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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 September 2010 (30.09.2010)

PCT

(10) International Publication Number
WO 2010/111149 A2

(51) International Patent Classification:
*G06F 12/02* (2006.01) *G06F 12/08* (2006.01)
*G06F 12/06* (2006.01) *G06F 13/16* (2006.01)

(21) International Application Number:
PCT/US2010/028034

(22) International Filing Date:
19 March 2010 (19.03.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
12/412,272 26 March 2009 (26.03.2009) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventor: **OSHINS, Jacob**; c/o Microsoft Corporation, International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (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, 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, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, 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, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (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, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

*[Continued on next page]*

(54) Title: VIRTUAL NON-UNIFORM MEMORY ARCHITECTURE FOR VIRTUAL MACHINES

(57) Abstract: Techniques for effectuating a virtual NUMA architecture for virtual machines and adjusting memory in virtual NUMA nodes are described herein.

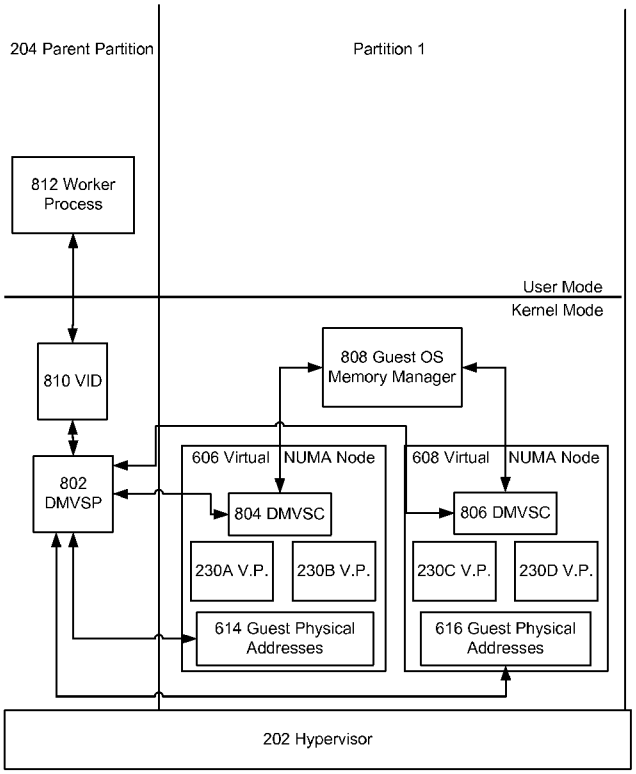



FIG. 8

WO 2010/111149 A2

**Declarations under Rule 4.17**:

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

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

**Published**:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

# VIRTUAL NON-UNIFORM MEMORY ARCHITECTURE FOR VIRTUAL MACHINES

## BACKGROUND

**[0001]** Virtualization technology allows for sharing hardware resources between multiple partitions, each of which can host a guest operating system. Generally, virtual machine technology can be used to consolidate servers and increase their portability. As virtual machines become larger, and their workloads increase, the ability to easily consolidate and/or migrate them from one computer system to another becomes more difficult. Accordingly, techniques for increasing the ability to consolidate and/or migrate larger virtual machines are desirable.

## SUMMARY

**[0002]** An example embodiment of the present disclosure describes a method. In this example, the method includes, but is not limited to receiving a request to instantiate a virtual machine, the request including a characteristic for the virtual machine; selecting a virtual NUMA node topology for the virtual machine based on the characteristic, the virtual NUMA node topology including a plurality of virtual NUMA nodes; instantiating the virtual machine on a computer system, the virtual machine including the plurality of virtual NUMA nodes; and adjusting, based on memory pressure in a specific virtual NUMA node of the plurality, an amount of guest memory assigned to the specific virtual NUMA node. In addition to the foregoing, other aspects are described in the claims, drawings, and text forming a part of the present disclosure.

**[0003]** An example embodiment of the present disclosure describes a method. In this example, the method includes, but is not limited to executing a virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system; determining memory pressure in each virtual NUMA node of the plurality; and adjusting, based on the memory pressure in each virtual NUMA node of the plurality, guest memory assigned to at least one virtual NUMA node of the plurality. In addition to the foregoing, other aspects are described in the claims, drawings, and text forming a part of the present disclosure.

**[0004]** An example embodiment of the present disclosure describes a method. In this example, the method includes, but is not limited to executing a first virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, each

virtual NUMA node of the plurality including a virtual processor and guest physical addresses, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system; and adding an additional virtual processor to a virtual NUMA node of the plurality. In addition to the foregoing, other aspects are described in the claims, drawings, and text forming a part of the present disclosure.

**[0005]** It can be appreciated by one of skill in the art that one or more various aspects of the disclosure may include but are not limited to circuitry and/or programming for effecting the herein-referenced aspects of the present disclosure; the circuitry and/or programming can be virtually any combination of hardware, software, and/or firmware configured to effect the herein-referenced aspects depending upon the design choices of the system designer.

**[0006]** The foregoing is a summary and thus contains, by necessity, simplifications, generalizations and omissions of detail. Those skilled in the art will appreciate that the summary is illustrative only and is not intended to be in any way limiting.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0007]** FIG. 1 depicts an example computer system wherein aspects of the present disclosure can be implemented.

**[0008]** FIG. 2 depicts an operational environment for practicing aspects of the present disclosure.

**[0009]** FIG. 3 depicts an operational environment for practicing aspects of the present disclosure.

**[0010]** FIG. 4 depicts how memory can be arranged in embodiments of the present disclosure.

**[0011]** FIG. 5 depicts an example operational environment practicing aspects of the present disclosure.

**[0012]** FIG.6 depicts an example operational environment practicing aspects of the present disclosure.

**[0013]** FIG. 7 depicts an example operational environment practicing aspects of the present disclosure.

**[0014]** FIG. 8 depicts an example block diagram that illustrates aspects of the present disclosure.

**[0015]** FIG. 9 depicts operational procedure for practicing aspects of the present disclosure.

**[0016]** FIG. 10 depicts an alternative embodiment of the operational procedure of FIG. 9.

**[0017]** FIG. 11 depicts operational procedure for practicing aspects of the present disclosure.

**[0018]** FIG. 12 depicts an alternative embodiment of the operational procedure of FIG. 11.

**[0019]** FIG. 13 depicts an alternative embodiment of the operational procedure of FIG. 12.

**[0020]** FIG. 14 depicts an alternative embodiment of the operational procedure of FIG. 12.

**[0021]** FIG. 15 depicts operational procedure for practicing aspects of the present disclosure.

**[0022]** FIG. 16 depicts an alternative embodiment of the operational procedure of FIG. 15.

**[0023]** FIG. 17 depicts an alternative embodiment of the operational procedure of FIG. 16.

## DETAILED DESCRIPTION

**[0024]** Embodiments may execute on one or more computers. FIG. 1 and the following discussion are intended to provide a brief general description of a suitable computing environment in which the disclosure may be implemented. One skilled in the art can appreciate that the computer system of FIG. 1 can in some embodiments effectuate computer systems 200, 300, 600, and 700. In these example embodiments, the computer systems can include some or all of the components described in FIG. 1 and circuitry configured to instantiate aspects of the present disclosure.

**[0025]** The term circuitry used through the disclosure can include hardware components such as hardware interrupt controllers, hard drives, network adaptors, graphics processors, hardware based video/audio codecs, and the firmware/software used to operate such hardware. In the same or other embodiments the term circuitry can include microprocessors configured to perform function(s) by firmware or by switches set in a certain way. In the same or other example embodiments the term circuitry can include one or more logical processors, e.g., one or more cores of a multi-core general processing unit. The logical processor(s) in this example can be configured by software instructions embodying logic operable to perform function(s) that are loaded from memory, e.g., RAM, ROM, firmware, and/or virtual memory. In example embodiments where circuitry includes a combination of hardware and software an implementer may write source code embodying logic that is subsequently compiled into machine readable code that can be processed by a logical processor. Since one skilled in the art can appreciate that the state of the art has evolved to a point where there is little difference between hardware, software, or a combination of hardware/software, the selection of hardware versus software to effectuate functions is merely a design choice. Thus, since one of skill in the art can appreciate that a software process can be transformed into an equivalent hardware

structure, and a hardware structure can itself be transformed into an equivalent software process, the selection of a hardware implementation versus a software implementation is one of design choice and left to an implementer.

**[0026]** Referring now to FIG. 1, an exemplary general purpose computing system is depicted. The general purpose computing system can include a conventional computer 20 or the like, including a logical processor 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the logical processor 21. The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory can include read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system 26 (BIOS), containing the basic routines that help to transfer information between elements within the computer 20, such as during start up, is stored in ROM 24. The computer 20 may further include a hard disk drive 27 for reading from and writing to a hard disk (not shown), a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media. The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are shown as connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer readable storage media provide non volatile storage of computer readable instructions, data structures, program modules and other data for the computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer readable storage media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs) and the like may also be used in the exemplary operating environment. Generally, in certain embodiments, such computer readable storage media can be used to store processor executable instructions embodying aspects of the present disclosure.

**[0027]** A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37 and program data 38. A user may enter commands and information into the computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a

microphone, joystick, game pad, satellite disk, scanner or the like. These and other input devices are often connected to the logical processor 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A display 47 or other type of display device can also be connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the display 47, computers typically include other peripheral output devices (not shown), such as speakers and printers. The exemplary system of FIG. 1 also includes a host adapter 55, Small Computer System Interface (SCSI) bus 56, and an external storage device 62 connected to the SCSI bus 56.

**[0028]** The computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. The remote computer 49 may be another computer, a server, a router, a network PC, a peer device or other common network node, and typically can include many or all of the elements described above relative to the computer 20, although only a memory storage device 50 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 can include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise wide computer networks, intranets and the Internet.

**[0029]** When used in a LAN networking environment, the computer 20 can be connected to the LAN 51 through a network interface or adapter 53. When used in a WAN networking environment, the computer 20 can typically include a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, can be connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used. Moreover, while it is envisioned that numerous embodiments of the present disclosure are particularly well-suited for computerized systems, nothing in this document is intended to limit the disclosure to such embodiments.

**[0030]** Referring now to FIG. 2 and 3, they depict high level block diagrams of computer systems. As shown by the figure, computer system 200 can include physical hardware devices such as a storage device 208, e.g., a hard drive, a network interface controller (NIC) 210, a graphics card 234, at least one logical processor 212, and random access

memory (RAM) 214. Computer system 200 can also include similar components as computer 20 of FIG. 1. While one logical processor is illustrated, in other embodiments computer system 200 may have multiple logical processors, e.g., multiple execution cores per processor and/or multiple processors that could each have multiple execution cores. Continuing with the description of FIG. 2, depicted is a hypervisor 202 that may also be referred to in the art as a virtual machine monitor. The hypervisor 202 in the depicted embodiment includes executable instructions for controlling and arbitrating access to the hardware of computer system 200. Broadly, the hypervisor 202 can generate execution environments called partitions such as child partition 1 through child partition N (where N is an integer greater than 1). In embodiments a child partition can be considered the basic unit of isolation supported by the hypervisor 202, that is, each child partition can be mapped to a set of hardware resources, e.g., memory, devices, logical processor cycles, etc., that is under control of the hypervisor 202 and/or the parent partition. In embodiments the hypervisor 202 can be a stand-alone software product, a part of an operating system, embedded within firmware of the motherboard, specialized integrated circuits, or a combination thereof.

**[0031]** In the depicted example the computer system 200 includes a parent partition 204 that can be also thought of as domain 0 in the open source community. Parent partition 204 can be configured to provide resources to guest operating systems executing in the child partitions 1-N by using virtualization service providers 228 (VSPs) that are also known as back-end drivers in the open source community. In this example architecture the parent partition 204 can gate access to the underlying hardware. Broadly, the VSPs 228 can be used to multiplex the interfaces to the hardware resources by way of virtualization service clients (VSCs) that are also known as front-end drivers in the open source community. Each child partition can include one or more virtual processors such as virtual processors 230 through 232 that guest operating systems 220 through 222 can manage and schedule threads to execute thereon. Generally, the virtual processors 230 through 232 are executable instructions and associated state information that provide a representation of a physical processor with a specific architecture. For example, one virtual machine may have a virtual processor having characteristics of an Intel x86 processor, whereas another virtual processor may have the characteristics of a PowerPC processor. The virtual processors in this example can be mapped to logical processors of the computer system such that the instructions that effectuate the virtual processors will be backed by logical processors. Thus, in these example embodiments, multiple virtual

processors can be simultaneously executing while, for example, another logical processor is executing hypervisor instructions. Generally speaking, and as illustrated by the figure, the combination of virtual processors, various VSCs, and memory in a partition can be considered a virtual machine such as virtual machine 240 or 242.

**[0032]** Generally, guest operating systems 220 through 222 can include any operating system such as, for example, operating systems from Microsoft®, Apple®, the open source community, etc. The guest operating systems can include user/kernel modes of operation and can have kernels that can include schedulers, memory managers, etc. Each guest operating system 220 through 222 can have associated file systems that can have applications stored thereon such as e-commerce servers, email servers, etc., and the guest operating systems themselves. The guest operating systems 220-222 can schedule threads to execute on the virtual processors 230-232 and instances of such applications can be effectuated.

**[0033]** Referring now to FIG. 3, it illustrates an alternative architecture that can be used. FIG. 3 depicts similar components to those of FIG. 2, however in this example embodiment the hypervisor 202 can include the virtualization service providers 228 and device drivers 224, and parent partition 204 may contain configuration utilities 236. In this architecture hypervisor 202 can perform the same or similar functions as the hypervisor 202 of FIG. 2. The hypervisor 202 of FIG. 3 can be a stand alone software product, a part of an operating system, embedded within firmware of the motherboard or a portion of hypervisor 202 can be effectuated by specialized integrated circuits. In this example parent partition 204 may have instructions that can be used to configure hypervisor 202 however hardware access requests may be handled by hypervisor 202 instead of being passed to parent partition 204.

**[0034]** Referring now to FIG. 4, it illustrates how memory can be arranged in embodiments that include virtual machines. For example, a computer system such as computer system 200 can have RAM 214 having memory addresses. Instead of reporting the system physical memory addresses to virtual machines, the hypervisor 202 can present different addresses for the system physical addresses, e.g., guest physical addresses (GPAs), to memory managers of guest operating systems. Guest operating systems can then manipulate the guest physical addresses and hypervisor 202 maintains the relationship by the GPAs and the SPAs. As shown by the figure, in an embodiment the GPAs and SPAs can be arranged into memory blocks. Broadly, a memory block can include one or more pages of memory. The relationship between the GPAs and the SPAs

can be maintained by shadow page table such those described in commonly assigned U.S. Patent Application No. 11/128,665 entitled "Enhanced Shadow Page Table Algorithms," the content of which is herein incorporated by reference by its entirety. In operation, when a guest operating system stores data in GPA of block 1, the data may actually be stored in a different SPA such as block 6 on the system.

**[0035]** Briefly FIG. 5 depicts an operational environment for practicing aspects of the present disclosure. For example, a number of computer systems 504-510 can be coupled together in a datacenter 500 (While four computer systems are depicted, one of skill in the art can appreciate that datacenter 500 can include more or fewer computer systems). The depicted computer systems can have different topologies and moreover, they can have different characteristics, e.g., different amounts of RAM, different RAM speeds, different amount of logical processors, and/or logical processors with different speeds.

**[0036]** The management system 502 can have components similar to computer system 20 of FIG. 1, and/or computer systems 200, 300, 600 or 700.. That is, in an embodiment the management system 502 may be a computer system that includes subject matter described below with respect to FIG. 6 or FIG. 7.

**[0037]** Continuing with the general overview of the figures, FIG. 6 depicts a computer system 600 having a symmetric multiprocessing topology (SMP) or 'flat' topology. Generally, SMP is a computer architecture that includes a plurality of processors that are connected to a single shared memory. In this arraignment, a memory controller 602 can manage the flow of data to and from memory. Memory access may be uniform with respect to each logical processor 212A-F and each logical processor can access the entire range of memory, i.e., system physical addresses 622-632. This topology works well for computer systems with a relatively small number of processors, but when the computer system includes many processors, all competing for access to the shared memory bus, performance of the system can decrease. Moreover, the complexity of the computer system significantly increases which in turn drives the price per processor up.

**[0038]** Briefly, computer system 600 can include the same, or similar components as computer 200 or 300. As shown by the figure, computer system 600 can have a plurality of logical processors 212A-212F (while six logical processors are depicted the computer system could have more or less) coupled together via a memory controller 602 that gates access to RAM 214. Similar to that described above, each logical processor 212A-212F can have different characteristics, e.g., clock speeds, cache size, etc. In this arraignment the memory controller 602 can manage the flow of data to and from the RAM 214.

