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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 February 2016 (04.02.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/018212 A1

(51) International Patent Classification:
G06F 9/54 (2006.01)

(21) International Application Number:
PCT/US2014/048403

(22) International Filing Date:
28 July 2014 (28.07.2014)

(25) Filing Language: English

(26) Publication Language: English

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

(72) Inventor: **SPRAGUE, Fred A.**; 3222 Shallow Pond Drive, Ft. Collins, Colorado 80528 (US).

(74) Agents: **HAQ, M. Aamir** et al.; Hewlett-packard Company, Intellectual Property Administration, Mail Stop 35 3404 E. Harmony Road, Fort Collins, Colorado 80528 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, 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, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

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

Published:

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

WO 2016/018212 A1

(54) Title: MULTI-CORE PROCESSOR INCLUDING A MASTER CORE AND SLAVE CORES

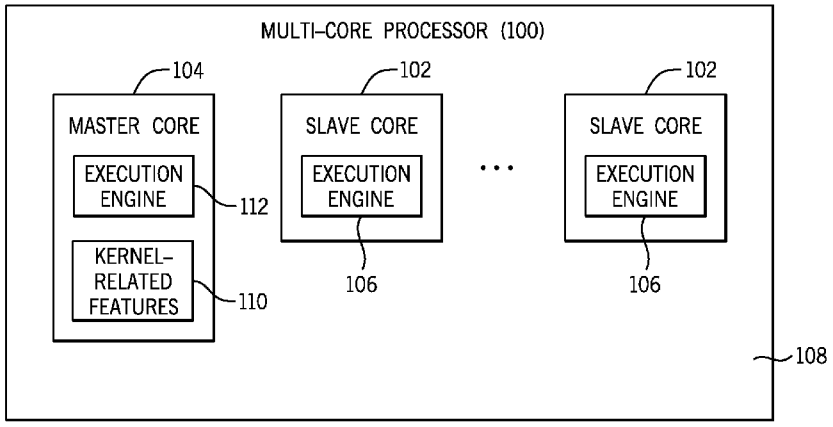

FIG. 1

(57) Abstract: A multi-core processor comprises a plurality of slave cores, the slave cores being without operating system kernel-related features, and the slave cores to execute respective instructions. A master core configures the operating system kernel-related features on behalf of the slave cores. The master core is to control usage of the operating system kernel-related features during execution of the instructions on the respective slave cores.

MULTI-CORE PROCESSOR INCLUDING A MASTER CORE AND SLAVE CORES

Background

[0001] A computing device can include a processor to perform computation tasks. In some examples, a processor can include multiple cores that are able to concurrently execute instructions.

Brief Description Of The Drawings

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

[0003] Fig. 1 is a block diagram of an example multi-core processor according to some implementations.

[0004] Fig. 2 is a flow diagram of example tasks of a multi-core processor according to some implementations.

[0005] Fig. 3 is a schematic diagram of an example arrangement that includes a slave core, a master core, a library interface, and a memory subsystem, according to some examples.

[0006] Fig. 4 is a schematic diagram of an example arrangement to migrate a process from one multi-core processor to another multi-core processor, according to further implementations.

Detailed Description

[0007] Each of multiple cores of a multi-core processor can include elements to support the execution of machine-readable instructions. The elements of a core can include an execution engine (or execution pipeline), which performs the execution of machine-readable instructions. In addition, the elements of the core can also include various features that are related to an operating system kernel. An “operating system kernel” (or more simply, a “kernel”) can refer to a part of an operating system of a computing device (*e.g.* desktop computer, notebook computer, tablet computer, server computer, smartphone, personal digital assistant, game appliance, etc.) that

manages various central tasks of the operating system, such as managing memory space in which machine-readable instructions are executed, managing requests for use of hardware components, managing a file system, and so forth.

[0008] The features related to a kernel (also referred to as "kernel-related features" or "operating system kernel-related features") are those features that are useable by and/or controlled by the kernel. Certain actions associated with machine-readable instructions, or more simply "instructions" (such as those of an application process), executed on a core can invoke or otherwise involve the kernel. An application process can refer to a process of an application program that executes in a computing device. Note that a process can refer to executable instructions, and can also refer to a thread of the process (for examples where multiple threads of a process can be executed).

[0009] As examples, if instructions executed on the core attempt to access a hardware component, *e.g.* a graphics controller, a network interface controller, etc., then the kernel is invoked to perform the access of the hardware component on behalf of the machine-readable instructions. In some examples, the features of the core that are used by the kernel to access a hardware component can include an Advanced Configuration and Power Interface (ACPI) interface logic that the kernel can access to obtain information about a specific hardware component.

