernel data address ranges with the hypervisor. In an example, a guest operating system may be modified to ensure that all kernel text addresses may come from a single common

address range. In such case, a guest operating system may be modified to share only single address range with the hypervisor, and the hypervisor, in turn, may be modified to identify a single kernel text address range instead of multiple address ranges. It may help the hypervisor to differentiate between guest kernel text ranges and guest kernel data ranges by analyzing just a single address range. In an example, a guest operating system may be modified to share page table address ranges with the hypervisor.

[0024] The hypervisor 102 may be modified to receive information related to guest address space shared by a guest operating system. In an example, hypervisor may include an instruction emulation module 108. The term "module" may refer to a software component (machine executable instructions), a hardware component or a combination thereof. A module may include, by way of example, components, such as software components, processes, tasks, co-routines, functions, attributes, procedures, drivers, firmware, data, databases, data structures, Application Specific Integrated Circuits (ASIC) and other computing devices. The module may reside on a volatile or non-volatile storage medium and configured to interact with a processor of a computing device.

[0025] In an example, the instruction emulation module 108 may emulate a task for the guest operating system. In an instance, the task may include emulation of a privileged instruction for the guest operating system. In another example, the task may include emulating a response to a page fault generated by the guest operating system. A page fault may occur, for instance, when the page (data) requested by a program is not available in the memory.

[0026] FIG. 2 is a flowchart of an example method 200 of improving performance of a virtual machine. The method 200, which is described below, may be executed on a computing device such as computing system 100 of FIG. 1. However, other computing devices may be used as well. At

block 202, information related to guest address space of a guest operating system may be shared with a hypervisor supporting the guest operating system in a paravirtualization environment. In an example, the information related to guest address space of a guest operating system may include information related to guest user address space and guest kernel address space. In an instance, the information related to guest kernel address space may include information related to guest kernel text address ranges and guest kernel data address ranges. In an example, the guest address space comprises guest kernel text address ranges from a single common address range. In another example, the guest address space may comprise a guest kernel text address range for a Dynamically Loadable Kernel Module (DLKM). In an instance, the guest kernel text address range in such case may also come from a single common address range.

[0027] In another example, the information related to guest address space of a guest operating system may include information related to guest page table address space of the guest operating system.

[0028] In an instance, the paravirtualization environment includes the guest operating system that may be configured to share the information related to guest address space with the hypervisor, and the hypervisor may be configured to receive the information related to guest address space shared by the guest operating system.

[0029] At block 204, the information related to guest address space may be used by the hypervisor during performance of a task for the guest operating system. In an instance, the task may include emulation of a privileged instruction for the guest operating system by the hypervisor. In an example, the emulation may comprise determining, based on the information related to guest address space shared with the hypervisor, if a page accessed by the privileged instruction belongs to a guest kernel text address space or guest kernel data address space. If the page belongs to the guest kernel text

address space, the hypervisor may pursue a “slow path” during emulation of the privileged instruction. On the other hand, if the page belongs to the guest kernel data address ranges, the hypervisor may pursue a “fast path” during emulation of the privileged instruction.

[0030] The term “fast path” may be defined to include a mechanism wherein the hypervisor may only look up in its own auxiliary metadata structure (guest address to host address map) to create address translation for the guest. The hypervisor may in this case try to retain page size of the guest address translation. In case of “slow path”, in addition to the process followed by the hypervisor under “fast path”, the hypervisor may further determine whether any binary translated text exists for the page being accessed. The hypervisor may perform an entire page scan for determining the possibility of translating other text instruction ahead of execution. Thus, the slow path may require synchronization across all virtual CPUs and a page scan algorithm may also try to limit the page size to a minimum value to avoid extensive scanning of pages having large page size.

[0031] In an example, the hypervisor may determine during emulation, based on the information related to guest address space shared with the hypervisor, if a page accessed by the privileged instruction belongs to a guest user address space or guest kernel address space. If the page accessed by the privileged instruction belongs to the guest kernel address space, the hypervisor may further determine if the page belongs to a guest kernel text address ranges or guest kernel data address ranges. If the page belongs to the guest kernel text address ranges, the hypervisor may pursue a slow path during emulation of the privileged instruction for the guest operating system. On the other hand, if the page belongs to the guest kernel data address ranges, the hypervisor may pursue a fast path during emulation of the privileged instruction for the guest operating system.