**[0039]** Hypervisor 202 can be instantiated and it can control the hardware of the computer system 600. Hypervisor 202 can manage one or more virtual machines 240 through 242 that each can have virtual NUMA nodes such as virtual NUMA nodes 606-612. Virtual NUMA nodes 606-612 can be used to organize the virtual machine's resources by reporting virtual topologies to guest applications or guest operating systems such as guest operating systems 220 and 222. As shown by the figure, each virtual NUMA node 606-612 can have one or more virtual processors 230A-D, 232A-D and guest physical addresses 614-616, and 618-620. Generally, hypervisor 202 can back each virtual NUMA node 606-612 with one or more logical processors and system physical addresses from RAM 214. That is, hypervisor 202 can set one or more logical processors as idea processors that can be used to run virtual processors threads.

**[0040]** Briefly, FIG. 7 depicts a computer system 700 having a topology that includes NUMA nodes 702-706. Computer systems with NUMA nodes are can be generally though of as computers that are made up of smaller computer systems. In this example, each NUMA node 606-612 can include one or more logical processors and local memory. The memory inside of a NUMA node is considered local memory and memory in other NUMA nodes is considered remote memory because the only the processors inside of the node are connected to the same memory bus. The NUMA nodes are interconnected by cache coherency domain interconnects which allow processors in one NUMA node to access memory in other NUMA nodes in a coherent way. Thus, system physical addresses 622-632 are uniform with respect to each processor. Or put another way, system physical address 20,000 is the same for every processor in the computer system. The difference is that for some processors memory address 20,000 is a local memory address, e.g., inside their NUMA node, and for other processors memory address 20,000 is remote, e.g., outside their NUMA node. Generally, local memory can be accessed faster than remote memory and the relationship between local v. remote access time is called a NUMA ratio. A NUMA ratio of 1 to 2 means that it costs twice as many processor cycles to access a particular remote system physical address than a local system physical address. NUMA alleviates bottlenecks caused by SMP systems by limiting the number of processors on any one memory bus and is generally less expensive than a SMP computer system with the same amount of logical processors.

**[0041]** Computer system 700 can include the same, or similar components as computer 200 or 300. As shown by the figure, in this operational environment computer system 700 includes three NUMA nodes 702-706 (although the computer can have more or less)

connected by interconnects 708. As illustrated by the figure, the number of processors within each NUMA node can be variable and each node can have its own RAM.

**[0042]** Similar to FIG. 7, a hypervisor 202 can control the hardware of the computer system 700. When guest operating systems or monolithic applications boot they can detect the topologies of the virtual machines 240 and 242 similar to that described above. Each virtual NUMA node 606-612 can be assigned one or more ideal processors and system physical addresses from the same NUMA node that can be used to run virtual processor's threads.

**[0043]** While computer system 600 and 700 are depicted as including two virtual machines 240 and 242, in other embodiments they can execute more or fewer virtual machines. Moreover, while each virtual machine is depicted has having two virtual NUMA nodes, in other embodiments the virtual machines can have more or fewer virtual NUMA nodes. Also, while virtual NUMA nodes are depicted as having two virtual processors, in other embodiments the virtual NUMA nodes can have more or fewer virtual processors. Furthermore, each virtual NUMA node can have a different topology than other virtual NUMA nodes, e.g., one virtual NUMA node can have 4 virtual processors and 8 gigabytes of RAM while another virtual NUMA node can have 2 virtual processors and 4 gigabytes of RAM.

**[0044]** FIG. 8 depicts a block diagram environment that can be used in aspects of the present disclosure. As shown by the figure, a component which manages memory assigned to a virtual machine that can be known as a dynamic memory virtualization service provider 802 (DMVSP) is illustrated and can be used to adjust the amount of memory accessible to a virtual NUMA node. As shown by the figure, DMVSP 802 can be associated with one or more ballooning drivers that can be known as virtualization service clients, namely dynamic memory virtualization service clients 804 and/or 806 (DMVSCs) (while one DMVSC per virtual NUMA node is depicted, in other embodiments one DMVSC per partition may be used). Broadly, DMVSCs 804 and/or 806 can provide information that can be used by DMVSP 802 to adjust the memory of virtual NUMA nodes and each DMVSC can also help commit and de-commit memory from the virtual NUMA node it is associated with. DMVSCs 804, 806, and DMVSP 802 can communicate by way of a virtualization bus described in U.S. Patent Application No. 11/128,647, entitled "Partition Bus," the content of which is incorporated by reference in its entirety. Moreover, further aspects of DMVSCs and DMVSPs are described in U.S.

Patent Application No. 12/345,469, entitled "Dynamic Virtual Machine Memory Management," the content of which is incorporated by reference in its entirety.

**[0045]** Continuing with the description of FIG. 8, the system can include a worker process 812 is depicted that can manage the child partition(s). The worker process 812 can work in conjunction with the a virtualization infrastructure driver 810 (VID) which can allocate memory to a child partition. For example, the VID 810 can establish and remove the relationships between guest physical addresses and system physical addresses. FIG. 8 also depicts a partition that can include a guest operating system such as guest operating system 220 which can include a memory manager 808. Generally, memory manager 808 can allocate memory to applications at their request and free the memory when it is no longer needed by applications.

**[0046]** The following are a series of flowcharts depicting implementations of processes. For ease of understanding, the flowcharts are organized such that the initial flowcharts present implementations via an overall "big picture" viewpoint and subsequent flowcharts provide further additions and/or details. Furthermore, one of skill in the art can appreciate that the operational procedure depicted by dashed lines are considered optional.

**[0047]** Turning now to FIG. 9, it depicts an operational procedure for practicing aspects of the present disclosure including operations 900-910. The operational procedure begins with operation 900 and operation 902 illustrates receiving a request to instantiate a virtual machine, the request including a characteristic for the virtual machine. For example, and referring to FIG. 6 or 7, hypervisor 202 can receive a request to create a virtual machine such as virtual machine 240. For example, the request can be received from a management system 502, a parent partition 204 of FIG. 2 or 3, etc. The request can be for a new virtual machine or it can be a request to instantiate a previously saved virtual machine. When virtual machine 240 is a new virtual machine, the characteristics of virtual machine, e.g., an amount of RAM assigned to the virtual machine, a number of virtual processors, or what type of I/O devices the virtual machine should have, can be set by, for example, an administrator.

**[0048]** Continuing with the description of FIG. 9, operation 904 shows selecting a virtual NUMA node topology for the virtual machine based on the characteristic, the virtual NUMA node topology including a plurality of virtual NUMA nodes. For example, a process in the parent partition 204 (and/or hypervisor 202) can determine a topology for the virtual machine 240 based on the received characteristic. For example, parent partition 204 can include information that identifies a default size for a virtual NUMA node such as

virtual NUMA node 606. A process in parent partition 204 can use the information that describes the default size and the desired characteristic to determine the number of virtual NUMA for virtual machine 240. In a specific example, the desired characteristic may be a 6 processor virtual machine with 10 gigabytes of RAM. If the default size of a virtual NUMA node includes 2 virtual processors and 4 gigabytes of RAM then management system 502 can generate a configuration file that indicates that virtual machine 240 will include 3 virtual NUMA nodes.

**[0049]** In an embodiment the default virtual NUMA node size can be set by an administrator or by management system 502. Turning to FIG. 5, management system 502 can execute one or more programs that can obtain information that identifies the physical topology of computer systems 504-510 in datacenter 500, e.g., information that identifies how many NUMA nodes (if any) each computer system 504-510 has, how much RAM each computer system 504-510 has, the speed of the RAM, how the RAM is arranged, processor speeds, how many cores each processor has, etc.

**[0050]** Generally, the size of the virtual NUMA node influences the operation of virtual machines in datacenter 500. For example, as the size of a virtual NUMA node increases, e.g., in memory and/or processors, the portability of the virtual NUMA node decrease. Or put another way, large virtual NUMA nodes may make it more difficult to migrate the virtual machine. This occurs because a virtual NUMA node needs to be assigned to either a NUMA node or a computer system that has enough 'flat' resources to effectuate the virtual NUMA node. If, for example, a virtual NUMA node is too large, e.g., it has too RAM or too many virtual processors, it won't be able to fit in smaller NUMA nodes in datacenter 500 thus limiting the ability to migrate the virtual machine. Moreover, if the larger virtual NUMA node is simply assigned to multiple smaller NUMA nodes performance of the virtual machine will decrease because of the difference that exists between local memory and remote memory access times.

**[0051]** On the other hand, as the size of the virtual NUMA node decreases performance of the guest operating system may be adversely impacted. This inefficiency may occur because the guest operating system will try to segregate applications and it's own execution to single virtual NUMA nodes. The guest operating system will be constrained in this case and performance will decrease.

**[0052]** Accordingly, in an embodiment management system 502 can strike a balance between portability and efficiency by determining an optimum virtual NUMA node size for datacenter 500. For example, in an embodiment a logical processor of management

system 502 can execute a program and determine the average size of a NUMA node in the datacenter, e.g., the average number of logical processors, average amount of RAM, etc., and set the size of the virtual NUMA node to be the same or smaller than the average NUMA node in the system. In another embodiment the program can be configured to set the size of the virtual NUMA node to be slightly smaller than the smallest NUMA node in the datacenter 500. In an embodiment, the size of the virtual NUMA node can be set to be slightly smaller than the average size or smallest size so that if the computer system becomes heavily committed, more than one virtual NUMA node can be assigned to a single NUMA node. In a specific example, if the smallest NUMA node has 4 logical processors and 8 gigabytes of RAM, then the size of the virtual NUMA node could be set to, for example, 2 virtual processors and 4 gigabytes of RAM.

**[0053]** Operation 906 shows instantiating the virtual machine on a computer system, the virtual machine including the plurality of virtual NUMA nodes. In an embodiment, hypervisor 202 can be executed by a logical processor and a virtual machine having the plurality of virtual NUMA nodes can be instantiated. For example, and referring to FIG. 6 and/or FIG. 7, virtual machine 240 having virtual NUMA nodes 606-608 can be effectuated by computer system 600 or 700. That is, VID 810 can back guest physical addresses of virtual machine 240 with system physical addresses from RAM and virtual processors with one or more logical processors. For example, blocks of guest physical addresses 614 can be backed with blocks of system physical addresses 622 and blocks of guest physical addresses 616 can be backed by blocks of system physical addressees 624. Hypervisor threads can then be scheduled on the logical processors backing the virtual processors and instructions indicative of the virtual processors can be executed. As shown by FIG. 6 and FIG. 7, the topology of each virtual machines can be created independently from the topology of the underlying hardware. That is, each virtual machine's topology is decoupled from the underlying physical topology of the computer system that effectuates it.

**[0054]** In an embodiment the virtual machine BIOS or boot firmware can describe the virtual machine's topology, e.g., whether it has virtual NUMA nodes, the size of any virtual NUMA nodes, and NUMA ratios for the virtual NUMA nodes, to a guest operating system of monolithic application. The data structure can be processed and guest OS 220 or an application and it can be used to by the OS or application to take advantage of the presence of virtual NUMA nodes. For example, guest operating system 220 can try to affinitize the threads of an application that is not NUMA aware to a virtual NUMA node so that execution of the application remains local. In another example a database

management program such as SQL Server can allocate locks local to the virtual NUMA nodes and the database can split up read/write requests across the virtual NUMA nodes. In yet another example the guest operating system 220 can create page pools for each virtual NUMA node in the virtual machine.

**[0055]** Continuing with the description of FIG. 9, operation 908 shows adjusting, based on memory pressure in a specific virtual NUMA node of the plurality, an amount of guest memory assigned to the specific virtual NUMA node. For example, a logical processor, logical processor 212A of FIG. 6 or 7 for example, can execute instructions indicative of DMVSP 802 and can adjust the amount of guest physical addresses that are available to a virtual NUMA node such as virtual NUMA node 606. That is, the DMVSP 802 can be executed and memory can be committed or de-committed based on the pressure that the virtual NUMA nodes are experiencing.

**[0056]** In an embodiment, memory pressure can identify how performance of guest operating system 220 is affected by the amount of memory that is available to each virtual NUMA node 606-608. This information can be calculated during the runtime of guest operating system 220 by, for example, one or more DMVSCs such as DMVSC 804 and/or 806 and sent to DMVSP 802. For example, memory pressure can be represented by a series of values that can identify a different level of memory pressure in a virtual NUMA node. As the resources in a virtual NUMA node become more stressed, i.e., as the amount of memory required to efficiently execute the current workload on a virtual NUMA node increases, the DMVSC 804 can revise the value and communicate this information to the DMVSP 802.

**[0057]** In an embodiment, the memory pressure information can be calculated by DMVSC 804 from information received from guest operating system 220. For example, DMVSC 804 can be configured to receive operating system paging information from the memory manager 808 for the virtual NUMA node 606. Guest operating system paging rates can be monitored through two counters that are exposed by memory manager 808 and a cache manager, namely the paging rate and the cache rotation rate.

**[0058]** In the same or another embodiment DMVSC 804 can receive physical memory notifications associated with virtual NUMA node 606 from memory manager 808 and use this information to compute the memory pressure of virtual NUMA node 606. For example, memory manager 808 can output high-memory notifications and low-memory notifications based on activity in guest operating system 220 that is associated with virtual NUMA node 606. Memory manager 808 can trigger these notifications based on a low-

memory threshold (LMT) and a high-memory threshold (HMT). In a specific example embodiment, the default level of available memory that signals a low-memory-resource notification event can be approximately 32 MB per 4 GB, to a maximum of 64 MB. The default level that signals a high-memory-resource notification event can be, for example, three times the default low-memory value. Intermediate memory availability levels between the two can be determined by dividing the interval between the high-memory threshold and low-memory threshold levels. One skilled in the art can appreciate that these values are exemplary and that changes can be made without departing from the spirit of the disclosure.

**[0059]** These notifications, along with others, can be used by DMVSC 804 to calculate the memory pressure of virtual NUMA node 606. For example, each level can be associated with a value, e.g., 0-4 and if any other performance counters are taken into account they can also be associated with values. The values for each performance counter can then be used to calculate the current memory pressure of virtual NUMA node 606. In a specific example, the memory pressure can be calculated by taking the higher or lower of the performance counter values. In another example the average value of the performance counters can be used as the memory pressure. In yet another embodiment a more sophisticated algorithm can be used to calculate the memory pressure that takes into account the previous performance counter values and assigns each performance counter a scalar to affect its relative weight in the calculation.

**[0060]** When a decision to commit memory is made DMVSP 802 can use a variety of techniques, one of which is a hot-add operation. For example, certain operating systems can support a hot-add, which allows ranges of physical memory to be added to a running operating system without requiring a system reboot. That is, memory manager 808 can be configured to support dynamic addition of memory to a running system. In a hot-add embodiment, DMVSC 804 can be configured to access a hot-add interface of memory manager 808 and DMVSC 804 can send a message to guest operating system 220 that describes the hot-added GPAs and which virtual NUMA node they are associated with. Memory manager 808 can then make the new memory available to guest operating system 220, drivers, applications, or any other processes that are running on virtual NUMA node 606. For example, DMVSC 804 can receive the hot-added memory addresses from DMVSP 802 after VID 810 generates the relationships between GPAs and SPAs.

**[0061]** Similarly, a hot-remove operation can be used to de-commit memory addresses from a virtual NUMA node such as virtual NUMA node 606. For example, DMVSC 804

can send a message to guest operating system 220 that indicates that memory has been hot-removed. DMVSC 804 can request that memory manager 808 provide blocks of GPAs from virtual NUMA node 606 for removal. In this example DMVSC 804 can then call a removal API of memory manager 808 and remove the GPAs from guest operating system 220. In an embodiment where hot-remove is used, the memory that is removed is not counted against the guest current commit, and memory manager 808 can adjust its internal counters using similar techniques that are used by an operating system to remove memory that is physically removed from a motherboard.

**[0062]** In another embodiment memory can be de-committed to a virtual NUMA node by using a ballooning technique. That is, memory can be de-committed by dissociating guest physical addresses in virtual NUMA 606 node from the physical addresses that back them. For example, logical processor 212B can execute instructions indicative of the DMVSC 804 and can send a message to memory manager 808 requesting that memory manager 808 reserve a certain amount of memory for use by DMVSC 804, e.g., one or more memory blocks. Memory manager 808 can lock memory for exclusive use within DMVSC 804 and DMVSC 804 can send the GPAs of the memory to DMVSP 802. In this example DMVSP 802 can send the GPAs to VID 810 and VID 810 can remove entries for these GPAs to SPAs in the shadow page table. In this example, memory manager 808 may include information that identifies that the GPAs are still valid, however in actuality the GPAs are no longer backed by system physical addresses. In this example memory manager 808 won't use the locked GPAs and the SPAs backing them can be reallocated.