[0010] As another example, when instructions executing on a core attempt to access memory, such as a read of or write to memory, then the kernel may be invoked under certain conditions. A core can include control logic for controlling a translation look-aside buffer (TLB), which includes entries that each includes information to translate a virtual address to a physical address. A virtual address is part of a virtual memory space as seen by machine-readable instructions executing on a core. A physical address identifies a location of physical memory.

[0011] The TLB has a specific number of entries that can store a relatively small subset of translations between virtual addresses and physical addresses, such as the translations for the most recently accessed virtual addresses. When instructions

issue a memory access to a virtual address, the core performs a lookup of the TLB to determine if the corresponding virtual-to-physical address translation is in the TLB. If so, then a TLB hit has occurred, and the mapped physical address from the TLB is used to access the respective location in physical memory.

[0012] However, if the accessed virtual address is not included in the TLB, then a TLB miss occurs. A TLB miss can result in an exception that is handled by the kernel. The kernel accesses data structures (such as page tables, for example) that store all virtual-to-memory address translations to retrieve the corresponding translation information associated with the TLB miss. The retrieved translation information can then be used to populate, using the TLB control logic of the core, the TLB. The physical address derived from the retrieved translation information can be used to complete the memory access.

[0013] In addition to the foregoing, a core can also include other kernel-related features.

[0014] The cores of a traditional multi-core processor are configured as full-functional cores with various kernel-related features used or controlled by a kernel. Instructions executing on a core can perform a system call when performing certain tasks, such as to access a hardware component, perform memory management, process control, and so forth. A system call triggers the kernel to perform a corresponding task, such as to access information of a hardware component, perform a memory management task (that includes handling a TLB miss, for example), process control (to schedule and/or control the execution of one or multiple processes), and so forth.

[0015] Upon triggering the kernel, a context switch is made from the user space (space of the application process, for example) to the kernel space. A “context” can include information regarding a state of execution of a process (*e.g.* the application process or the kernel), and data used by the process. Context switching is wasteful of resources of a core, since the content of cache memory and registers is swapped with the different context.

[0016] In accordance with some implementations, as shown in Fig. 1, a multi-core processor 100 is provided with multiple simplistic slave cores 102 and a master core 104 (or multiple master cores). In the ensuing discussion, reference is made to a multi-core processor that has one master core; however, it is noted that techniques or mechanisms according to some implementations are applicable to a multi-core processor that includes multiple master cores. The slave cores 102 are relatively simple cores without kernel-related features (*e.g.* TLB control logic, ACPI interface logic, etc.). However, each slave core 102 includes a respective execution engine 106 that is able to execute instructions of application processes, for example.

[0017] The master core 104 and the slave cores 102 can be integrated into a single integrated circuit chip or die, for example. In other examples, the master core 104 and the slave cores 102 can be provided on a circuit board. In Fig. 1, the master core 104 and the slave cores 102 are arranged on a support structure 108, where the support structure 108 can be an integrated circuit chip or die, or a circuit board, as examples.

[0018] Without kernel-related features, the slave cores 102 are unable to enter into a privileged mode (also referred to as a "ring 0 mode" in some examples) in which the kernel is executed on the respective slave core 102 to perform a requested task. As a result, instructions can run on a slave core 102 with reduced interruption, since context switching does not have to be performed between a user space and kernel space. This can increase the efficiency of usage of a slave core 102 by an application process, and can also increase the security of the system as the slave core cannot access kernel-related features except through the defined interface to the master core. Additionally, the slave cores 102 can have simpler designs (than fully functional slave cores) that take up less space on a die and consume less power.

[0019] The master core 104 includes kernel-related features 110. At least a portion of the kernel-related features 110 is configured by the master core 104 for the slave cores 102. In some examples, the kernel-related features 110 can be divided into multiple groups of kernel-related features 110, where each group of

kernel-related features is dedicated to a respective one of the slave cores 102. A group of kernel-related features is dedicated to a slave core 102 if the group of kernel-related features is for use in relation to tasks of instructions executing on the slave core 102. Note that another portion of the kernel-related features 110 is dedicated to the master core 104.

[0020] In examples where multiple master cores 104 are included in a multi-core processor, each of the master cores 104 can include its respective kernel-related features 110.

[0021] For example, different TLB control logic (and respective groups of TLBs) can be configured for different slave cores 102, where a first TLB control logic and a respective group of TLBs can be configured for a first slave core 102, a second TLB control logic and a respective group of TLBs can be configured for a second slave core 102, and so forth. In this manner, by dedicating respective TLB control logic and a respective group of TLBs to each slave core 102, more efficient TLB operation can be achieved. This can avoid the situation where TLBs and respective control logic are shared by multiple application processes, which can result in reduced efficiency if the content of TLBs have to be swapped depending on which of the multiple application processes is active. Note that although the TLB control logic can be part of the master core 104, the TLBs themselves may be located separately from the master core 104; for example, the TLBs can be part of a memory subsystem.

