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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 December 2014 (18.12.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/200521 A1

(51) International Patent Classification:
G06F 9/50 (2006.01) *G06F 11/34* (2006.01)

(21) International Application Number:
PCT/US2013/060243

(22) International Filing Date:
18 September 2013 (18.09.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
13/917,634 13 June 2013 (13.06.2013) US

(71) Applicant: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, WA 98052-6399 (US).

(72) Inventors: **BARAKAT, Youssef**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **BROWN, Tristan**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **FATEMIEH, Omid**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **KIM, Minsang**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **RAFFMAN, Andrew**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US). **WOHLGEMUTH, Jason**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, WA 98052-6399 (US).

(74) Agent: **BANOWSKY, James R.**; Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, 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, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM,

[Continued on next page]

(54) Title: OPERATING SYSTEM-MANAGED INTERRUPT STEERING IN MULTIPROCESSOR SYSTEMS

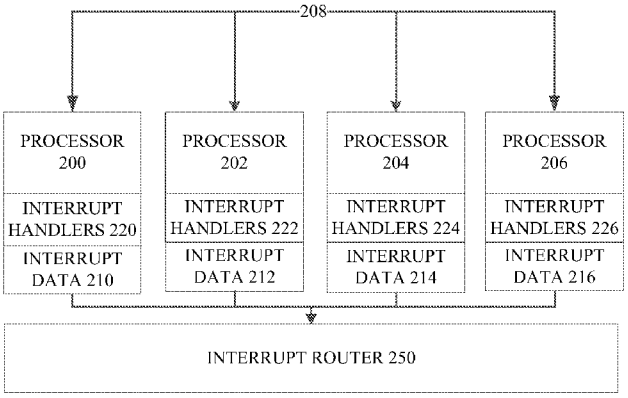

FIG. 2

(57) Abstract: An operating system is provided in which an interrupt router dynamically steers each interrupt to one or more processors within set of processors based on overall load information from the set of processors. An interrupt source is assigned to a processor based on the load imposed by the interrupt source and the target overall load for the processor. For example, each processor can maintain information about each interrupt it processes over time. The operating system receives this historical load information to determine an expected load for interrupts of a given type from a given device, an overall load on the system, and a target load for each processor. Given a set of interrupt sources, their expected loads, and target load for each processor, each interrupt source can be assigned dynamically to a processor during runtime of the system. On a regular basis, these assignments can be changed given current operating conditions of the system.

WO 2014/200521 A1

TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to 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:

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

OPERATING SYSTEM-MANAGED INTERRUPT STEERING IN MULTIPROCESSOR SYSTEMS

BACKGROUND

[0001] All modern computing platforms implement a mechanism called interrupt handling. In general, a device generates a signal, called an interrupt, to the system to request an asynchronous service to be performed. In response to the interrupt, the system executes an interrupt handler. An interrupt handler is a computer program that, when executed by a processor, causes the system to perform the requested service, or other appropriate action, in response to that interrupt. The interrupt handler may in turn schedule additional operations to be performed to assist in responding to, or processing the data associated with, that interrupt.

[0002] In modern computing platforms, there typically are several devices that can generate interrupts, each with its own interrupt handler. The rate at which each device generates interrupts generally is variable. The amount of time taken to process each interrupt also can vary.

[0003] In a multiprocessor system, each individual interrupt is typically directed at a single processor; however some interrupt architectures allow an interrupt to be directed at a cluster of processors. In general, multiprocessor systems are designed such that interrupts are distributed among a subset of the system's processors while attempting to optimize one or more of overall system throughput, latency and power consumption. For example, in some systems, all interrupts are directed to a dedicated processor that executes the interrupt handlers. In some systems, each device or process that generates an interrupt, i.e., an interrupt source, is statically assigned a processor for its interrupt handler.

[0004] In some systems, a hardware-based interrupt controller can dynamically assign each interrupt to one of the processors based on information local to the interrupt controller. In some systems, the interrupt controller assigns a set of processors to handle each interrupt in a round robin fashion among the set of processors, each of which is programmed with the various interrupt handlers. In some systems, the interrupt controller broadcasts each interrupt to a set of processors, each of which selects whether to accept the interrupt. In some systems, the interrupt is directed to the processor, within a subset of processors, which is currently handling the lowest priority task.

SUMMARY

[0005] This Summary introduces selected concepts in simplified form that are further described below in the Detailed Description. This Summary is intended neither to identify key or essential features of the claimed subject matter, nor to limit the scope of the claimed subject matter.

[0006] An operating system is provided in which an interrupt router dynamically steers each interrupt source to a processor within set of processors based on overall load information from the set of processors. An interrupt source is assigned to a processor based on the load imposed by the associated interrupts and the target overall load for the processor. For example, each processor can maintain information about each interrupt it processes over time. The operating system receives this historical load information to determine an expected load due to interrupts of a given type from a given device, an overall load on the system, and a target load for each processor. Given a set of interrupt sources, their expected loads, and target load for each processor, each interrupt source can be assigned dynamically to a processor during runtime of the system. On a regular basis, these assignments can be changed given current operating conditions of the system. The assignments also can be determined based on the current power state of each processor, so as to avoid activating an idle processor solely to process an interrupt, and to allow processors to become idle, to save power.

[0007] One challenge is measuring the overall load due to an interrupt source, because an interrupt handler can invoke additional processing on the same processor in response to handling an interrupt.

[0008] Accordingly, in one aspect, in a computer including a plurality of processors, an interrupt router receives information about interrupts. The interrupt router determines a load on the computer due to interrupt handling by the plurality of processors for the interrupts. The interrupt router assigns each interrupt source to a selected one or more of the plurality of processors, such selection being a function of the determined load so as to distribute the load among the processors. The interrupt router can periodically repeat determining a load on the computer due to interrupt handling, and assigning each interrupt source to a selected one of the plurality of processors.

[0009] In one implementation, the interrupt router identifies a number of processors, from among the plurality of processors, available for processing interrupts, and selects a number of processors, from among the identified number of processors available for processing interrupts, such that the number of selected processors matches the determined load

divided by a target per-processor load. The load can be determined by, in response to each interrupt, executing an interrupt handler for the interrupt on the processor assigned to the interrupt, and storing, in memory associated with the processor assigned to the interrupt, data indicating an amount of processing time consumed due to executing the interrupt handler. The interrupt router aggregates the stored data from the plurality of processors for the interrupts.

[0010] The amount of processing time can be determined by storing a system time stamp when beginning execution of the interrupt handler, computing a difference between a system time stamp observed when ending execution of the interrupt handler, and storing the computed difference. Determining the amount of processing time can further include, for any process invoked by the interrupt handler, storing a system time stamp when beginning execution of the process, computing a difference between a system time stamp observed when ending execution of the process, and storing data indicative of the computed difference for the process and the computed difference for the interrupt handler. Determining the amount of processing time can further include, for any preemptive activity, such as an interrupt or other process preempting interrupt processing by another interrupt handler or process performing associated work, computing a difference between system time stamps observed when pausing the preempted interrupt processing and when restarting the preempted interrupt processing to have an amount of time for executing the preemptive activity, such that the computed difference for the interrupt handler excludes the amount of time for executing the preemptive activity.