**[0063]** Guest physical addresses that have been dissociated can be re-associated with physical addresses. In this example a request to commit memory pages can be received by VID 810 and VID 810 can obtain SPAs to satisfy the request and send the range of addresses to DMVSP 802. In an embodiment VID 810 can be configured to obtain a contiguous range of SPAs in order to increase system efficiency. In this example VID 810 can determine that the guest operating system 220 has GPAs that are locked for exclusive use by the DMVSC 804 associated with virtual NUMA node 606. VID 810 can create relationships between the locked GPAs and the SPAs and send a message to DMVSP 802. DMVSP 802 can then send a message to DMVSC 804 and DMVSC 804 can send a message to memory manager 808 indicating that the GPAs can be unlocked and returned to a memory pool of memory manager 808 that is associated with virtual NUMA node 606.

**[0064]** In an embodiment VID 810 can determine whether to use a hot-add technique or a ballooning technique depending on whether GPAs are ballooned. For example, when VID 810 receives SPAs to commit to NUMA node 606 it can determine whether any GPAs are locked by DMVSC 804. In the instance that there are locked GPAs, VID 810 can back them with SPAs before it hot-adds memory. Before memory is committed to virtual NUMA node 606 it can be zeroed and its associated cache lines can be flushed for security reasons. By zeroing the memory the contents of memory previously associated with one partition do not leak to another partition.

**[0065]** Turning now to FIG. 10, it depicts an alternative embodiment of the operational procedure of FIG. 9 including additional operations 1010-1020. Operation 1010 shows determining that memory pressure in a second virtual NUMA node of the plurality is greater than a predetermined value; and migrating the second virtual NUMA node to a second NUMA node of the computer system. For example, and turning to FIG. 7, in an embodiment the memory pressure in a second virtual NUMA node 608 can increase. That is, a value indicative of the memory pressure can be received by the DMVSP 802 which indicates that virtual NUMA node 608 is stressed. In this example virtual machine 240 or individual virtual NUMA node 608 can have a target pressure value and the current pressure value may be greater than a target value that was set by an administrator. The target pressure values can be stored in a data structure that can be accessed by DMVSP 802. Current pressure values of the running virtual machines or virtual NUMA nodes can then be received. DMVSP 802 may serially step through the list of running virtual machines or virtual NUMA nodes and commit memory in order to reduce the memory pressure values to the target values, and de-commit memory to increase the pressure to the target values.

**[0066]** In an example, DMVSP 802 can be configured to determine that the NUMA node currently hosting virtual NUMA nodes 606 and 608, e.g., NUMA node 702, is unable to allocate enough memory in order to obtain the target memory pressure values for both virtual NUMA nodes. In this example DMVSP 802 can be configured to send a signal to hypervisor 202 and hypervisor 202 can be configured to attempt to move one of the virtual NUMA nodes off NUMA node 702. Hypervisor 202 can check the current workload of NUMA nodes 702-706 and determine that, for example, NUMA node 704 can host the virtual NUMA node and allocate enough resources to it to decrease the memory pressure to the target value. In this example hypervisor 202 can be configured to reassign virtual NUMA node 608 to NUMA node 704. That is, hypervisor 202, in conjunction with VID

810, can remap the guest physical addresses 616 to system physical addresses 714, and set logical processors 212E and F as ideal processors for virtual processors 230C and D.

**[0067]** Continuing with the description of FIG. 10, operation 1012 illustrates de-committing at least one memory block of guest memory from the specific virtual NUMA node; and committing the de-committed at least one memory block of guest memory to a second virtual NUMA node. For example, DMVSP 802 can be configured to de-commit memory from, for example, virtual NUMA node 606 and commit the memory to virtual NUMA node 608. In this example, virtual NUMA nodes 606 and 608 may be backed by a single NUMA node, or a 'flat' architecture. In this example embodiment the DMVSP 802 can attempt to free up memory from virtual NUMA node 606 when, for example, there is no available memory available that can be committed to virtual NUMA node 608. In another example, DMVSP 802 can be configured to de-commit memory from, for example, virtual NUMA node 610 and commit the memory to virtual NUMA node 608. That is, memory can be taken from one virtual machine and given to another.

**[0068]** In a specific example, and referring to FIG. 6, virtual NUMA nodes 606 and 608 can be mapped to the resources of computer system 600. In this example, DMVSP 802 can check the other virtual NUMA nodes in, for example, order of memory priority starting with, for example, low priority virtual NUMA nodes in virtual machine 240 or the lowest-priority virtual machine. If, for example, a virtual NUMA node is detected, such as virtual NUMA node 606, that has a memory pressure value that is less than a target threshold, the DMVSP 802 can initiate a memory de-commit and remove memory from virtual NUMA node 606. After the de-commit is complete a commit operation can be initiated and memory can be hot-added to virtual NUMA node 608 or ballooned guest physical addresses can be re-associated with system physical addresses.

**[0069]** In a specific example, and referring to FIG. 7, DMVSP 802 can check the other virtual NUMA nodes that are backed by the same NUMA node 702 in, for example, order of memory priority. If, for example, a virtual NUMA node on the same NUMA node as virtual NUMA node 608 is detected that has a memory pressure value that is less than a target threshold, DMVSP 802 can initiate a memory de-commit. After the de-commit is complete a commit operation can be initiated and memory can be hot-added to virtual NUMA node 608 or ballooned guest physical addresses can be re-associated with system physical addresses.

**[0070]** Continuing with the description of FIG. 10, operation 1014 depicts determining that at least one block of guest memory of the specific virtual NUMA node is dissociated

with system memory; and mapping the at least one memory block of guest memory onto at least one memory block of system memory. For example, in an embodiment DMVSP 802 can be executed by a logical processor and a determination can be made to back GPAs in virtual NUMA node 606 with SPAs 624. For example, GPAs may be reserved by DMVSC 804 and the SPAs may be reallocated to either another virtual NUMA node or the parent partition 204. In this example a request to commit memory pages can be received by VID 810 and VID 810 can obtain SPAs to satisfy the request and send the range of addresses to the DMVSP 802. In an embodiment VID 810 can be configured to obtain a contiguous range of SPAs in order to increase system efficiency. In a NUMA embodiment, VID 810 can be configured to obtain a contiguous range of SPAs from the same NUMA node that is running virtual NUMA node 606. VID 810 can create relationships between the locked GPAs and the SPAs and send a message to DMVSP 802. DMVSP 802 can then send a message to DMVSC 804 and DMVSC 804 can send a message to memory manager 808 indicating that the GPAs can be unlocked and returned to a memory pool associated with virtual NUMA node 606.

**[0071]** Continuing with the description of FIG. 10, operation 1016 illustrates mapping the specific virtual NUMA node onto a first NUMA node of the computer system; and migrating the specific virtual NUMA node onto a second NUMA node of the computer system. For example, and referring to FIG. 7, guest operating system 220 can be spread across at least two NUMA nodes such as NUMA nodes 702 and 704. For example, and referring to FIG. 7, hypervisor 202 can schedule virtual NUMA nodes 606 and 608 to run on NUMA node 702. In this example hypervisor 202 can receive a signal that indicates that NUMA node 702 is stressed. For example, guest operating system 220 may generate signals that indicate that virtual NUMA nodes 606 and 608 are low on memory. In this example, hypervisor 202 can be configured to reduce the workload on the stressed NUMA node by moving virtual NUMA node 608 off NUMA node 702.

**[0072]** Continuing with the description of FIG. 10, operation 1018 illustrates adding a virtual processor to the specific virtual NUMA node. For example, in an embodiment a virtual processor such as virtual processor 230B can be added during the runtime execution of virtual machine 240 using, for example, a processor hot-add operation. That is, virtual NUMA node 606 may only have a single virtual processor 230A at one point and then another is added. In an embodiment the newly added processor can be assigned to the processor backing virtual processor 230A or another logical processor can be allocated to run virtual processor 230B threads. In a NUMA embodiment, if a another

logical processor is being used to back the virtual processor 230B, it can be allocated from the same NUMA node 702 that is backing other virtual processors in virtual NUMA node 606.

**[0073]** Continuing with the description of FIG. 10, operation 1020 illustrates receiving a request to execute a virtual processor of the virtual machine, the virtual processor assigned to a logical processor, the logical processor assigned to a NUMA node and the virtual processor assigned to the virtual NUMA node; determining that the logical processor can not execute the virtual processor; and selecting a second logical processor to execute the virtual processor, the second logical processor from a second NUMA node. For example, and referring to FIG. 7, in an embodiment hypervisor 202 can receive a request to execute a virtual processor thread from virtual processor 230A and attempt to schedule the thread on an idea processor 212A, e.g., a processor backing virtual processor 230A. In this example, hypervisor 202 can detect that logical processor 212A is overcommitted and unable to execute the virtual processor thread. In this case, hypervisor 202 can be executed and it can select another logical processor to execute the virtual processor thread. For example, hypervisor 202 can attempt to select a different logical processor on the same NUMA node. If, for example, the NUMA node is overcommitted hypervisor 202 can be configured to select a remote processor to execute virtual processor 230A, e.g., logical processor 212E. In this example the decision as to whether or not to wait or schedule the thread on a remote node can be made using the NUMA ratio associated with NUMA node 704. If the NUMA ratio is low, and the expected wait for the ideal processor is long, then the decision to schedule the thread on NUMA node 704 can be made. On the other hand, if the NUMA ratio is high, and the expected wait time is low, then the decision to wait can be made.

**[0074]** Turning now to FIG. 11, it depicts an operational procedure for practicing aspects of the present disclosure including operations 1100, 1102, 1104, and 1106. Operation 1100 beings the operational procedure and operation 1102 shows executing a virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system. For example, hypervisor 202 can execute a virtual machine having a plurality of virtual NUMA nodes. As shown by FIG. 6, virtual machine 240 can be created that includes virtual NUMA nodes 606 and 608. The virtual NUMA nodes can each have one or more virtual processors 230A-D and guest physical addresses 614 and 616. In this embodiment virtual NUMA nodes 606 and 608 can be

created independently from the topology of the underlying hardware. That is, the topology of the virtual machine is unrelated to the underlying hardware such as is depicted by FIG. 6 and FIG. 7. Thus, in this embodiment, each virtual machine's topology is decoupled from the underlying physical topology of the computer system that effectuates it.

**[0075]** Continuing with the description of FIG. 11, operation 1104 illustrates determining memory pressure in each virtual NUMA node of the plurality. For example, and referring to FIG. 8, memory pressure for each virtual NUMA node 606 and 608 can be obtained, e.g., generated and/or received. Memory pressure information can identify how performance of the guest is affected by the amount of memory that is available to each virtual NUMA node 606-608. This information can be calculated during the runtime of guest operating system 220 by, for example, one or more DMVSCs such as DMVSC 804 and/or 806 and sent to the DMVSP 802. That is, in a specific embodiment a logical processor can execute instructions indicative of the DMVSC 804 or 806 and generate memory pressure information for each virtual NUMA node. This information can then be sent to the DMVSP 802 for example.

**[0076]** In an example embodiment the memory pressure information can include a series of values ranging from 0-4, and each value can identify a different level of memory pressure that the guest OS is experiencing due to the resources of the virtual NUMA nodes 606-608. As the guest operating system becomes more stressed, i.e., as the amount of memory required to efficiently execute the current workload increases, the DMVSCs 804 and 806 can revise their values and communicate this information to the DMVSP 802.

**[0077]** Continuing with the description of FIG. 11, operation 1106 shows adjusting, based on the memory pressure in each virtual NUMA node of the plurality, guest memory assigned to at least one virtual NUMA node of the plurality. In an embodiment that includes operation 1206, a logical processor 212A can execute instructions indicative of DMVSP 802 and can adjust the amount of guest physical addresses in a virtual NUMA node 606 for example. That is, DMVSP 802 can commit or de-commit memory based on the memory pressure in virtual NUMA node 606, e.g., if processes allocated to virtual NUMA node 606 are stressed, memory can be committed.

**[0078]** In an embodiment, when the logical processor 212, configured by DMVSP 802, determines to commit or de-commit memory it can do so on a per memory block basis. For example, DMVSP 802 can commit/de-commit a memory block and check how the memory status changes. If memory status has not changed, DMVSP 802 can commit/de-commit another memory block.

**[0079]** Turning now to FIG. 12, it depicts an alternative embodiment of the operational procedure 1100 of FIG. 11 including operations 1208-1216. As illustrated by the figure, operation 1208 shows sending the virtual machine to a second computer system. For example, and referring to FIG. 5, in an embodiment the state of a virtual machine can be saved in one or more configuration files and sent to another computer system, e.g., sent from computer 504 to 506. The hypervisor of computer system 506 can read the file or files and instantiate the virtual machine.

**[0080]** Virtual machine topology affects the ability to migrate and restore virtual machines. Specifically, the decision to allow the topology of the underlying hardware to be detected and the size of the virtual NUMA nodes influence how well a virtual machine will execute and whether it can be easily migrated. For example, the size of the virtual NUMA nodes affects the ability to migrate, e.g., as the size of a virtual NUMA node increases the portability of the virtual NUMA node decrease and as the size of the virtual NUMA node decreases so does performance of the virtual machine. In addition, virtual machines that can detect underlying computer's topology can not be easily migrated due to the fact that NUMA aware operating systems and applications optimize themselves at boot time based on first topology they detect and these optimizations may not work well on computers that the virtual machine may be migrated to in the future. Thus, by exposing virtual NUMA nodes to the guest operating system when it boots the operating system can be optimized to use NUMA nodes. By sizing the virtual NUMA nodes correctly, the virtual machine can be optimized for many diverse computer systems in datacenter 500.

**[0081]** For example, and referring to FIG. 6, virtual machine 240 can include two or more virtual NUMA nodes 606 and 608. The hypervisor 202 can back the virtual processors 230A-D with logical processors 212A-D. When the guest operating system 220 boots it can detect the virtual NUMA nodes 606 and 608 and can be configured to optimize the scheduling and execution of processes to use the virtual NUMA nodes 606 and 608. After some time the virtual machine 240 may be migrated to a computer system having a physical topology similar to that depicted by FIG. 7. The hypervisor 202 of FIG. 7 can back the virtual processors 230A and B with logical processors 212A and B and back virtual processors 230C and D with logical processors 212E and F. Guest operating system 220 can continue to operate in the same manner as it was when running on the computer system of FIG. 6 even though the underlying computer topology has changed from SMP to NUMA.

**[0082]** Continuing with the description of FIG. 12, operation 1210 shows mapping a first virtual NUMA node of the plurality onto a first NUMA node of the computer system; and mapping a second virtual NUMA node of the plurality onto the first NUMA node of the computer system. For example, and referring to FIG. 7, a logical processor such as logical processor 212A can execute hypervisor instructions and can affinitize virtual NUMA nodes such as virtual NUMA node 606 and 608 to a NUMA node such as NUMA node 702. More specifically, the logical processor can back guest physical addresses 614 and 616 with system physical addresses from RAM 214 of NUMA node 702 and can back virtual processors 230A through D with logical processors 212A through D.

**[0083]** Continuing with the description of FIG. 12, operation 1212 shows mapping a first virtual NUMA node of the plurality onto a first NUMA node of the computer system; and mapping a second virtual NUMA node of the plurality onto a second NUMA node of the computer system. For example, and referring to FIG. 7, a logical processor such as logical processor 212A can execute hypervisor instructions and can assign a virtual NUMA node 606 to NUMA node 702 and assign virtual NUMA node 608 to NUMA node 704. In this example, when hypervisor 202 is executed, a hypervisor scheduler can schedule threads from virtual processors 230A-B on logical processors 212A-D and schedule threads from virtual processor 230C or D on logical processors 212E through G.

**[0084]** Turning now to FIG. 13, it illustrates an alternative embodiment of the operational procedure of FIG. 12 including operation 1314 which shows determining that memory pressure in the second virtual NUMA node is greater than a predetermined value; and migrating the second virtual NUMA node to a second NUMA node of the computer system. For example, and turning to FIG. 7, in an embodiment the memory pressure in the second virtual NUMA node 608 can increase. That is, a value indicative of the memory pressure can be received by the DMVSP 802 which indicates that the virtual NUMA node 608 is stressed. In this example each virtual NUMA node 606-612, and/or virtual machine 240-242, can have a target pressure value and the current pressure value for virtual NUMA node 608 may be greater than a target value that was set by an administrator. The target pressure values can be stored in a data structure that can be accessed by DMVSP 802. Current pressure values of the running virtual machines or virtual NUMA nodes can then be received. DMVSP 802 may serially step through the list of running virtual machines or virtual NUMA nodes and commit memory in order to reduce the memory pressure values to the target values, and de-commit memory to increase the pressure to the target values.

**[0085]** Turning now to FIG. 14, it illustrates an alternative embodiment of the operational procedure of FIG. 12 including operation 1416 which shows determining that memory pressure of a second virtual machine is greater than a predetermined value; and migrating the second virtual NUMA node of the virtual machine to the first NUMA node of the computer system. In an embodiment at least two virtual machines can be executing, e.g., virtual machines 240 and 242 of. In this example virtual NUMA nodes of virtual machine 240 can be mapped to, for example, NUMA nodes 702 and 704 of FIG. 7 and the virtual NUMA nodes of virtual machine 242 can be mapped to, for example NUMA node 706. In this example each virtual machine 240 and 242, and/or each virtual NUMA node 606-612 can have a target pressure value that can be stored in a data structure that can be accessed by DMVSP 802. In this example the memory pressure in the second virtual machine 242 may increase due to activity in the virtual machine, e.g., virtual machine 242 received many read/write requests, and the value can be received by DMVSP 802. DMVSP 802 can receive current pressure values of the running virtual machines or virtual NUMA nodes and serially step through the list of running virtual machines or virtual NUMA nodes to determine whether memory can be committed to virtual machine 242 in order to alleviate the memory pressure.