[0032] In an example, the guest operating systems may define a fixed address range for its page tables. In an instance, if the information related to guest address space of a guest operating system includes guest page table address space of the guest operating system, the hypervisor may emulate a response to a page fault generated by the guest operating system. In an instance, the response may include updating a translation lookaside buffer (TLB) with information related to guest page table address space of the guest operating system during scheduling of a virtual CPU associated with the guest operating system.

[0033] In an example, as described herein, the emulation may be performed by an instruction emulation module (for example, 108) in the hypervisor (for example, 102).

[0034] FIG. 3 is a block diagram of an example system 300 to improve performance of a virtual machine. System 300 includes a processor 302 and a machine-readable storage medium 304 communicatively coupled through a system bus. In an example, system 300 may be analogous to computing system 102 of FIG. 1. Processor 302 may be any type of Central Processing Unit (CPU), microprocessor, or processing logic that interprets and executes machine-readable instructions stored in machine-readable storage medium 304. Machine-readable storage medium 304 may be a random access memory (RAM) or another type of dynamic storage device that may store information and machine-readable instructions that may be executed by processor 302. For example, machine-readable storage medium 304 may be Synchronous DRAM (SDRAM), Double Data Rate (DDR), Rambus DRAM (RDRAM), Rambus RAM, etc. or storage memory media such as a floppy disk, a hard disk, a CD-ROM, a DVD, a pen drive, and the like. In an example, machine-readable storage medium may be a non-transitory machine-readable medium. Machine-readable storage medium 304 may store instructions 306 and 308. In an example, instructions 306 may be executed by processor 302 to share information related to guest address

space of a guest operating system with a hypervisor supporting the guest operating system in a paravirtualization environment, wherein the guest operating system to share the information related to guest address space with the hypervisor, and the hypervisor to receive the information related to guest address space shared by the guest operating system. Instructions 308 may be executed by processor 302 to use the information related to guest address space during performance of a task for the guest operating system by the hypervisor.

[0035] For the purpose of simplicity of explanation, the example method of FIG. 2 is shown as executing serially, however it is to be understood and appreciated that the present and other examples are not limited by the illustrated order. The example systems of FIGS. 1 and 3, and method of FIG. 2 may be implemented in the form of a computer program product including computer-executable instructions, such as program code, which may be run on any suitable computing device in conjunction with a suitable operating system (for example, Microsoft Windows, Linux, UNIX, and the like). Embodiments within the scope of the present solution may also include program products comprising non-transitory computer-readable media for carrying or having computer-executable instructions or data structures stored thereon. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computer. By way of example, such computer-readable media can comprise RAM, ROM, EPROM, EEPROM, CD-ROM, magnetic disk storage or other storage devices, or any other medium which can be used to carry or store desired program code in the form of computer-executable instructions and which can be accessed by a general purpose or special purpose computer. The computer readable instructions can also be accessed from memory and executed by a processor.

[0036] It should be noted that the above-described examples of the present solution is for the purpose of illustration only. Although the solution has been

described in conjunction with a specific embodiment thereof, numerous modifications may be possible without materially departing from the teachings and advantages of the subject matter described herein. Other substitutions, modifications and changes may be made without departing from the spirit of the present solution.

Claims:

1. A method of improving performance of a virtual machine, comprising:

sharing information related to guest address space of a guest operating system with a hypervisor supporting the guest operating system in a paravirtualization environment, wherein the guest operating system is configured to share the information related to guest address space with the hypervisor, and the hypervisor is configured to receive the information related to guest address space shared by the guest operating system; and

using, by the hypervisor, the information related to guest address space during performance of a task for the guest operating system.

2. The method of claim 1, wherein the task comprises emulation of a privileged instruction for the guest operating system by the hypervisor.

3. The method of claim 2, wherein the emulation comprises:

determining, based on the information related to guest address space shared with the hypervisor, if a page accessed by the privileged instruction belongs to a guest kernel text address ranges or guest kernel data address ranges;

if the page belongs to the guest kernel text address ranges, pursuing, by the hypervisor, a slow path during emulation of the privileged instruction; and

if the page belongs to the guest kernel data address ranges, pursuing, by the hypervisor, a fast path during emulation of the privileged instruction.

4. The method of claim 2, wherein the emulation comprises:

determining, based on the information related to guest address space shared with the hypervisor, if a page accessed by the privileged instruction belongs to a guest user address space or guest kernel address space;

if the page accessed by the privileged instruction belongs to the guest kernel address space, further determining if the page belongs to a guest kernel text address ranges or guest kernel data address ranges; and

if the page belongs to the guest kernel text address ranges, pursuing, by the hypervisor, a slow path during emulation of the privileged instruction for the guest operating system.