[0011] These various aspects and implementations can be embodied in a computer-implemented process, computer, or an article of manufacture including computer storage media.

[0012] In the following description, reference is made to the accompanying drawings which form a part hereof, and in which are shown, by way of illustration, specific example implementations of this technique. It is understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the disclosure.

DESCRIPTION OF THE DRAWINGS

[0013] FIG. 1 is a block diagram of an example computer in which operating system-manage interrupt steering can be implemented.

[0014] FIG. 2 is a block diagram of an example multi-processor system implementing an operating system-based interrupt router.

[0015] FIG. 3 illustrates a data structure used in tracking interrupt handlers on processors.

[0016] FIG. 4 is a flow chart of an example implementation of identifying available processors to assign to interrupt sources.

[0017] FIG. 5 is a flow chart of an example implementation of assigning interrupt sources to processors.

[0018] FIG. 6 is a flow chart of an example implementation of reassigning interrupt sources to processors.

[0019] FIG. 7 is a flow chart of an example implementation of tracking impact of interrupt sources on a computer system.

DETAILED DESCRIPTION

[0020] The following section provides an example operating environment in which operating-system based interrupt steering can be implemented.

[0021] Referring to Fig. 1, the following description is intended to provide a brief, general description of a general purpose computer that may use such an interrupt router. The computer can be any of a variety of general purpose or special purpose computing hardware configurations. Examples of well-known computers that may be suitable include, but are not limited to, personal computers, server computers, hand-held or laptop devices (for example, media players, notebook computers, cellular phones, personal data assistants, voice recorders), multiprocessor systems, microprocessor-based systems, set top boxes, game consoles, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

[0022] FIG. 1 illustrates an example of a suitable computer. This is only one example of a suitable computer and is not intended to suggest any limitation as to the scope of use or functionality of such a computer.

[0023] With reference to FIG. 1, an example computer 100, in a basic configuration, includes at least one processing unit 102 and memory 104. The computer may include multiple processing units and/or additional co-processing units such as graphics processing unit 120. Depending on the exact configuration and type of computer, memory 104 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. This configuration is illustrated in FIG. 1 by dashed line 106.

[0024] Additionally, computer 100 may also have additional features/functionality. For example, computer 100 may also include additional storage (removable and/or non-

removable) including, but not limited to, magnetic or optical disks or tape. Such additional storage is illustrated in FIG. 1 by removable storage 108 and non-removable storage 110. Computer storage media includes volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer program instructions, data structures, program modules or other data. Memory 104, removable storage 108 and non-removable storage 110 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the information and which can accessed by computer 100. Any such computer storage media may be part of computer 100.

[0025] Computer 100 may also contain communications connection(s) 112 that allow the device to communicate with other devices over a communication medium. Communication media typically carry computer program instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and include any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal, thereby changing the configuration or state of the receiving device of the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Communications connections 112 are devices that interface with the communication media to transmit data over and receive data from communication media, such as a network interface.

[0026] Computer 100 may have various input device(s) 114 such as a keyboard, mouse, pen, camera, touch input device, and so on. Output device(s) 116 such as a display, speakers, a printer, and so on may also be included. All of these devices are well known in the art and need not be discussed at length here. Various input and output devices can implement a natural user interface (NUI), which is any interface technology that enables a user to interact with a device in a "natural" manner, free from artificial constraints imposed by input devices such as mice, keyboards, remote controls, and the like.

[0027] Examples of NUI methods include those relying on speech recognition, touch and stylus recognition, gesture recognition both on screen and adjacent to the screen, air

gestures, head and eye tracking, voice and speech, vision, touch, gestures, and machine intelligence, and may include the use of touch sensitive displays, voice and speech recognition, intention and goal understanding, motion gesture detection using depth cameras (such as stereoscopic camera systems, infrared camera systems, and other camera systems and combinations of these), motion gesture detection using accelerometers or gyroscopes, facial recognition, three dimensional displays, head, eye , and gaze tracking, immersive augmented reality and virtual reality systems, all of which provide a more natural interface, as well as technologies for sensing brain activity using electric field sensing electrodes (EEG and related methods).

[0028] Each component of this system that operates on a computer generally is implemented by software, such as one or more computer programs, which include computer-executable instructions and/or computer-interpreted instructions, such as program modules, being processed by the computer. Generally, program modules include routines, programs, objects, components, data structures, and so on, that, when processed by a processing unit, instruct the processing unit to perform particular tasks or implement particular abstract data types. This computer system may be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices.

[0029] Alternatively, or in addition, the functionality described herein can be performed, at least in part, by one or more hardware logic components. For example, and without limitation, illustrative types of hardware logic components that can be used include Field-programmable Gate Arrays (FPGAs), Program-specific Integrated Circuits (ASICs), Program-specific Standard Products (ASSPs), System-on-a-chip systems (SOCs), Complex Programmable Logic Devices (CPLDs), etc.

[0030] Given a computer such as described above, an operating system based interrupt router that performs dynamic interrupt steering at runtime among multiple processors assumes that the computer includes a central processing unit (e.g., 120 in Fig. 1) above with multiple processors. The use of the term "processor" herein is intended to include any logical processor, including but not limited to a single instance of a hardware processor, one of multiple processors, a "core" in a multiple core processor or any other processing unit that can be managed independently of other processing units.

[0031] Interrupt handling generally is managed by the operating system, firmware or other low level software on the computer. In one implementation, the operating system includes an interrupt router in the kernel that is executed on one or more of the processors and that dynamically steers interrupts to a subset of processors which may include the processor on which the interrupt router is executed. Alternatively, other components of an operating system, hardware abstraction layer, firmware or other low level computer programs of a computer can implement such functionality.

[0032] Referring now to Fig. 2, an example implementation of such a system will now be described in the context of an illustrative example. In Fig. 2, there are four processors 200, 202, 204 and 206, interconnected by a bus 208, each with a respective interrupt data 210, 212, 214 and 216 for tracking information about interrupts processed by that processor. Each processor also has a set of one or more interrupt handlers 220, 222, 224, 226, respectively, for processing interrupts from various interrupt sources. The system also includes an interrupt router 250 which is part of the operating system that manages access to the resources in the multiprocessor computer. The interrupt router 250 periodically obtains the interrupt data 210, 212, 214, 216 from the processors. The interrupt router may be a software component executing on one or more of the processors, or may be implemented or supported by hardware components. The interrupt router determines an interrupt source-to-processor assignment based on the interrupt data, and sets an interrupt controller (not shown) with the assignments to direct interrupts in the system to their assigned processors.

[0033] It should be understood that the number of processors, i.e., four, in Fig. 2 is merely illustrative and not limiting of the present invention. Further, interrupt routing can be performed by a subset of the processors in the system. Fig. 2 illustrates merely an example of a subset of the processors of a multiprocessor system that participate in interrupt routing.