**[0086]** In a situation where the pressure can not be relieved by committing or de-committing memory, DMVSP 802 can be configured to send a signal to hypervisor 202 and hypervisor 202 can attempt to reallocate the resources of computer system in order to relieve memory pressure. For example, hypervisor 202 can check the current workload of the NUMA nodes 702-706 and determine that, for example, NUMA node 702 can host virtual NUMA nodes from virtual machine 240 and reassign virtual NUMA node 608 to NUMA node 702. That is, hypervisor 202, in conjunction with VID 810, can remap the guest physical addressee 616 to system physical addresses 712, and set logical processors 212A and D as ideal processors for virtual processors 230C and D. Then hypervisor can remap virtual NUMA node 610 to NUMA node 704 and adjust the memory inside each virtual NUMA node 610-612 of virtual machine 242 in order to decrease its memory pressure.

**[0087]** Turning now to FIG. 15, it illustrates an operational procedure for practicing aspects of the present disclosure including operations 1500, 1502, and 1504. Operation 1500 begins the operational procedure and operation 1502 shows executing a first virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, each virtual NUMA node of the plurality including a virtual processor and guest

physical addresses, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system. For example, hypervisor 202 of FIG. 7 can execute virtual machine 240 having a plurality of virtual NUMA nodes 608-610. The virtual NUMA nodes 606 and 608 can each have one or more virtual processors 230A-D and guest physical addresses 614 and 616. In this embodiment the virtual NUMA nodes 606 and 608 can be mapped to the resources of the computer system 700. For example, logical processors 212A and B can be set as ideal processors for virtual processor 230A and B and guest physical addresses 614 can be backed by system physical addresses 710. Likewise, logical processors 212E and F can be set as ideal processors for virtual processor 230C and D and guest physical addresses 616 can be mapped to system physical addresses 714. In this embodiment the virtual NUMA nodes 606 and 608 are independent from the topology of the underlying hardware. That is, the topology of the virtual machine is unrelated to the underlying hardware such as is depicted by FIG. 6 and FIG. 7. Thus, in this embodiment, the virtual machine's topology is decoupled from the underlying physical topology of the computer system that effectuates it.

**[0088]** Continuing with the description of FIG. 15, operation 1504 shows adding an additional virtual processor to a virtual NUMA node of the plurality. For example, in an embodiment an additional virtual processor can be added to a virtual NUMA node such as, for example, virtual NUMA node 606. In this example a virtual processor such as virtual processor can be added during the runtime execution of virtual machine 240 using, for example, a processor hot-add operation. In an embodiment the newly added processor can be assigned to the processor backing virtual processor 230A or another logical processor can be set as the ideal processor to run the newly added virtual processor's threads. In a NUMA embodiment logical processors can be allocated from the same NUMA node 702 that is backing the virtual NUMA node 606.

**[0089]** Turning now to FIG. 16, it depicts an alternative embodiment of the operational procedure depicted by FIG. 15 including additional operations 1606-1612. Operation 1606 shows determining memory pressure in each virtual NUMA node of the plurality; and adjusting, based on the memory pressure in each virtual NUMA node of the plurality, guest memory assigned to at least one virtual NUMA node of plurality. Referring to FIG. 8, memory pressure for each virtual NUMA node 606 and 608 can be obtained, e.g., generated and/or received. Memory pressure information can identify how performance of the guest is affected by the amount of memory that is available to each virtual NUMA node. DMVSCs 804 and 806 can be configured to receive, for example, physical memory

notifications and/or guest operating system paging information from memory manager 808 and use it to compute the memory pressure of each virtual NUMA node 606 and 608.

**[0090]** Continuing with the example, a logical processor 212A for example, can execute instructions indicative of DMVSP 802 and can adjust the amount of guest physical addresses in a virtual NUMA node 606 for example. That is, DMVSP 802 commit or de-commit memory based on the memory pressure that the guest OS is experiencing due to the resources in of the virtual NUMA node 606.

**[0091]** Continuing with the description of FIG. 16, operation 1608 illustrates removing a virtual processor from a virtual NUMA node of the plurality. In an embodiment that includes 1706, hypervisor 202 can be executed by a logical processor and a virtual processor can be removed from the virtual NUMA node 606. For example, the hypervisor 202 can access a hot-remove API of the guest operating system 220 and remove, for example, virtual processor 230B from virtual NUMA node 608.

**[0092]** Turning now to operation 1610, it shows reporting NUMA ratios for the plurality of virtual NUMA nodes to a guest operating system. For example, in an embodiment the hypervisor 202 can generate NUMA ratios for virtual NUMA nodes 606-608 and this information can be reported to the guest operating system 220 of either FIG. 6 or FIG. 7. In an embodiment hypervisor 202 can create a data structure in a virtual machine's firmware table that indicates the NUMA ratio for the virtual NUMA nodes and when the guest operating system 220 boots the guest can read the table and use the information to make scheduling decisions. For example, a guest operating system, or NUMA aware application, may use the NUMA ratios to determine whether or not to use resources from a remote NUMA node. For example, an operating system may have a pending thread that is ready to be executed. The operating system in this example can also be configured to wait a certain amount of time for an ideal processor to become free otherwise it will schedule the thread on a remote processor that has less than a predetermined NUMA ratio. In this case the amount of time the scheduler is willing to wait is dependent upon the NUMA ratio.

**[0093]** Turning now to operation 1612, it shows migrating the virtual machine from a first NUMA node to a plurality of NUMA nodes. For example, in an embodiment hypervisor instructions can be executed by a logical processor and virtual machine 240 can be mapped to a plurality of NUMA nodes 704 and 706. In this example computer system 700 may have been under heavy use, e.g., virtual machine 242 may have been using a majority of the resources, and so virtual machine 242 was migrated off computer 700. In this

situation hypervisor 202 can reallocate the resources of the computer system 700 and remap virtual machine 240 to NUMA nodes 704 and 706.

**[0094]** Turning now to FIG. 17, it depicts an alternative embodiment of the operational procedure of FIG. 16 including operations 1714 and 1716. In an embodiment operation 1606 can include operation 1714 which shows de-committing memory from a first virtual NUMA node of the plurality based on a determination that the current memory pressure of the virtual NUMA node is lower than a target threshold. For example, in an embodiment DMVSP 802 can be configured to de-commit memory from a virtual NUMA node 606 when the memory pressure of the virtual NUMA node 608 is lower than a target threshold. For example, in an embodiment the DMVSP 802 can attempt to free up memory from virtual NUMA node 606 when, for example, there is no memory available in the parent partition that can be committed and the virtual NUMA node 608 is experiencing unacceptable pressure. If memory is de-committed from a virtual NUMA node 606, an asynchronous message can be sent to the DMVSC 804 directing it to de-commit memory. When the DMVSC 804 associated with guest OS 220 responds, it can indicate new memory pressure within the virtual NUMA node 606. In some cases, the memory pressure may be increased in response to the memory removal operation.

**[0095]** Continuing with the description of FIG. 17, operation 1716 shows committing memory to a first virtual NUMA node of the plurality based on a determination that the current memory pressure of a guest operating system is greater than a target threshold. For example, in an embodiment DMVSP 802 can be configured to commit memory to virtual NUMA node 606 when the memory pressure of virtual NUMA node 606 is greater than a target threshold. In this case, if memory is available it can be allocated to the virtual NUMA node 606. That is, the DMVSP 802 can obtain memory pressure information that identifies how performance of the virtual NUMA node 606 is affected by the amount of memory available and add memory to the virtual NUMA node 606. In a specific example the memory pressure information can be a value. In this example the DMVSP 802 can compare the current memory pressure value to a table of information that indicates a minimum value for the virtual NUMA node 606 and adjust the memory until the memory pressure of the virtual NUMA node 606 is equal to the minimum. For example, an administrator may configure a guest operating system that runs a critical application to have a low minimum values for the virtual NUMA nodes 606 and 608.

**[0096]** The foregoing detailed description has set forth various embodiments of the systems and/or processes via examples and/or operational diagrams. Insofar as such block

diagrams, and/or examples contain one or more functions and/or operations, it will be understood by those within the art that each function and/or operation within such block diagrams, or examples can be implemented, individually and/or collectively, by a wide range of hardware, software, firmware, or virtually any combination thereof.

**[0097]** While particular aspects of the present subject matter described herein have been shown and described, it will be apparent to those skilled in the art that, based upon the teachings herein, changes and modifications may be made without departing from the subject matter described herein and its broader aspects and, therefore, the appended claims are to encompass within their scope all such changes and modifications as are within the true spirit and scope of the subject matter described herein.

**What is Claimed:**

1. A method, comprising:

receiving a request to instantiate a virtual machine (240), the request including a characteristic for the virtual machine (240);

selecting a virtual NUMA node topology for the virtual machine (240) based on the characteristic, the virtual NUMA node topology including a plurality of virtual NUMA nodes (606-608);

instantiating the virtual machine (240) on a computer system (606), the virtual machine including the plurality of virtual NUMA nodes; and

adjusting, based on memory pressure in a specific virtual NUMA node (606) of the plurality, an amount of guest memory assigned to the specific virtual NUMA node.

2. The method of claim 1, further comprising:

determining that memory pressure in a second virtual NUMA node (608) of the plurality is greater than a predetermined value; and

migrating the second virtual NUMA node (608) to a second NUMA node (704) of the computer system (700).

3. The method of claim 1, wherein adjusting the amount of guest memory further comprises:

de-committing at least one memory block of guest memory from the specific virtual NUMA node (606); and

committing the de-committed at least one memory block of guest memory to a second virtual NUMA node (608).

4. The method of claim 1, wherein adjusting the amount of guest memory further comprises:

determining that at least one block of guest memory of the specific virtual NUMA node (606) is dissociated with system memory; and

mapping the at least one memory block of guest memory onto at least one memory block of system memory.

5. The method of claim 1, further comprising:

mapping the specific virtual NUMA node (606) onto a first NUMA node of the computer system (702); and

migrating the specific virtual NUMA node (606) onto a second NUMA node of the computer system (704).

6. The method of claim 1, further comprising:

adding a virtual processor (230B) to the specific virtual NUMA node (606).

7. The method of claim 1, further comprising:

receiving a request to execute a virtual processor (230A) of the virtual machine (240), the virtual processor (230A) assigned to a logical processor (212A), the logical processor (212A) assigned to a NUMA node and the virtual processor assigned to the virtual NUMA node (702);

determining that the logical processor (230A) can not execute the virtual processor (212A); and

selecting a second logical processor (230E) to execute the virtual processor (230A), the second logical processor from a second NUMA node (704).

8. A computer system, comprising:

circuitry for executing a virtual machine (240), the virtual machine (240) having a topology that includes a plurality of virtual NUMA nodes (606-608), wherein the topology of the virtual machine (240) is generated independently from the physical topology of the computer system (700);

circuitry for determining memory pressure in each virtual NUMA node (606-608) of the plurality; and

circuitry for adjusting, based on the memory pressure in each virtual NUMA node (606-608) of the plurality, guest memory assigned to at least one virtual NUMA node (606) of the plurality.

9. The computer system of claim 8, further comprising:

circuitry for sending the virtual machine (240) to a second computer system (510).

10. The computer system of claim 8, further comprising:

circuitry for mapping a first virtual NUMA node (606) of the plurality onto a first NUMA node (702) of the computer system (700); and

circuitry for mapping a second virtual NUMA node (608) of the plurality onto the first NUMA node (702) of the computer system (700).

11. The computer system of claim 8, further comprising:

circuitry for mapping a first virtual NUMA node (606) of the plurality onto a first NUMA node (702) of the computer system (700); and

circuitry for mapping a second virtual NUMA node (608) of the plurality onto a second NUMA node (704) of the computer system (700).

12. The computer system of claim 10, further comprising:

circuitry for determining that memory pressure in the second virtual NUMA node (608) is greater than a predetermined value; and

circuitry for migrating the second virtual NUMA node (608) to a second NUMA node of the computer system (704).

13. The computer system of claim 11, further comprising:

circuitry for determining that memory pressure of a second virtual machine (242) is greater than a predetermined value; and

circuitry for migrating the second virtual NUMA node (608) of the virtual machine (240) to the first NUMA node (704) of the computer system (700).

14. A computer readable storage medium including processor executable instructions, the computer readable storage medium comprising:

instructions for executing a first virtual machine (240), the virtual machine (240) having a topology that includes a plurality of virtual NUMA nodes (606-608), each virtual NUMA node of the plurality including a virtual processor (230A-C) and guest physical addresses, wherein the topology of the virtual machine (240) is generated independently from the physical topology of the computer system (700); and

instructions for adding an additional virtual processor (230D) to a virtual NUMA node of the plurality (608).

15. The computer readable storage medium of claim 14, further comprising:

instructions for determining memory pressure in each virtual NUMA node (606-608) of the plurality; and

instructions for adjusting, based on the memory pressure in each virtual NUMA node (606-608) of the plurality, guest memory assigned to at least one virtual NUMA node of plurality (606).

16. The computer readable storage medium of claim 14, further comprising:

instructions for removing a virtual processor (230D )from a virtual NUMA node of the plurality (608).

17. The computer readable storage medium of claim 14, further comprising:

instructions for reporting NUMA ratios for the plurality of virtual NUMA nodes (606-608) to a guest operating system.

18. The computer readable storage medium of claim 14, further comprising:

instructions for migrating the virtual machine (240) from a first NUMA node (606) to a plurality of NUMA nodes (606-608).

19. The computer readable storage medium of claim 15, wherein the instructions for adjusting the guest memory further comprise:

instructions for de-committing memory from a first virtual NUMA node (606) of the plurality based on a determination that the current memory pressure of the virtual NUMA node (606) is lower than a target threshold.

20. The computer readable storage medium of claim 15, wherein the instructions for adjusting the guest memory further comprise:

instructions for committing memory to a first virtual NUMA node (606) of the plurality based on a determination that the current memory pressure of a guest operating system (220) is greater than a target threshold.

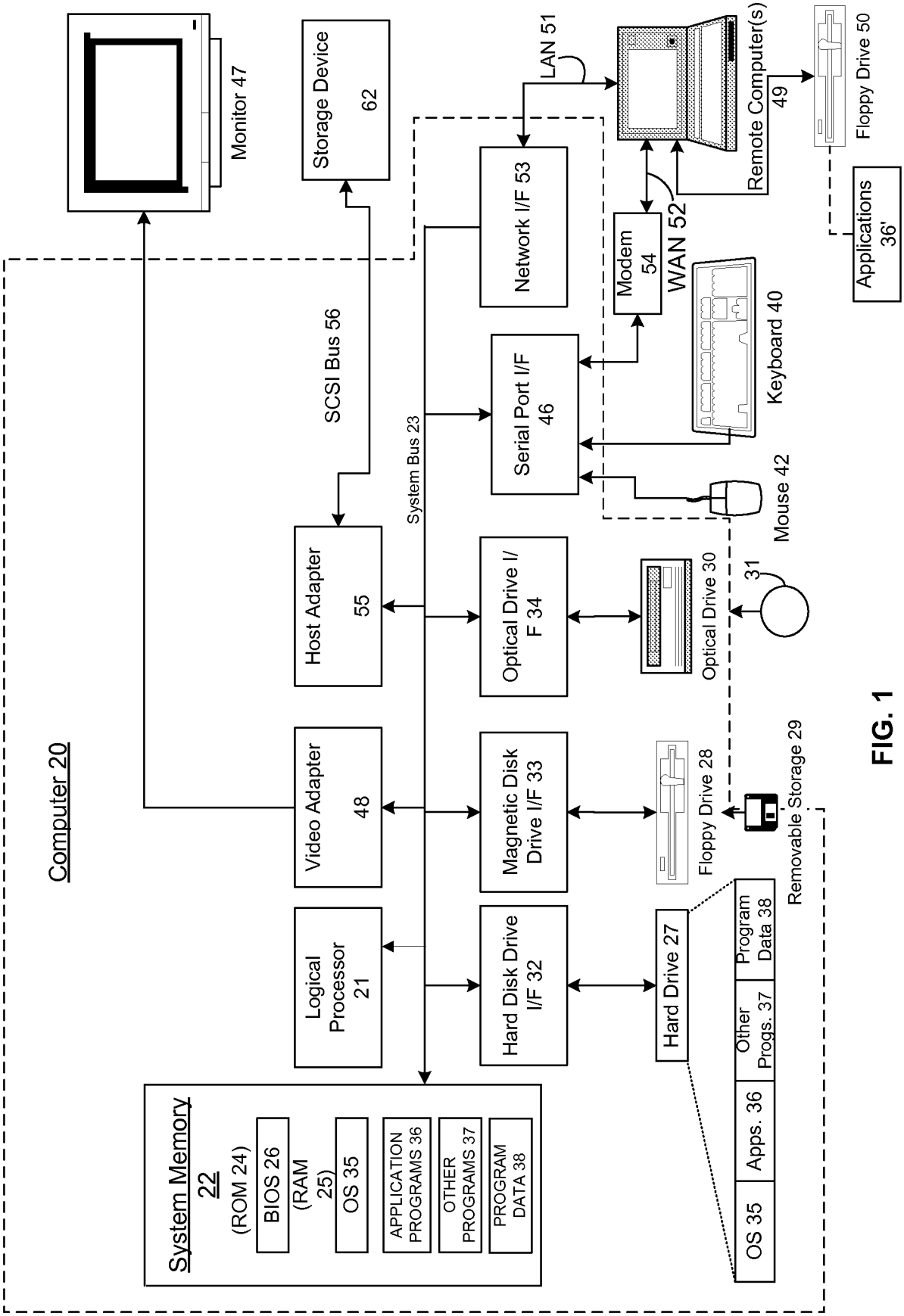

Computer 20
System Memory 22
(ROM 24)
BIOS 26
(RAM 25)
OS 35
APPLICATION PROGRAMS 36
OTHER PROGRAMS 37
PROGRAM DATA 38
Logical Processor 21
Video Adapter 48
Host Adapter 55
Hard Disk Drive I/F 32
Magnetic Disk Drive I/F 33
Optical Drive I/F 34
Serial Port I/F 46
Network I/F 53
Modem 54
WAN 52
LAN 51
Remote Computer(s) 49
Floppy Drive 50
Applications 36'
Monitor 47
Storage Device 62
SCSI Bus 56
System Bus 23
Keyboard 40
Mouse 42
Optical Drive 30
31
Floppy Drive 28
Removable Storage 29
Hard Drive 27
OS 35
Apps. 36
Other Progs. 37
Program Data 38