[0022] By configuring kernel-related features by the master core 104 for the slave cores 102, the kernel-related features are available for use during execution of instructions on the slave cores 102, without having to include such kernel-related features in the slave cores 102. Configuring kernel-related features by the master core 104 for a slave core 102 can refer to setting up or otherwise establishing the kernel-related features for use by or on behalf of the slave core 102.

[0023] During execution of instructions (such as of application processes) on the slave cores 102, the master core 104 controls usage of the kernel-related features 110 on behalf of the slave cores 102 when appropriate, which can reduce context

switching, cache line loss, and so forth at the slave cores 102, and which can lead to improved performance. As an example, if TLB content is to be updated for access of a file by an application process executing on a slave core 102, the master core 104 can be triggered to perform the update of the TLB content.

[0024] As further shown in Fig. 1, the master core 104 can also include an execution engine 112 for executing instructions on the master core 104. Although not shown in Fig. 1, each master core 104 and slave core 102 can include local cache memory, registers, bus interfaces, and so forth.

[0025] Fig. 2 is a flow diagram of tasks that can be performed by the multi-core processor 100 of Fig. 1. Instructions are executed (at 202) on respective slave cores 102 of the multi-core processor 100, where the slave cores 102 are without kernel-related features. The master core 104 configures (at 204) kernel-related features (110 in Fig. 1) for the slave cores 102. The master core 104 controls (at 206) usage of the kernel-related features during execution of the instructions on the respective slave cores 102.

[0026] In some examples, threads of a single process can run on multiple slave cores 102 of the multi-processor core 100 of Fig. 1. These threads of the single process can share a context domain—in other words, the threads share data that can be stored in a memory subsystem. If instructions executing on multiple slave cores 102 share a context domain, then a coherency mechanism can be provided to ensure data coherency of the shared data between the multiple slave cores 102.

[0027] In a computing device that includes multiple multi-processor cores 100, the different multi-processor cores 100 can execute respective different processes. In this execution mode, context switching between processes does not occur, which can result in increased performance for the processes.

[0028] Cores that are not actively executing processes can be powered down, or at least have their execution clocks stopped, thereby providing power savings.

[0029] Fig. 3 is a schematic diagram of an arrangement that includes a slave core 102, the master core 104, an interface library 306, and a memory subsystem 312. The memory subsystem 312 can include memory media (*e.g.* one or multiple memory devices and associated memory management logic located locally or remotely).

[0030] The interface library 306 includes library routines 308 (which can be in the form of machine-readable instructions) that can be invoked in response to calls from a process (*e.g.* 302) executed by the execution engine 106 of the slave core 102. The interface library 306 can be implemented as part of the multi-core processor 100 of Fig. 1, or external of the multi-core processor 100. In some examples, the interface library 306 can be part of the master core 104.

[0031] Fig. 3 shows an example in which the process 302 makes a call (304) to write a file, which can be stored in the memory subsystem 312. Note that the call (304) differs from a system call performed by a process to trigger a kernel to perform a requested task.

[0032] Instead, the call (304) triggers execution of a library routine 308 in the library interface 306. Note that different calls (corresponding to different actions) can cause triggering of different library routines 308. Once the process 302 makes the call (304), the process 302 stops or pauses while waiting for a response to the call (304), or in other cases (such as this example of writing to a file), continues to execute without interruption. The system response to the invoked library routine 308 can be handled by the master core 104. The invoked library routine 308 can provide information pertaining to the call (304) to a kernel-related features management logic 310 that is part of the master core 104. The kernel-related features management logic 310 can be implemented as hardware circuitry or as hardware processing circuitry and machine-readable instructions executed on the hardware processing circuitry. In some examples, the kernel-related features management logic 310 can configure kernel-related features for a slave core 102, and control usage of the kernel-related features during execution of instructions on a slave core.

[0033] The kernel-related features management logic 310 determines from the information pertaining to the call (304) that a write to a file is requested. In response, the kernel-related features management logic 310 can interact with the memory subsystem 312 to determine an address for the file. The determination of the address can be based on obtaining a virtual address-to-physical address translation information, for example. This address translation information can be returned to the kernel-related features management logic 310, which can update a respective TLB, for example.

[0034] The master core 104 then can instruct, through the invoked library routine 308, the process 302 in the slave core 102 to continue running. In some cases (such as when writing a file), the slave core 102 does not pause in execution but simply writes the file contents to memory and continues execution, depending on another core (the master core104 in this case) to consume the write and deliver the write to the appropriate destination.