[0034] Referring now to Fig. 3, an example implementation of a data structure 300 maintained and used by interrupt router 250 for aggregating interrupt data from multiple processors, will now be described. The data structure 300 includes a list 302 of interrupt sources. Each entry 304 in the list 302 represents an interrupt source and points to an array 306 representing the interrupt handler(s) for interrupts from that source. The array 306 includes an entry 308 that represents each interrupt handler. The entry 308 points to a list 310 of pointers 314 to data structures 312 maintained by each processor for that interrupt handler for storing the interrupt data for that processor, which includes data about

the load placed on the processor related to that interrupt handler. How this data structure is used by each processor and the interrupt router to monitor current system status will be described in more detail below. It should be understood that other data structures can be used to aggregate and store the per-processor, per-interrupt handler performance data for use by the interrupt router and that the foregoing is merely an example.

[0035] Other data structures, not shown, include, for each processor, a count of the number of interrupt sources assigned to the processor as part of the interrupt data for each processor. Additionally, a current list of interrupt source-to-processor assignments also is stored by the interrupt router.

[0036] Steering of interrupts to different processors generally involves four steps. First, the interrupt router identifies which processors, of the available processors in the system, are to be used. Second, the interrupt router determines how to distribute interrupts among the identified processors. This interrupt source-to-processor assignment is set in an interrupt controller and used for a period of time. Third, the interrupt router dynamically changes the distribution of interrupts in response to system conditions during runtime. The result of this process changes the current interrupt source-to-processor assignments to new assignments. Fourth, the interrupt router tracks the impact of the interrupts from the various interrupt sources on system performance, which is used to determine future interrupt source-to-processor assignments. It should be understood that the process may assign interrupt sources to individual processor clusters as well as or instead of individual processors.

[0037] Referring now to Fig. 4, an example implementation of a way to identify processors to be used for handling interrupts will now be described. The output of this process is an indication of the set of processors to which interrupts are distributed.

[0038] This process begins by identifying 400 an initial set of available processors. This initial set can be all of the available processors, a predetermined subset of the available processors, or other set of processors identified based on a specified characteristic. As an example, the initial set of processors can be all active, i.e., non-idle, processors. Next, some of these processors can be eliminated 402 based on the time for which they have been active. The amount of time can be a tunable parameter, which, when set to zero, disables this step.

[0039] The interrupt router then determines 404 the load on the system due to interrupt handling, in a manner described in more detail below. Then, a number of processors to be

used for handling interrupts, given a target per-processor load and the determined load, is then computed 406.

[0040] In one example implementation, a total sum of all processor time used to handle all interrupt-related work is computed. This total amount of time is divided by the actual elapsed time for the entire system to process the interrupts, to determine a percent of time spent processing interrupts and related work. This percent of time is then divided by a target per-processor load, which can be a tunable parameter, to determine a target number of processors to handle that load. For example, if each of eight processors handled interrupts for 20ms, the total load is 160ms. If this processing occurred in 100ms, then the total load is 160% (of one of the processors). If the target processing load is 40%, then the target number of processors is four.

[0041] Other algorithms can be used to select the number of processors. For example, statistics about which interrupt sources are primarily responsible for the load can be examined. If one interrupt source is responsible for a load that is significantly greater than the target per-processor load, then it is possible that this interrupt source can be assigned to one processor and fewer processors can be used for the remaining interrupts.

[0042] The computed number can be capped 408 to the number of processors in the initial set of processors identified at 400. If this number of processors is less than the size of the initial set, as determined at 410, then a subset of the initial set is selected 412.

[0043] In one example implementation, the subset can be selected by using a same algorithm used by the kernel to assign threads to processors thus maximizing a number of processors that can be idle to reduce power consumption.

[0044] In this example implementation, a set of processors is selected (see Fig. 4) and then interrupts are distributed among the selected processors. An example implementation of this distribution process will now be described in connection with Fig. 5.

[0045] First, interrupt sources are sorted 500 in order of the load imposed on the system by the interrupt source. Next, an interrupt source from the list is selected 502. A processor from the set of processors is selected 504 and assigned to the selected interrupt source in the sorted list. If interrupt sources remain, as determined in 506, the process continues by repeating the selection steps 502 and 504, until all interrupt sources are assigned one of the processors. After assignments are completed, the interrupt controller is set 508 to direct interrupts to their assigned processors.

[0046] There are a variety of different ways to match processors and interrupts, and the process in Fig. 5 is merely one way of implementing such a matching.

[0047] Using the example of Fig. 5, the selection of the interrupt source in step 502 and the selection of a processor in step 504 can be performed using the following example implementation. The processors are selected in a serpentine manner: stepping through the list from beginning to end and then from the end to the beginning, and so on, assigning the next interrupt source in the sorted list to the selected processor, until all of the interrupt sources are assigned.

[0048] Other implementations are possible, with the general objective of evenly dividing the load due to handling interrupts among the processors, trading off computational complexity against a more optimal distribution.

[0049] In one example, a worst-fit bin packing algorithm can be used. In this algorithm, each subsequent interrupt source is assigned to the processor that will have the least total load after the assignment.

[0050] In another example, the serpentine selection of the next processor is followed, with the condition that the current processor has a higher total load than the next processor; otherwise the current processor is used again for the next interrupt.

[0051] In another example, an entire processor is first reserved for each interrupt for which the load is greater than the target per-processor load. Then, the remaining interrupts are assigned to the remaining processors.

[0052] In another example, interrupt sources having little impact on the system can be assigned permanently to certain processors and removed from this process of assigning interrupt sources to processors.

[0053] Given an assignment of interrupt sources and processors, this assignment can be dynamically changed during operation of the computer as system conditions change. The result of this process is to allow the system to transition active processors into idle states and idle processors into active states, while not missing interrupts.

[0054] An example implementation is shown in Fig. 6. Other implementations are possible, and depend on the nature of the processors being used. In the implementation shown in Fig. 6, the processors have both non-interruptible and interruptible idle states, and, in either idle state, a "wake" command is issued to the processor in order to query the processor and/or cause it to change state. In general, the process involves identifying the differences between the current interrupt source-to-processor assignments and the new interrupt source-to-processor assignments, and determining whether any of these differences involve changing the state of the processor. If a processor has a state change, the state is changed before allowing interrupt sources to be assigned to it.

[0055] Thus, in Fig, 6, in this implementation, each interrupt source-to-processor assignment is processed 600 to identify interrupt sources for which the target processor changed. For each interrupt source for which the current target processor is to be changed, the count of interrupt sources for the new target processor is incremented 602. On each such increment, if the target processor's count of interrupt sources changes from zero to one, as determined in 604, the processor is added 606 to a list of processors to be "awakened". "Wake" instructions then are sent 608 to each processor on this list. In some implementations, waiting for an acknowledgement from the processor ensures that an interrupt is not lost.

[0056] Each interrupt source-to-processor assignment is again processed 610 and the current assignment is set to the new assignment in the interrupt controller. For each interrupt source where the target processor changed, the count of interrupt sources assigned to the previous target processor for that interrupt is decremented 612. On each such increment, if the previous target processor's count of interrupt sources changes from one to zero, as determined at 614, the processor is added to a list. Instructions are then sent 616 to each processor on this list, in response to which each processor can evaluate whether it is idle and can transition into a power saving state due to no longer having any interrupts targeted at it.