5. The method of claim 4, further comprising:

if the page belongs to the guest kernel data address ranges, pursuing, by the hypervisor, a fast path during emulation of the privileged instruction for the guest operating system.

6. A system for improving performance of a virtual machine, comprising:

a guest operating system of a virtual machine;

a hypervisor supporting the guest operating system in a paravirtualization environment, wherein the guest operating system to provide information related to guest address space of the guest operating system with the hypervisor, and the hypervisor to receive the information related to guest address space from the guest operating system; and

an instruction emulation module to use the information related to guest address space during performance of a task for the guest operating system.

7. The system of claim 6, wherein in the paravirtualization environment the guest operating system is modified to provide the information related to guest address space to the hypervisor.

8. The system of claim 6, wherein in the paravirtualization environment the hypervisor is modified to receive the information related to guest address space from the guest operating system.

9. The system of claim 6, wherein the information related to guest address space includes information related to guest kernel text address ranges of the guest operating system, and the task includes emulation of a privilege instruction for the guest operating system by the hypervisor.

10. The system of claim 8, wherein the guest operating system is modified to ensure that the guest kernel text address ranges including a text address range for a DLKM module are from a single address range.

11. A non-transitory machine-readable storage medium comprising instructions to improving performance of a virtual machine, the instructions executable by a processor to:

share information related to guest address space of a guest operating system with a hypervisor supporting the guest operating system in a paravirtualization environment, wherein the guest operating system to share the information related to guest address space with the hypervisor, and the hypervisor to receive the information related to guest address space shared by the guest operating system; and

use, by the hypervisor, the information related to guest address space during performance of a task for the guest operating system.

12. The storage medium of claim 11, wherein the information related to guest address space includes information related to guest page table address space of the guest operating system, and the task includes emulating a response to a page fault generated by the guest operating system.

13. The storage medium of claim 12, wherein the response includes updating, by the hypervisor, a translation lookaside buffer (TLB) with information related to guest page table address space of the guest operating system during scheduling of a virtual CPU associated with the guest operating system.

14. The storage medium of claim 12, wherein the guest address space comprises guest kernel text address ranges from a single address range.

15. The storage medium of claim 14, wherein the guest address space comprises a guest kernel text address range for a Dynamically Loadable Kernel Module (DLKM).

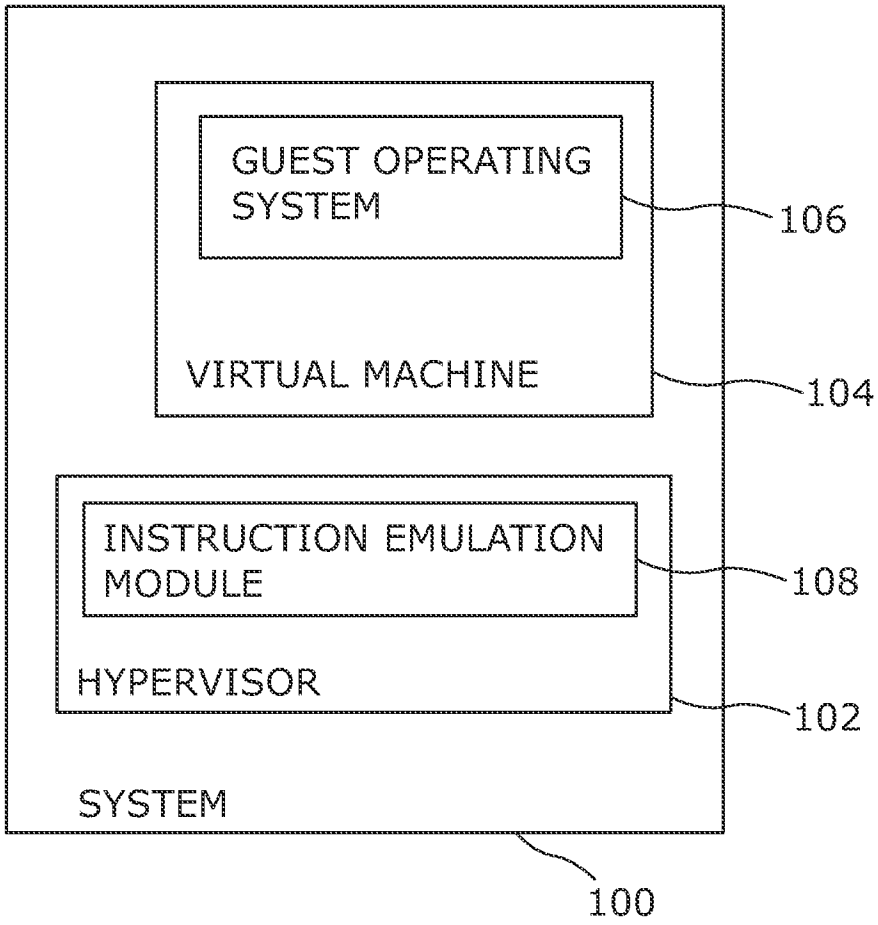
GUEST OPERATING SYSTEM
106
VIRTUAL MACHINE
104
INSTRUCTION EMULATION MODULE
108
HYPERVISOR
102
SYSTEM
100