FIG. 1

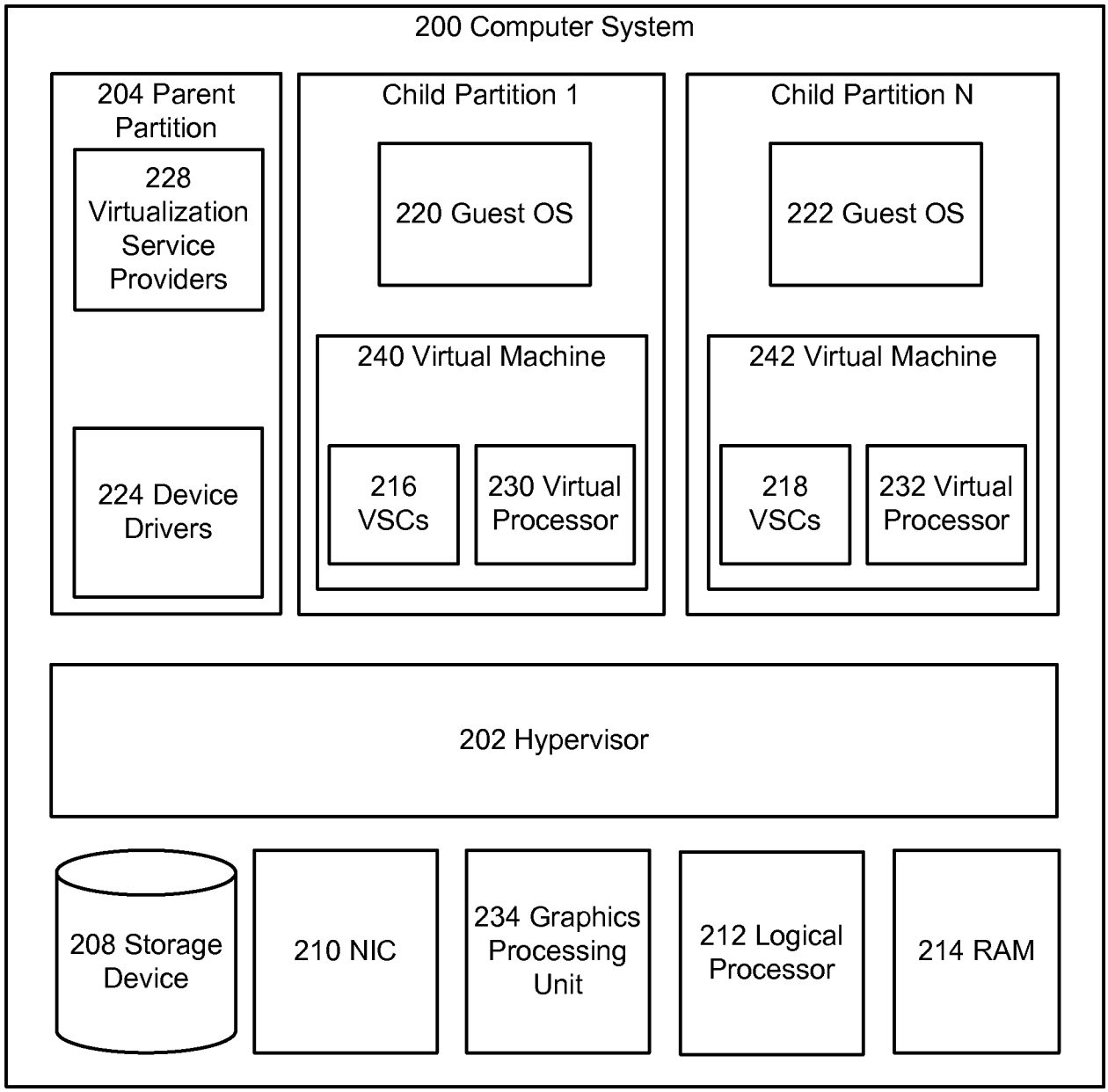

200 Computer System
204 Parent Partition
228 Virtualization Service Providers
224 Device Drivers
Child Partition 1
220 Guest OS
240 Virtual Machine
216 VSCs
230 Virtual Processor
Child Partition N
222 Guest OS
242 Virtual Machine
218 VSCs
232 Virtual Processor
202 Hypervisor
208 Storage Device
210 NIC
234 Graphics Processing Unit
212 Logical Processor
214 RAM

FIG. 2

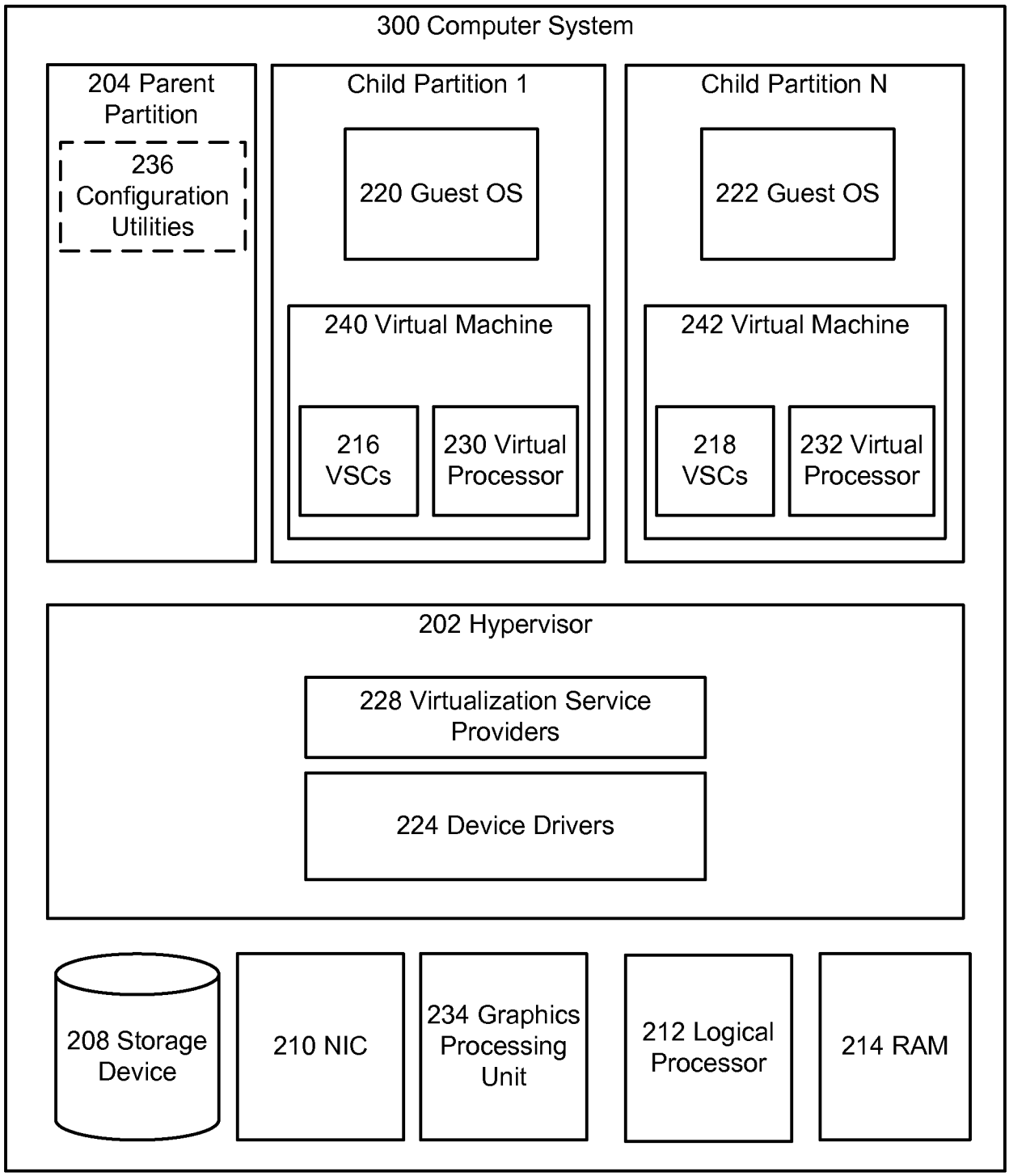

300 Computer System
204 Parent Partition
236 Configuration Utilities
Child Partition 1
220 Guest OS
240 Virtual Machine
216 VSCs
230 Virtual Processor
Child Partition N
222 Guest OS
242 Virtual Machine
218 VSCs
232 Virtual Processor
202 Hypervisor
228 Virtualization Service Providers
224 Device Drivers
208 Storage Device
210 NIC
234 Graphics Processing Unit
212 Logical Processor
214 RAM

FIG. 3

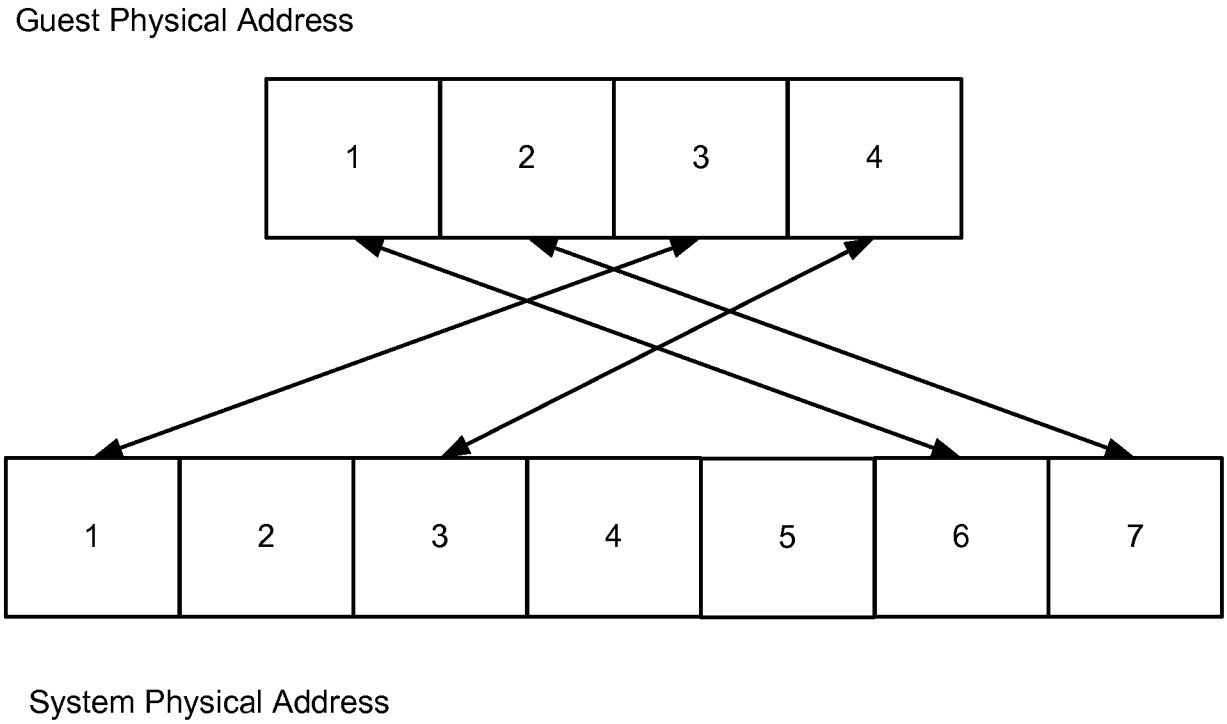

Guest Physical Address
1
2
3
4
1
2
3
4
5
6
7
System Physical Address

FIG. 4

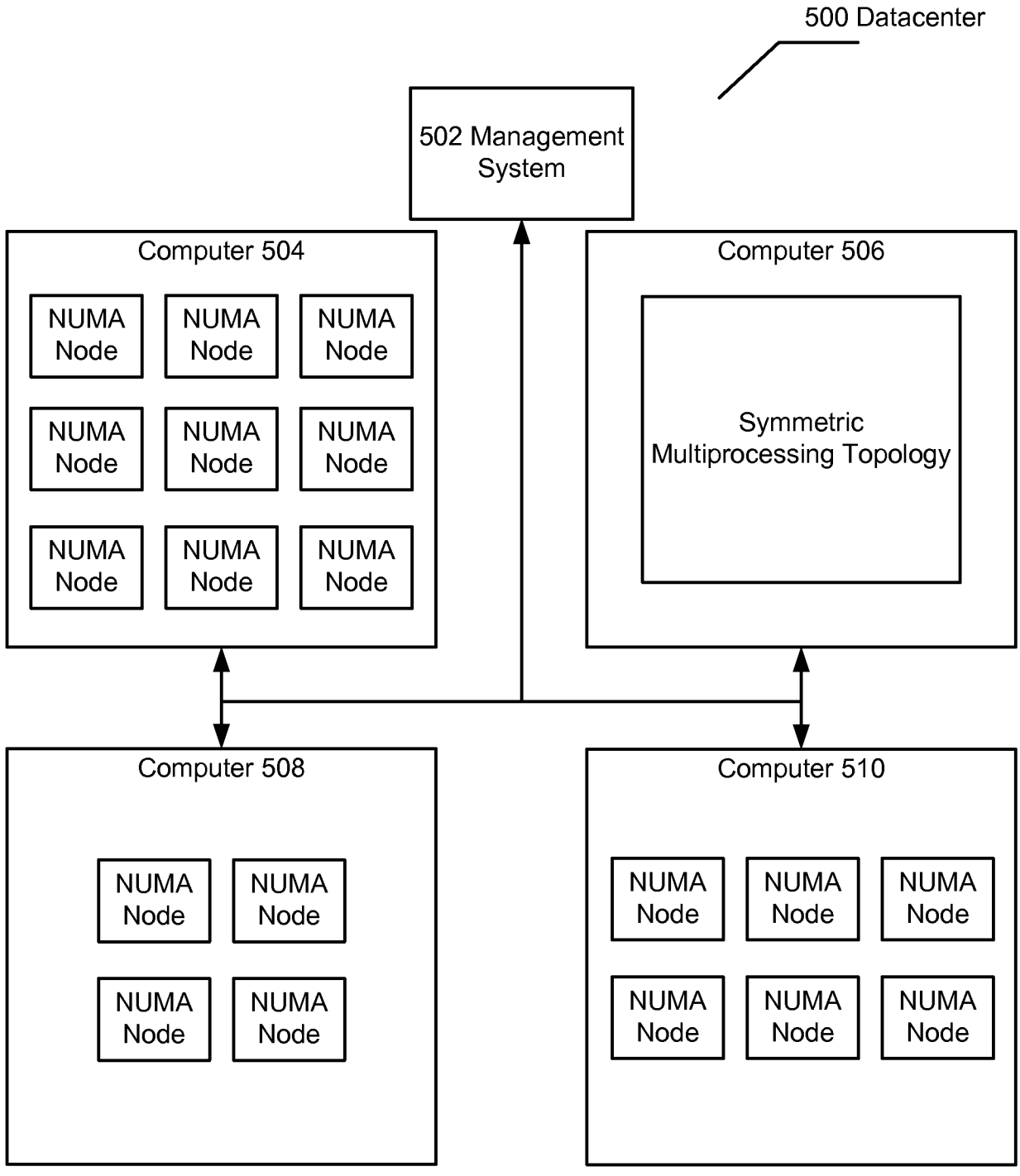

500 Datacenter
502 Management System
Computer 504
NUMA Node
NUMA Node
NUMA Node
NUMA Node
NUMA Node
NUMA Node
NUMA Node
NUMA Node
NUMA Node
Computer 506
Symmetric Multiprocessing Topology
Computer 508
NUMA Node
NUMA Node
NUMA Node
NUMA Node
Computer 510
NUMA Node
NUMA Node
NUMA Node
NUMA Node
NUMA Node
NUMA Node