[0057] The process in Fig. 6 can be modified depending on how the processors handle interrupts, power saving states, and transitions among those states. The state of each processor can be identified during this process. For example, if processors do not support wake commands or non-interruptible states, then other steps can be eliminated.

[0058] An example implementation of a process for tracking the impact of interrupt handling as associated work on the system will now be described in connection with Fig. 7.

[0059] A challenge with measuring the impact of interrupts is the fact that an interrupt handler can invoke additional processes and generate additional interrupts, or other work generated by an interrupt handler that is performed on the same processor as the interrupt handler. Such additional work, in turn, can create yet more additional work on a processor. Measurement and tracking of such work uses the data structure described above, an example of which is shown in Fig. 3.

[0060] For each interrupt source and processor in the system, fields are allocated 700 in this data structure to accumulate the amount of time consumed by the interrupt source's associated interrupt handler and any additional work it invokes on the processor.

[0061] When an interrupt handler is executed, the amount of time it spends executing is tracked and stored 702 on its processor. For example, a time stamp is read both prior to and after such execution. The difference is calculated and stored in the data structure. If the interrupt handler schedules additional work, such as by invoking another process, the amount of time the additional work is performed also is determined. For example, when additional work is queued on the processor, a pointer to the data structure can be queued along with that work. If the interrupt handler, or its related work, is preempted by another higher priority task, then the higher priority task adjusts the time stamp data of the preempted interrupt handler or related work.

[0062] Periodically, the interrupt data from the processors is accumulated 704 by the interrupt router, and stored 706 in the allocated data structure so as to accumulate the interrupt handling statistics across the system.

[0063] It should be understood that the foregoing is only an example implementation and that other implementations of tracking also are possible. In general, each interrupt handler and its additional work track the amount of time spent processing the related interrupt. This data is collected over the multiple interrupts and multiple processors.

[0064] With such information, interrupt sources can be dynamically assigned to processors, and such assignments can be dynamically changed, during runtime of the computer. By managing the assignments efficiently, interrupts can be distributed among processors while maximizing a number of processors that can be idle to reduce power consumption.

[0065] Any or all of the aforementioned alternate embodiments described herein may be used in any combination desired to form additional hybrid embodiments. It should be understood that the subject matter defined in the appended claims is not necessarily limited to the specific implementations described above. The terms “article of manufacture”, “process”, “machine” and “composition of matter” in the preambles of the appended claims are intended to limit the claims to subject matter deemed to fall within the scope of patentable subject matter defined by the use of these terms in 35 U.S.C. §101. The specific implementations described above are disclosed as examples only.

[0066] What is claimed is:

CLAIMS

1. A process performed by a computer including a plurality of processors, comprising:

receiving, into memory for an interrupt router, information about interrupts;

the interrupt router determining a load on the computer due to interrupt handling by the plurality of processors for the interrupts; and

the interrupt router assigning each interrupt source to a selected one of the plurality of processors, such selection being a function of the determined load so as to distribute the load among the processors.

2. The computer-implemented process of claim 1, wherein assigning comprises:

the interrupt router identifying a number of processors, from among the plurality of processors, available for processing interrupts;

the interrupt router selecting a number of processors, from among the identified number of processors available for processing interrupts, such that the number of selected processors matches the determined load divided by a target per-processor load.

3. The computer-implemented process of claim 1, wherein determining the load comprises:

in response to each interrupt from an interrupt source, executing an interrupt handler for the interrupt on the processor assigned to the interrupt source;

storing, in memory associated with the processor assigned to the interrupt source, data indicating an amount of processing time consumed due to executing the interrupt handler;

the interrupt router aggregating stored data from the plurality of processors for the interrupts.

4. The computer-implemented process of claim 3, further comprising determining the amount of processing time by:

storing a system time stamp when beginning execution of the interrupt handler;

computing a difference between a system time stamp observed when ending execution of the interrupt handler; and

storing the computed difference.

5. The computer-implemented process of claim 4, wherein determining the amount of processing time further comprises:

for any process invoked by the interrupt handler,

storing a system time stamp when beginning execution of the process;

computing a difference between a system time stamp observed when ending execution of the process; and

storing data indicative of the computed difference for the process and the computed difference for the interrupt handler.

6. The computer-implemented process of claim 5, wherein determining the amount of processing time further comprises:

for any preemptive activity preempting interrupt processing by the interrupt handler and associated processes,

computing a difference between system time stamps observed when pausing the preempted interrupt processing and when restarting the preempted interrupt processing to have an amount of time for executing the preemptive activity; and

wherein the computed difference for the interrupt handler excludes the amount of time for executing the preemptive activity.

7. The computer-implemented process of claim 1, further comprising the interrupt router periodically repeating the steps of:

determining a load on the computer due to interrupt handling by the plurality of processors for the interrupts since a previous assignment of interrupt sources to processors; and

assigning each interrupt source to a selected one of the plurality of processors, such selection being a function of the determined load since a previous assignment of interrupt sources to processors.

8. An article of manufacture comprising:

a computer storage medium;

computer program instructions stored on the computer storage medium which, when read from storage and processed by a processing device, instruct the processing device to perform a process comprising:

receiving, into memory for an interrupt router, information about interrupts;

the interrupt router determining a load on the computer due to interrupt handling by the plurality of processors for the interrupts; and

the interrupt router assigning each interrupt source to a selected one of the plurality of processors, such selection being a function of the determined load so as to distribute the load among the processors.

9. A computer comprising:

a plurality of processors, each processor being programmed to maintain information about interrupts handled on the processor, such information including an amount of processor time consumed due to executing the interrupt handler;

an interrupt router executed on one or more of the plurality of processors, the interrupt router being configured to:

receive the information about interrupts from the processors;

determine a load on the computer due to interrupt handling by the plurality of processors for the interrupts; and

assign each interrupt source to a selected one of the plurality of processors, such selection being a function of the determined load so as to distribute the load among the processors.

10. The computer of claim 9, wherein the interrupt router is configured to identify a number of processors, from among the plurality of processors, available for processing interrupts, and select a number of processors, from among the identified number of processors available for processing interrupts, such that the number of selected processors matches the determined load divided by a target per-processor load.