[0035] In further cases, it may not be the master core 104 that handles the library response; instead, another slave core can handle the response for the invoked library routine 308. An example of this may be when a database request is issued by a first slave core, in which case a second slave core has access to the database and the first slave core makes a request for data from that database. For this type of request all communications between the two slave cores would be through a common memory interface handled by the interface library (306)

[0036] More generally, instead of providing the library interface 306 that includes library routines 308, an interface can be provided between the slave core 102 and the master core 104, to allow a call made by a process running on the slave core 102 to trigger a respective action at the master core 104. In this way, execution of a kernel at the slave core 102 can be avoided, which can improve efficiency of the slave core 102.

[0037] Certain tasks, such as memory management tasks (which can involve management with respect to TLBs) and hardware component accesses, can be

performed by the master core 104 on behalf of a slave core 102, without triggering kernel interaction. By providing the library interface 306 (or other like interface), an application process does not have to be aware that no kernel is involved in response to calls made by the application process to perform certain tasks.

[0038] However, there may be certain situations where a kernel has to be triggered, such as when a memory exception occurs due to memory running out of space. In such cases, the master core 104 is involved to resolve the request.

[0039] By employing the master core 104 to handle kernel functions, the slave cores 102 do not have to pause or context switch in some cases. Traditionally, even simple kernel functions, such as "getPid()" (to obtain the identifier of a process) or "getTimeOfDay()" (to obtain a time of day), trigger a kernel context switch to return a simple numeric value. The kernel context switch refers to performing a context switch between the user space and kernel space, which pauses the user process. By employing the master core 104 to perform kernel functions (such as those above), the kernel functions can return values that are accessible to the slave cores directly without kernel involvement. The returned values can be in write-protected memory.

[0040] Other functions, such as "openFile()" (to open a file), involve kernel action that would determine physical access to the file and map the file into the slave core 102 address space. In the case of openFile(), the user process running on a slave core 102 may or may not be aware of the address mapping. If not aware of the address mapping, then the interface library 306 can hide that mapping through existing interfaces such as "read()" accessing the mapped addresses without any kernel involvement after the initial mapping.

[0041] Kernel functions such as adding additional memory resources or handling illegal memory access can also automatically trigger kernel involvement, pausing the slave core 102 until the master core 104 can resolve the issue.

[0042] Fig. 4 shows an example of a computing device that includes multiple multi-core processors (100-A and 100-B shown in Fig. 4). In accordance with further implementations, it is possible to move or migrate (at 404) a process (*e.g.* 402) running on a slave core 102-A of the multi-core processor 100-A to a slave core 102-B of the multi-core processor 100-B. The process 402 can refer generally to instructions that can execute on a slave core.

[0043] The "context" of a process in this implementation is primarily the mapping of the process' memory addresses to the physical memory space. That mapping is held and controlled in the master core 104. To migrate a process from one slave core (102-A) to another slave core (102-B), the same mapping can be created in the destination slave core (102-B) to match the mapping in the source slave core, to allow the destination slave core to continue execution where the source slave core left off.

[0044] The moving of the process 402 is managed by migration logic 406-A and 406-B in respective master cores 104-A and 104-B of the multi-core processors 100-A and 100-B. Each migration logic 406-A and 406-B can be implemented as hardware circuitry or as a combination of hardware processing circuitry and machine-readable instructions executed on the hardware processing circuitry.

[0045] A migration request can be submitted to the migration logic 406-A of the master core 104-A that it is desired to move the process 402 from the multi-core processor 100-A to the multi-core processor 100-B. This migration request can be submitted by the process 402 itself, or by another entity (*e.g.* a user, an operating system, an application, etc.). Moving the process 402 from one multi-core processor 100-A to another multi-core processor 100-B includes moving the context of the process 402 between the multi-core processors 100-A and 100-B. The context can include data in a local cache memory and registers of the slave core 102-A.

[0046] In response to the moving of the process 402 from the multi-core processor 100-A to the multi-core processor 100-B, the migration logic 406-B in the master core 102-B of the multi-core processor 100-B configures kernel-related

features for the slave core 102-B of the multi-core processor 100-B. Also, the context of the process 402 is moved from the slave core 102-A to the slave core 102-B.

[0047] After configuring the kernel-related features for the slave core 102-B, the process 402 can start running on the slave core 102-B.

[0048] Certain logic (*e.g.* kernel-related features management logic 310 of Fig. 3 and migration logic 406-A and 406-B of Fig. 4) can be implemented with hardware processing circuitry or as a combination of machine-readable instructions and hardware processing circuitry. The machine-readable instructions can be executed on the hardware processing circuitry.