FIG. 5

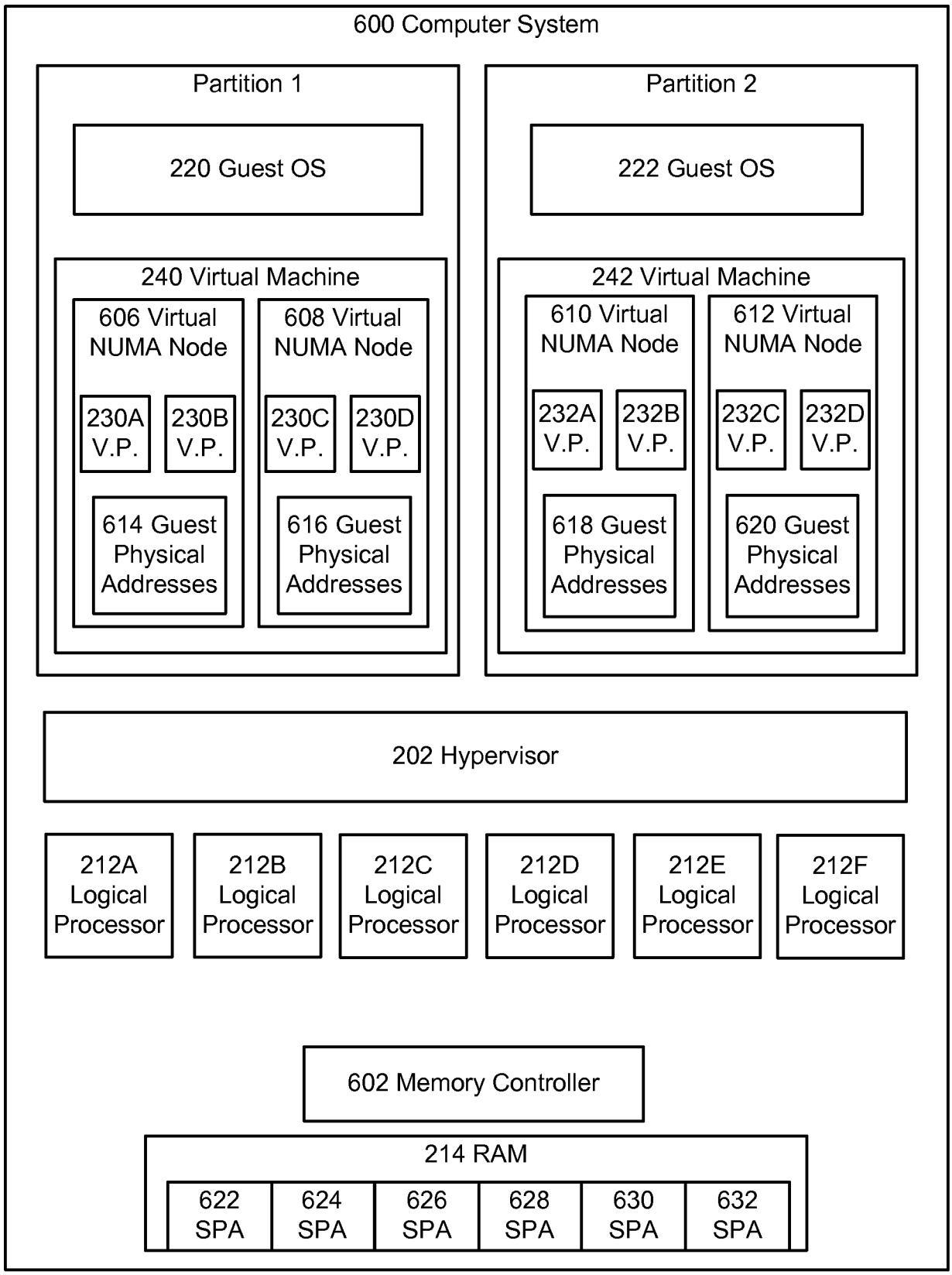

600 Computer System
Partition 1
220 Guest OS
240 Virtual Machine
606 Virtual NUMA Node
230A V.P.
230B V.P.
614 Guest Physical Addresses
608 Virtual NUMA Node
230C V.P.
230D V.P.
616 Guest Physical Addresses
Partition 2
222 Guest OS
242 Virtual Machine
610 Virtual NUMA Node
232A V.P.
232B V.P.
618 Guest Physical Addresses
612 Virtual NUMA Node
232C V.P.
232D V.P.
620 Guest Physical Addresses
202 Hypervisor
212A Logical Processor
212B Logical Processor
212C Logical Processor
212D Logical Processor
212E Logical Processor
212F Logical Processor
602 Memory Controller
214 RAM
622 SPA
624 SPA
626 SPA
628 SPA
630 SPA
632 SPA

FIG. 6

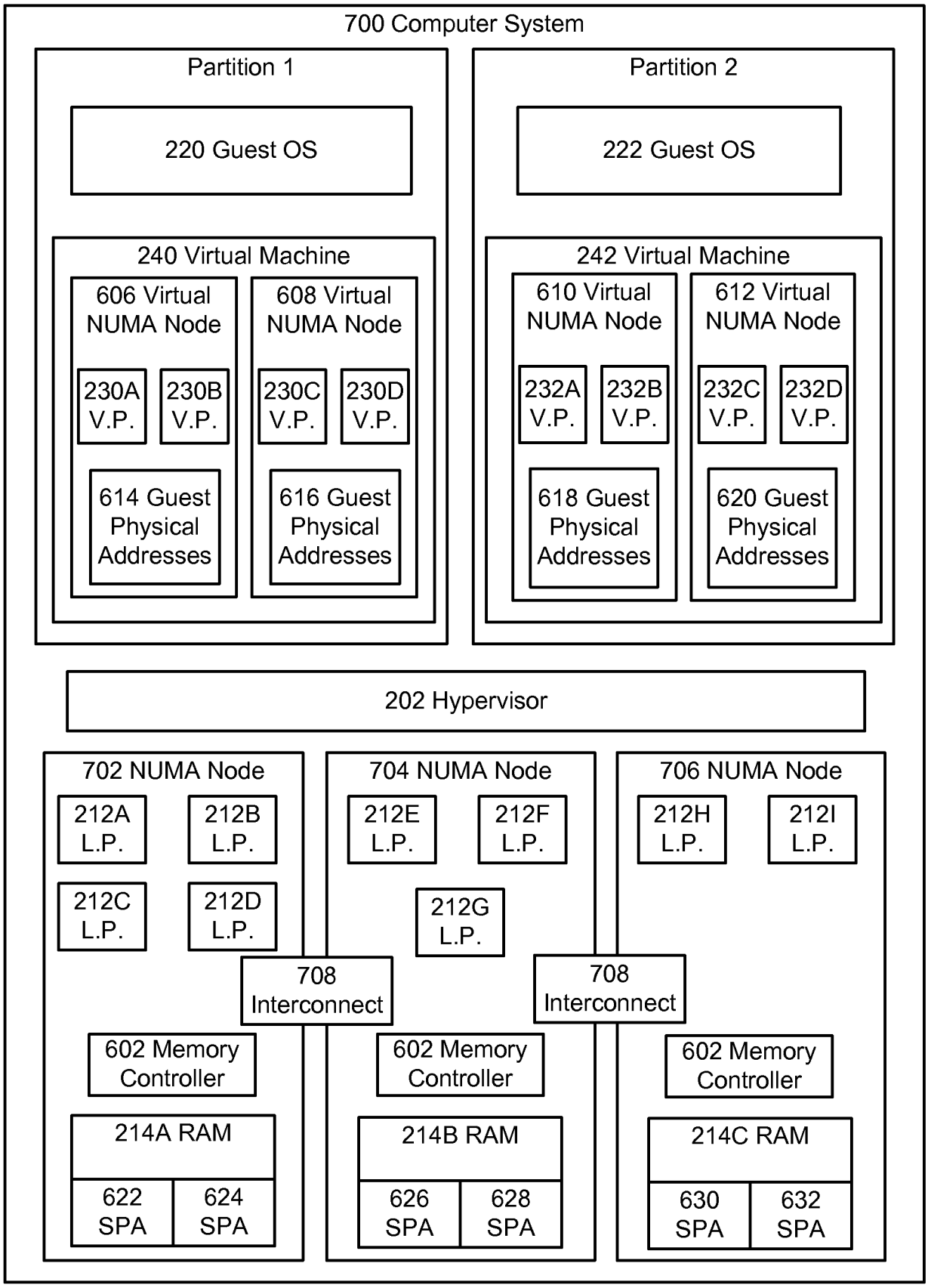

700 Computer System
Partition 1
220 Guest OS
240 Virtual Machine
606 Virtual NUMA Node
608 Virtual NUMA Node
230A V.P.
230B V.P.
230C V.P.
230D V.P.
614 Guest Physical Addresses
616 Guest Physical Addresses
Partition 2
222 Guest OS
242 Virtual Machine
610 Virtual NUMA Node
612 Virtual NUMA Node
232A V.P.
232B V.P.
232C V.P.
232D V.P.
618 Guest Physical Addresses
620 Guest Physical Addresses
202 Hypervisor
702 NUMA Node
704 NUMA Node
706 NUMA Node
212A L.P.
212B L.P.
212C L.P.
212D L.P.
212E L.P.
212F L.P.
212G L.P.
212H L.P.
212I L.P.
708 Interconnect
708 Interconnect
602 Memory Controller
602 Memory Controller
602 Memory Controller
214A RAM
622 SPA
624 SPA
214B RAM
626 SPA
628 SPA
214C RAM
630 SPA
632 SPA

FIG. 7

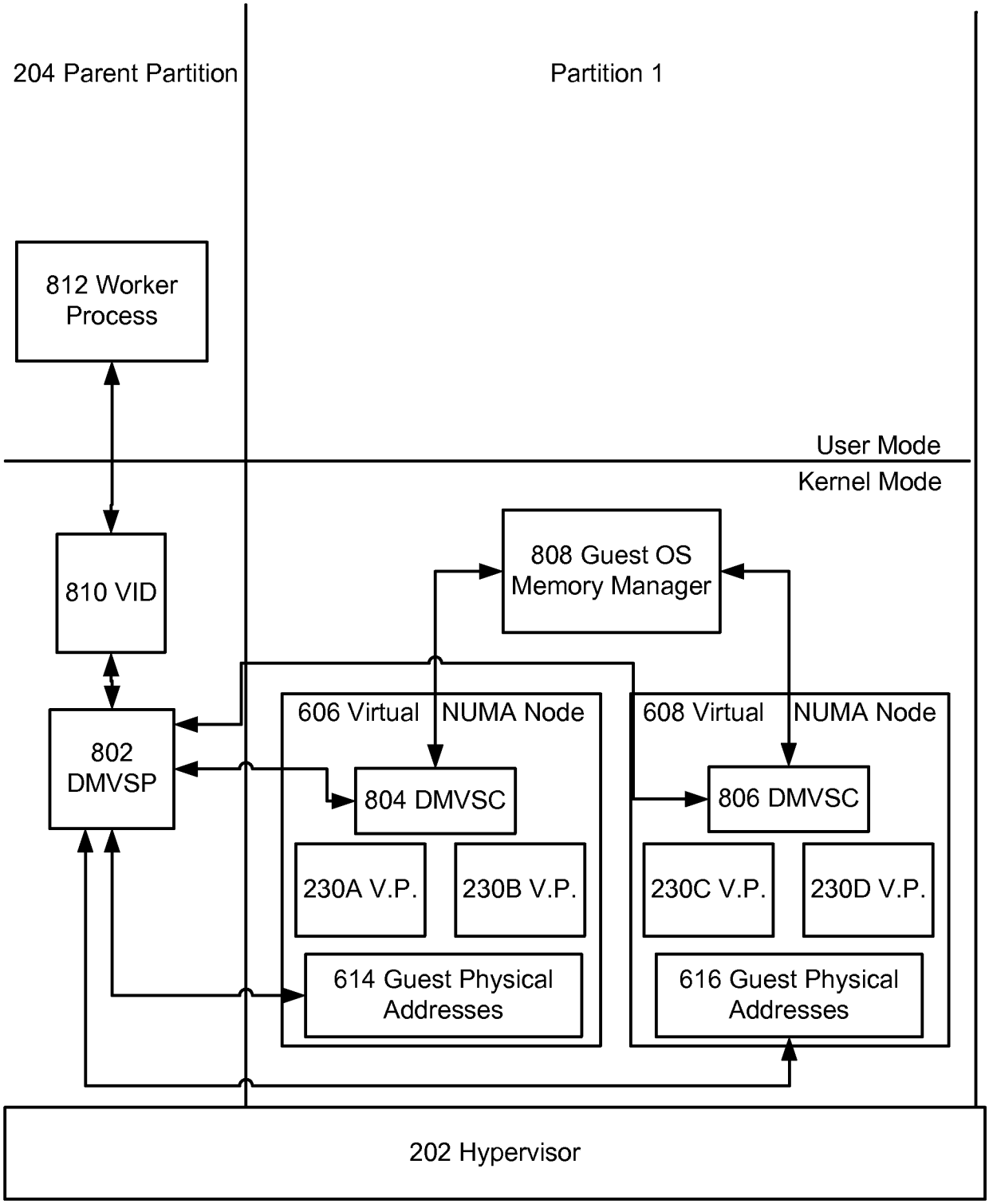

204 Parent Partition
Partition 1
812 Worker Process
User Mode
Kernel Mode
810 VID
808 Guest OS Memory Manager
802 DMVSP
606 Virtual NUMA Node
608 Virtual NUMA Node
804 DMVSC
806 DMVSC
230A V.P.
230B V.P.
230C V.P.
230D V.P.
614 Guest Physical Addresses
616 Guest Physical Addresses
202 Hypervisor

FIG. 8

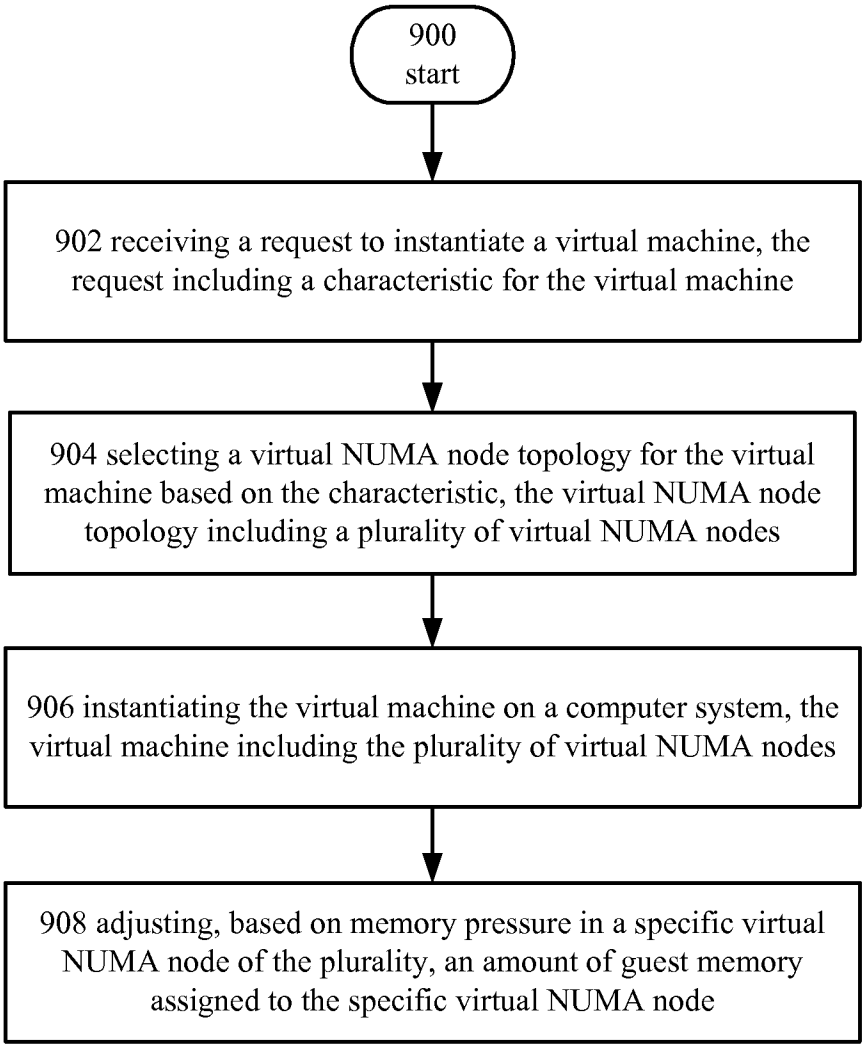

900
start
902 receiving a request to instantiate a virtual machine, the request including a characteristic for the virtual machine
904 selecting a virtual NUMA node topology for the virtual machine based on the characteristic, the virtual NUMA node topology including a plurality of virtual NUMA nodes
906 instantiating the virtual machine on a computer system, the virtual machine including the plurality of virtual NUMA nodes
908 adjusting, based on memory pressure in a specific virtual NUMA node of the plurality, an amount of guest memory assigned to the specific virtual NUMA node