Fig. 1

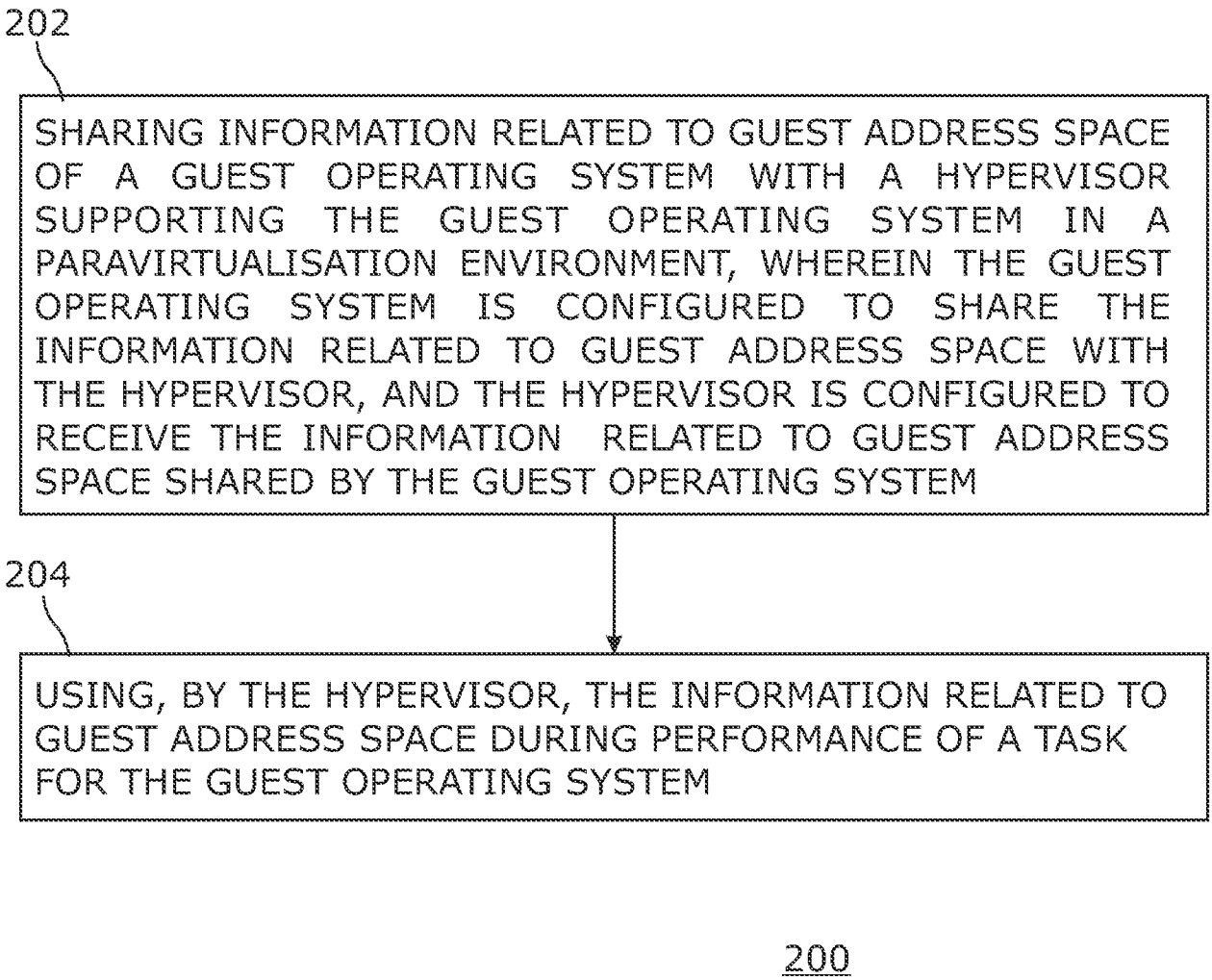
202
SHARING INFORMATION RELATED TO GUEST ADDRESS SPACE OF A GUEST OPERATING SYSTEM WITH A HYPERVISOR SUPPORTING THE GUEST OPERATING SYSTEM IN A PARAVIRTUALISATION ENVIRONMENT, WHEREIN THE GUEST OPERATING SYSTEM IS CONFIGURED TO SHARE THE INFORMATION RELATED TO GUEST ADDRESS SPACE WITH THE HYPERVISOR, AND THE HYPERVISOR IS CONFIGURED TO RECEIVE THE INFORMATION RELATED TO GUEST ADDRESS SPACE SHARED BY THE GUEST OPERATING SYSTEM
204
USING, BY THE HYPERVISOR, THE INFORMATION RELATED TO GUEST ADDRESS SPACE DURING PERFORMANCE OF A TASK FOR THE GUEST OPERATING SYSTEM
200

