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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 October 2009 (01.10.2009)

PCT

(10) International Publication Number
WO 2009/120427 A1

(51) International Patent Classification:
G06F 1/26 (2006.01) *G06F 1/32* (2006.01)
G06F 9/46 (2006.01) *G06F 12/00* (2006.01)

(21) International Application Number:
PCT/US2009/034209

(22) International Filing Date:
16 February 2009 (16.02.2009)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
12/057,716 28 March 2008 (28.03.2008) US

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

(72) Inventors: **MARSHALL, Allen**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US). **DENG, Yimin**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US). **JUDGE, Nicholas S.**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US). **KISHAN, Arun U.**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US). **RITZ, Andrew J.**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US).

(74) Agent: **ALLEN,Michael B**; Microsoft Corporation, One Microsoft Way, Redmond, WA 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, 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, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, 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,

[Continued on next page]

(54) Title: POWER-AWARE THREAD SCHEDULING AND DYNAMIC USE OF PROCESSORS

(57) Abstract: Techniques and apparatuses for providing power-aware thread scheduling and dynamic use of processors are disclosed. In some aspects, a multi-core system is monitored to determine core activity. The core activity may be compared to a power policy that balances a power savings plan with a performance plan. One or more of the cores may be parked in response to the comparison to reduce power consumption by the multi-core system. In additional aspects, the power-aware scheduling may be performed during a predetermined interval to dynamically park or unpark cores. Further aspects include adjusting the power state of unparked cores in response to the comparison of the core activity and power policy.

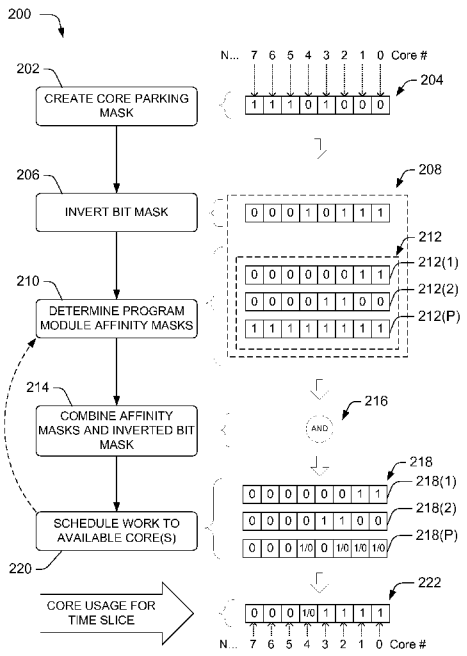

Fig. 2

WO 2009/120427 A1

ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, 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))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

POWER-AWARE THREAD SCHEDULING AND DYNAMIC USE OF PROCESSORS

BACKGROUND

[0001] Computer system management of power consumption is important to extend the operational ability of a battery and to reduce overall power consumption, which can be both fiscally and environmentally beneficial. Even for non-mobile computers, reducing power requirements is beneficial to save important global resources and prolong operation when relying on a battery backup system, such as during a utility power interruption.

[0002] Although most components of a computing system use power during system operation, the processor uses a disproportionate share of the system power. Many computer systems, including consumer based systems, include multiple processors and/or processors with multiple cores. Multiple processor enable the computers to execute increasing levels of work in parallel however additional processors may also increase power consumption. Most modern processors feature very low power idle power states, which may be applied per-core on a multi-core system, and which may be controlled by an operating system. In addition, processor frequency may be scaled on a per-core or per core group basis to reduce power usage by the system.

SUMMARY

[0003] This summary is provided to introduce simplified concepts of providing power-aware thread scheduling and dynamic use of processors, which is further described below in the Detailed Description. This summary is not intended to identify essential features of the claimed subject matter, nor is it intended for use in determining the scope of the claimed subject matter.