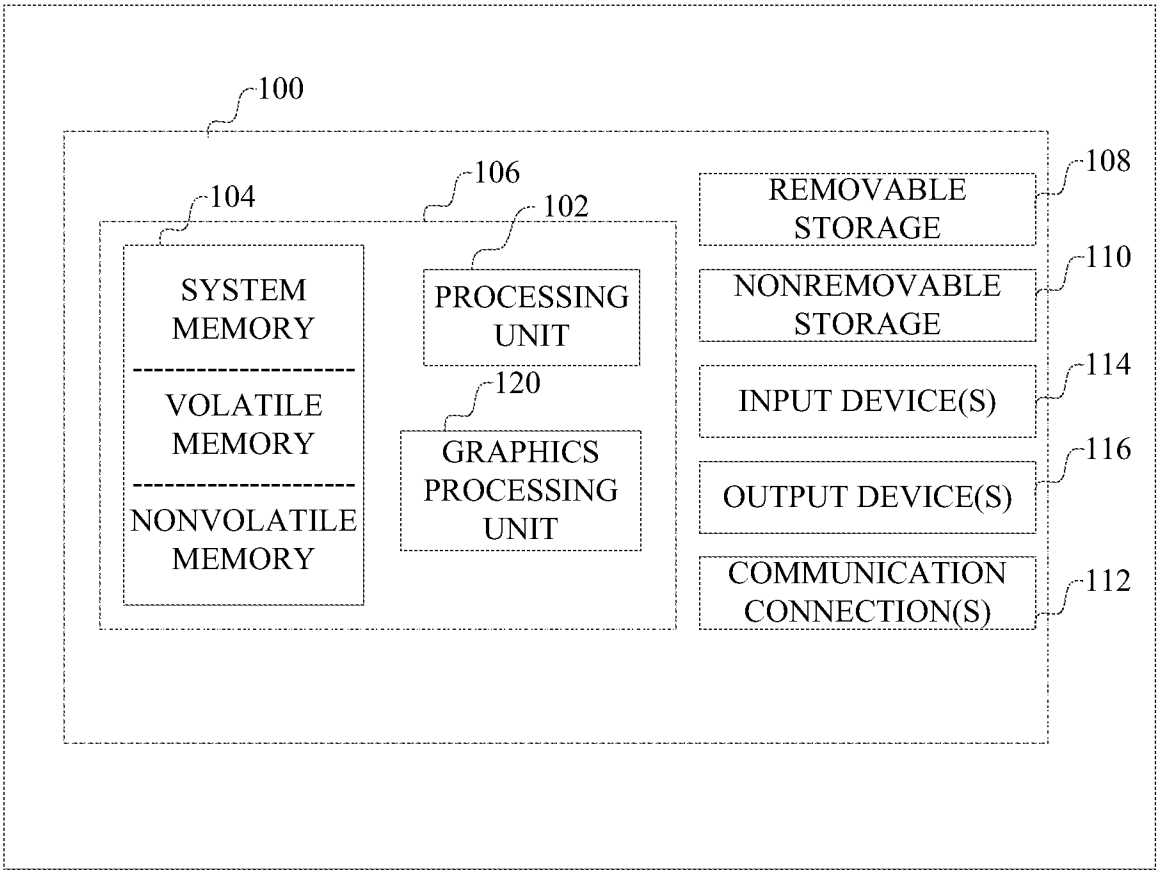
100
104
SYSTEM MEMORY
VOLATILE MEMORY
NONVOLATILE MEMORY
106
102
PROCESSING UNIT
120
GRAPHICS PROCESSING UNIT
108
REMOVABLE STORAGE
110
NONREMOVABLE STORAGE
114
INPUT DEVICE(S)
116
OUTPUT DEVICE(S)
112
COMMUNICATION CONNECTION(S)

FIG. 1

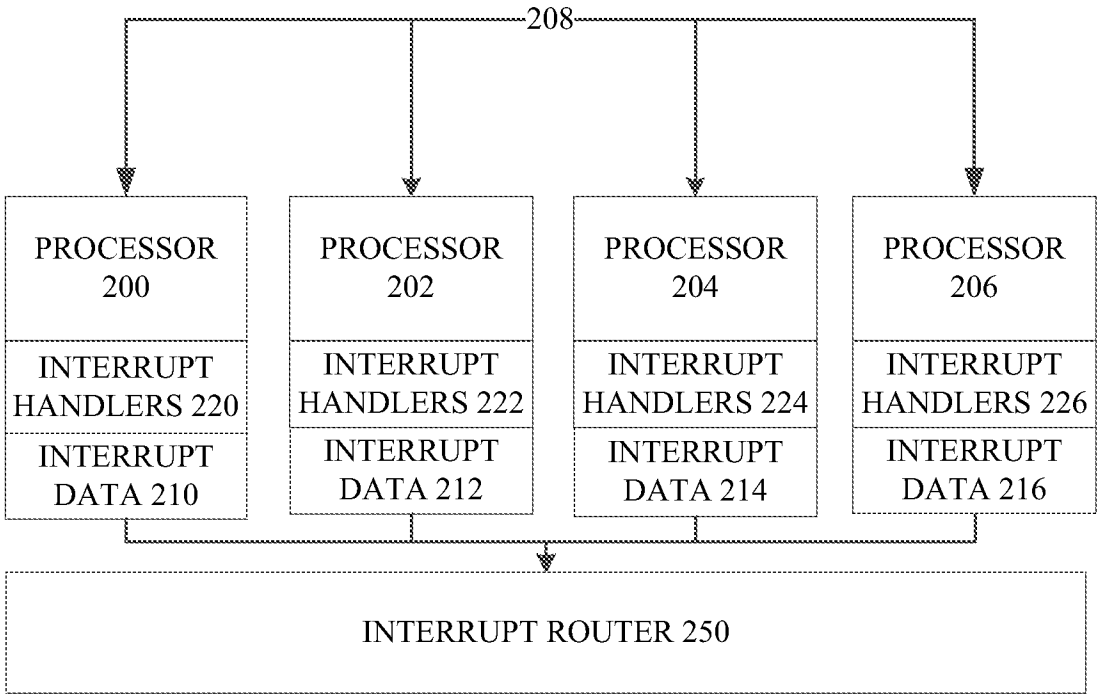
208
PROCESSOR 200
INTERRUPT HANDLERS 220
INTERRUPT DATA 210
PROCESSOR 202
INTERRUPT HANDLERS 222
INTERRUPT DATA 212
PROCESSOR 204
INTERRUPT HANDLERS 224
INTERRUPT DATA 214
PROCESSOR 206
INTERRUPT HANDLERS 226
INTERRUPT DATA 216
INTERRUPT ROUTER 250