Fig. 2

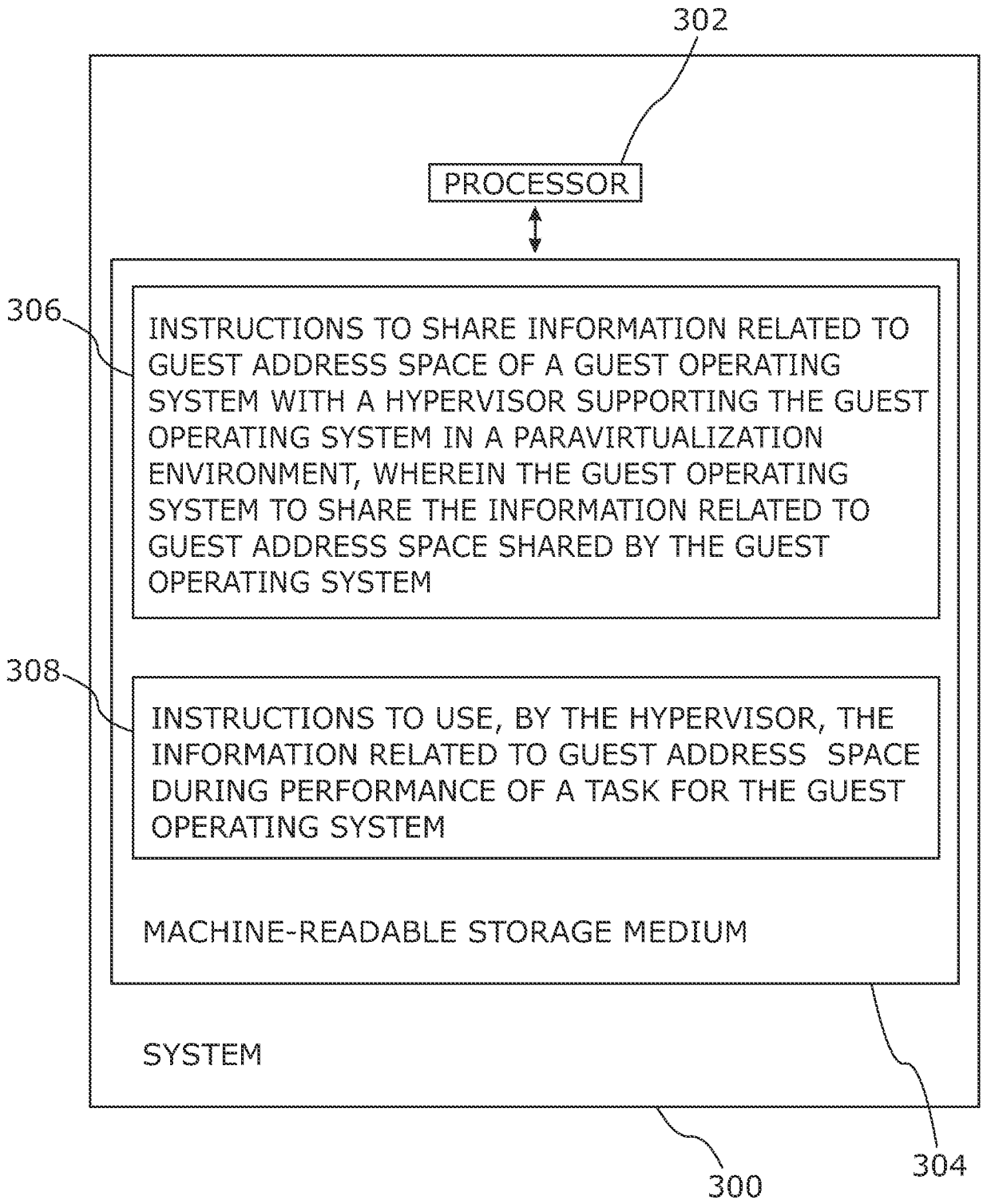
302
PROCESSOR
306
INSTRUCTIONS TO SHARE INFORMATION RELATED TO GUEST ADDRESS SPACE OF A GUEST OPERATING SYSTEM WITH A HYPERVISOR SUPPORTING THE GUEST OPERATING SYSTEM IN A PARAVIRTUALIZATION ENVIRONMENT, WHEREIN THE GUEST OPERATING SYSTEM TO SHARE THE INFORMATION RELATED TO GUEST ADDRESS SPACE SHARED BY THE GUEST OPERATING SYSTEM
308
INSTRUCTIONS TO USE, BY THE HYPERVISOR, THE INFORMATION RELATED TO GUEST ADDRESS SPACE DURING PERFORMANCE OF A TASK FOR THE GUEST OPERATING SYSTEM
MACHINE-READABLE STORAGE MEDIUM
SYSTEM
300
304