FIG. 9

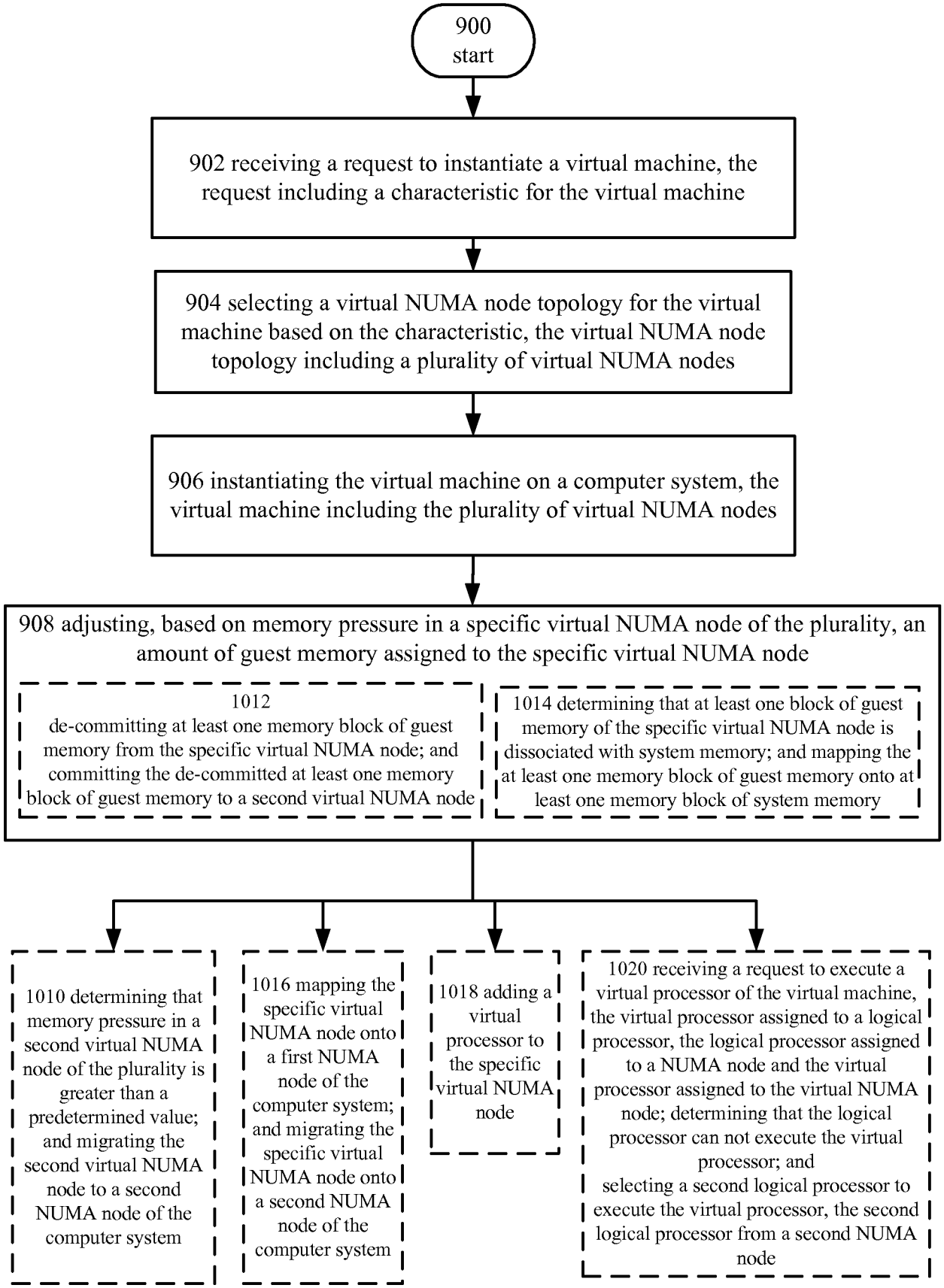

900
start
902 receiving a request to instantiate a virtual machine, the request including a characteristic for the virtual machine
904 selecting a virtual NUMA node topology for the virtual machine based on the characteristic, the virtual NUMA node topology including a plurality of virtual NUMA nodes
906 instantiating the virtual machine on a computer system, the virtual machine including the plurality of virtual NUMA nodes
908 adjusting, based on memory pressure in a specific virtual NUMA node of the plurality, an amount of guest memory assigned to the specific virtual NUMA node
1012 de-committing at least one memory block of guest memory from the specific virtual NUMA node; and committing the de-committed at least one memory block of guest memory to a second virtual NUMA node
1014 determining that at least one block of guest memory of the specific virtual NUMA node is dissociated with system memory; and mapping the at least one memory block of guest memory onto at least one memory block of system memory
1010 determining that memory pressure in a second virtual NUMA node of the plurality is greater than a predetermined value; and migrating the second virtual NUMA node to a second NUMA node of the computer system
1016 mapping the specific virtual NUMA node onto a first NUMA node of the computer system; and migrating the specific virtual NUMA node onto a second NUMA node of the computer system
1018 adding a virtual processor to the specific virtual NUMA node
1020 receiving a request to execute a virtual processor of the virtual machine, the virtual processor assigned to a logical processor, the logical processor assigned to a NUMA node and the virtual processor assigned to the virtual NUMA node; determining that the logical processor can not execute the virtual processor; and
selecting a second logical processor to execute the virtual processor, the second logical processor from a second NUMA node

FIG. 10

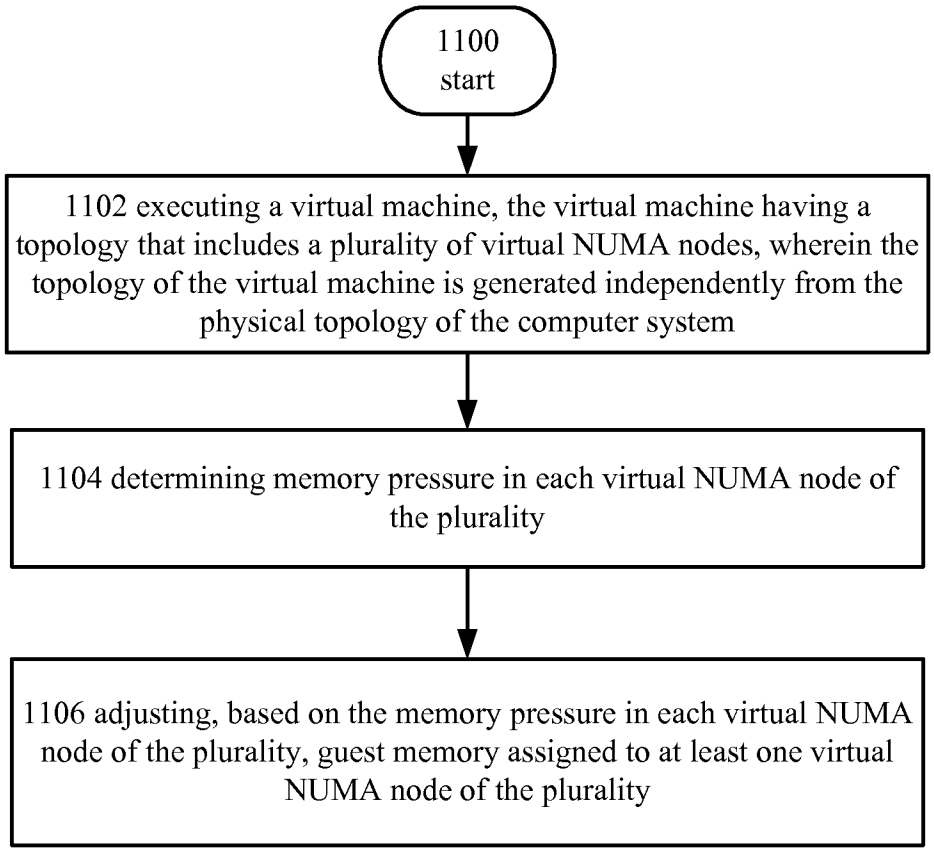

1100
start
1102 executing a virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system
1104 determining memory pressure in each virtual NUMA node of the plurality
1106 adjusting, based on the memory pressure in each virtual NUMA node of the plurality, guest memory assigned to at least one virtual NUMA node of the plurality

FIG. 11

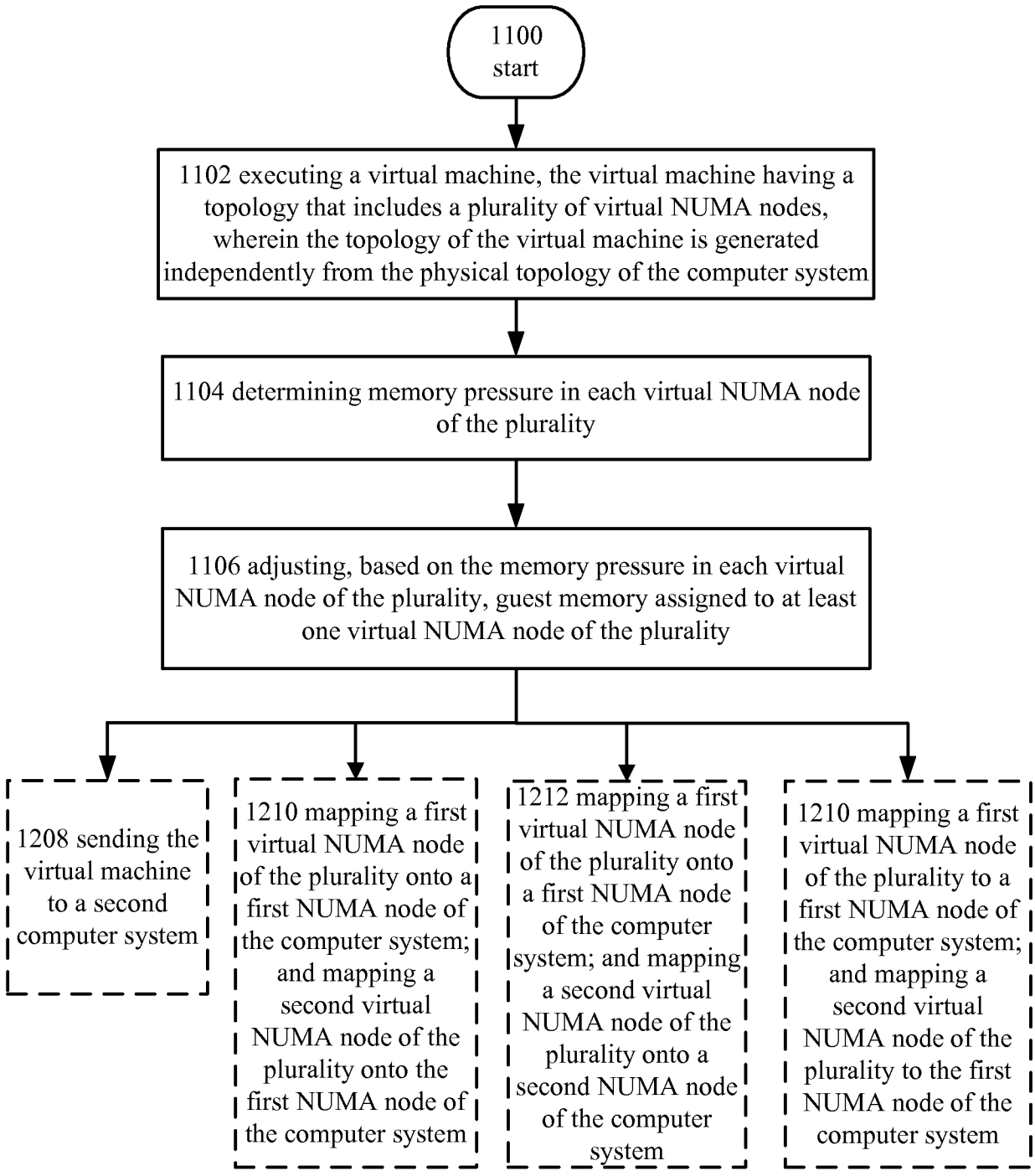

1100
start
1102 executing a virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system
1104 determining memory pressure in each virtual NUMA node of the plurality
1106 adjusting, based on the memory pressure in each virtual NUMA node of the plurality, guest memory assigned to at least one virtual NUMA node of the plurality
1208 sending the virtual machine to a second computer system
1210 mapping a first virtual NUMA node of the plurality onto a first NUMA node of the computer system; and mapping a second virtual NUMA node of the plurality onto the first NUMA node of the computer system
1212 mapping a first virtual NUMA node of the plurality onto a first NUMA node of the computer system; and mapping a second virtual NUMA node of the plurality onto a second NUMA node of the computer system
1210 mapping a first virtual NUMA node of the plurality to a first NUMA node of the computer system; and mapping a second virtual NUMA node of the plurality to the first NUMA node of the computer system

FIG. 12

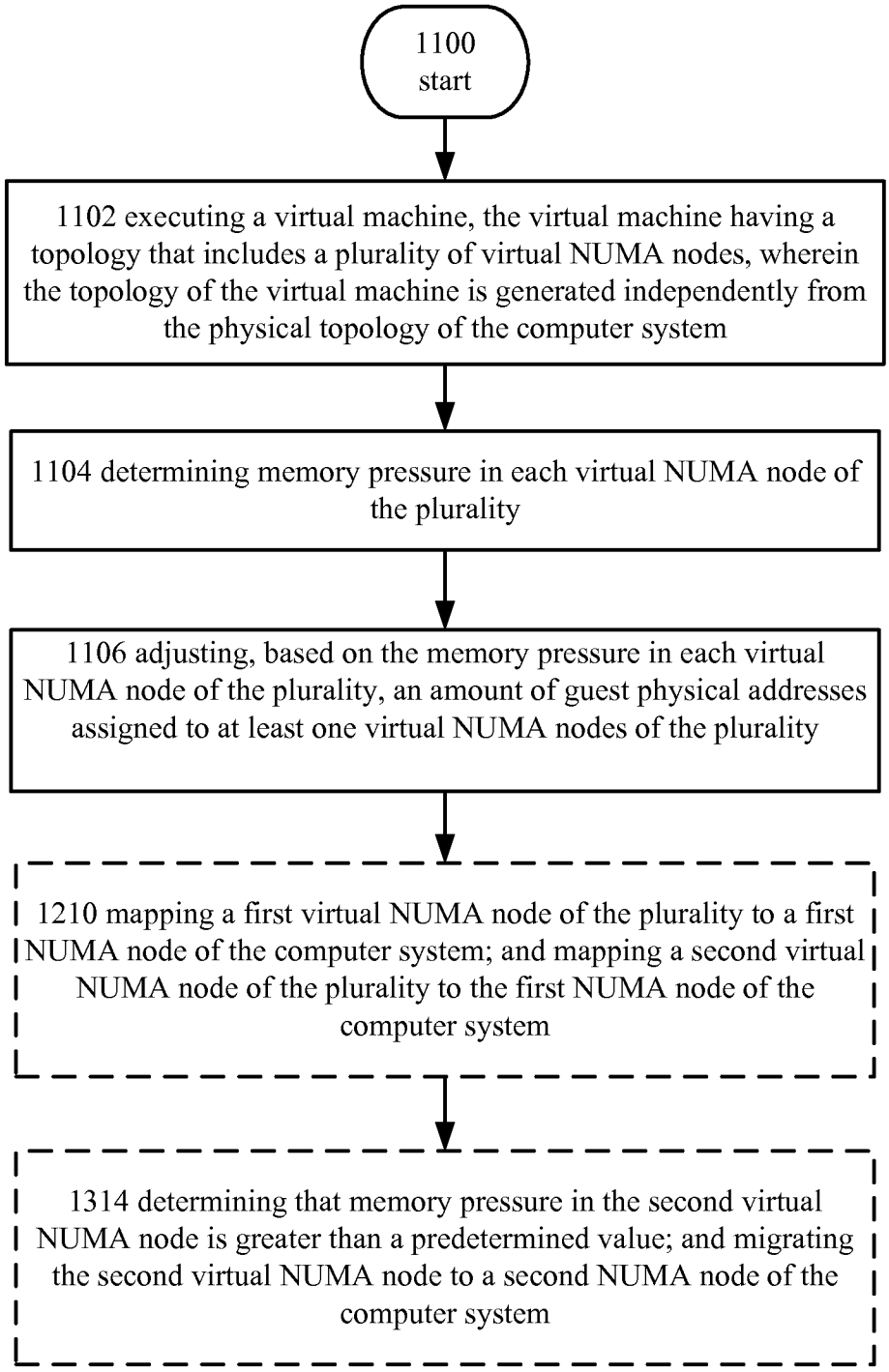

1100
start
1102 executing a virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system
1104 determining memory pressure in each virtual NUMA node of the plurality
1106 adjusting, based on the memory pressure in each virtual NUMA node of the plurality, an amount of guest physical addresses assigned to at least one virtual NUMA nodes of the plurality
1210 mapping a first virtual NUMA node of the plurality to a first NUMA node of the computer system; and mapping a second virtual NUMA node of the plurality to the first NUMA node of the computer system
1314 determining that memory pressure in the second virtual NUMA node is greater than a predetermined value; and migrating the second virtual NUMA node to a second NUMA node of the computer system