[0049] Storage media can be used to store machine-readable instructions, where the storage media can include different forms of memory including semiconductor memory devices such as dynamic or static random access memories (DRAMs or SRAMs), erasable and programmable read-only memories (EPROMs), electrically erasable and programmable read-only memories (EEPROMs) and flash memories; magnetic disks such as fixed, floppy and removable disks; other magnetic media including tape; optical media such as compact disks (CDs) or digital video disks (DVDs); or other types of storage devices. Note that the instructions discussed above can be provided on one computer-readable or machine-readable storage medium, or can be provided on multiple computer-readable or machine-readable storage media distributed in a large system having possibly plural nodes. Such computer-readable or machine-readable storage medium or media is (are) considered to be part of an article (or article of manufacture). An article or article of manufacture can refer to any manufactured single component or multiple components. The storage medium or media can be located either in the machine running the machine-readable instructions, or located at a remote site from which machine-readable instructions can be downloaded over a network for execution.

[0050] In the foregoing description, numerous details are set forth to provide an understanding of the subject disclosed herein. However, implementations may be

practiced without some of these details. Other implementations may include modifications and variations from the details discussed above. It is intended that the appended claims cover such modifications and variations.

What is claimed is:

1. A multi-core processor comprising:

a plurality of slave cores, the slave cores being without operating system kernel-related features, and the slave cores to execute respective processes; and

a master core to configure the operating system kernel-related features on behalf of the slave cores,

wherein the master core is to control usage of the operating system kernel-related features during execution of the processes on the respective slave cores.

2. The multi-core processor of claim 1, wherein the master core is to configure dedicated groups of the operating system kernel-related features for the respective slave cores, wherein each of the groups is dedicated to a respective one of the slave cores.

3. The multi-core processor of claim 1, wherein the operating system kernel-related features established by the master core includes one or a combination of control logic to control translation look-aside buffers (TLBs) and interface logic to access information of a hardware component.

4. The multi-core processor of claim 1, wherein the master core includes an interface to handle a request from a given slave core of the slave cores to perform a task with respect to memory management, access of a hardware component, or process control.

5. The multi-core processor of claim 1, wherein the master core is to perform memory management or access of a hardware component for a process running on a given one of the slave cores, without invoking an operating system kernel.

6. The multi-core processor of claim 1, wherein when a request of a given process running on a given one of the slave cores involves use of a respective one of the operating system kernel-related features, the master core manages use of the respective operating system kernel-related feature.

7. The multi-core processor of claim 1, wherein a process running on a given one of the slave cores is executable to call a library including library routines when processing a request, the call of the library triggering performance of a task related to the request by at least one of the slave cores or the master core.

8. The multi-core processor of claim 7, wherein the task performed by the master core includes an access of a memory subsystem to identify an address corresponding to the request.

9. The multi-core processor of claim 1, wherein instructions running on respective multiple slave cores of the plurality of slave cores are able to share a context.

10. A method comprising:

executing processes on respective slave cores of a multi-core processor, the slave cores being without operating system kernel-related features;

configuring, by a master core in the multi-core processor, the operating system kernel-related features for the slave cores; and

controlling, by the master core, usage of the operating system kernel-related features during execution of the processes on the respective slave cores.

11. The method of claim 10, further comprising:

moving a given process of the processes from a first of the slave cores to a second multi-core processor, the second multi-core processor including slave cores and a master core;

in response to the moving, establishing, by the master core of the second multi-core processor, operating system kernel-related features for a given one of the slave cores of the second multi-core processor; and

after establishing the operating system kernel-related features for the given slave core of the second multi-core processor, run the given process on the given slave core of the second multi-core processor.

12. The method of claim 10, wherein configuring the operating system kernel-related features for the slave cores comprises configuring translation look-aside buffers for the slave cores.

13. The method of claim 10, wherein configuring the operating system kernel-related features for the slave cores comprises configuring interface logic to access information of hardware components for the slave cores.

14. A computing device comprising:

a plurality of multi-core processors, where each of the multi-core processors comprises:

a plurality of slave cores, the slave cores being without operating system kernel-related features, and the slave cores to execute respective instructions; and

a master core to configure the operating system kernel-related features on behalf of the slave cores,

wherein the master core is to control usage of the operating system kernel-related features during execution of the instructions on the respective slave cores.

15. The computing device of claim 14, wherein the instructions executable on the slave cores of a first of the multi-core processors include instructions of threads of a first process that share a context domain, and wherein the instructions executable on the slave cores of a second of the multi-core processors include instructions of threads of a second process that share a context domain.