Fig. 3

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/042983

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/455(2006.01)i

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F 9/455; G06F 3/06; G06F 12/10; G06F 9/46; 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: Virtual Machine (VM), Virtual Machine Monitor (VMM), paravirtualization, supervisor, guest address space, task, emulation, Translation Lookaside Buffer (TLB), Dynamically Loadable Kernel Module (DLKM), 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 2012-0047348 A1 (SCOTT W. DEVINE et al.) 23 February 2012 See paragraphs [0060]-[0096]; claim 1; and figures 5, 9.	1-2,6-9,11
A		3-5,10,12-15
A	US 2014-0101402 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 10 April 2014 See paragraphs [0048]-[0055]; claim 1; and figures 1A-1B.	1-15
A	US 2014-0173628 A1 (DYNAVISOR, INC.) 19 June 2014 See paragraphs [0069]-[0070]; claims 1, 15; and figure 2.	1-15
A	US 2014-0101362 A1 (INTERNATIONAL BUSINESS MACHINES CORP.) 10 April 2014 See paragraphs [0042]-[0048]; claim 1; and figures 1A-1B.	1-15
A	US 2014-0208034 A1 (WIND RIVER SYSTEMS, INC.) 24 July 2014 See paragraphs [0023]-[0029]; claims 1, 10; and figure 2.	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
30 March 2016 (30.03.2016)	**31 March 2016 (31.03.2016)**

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
HAN, JOONG SUB
Telephone No. +82-42-481-3578

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2015/042983

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2012-0047348 A1	23/02/2012	US 2009-300263 A1	03/12/2009
		US 2009-300264 A1	03/12/2009
		US 2009-300611 A1	03/12/2009
		US 2009-300612 A1	03/12/2009
		US 2009-300645 A1	03/12/2009
		US 2013-283004 A1	24/10/2013
		US 8074045 B2	06/12/2011
		US 8086822 B2	27/12/2011
		US 8127107 B2	28/02/2012
		US 8245227 B2	14/08/2012
		US 8464022 B2	11/06/2013
		US 8868880 B2	21/10/2014
		US 9009727 B2	14/04/2015
US 2014-0101402 A1	10/04/2014	US 2014-101361 A1	10/04/2014
		WO 2014-058762 A2	17/04/2014
US 2014-0173628 A1	19/06/2014	US 2014-173600 A1	19/06/2014
		US 2014-189690 A1	03/07/2014
		WO 2014-100273 A1	26/06/2014
		WO 2014-100279 A1	26/06/2014
		WO 2014-100281 A1	26/06/2014
US 2014-0101362 A1	10/04/2014	GB 201506442 D0	03/06/2015
		GB 2520909 A	03/06/2015
		US 2014-101365 A1	10/04/2014
		WO 2014-058774 A1	17/04/2014
US 2014-0208034 A1	24/07/2014	None	