[0004] Exemplary techniques and apparatuses for providing power-aware thread scheduling and dynamic use of processors are disclosed. According to one or more embodiments, a multi-core system is monitored to determine core activity. A power policy is retrieved to initiate a performance and power savings plan for the cores. One or more of the cores of the multi-core system are parked (placed into a system-specified low power state) based on the power policy and core activity. When one or more cores are parked, the unparked cores are left to handle all of the remaining system activity. In some embodiments, the power policy may be modified to include additional factors influencing power savings or system performance. In at least one other embodiment, the multi-core system may dynamically adjust a power state of one or more unparked cores in addition to parking the one or more cores.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The detailed description is described with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The same reference number in different figures refers to similar or identical items.

[0006] Fig. 1 is an illustrative system that may be used to implement at least one embodiment of power-aware thread scheduling and dynamic use of processors.

[0007] Fig. 2 shows a flow diagram of at least one embodiment of a process of creating a core parking mask and implementing the mask with a thread scheduler to enable allocation of work to processors.

[0008] Figs. 3A and 3B show illustrative core utilizations in accordance with at least one embodiment of the disclosure. More specifically, Fig. 3B shows an

illustrative revision of the core utilizations in response to an illustrative system core utilization as shown in Fig. 3A.

[0009] Fig. 4 shows a flow diagram of at least one embodiment of an illustrative process of evaluating thread scheduling and dynamic use of processors and determining a revised configuration for the processors.

[0010] Fig. 5 shows a flow diagram of at least one embodiment of an illustrative process of providing power-aware thread scheduling and dynamic use of processors.

[0011] Fig. 6 shows another flow diagram of at least one embodiment of an illustrative process of providing power-aware thread scheduling and dynamic use of processors, further including domain idle accounting.

[0012] Fig. 7 shows an illustrative flow diagram of at least one embodiment of providing additional settings for power-aware thread scheduling and dynamic use of processors.

DETAILED DESCRIPTION

[0013] Processors may enable low power idle power states, including an idle state that consumes no power (zero watts). An operating system may direct one or more processors (or simply "cores") to an idle power state (i.e., processor sleep state) when there is no useful work to perform. Maximizing time spent in these low power states may increase system energy efficiency and/or extend battery performance. In addition to these processor idle power states, processors may also provide controls for scaling the processor's frequency, either alone or in conjunction with a simultaneous reduction in processor core voltage. These

controls can be collectively referred to as processor power management (PPM) features.

[0014] Processors may facilitate the execution of billions of executions per second. While having such a high capacity for executing computer instructions, processors may have considerable variance in workload over short periods of time. For example, periods as short as a delay between a typist's keystrokes may enable the PPM to reduce processor power momentarily or even enter a brief sleep state. Although a fraction of a second of power may seem negotiable, over longer periods of time the cumulative power savings may be significant.

[0015] Therefore, the PPM may reduce power demands by directing unused processors to a low power state or a sleep state ("parked" state) when the processors do not have adequate workload to justify higher power states. Parked cores may be placed in a processor idle power state (ACPI C-state) using a minimal amount of power or no power at all. The active work to be done on the system will be time multi-plexed on the unparked processors.

[0016] Accordingly, techniques and apparatuses to facilitate providing power-aware thread scheduling and dynamic use of processors are disclosed herein in the various sections that follow.

Illustrative Environment

[0017] Fig. 1 is an illustrative system 100 that may be used to implement at least one embodiment of power-aware thread scheduling and dynamic use of processors. The system 100 includes a computing device 102. For example, the computing device may be a mobile computer 102(1), a desktop computer 102(2), and/or a server 102(N), among other possible computing devices. In a very basic configuration, computing device 102 typically includes one or more processors

("processors") 104. For example, the processors 104 may be at least one of multiple independent processors configured in parallel or in series and a multi-core processing unit, either singly or in various combinations. A multi-core processor may have two or more processors ("cores") included on the same chip or integrated circuit. The terms "processor," "core," and "logical processor" may be used interchangeable throughout this disclosure unless specifically stated otherwise with reference to a particular element.