FIG. 2

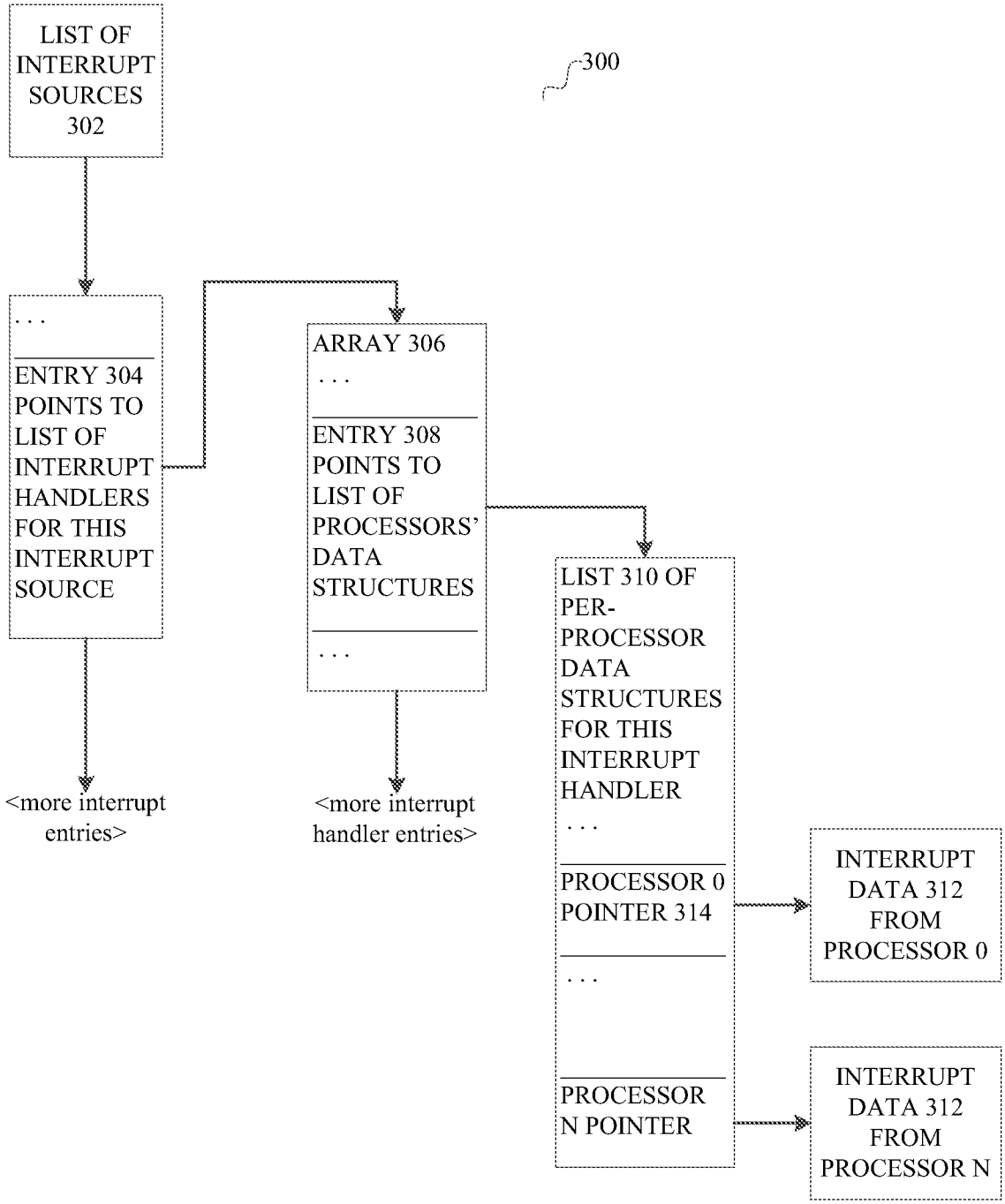
LIST OF INTERRUPT SOURCES 302
300
. . .
ENTRY 304 POINTS TO LIST OF INTERRUPT HANDLERS FOR THIS INTERRUPT SOURCE
<more interrupt entries>
ARRAY 306
. . .
ENTRY 308 POINTS TO LIST OF PROCESSORS' DATA STRUCTURES
. . .
<more interrupt handler entries>
LIST 310 OF PER-PROCESSOR DATA STRUCTURES FOR THIS INTERRUPT HANDLER
. . .
PROCESSOR 0 POINTER 314
. . .
PROCESSOR N POINTER
INTERRUPT DATA 312 FROM PROCESSOR 0
INTERRUPT DATA 312 FROM PROCESSOR N

FIG. 3

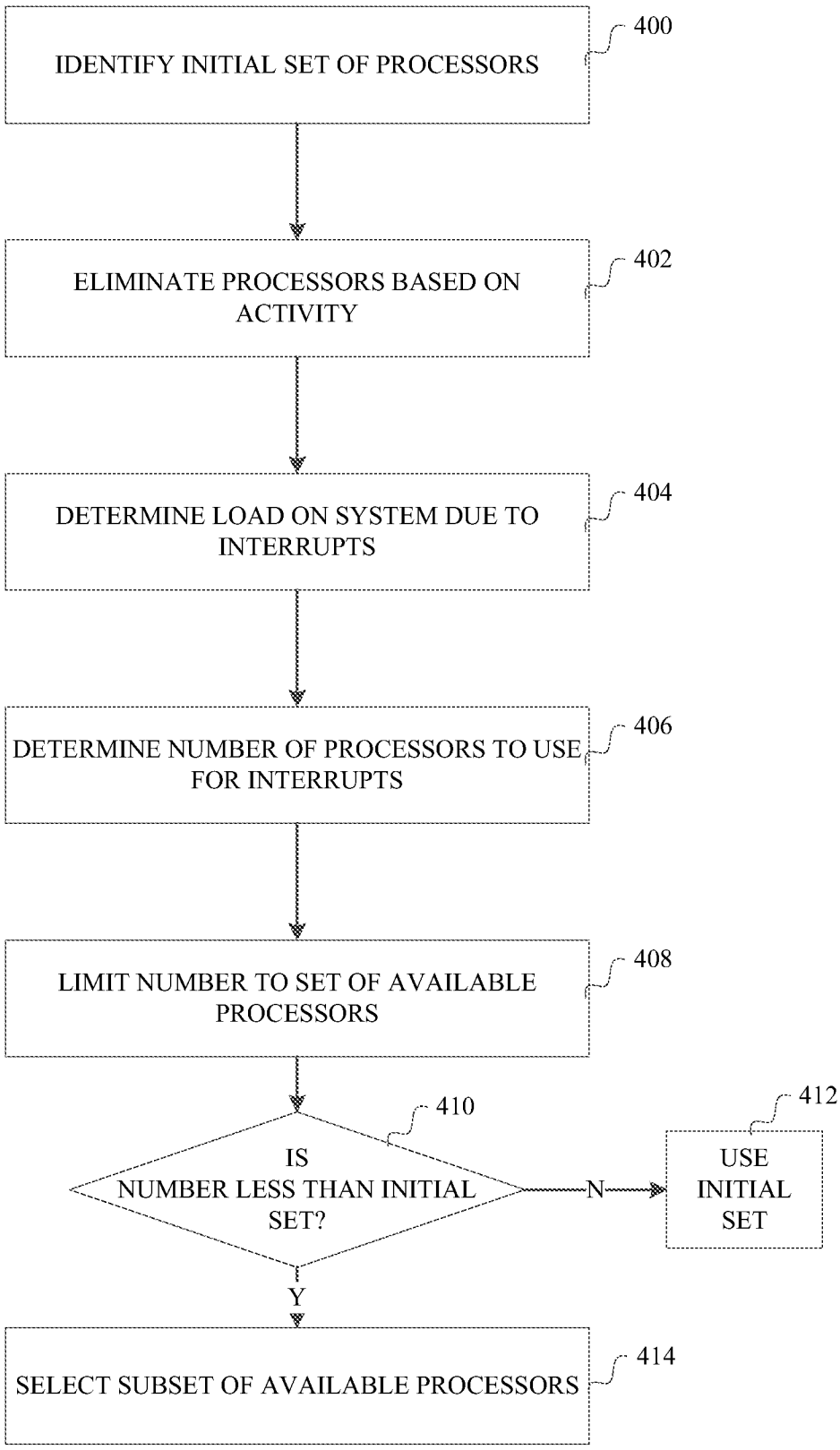
IDENTIFY INITIAL SET OF PROCESSORS
400
ELIMINATE PROCESSORS BASED ON ACTIVITY
402
DETERMINE LOAD ON SYSTEM DUE TO INTERRUPTS
404
DETERMINE NUMBER OF PROCESSORS TO USE FOR INTERRUPTS
406
LIMIT NUMBER TO SET OF AVAILABLE PROCESSORS
408
IS NUMBER LESS THAN INITIAL SET?
410
N
USE INITIAL SET
412
Y
SELECT SUBSET OF AVAILABLE PROCESSORS
414