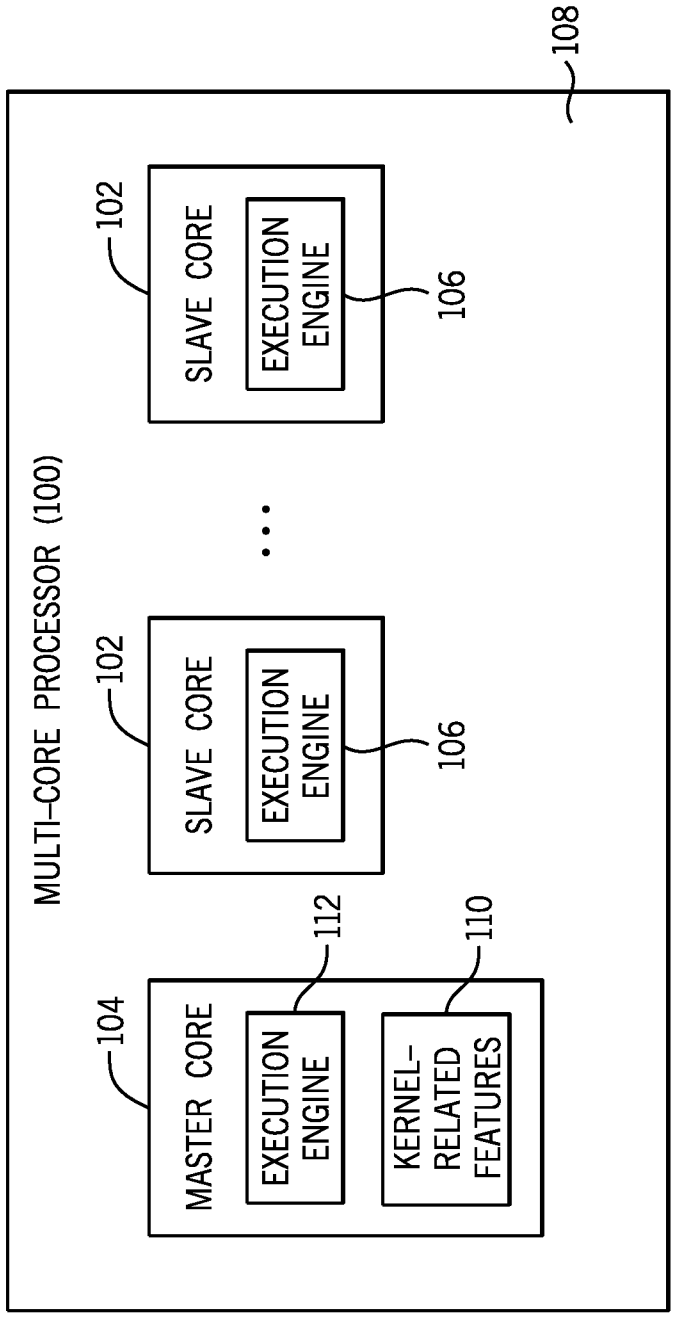
108
MULTI-CORE PROCESSOR (100)
102
SLAVE CORE
EXECUTION ENGINE
106
102
SLAVE CORE
EXECUTION ENGINE
106
104
MASTER CORE
EXECUTION ENGINE
112
KERNEL-RELATED FEATURES
110

FIG. 1

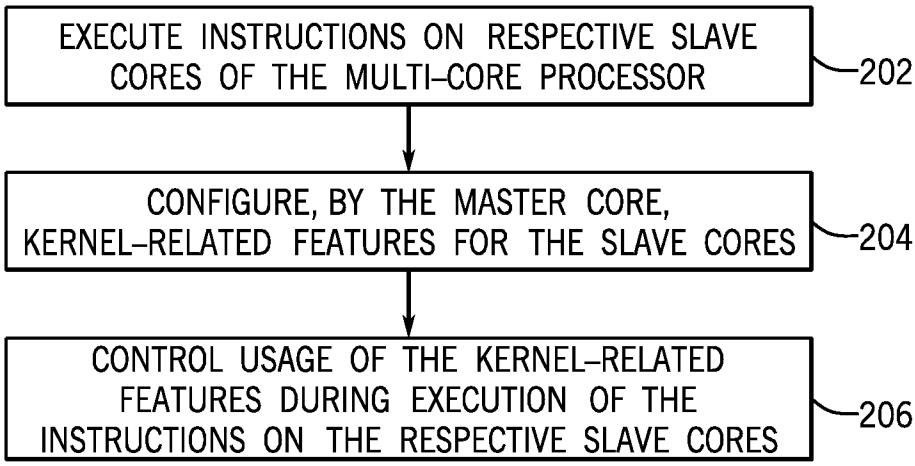
EXECUTE INSTRUCTIONS ON RESPECTIVE SLAVE CORES OF THE MULTI-CORE PROCESSOR
202
CONFIGURE, BY THE MASTER CORE, KERNEL-RELATED FEATURES FOR THE SLAVE CORES
204
CONTROL USAGE OF THE KERNEL-RELATED FEATURES DURING EXECUTION OF THE INSTRUCTIONS ON THE RESPECTIVE SLAVE CORES
206