FIG. 13

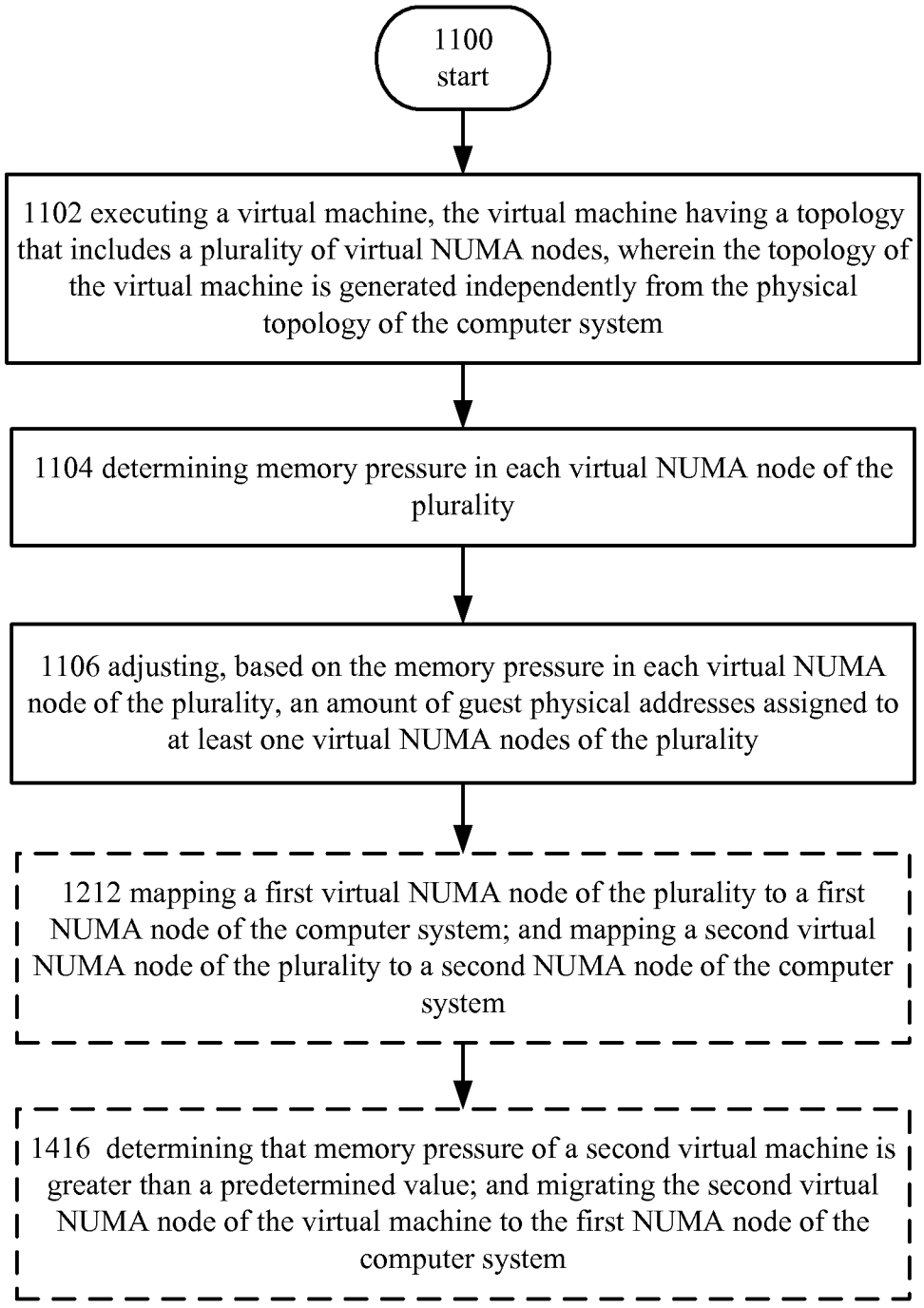

1100
start
1102 executing a virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system
1104 determining memory pressure in each virtual NUMA node of the plurality
1106 adjusting, based on the memory pressure in each virtual NUMA node of the plurality, an amount of guest physical addresses assigned to at least one virtual NUMA nodes of the plurality
1212 mapping a first virtual NUMA node of the plurality to a first NUMA node of the computer system; and mapping a second virtual NUMA node of the plurality to a second NUMA node of the computer system
1416 determining that memory pressure of a second virtual machine is greater than a predetermined value; and migrating the second virtual NUMA node of the virtual machine to the first NUMA node of the computer system

FIG. 14

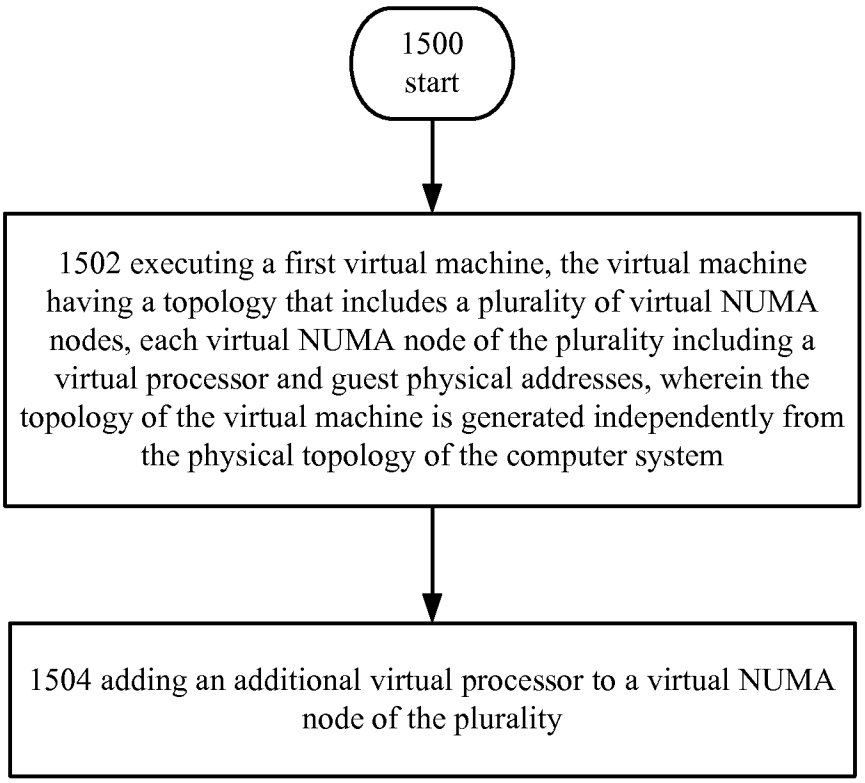

1500
start
1502 executing a first virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, each virtual NUMA node of the plurality including a virtual processor and guest physical addresses, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system
1504 adding an additional virtual processor to a virtual NUMA node of the plurality

FIG. 15

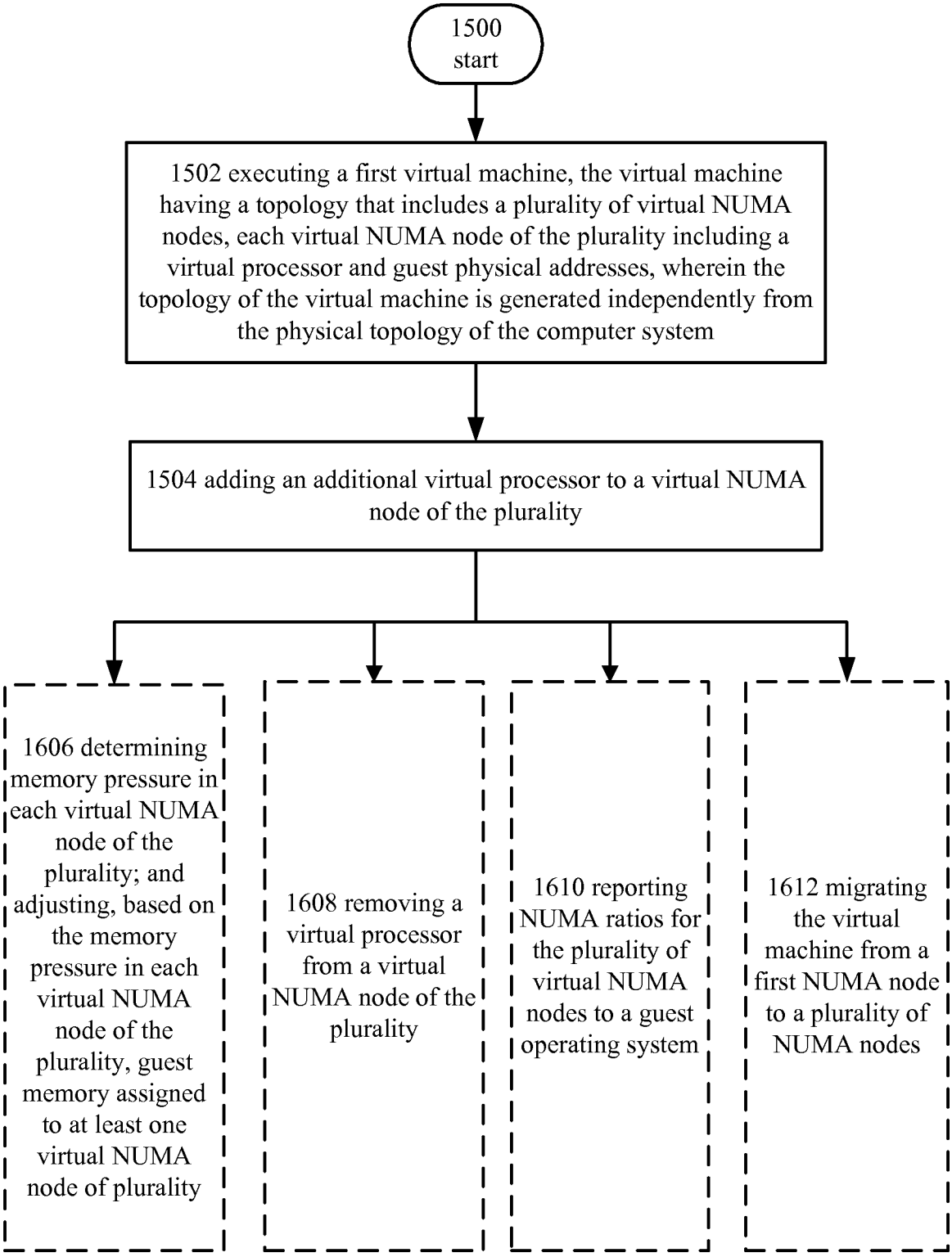

1500
start
1502 executing a first virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, each virtual NUMA node of the plurality including a virtual processor and guest physical addresses, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system
1504 adding an additional virtual processor to a virtual NUMA node of the plurality
1606 determining memory pressure in each virtual NUMA node of the plurality; and adjusting, based on the memory pressure in each virtual NUMA node of the plurality, guest memory assigned to at least one virtual NUMA node of plurality
1608 removing a virtual processor from a virtual NUMA node of the plurality
1610 reporting NUMA ratios for the plurality of virtual NUMA nodes to a guest operating system
1612 migrating the virtual machine from a first NUMA node to a plurality of NUMA nodes

FIG. 16

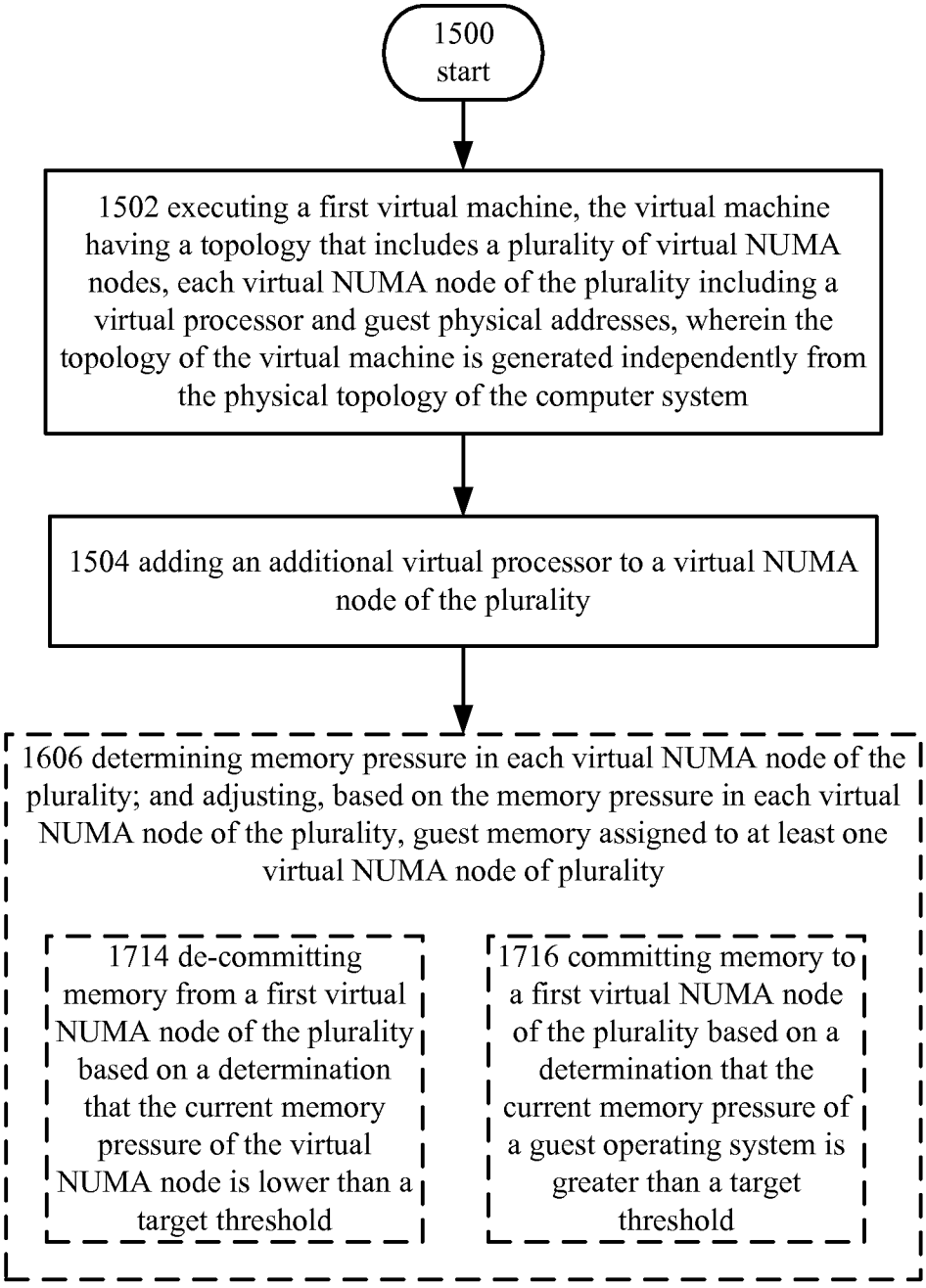

1500
start
1502 executing a first virtual machine, the virtual machine having a topology that includes a plurality of virtual NUMA nodes, each virtual NUMA node of the plurality including a virtual processor and guest physical addresses, wherein the topology of the virtual machine is generated independently from the physical topology of the computer system
1504 adding an additional virtual processor to a virtual NUMA node of the plurality
1606 determining memory pressure in each virtual NUMA node of the plurality; and adjusting, based on the memory pressure in each virtual NUMA node of the plurality, guest memory assigned to at least one virtual NUMA node of plurality
1714 de-committing memory from a first virtual NUMA node of the plurality based on a determination that the current memory pressure of the virtual NUMA node is lower than a target threshold
1716 committing memory to a first virtual NUMA node of the plurality based on a determination that the current memory pressure of a guest operating system is greater than a target threshold

FIG. 17