FIG. 4

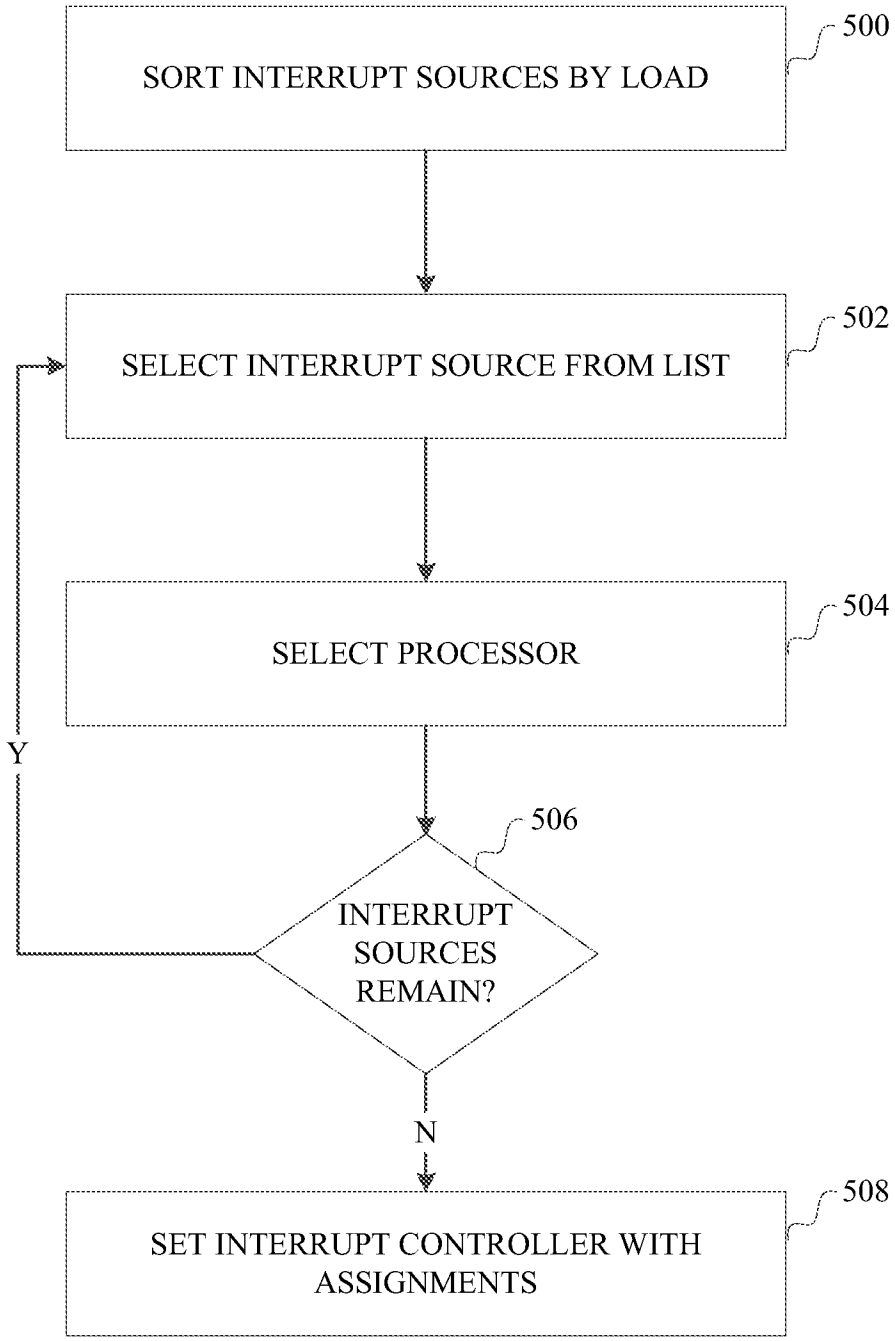
SORT INTERRUPT SOURCES BY LOAD
500
SELECT INTERRUPT SOURCE FROM LIST
502
SELECT PROCESSOR
504
Y
506
INTERRUPT SOURCES REMAIN?
N
SET INTERRUPT CONTROLLER WITH ASSIGNMENTS
508