FIG. 2

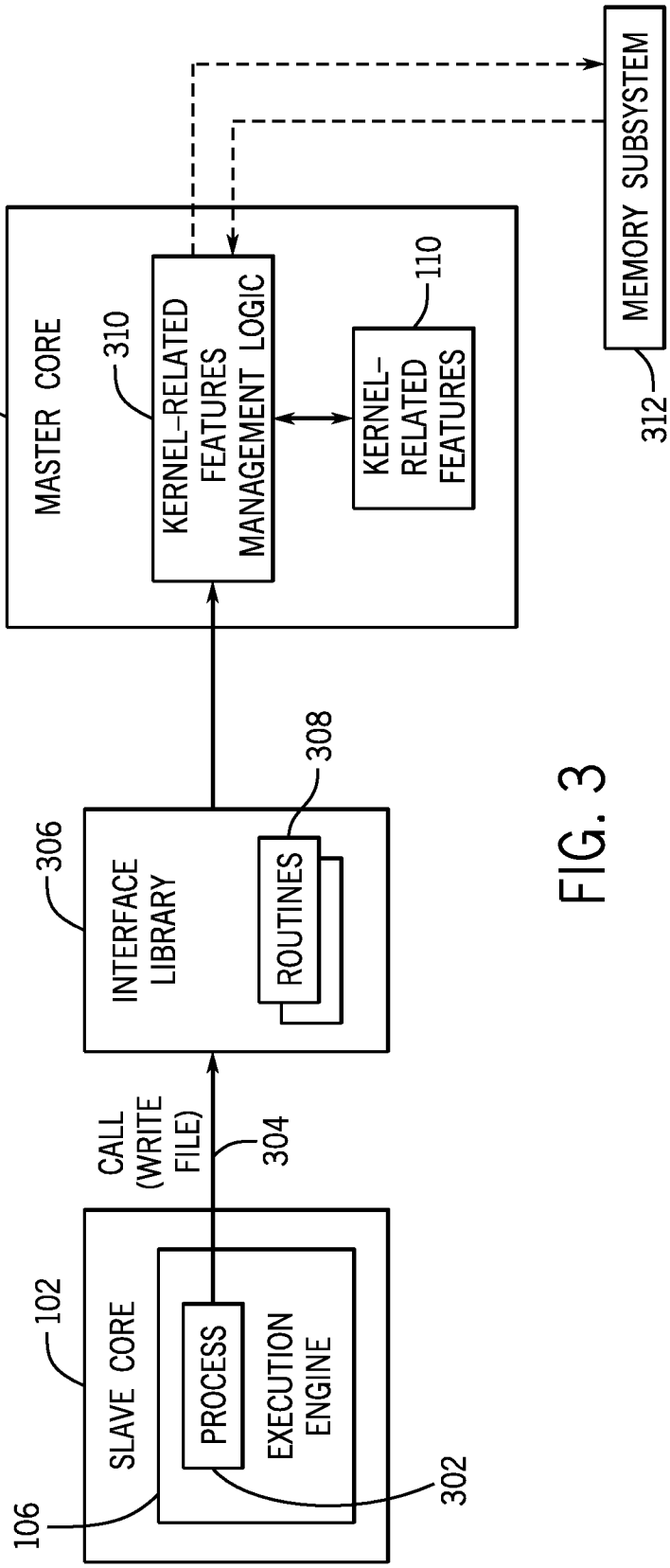
MEMORY SUBSYSTEM
312
104
MASTER CORE
310
KERNEL-RELATED FEATURES MANAGEMENT LOGIC
110
KERNEL-RELATED FEATURES
306
INTERFACE LIBRARY
308
ROUTINES
CALL (WRITE FILE)
304
102
SLAVE CORE
PROCESS
EXECUTION ENGINE
106
302

FIG. 3

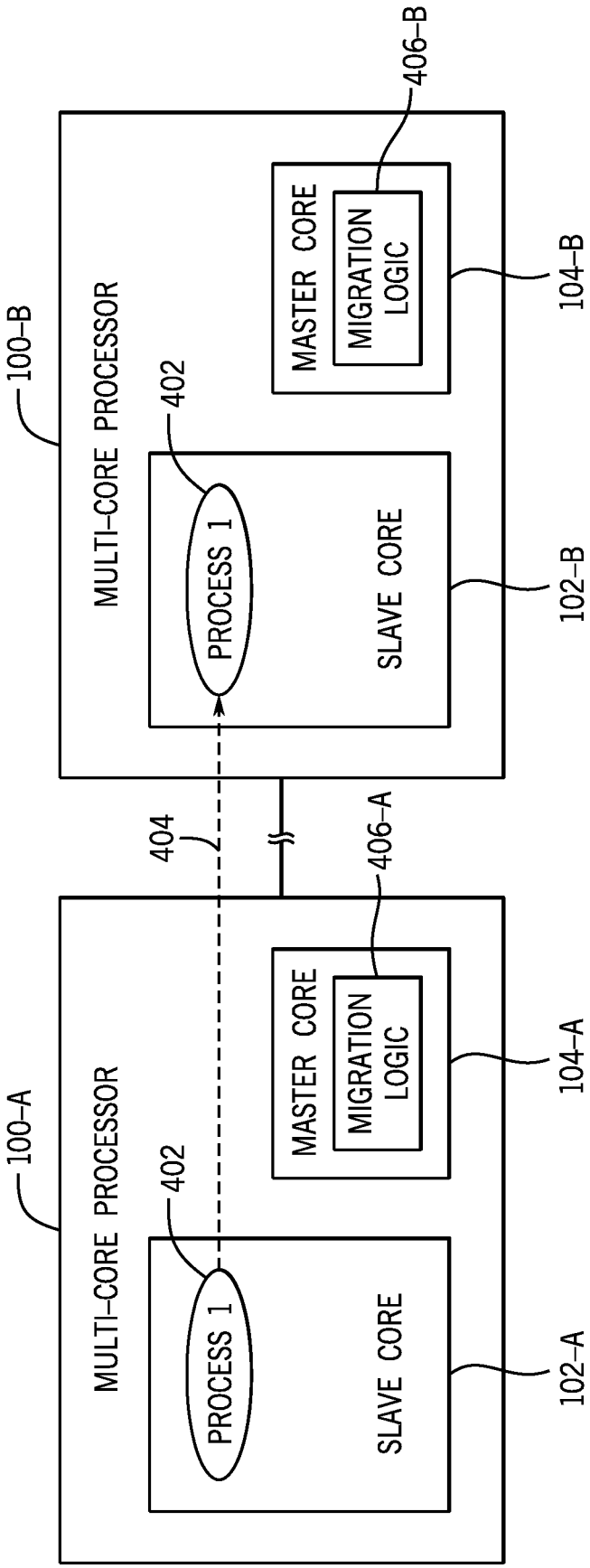
100-B
MULTI-CORE PROCESSOR
402
PROCESS 1
SLAVE CORE
102-B
MASTER CORE
MIGRATION LOGIC
406-B
104-B
404
100-A
MULTI-CORE PROCESSOR
402
PROCESS 1
SLAVE CORE
102-A
MASTER CORE
MIGRATION LOGIC
406-A
104-A

FIG. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/048403

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/54(2006.01)i

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F 9/54; G06F 9/30; G06F 17/50; G06F 9/445; G06F 9/46; G06F 1/32; H04B 1/40; G06F 9/50

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords:multi-core, master core(master processor), slave core(slave processor), kernel, operating system, kernel-related features

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2009-0307464 A1 (STEINBERG EREZ et al.) 10 December 2009 See abstract; paragraphs [0012]-[0020], [0025]-[0027]; claims 9-16 and figures 1-2.	1,10,14
A		2-9,11-13,15
Y	KR 10-2008-0065791 A (SEOUL NATIONAL UNIVERSITY INDUSTRY FOUNDATION) 15 July 2008 See abstract; paragraphs [16]-[19], [34]-[55]; claims 1-8 and figures 6-7.	1,10,14
A		2-9,11-13,15
A	US 2013-0179710 A1 (KUO-HAN CHANG et al.) 11 July 2013 See abstract; paragraphs [0008]-[0011], [0028]-[0032] and claims 1-7.	1-15
A	KR 10-2011-0085767 A (LG ELECTRONICS INC.) 27 July 2011 See abstract; paragraphs [0009]-[0010], [0070]-[0073], [0076]-[0082]; claims 1-8 and figure 3.	1-15
A	US 2012-0029900 A1 (PAPARIELLO FRANCESCO) 02 February 2012 See abstract; paragraphs [0028]-[0044], [0089]-[0093]; claims 12-20 and figures 3-4b.	1-15

☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
21 April 2015 (21.04.2015)	**22 April 2015 (22.04.2015)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea
Facsimile No. ++82 42 472 7140

Authorized officer
HONG, Kyoung Ah
Telephone No. +82-42-481-5668

Form PCT/ISA/210 (second sheet) (January 2015)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2014/048403

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2009-0307464 A1	10/12/2009	US 8711154 B2	29/04/2014
KR 10-2008-0065791 A	15/07/2008	None	
US 2013-0179710 A1	11/07/2013	CN102566739 A	11/07/2012
		CN102566739 B	26/11/2014
		TW 201329686 A	16/07/2013
		TW I443504 B	01/07/2014
		US 8977880 B2	10/03/2015
KR 10-2011-0085767 A	27/07/2011	None	
US 2012-0029900 A1	02/02/2012	None	