FIG. 5

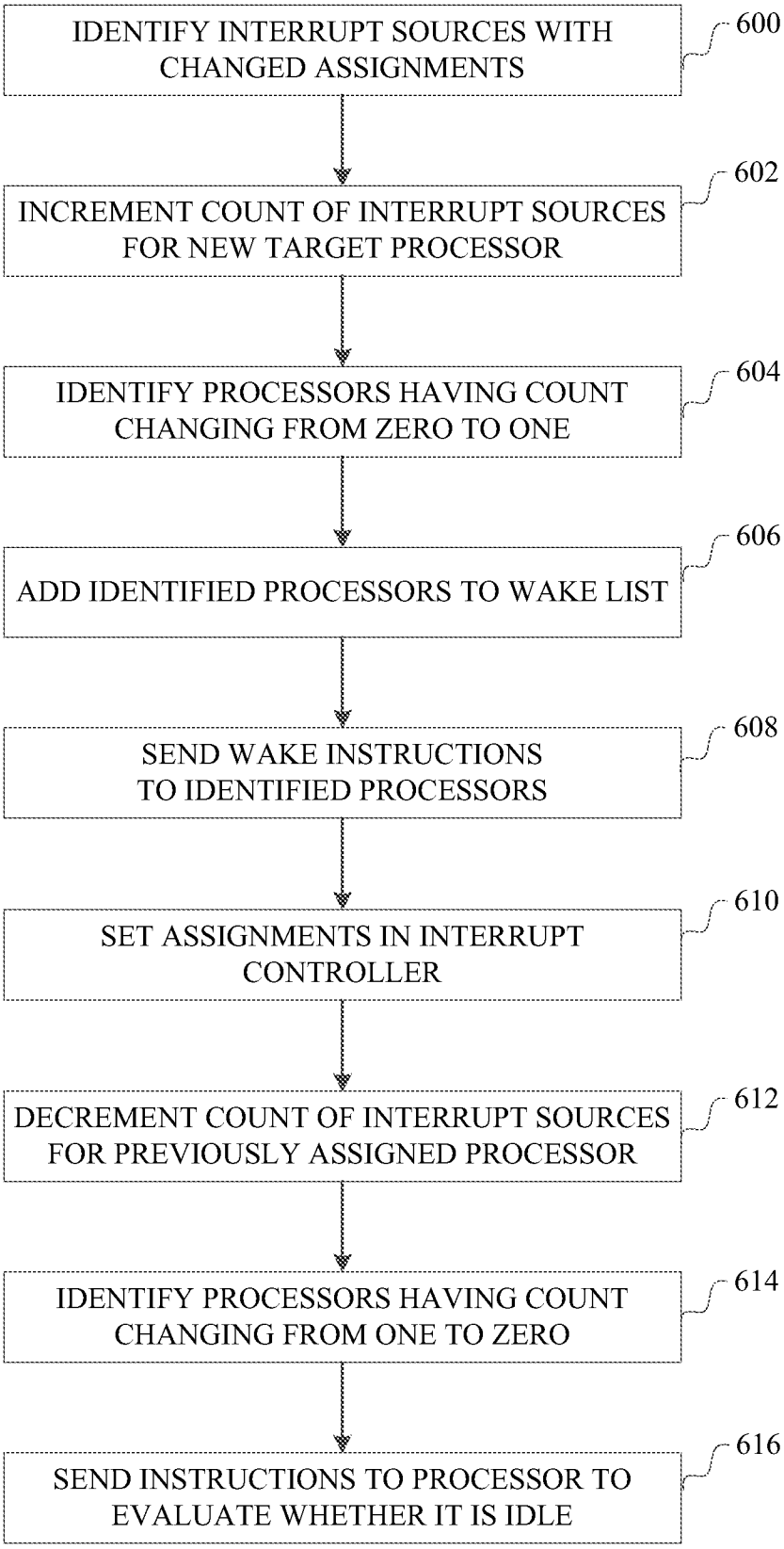
IDENTIFY INTERRUPT SOURCES WITH CHANGED ASSIGNMENTS
600
INCREMENT COUNT OF INTERRUPT SOURCES FOR NEW TARGET PROCESSOR
602
IDENTIFY PROCESSORS HAVING COUNT CHANGING FROM ZERO TO ONE
604
ADD IDENTIFIED PROCESSORS TO WAKE LIST
606
SEND WAKE INSTRUCTIONS TO IDENTIFIED PROCESSORS
608
SET ASSIGNMENTS IN INTERRUPT CONTROLLER
610
DECREMENT COUNT OF INTERRUPT SOURCES FOR PREVIOUSLY ASSIGNED PROCESSOR
612
IDENTIFY PROCESSORS HAVING COUNT CHANGING FROM ONE TO ZERO
614
SEND INSTRUCTIONS TO PROCESSOR TO EVALUATE WHETHER IT IS IDLE
616

FIG. 6

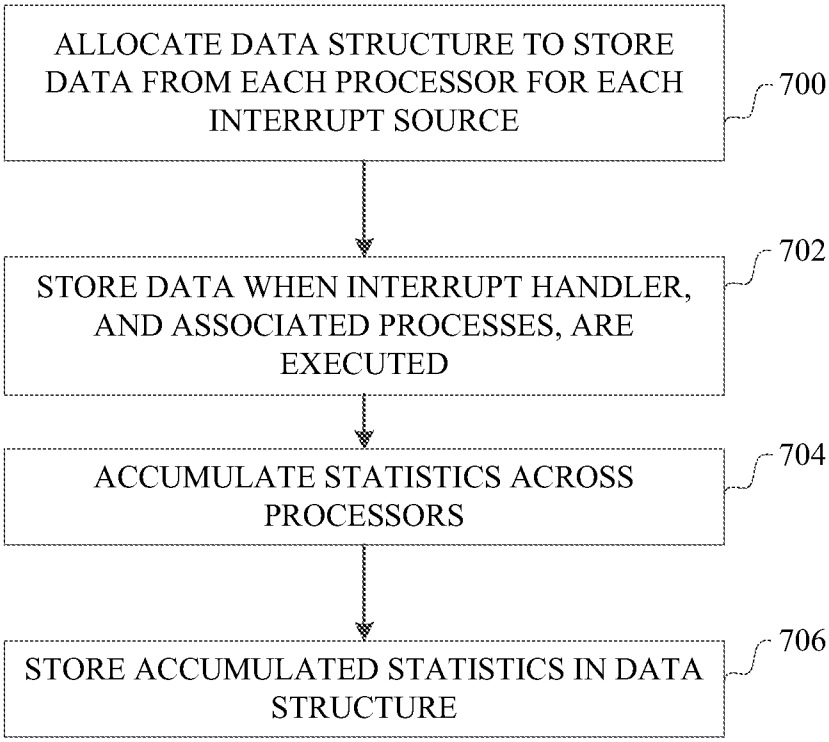
ALLOCATE DATA STRUCTURE TO STORE DATA FROM EACH PROCESSOR FOR EACH INTERRUPT SOURCE
700
STORE DATA WHEN INTERRUPT HANDLER, AND ASSOCIATED PROCESSES, ARE EXECUTED
702
ACCUMULATE STATISTICS ACROSS PROCESSORS
704
STORE ACCUMULATED STATISTICS IN DATA STRUCTURE
706

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/060243

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/50 G06F11/34
ADD.

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2010/057967 A1 (MURAKAMI TAKEO [JP] ET AL) 4 March 2010 (2010-03-04) paragraphs [0034], [0035] paragraphs [0075] - [0081] -----	1,2,7-10
X	US 7 581 052 B1 (SOLOMITA ETHAN [US]) 25 August 2009 (2009-08-25) column 2, lines 58-65 column 9, lines 13-20 column 9, line 65 column 15, lines 63-66 column 16, lines 50-56 column 24, line 37 - column 25, line 5 ----- -/--	1-10

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
17 February 2014	25/02/2014

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Kamps, Stefan

1

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/060243

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 2013/066124 A1 (SAMSUNG ELECTRONICS CO LTD [KR]) 10 May 2013 (2013-05-10) paragraph [0007] paragraphs [0033] - [0037] paragraph [0042] figure 3	1,2,7-10
A	US 2006/123422 A1 (FELTER WESLEY M [US] ET AL) 8 June 2006 (2006-06-08) the whole document	1-10
A	Richard Mcdougall: "Solaris (TM) Internals: Solaris 10 and OpenSolaris Kernel Architecture, Second Edition - Chapter 3.11" In: "Solaris (TM) Internals: Solaris 10 and OpenSolaris Kernel Architecture, Second Edition - Chapter 3.11", 10 July 2006 (2006-07-10), Prentice Hall, XP055102096, ISBN: 978-0-13-148209-8 * section 3.11.6 *	1-10
L	"Non-legal human translation of WO2013066124", , 17 February 2014 (2014-02-17), XP055102434, Retrieved from the Internet: URL:http://dummy.net [retrieved on 2014-02-17] the whole document	1-10

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2013/060243

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2010057967 A1	04-03-2010	JP 2010055296 A	11-03-2010
		US 2010057967 A1	04-03-2010
US 7581052 B1	25-08-2009	NONE	
WO 2013066124 A1	10-05-2013	KR 20130049110 A	13-05-2013
		WO 2013066124 A1	10-05-2013
US 2006123422 A1	08-06-2006	US 2006123422 A1	08-06-2006
		US 2008184256 A1	31-07-2008