[0018] In addition, the computing device 102 includes system memory 106. Depending on the exact configuration and type of computing device, system memory 106 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory 106 typically includes an operating system 108, one or more program modules 110, and may include program data 112.

[0019] The operating system 108 may include a kernel power manager 114 that is responsible for directing the use of processor power management (PPM) features. The kernel power manager 114 may adjust the performance (e.g., speed) of the processors 104 using a performance state (p-state) or linear throttle state (t-state). For example, the kernel power manager 114 may balance a power consumption of the processors 104 with a current workload to conserve energy when possible. Additionally or alternatively, the kernel power manager 114 may enable the processors 104 to provide a maximum processing capacity in response to a workload demand. Further, the kernel power manager 114 may direct one or more of the processors 104 into a low power sleep state when no active threads are ready to run, such as threads of the program module 110.

[0020] The operating system 108 may provide modules for queuing, scheduling, prioritizing, and dispatching units of work (threads) across all available processors

104 in the system 100, which may be represented as a collection of modules collectively referred to as a kernel thread scheduler 116. When an active thread is ready to be run, the kernel scheduler 116, via one or more modules, dispatches the thread to any available core for processing.

[0021] Generally speaking, the kernel power manager 114 and kernel scheduler 116 have competing interests in control and operation of the processors 104. The kernel power manager is configured to minimize the power consumption of the processors 104, and therefore tries to reduce the frequency and/or power state of one or more of the processors 104. Conversely, the kernel scheduler 116 is configured to maximize processing throughput, and thus favors distributing work to all processors 104.

[0022] The computing device 102 may have additional features or functionality. For example, the computing device 102 may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in Fig. 1 by a removable storage 118 and a non-removable storage 120. The computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. The system memory 106, the removable storage 118 and the non-removable storage 120 are all examples of the computer storage media. Thus, the 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 desired

information and which can be accessed by the computing device 102. Any such computer storage media may be part of the computer device 102.

[0023] The computing device 102 may also have one or more input device 122 such as keyboard, mouse, pen, voice input device, touch input device, etc. One or more output device 124 such as a display, speakers, printer, etc. may also be included either directly or via a connection to the computing device 102.

[0024] The computing device 100 may also include a communication connection 126 that allows the device to communicate with other computing devices, such as over a network. The communication connection 126 is one example of communication media. The communication media may typically be embodied by computer readable instructions, data structures, or program modules. 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. 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. The computer readable media can be any available media that can be accessed by the computing device 102. By way of example, and not limitation, the computer readable media may comprise the "computer storage media" and the "communications media."

[0025] Various modules and techniques may be described herein in the general context of computer-executable instructions, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. for performing particular tasks or implement particular abstract data types. These program modules and the like may be executed as native code or may be downloaded and executed, such as in a virtual machine or other just-in-time compilation execution

environment. Typically, the functionality of the program modules may be combined or distributed as desired in various embodiments. An implementation of these modules and techniques may be stored on or transmitted across some form of the computer readable media.

Illustrative Processor Configuration

[0026] Fig. 2 shows a flow diagram of at least one embodiment of a process 200 of creating a core parking mask and implementing the mask with the thread scheduler to enable allocation of work to processors. The process 200 is illustrated as a collection of blocks in a logical flow graph, which represent a sequence of operations that can be implemented in hardware, software, or a combination thereof. In the context of software, the blocks represent computer-executable instructions that, when executed by one or more processors, perform the recited operations. Generally, computer-executable instructions include routines, programs, objects, components, data structures, and the like that perform particular functions or implement particular abstract data types. The order in which the operations are described is not intended to be construed as a limitation, and any number of the described blocks can be combined in any order and/or in parallel to implement the process. Other processes described through this disclosure, in addition to process 200, shall be interpreted accordingly. For discussion purposes, the process 200 is described with reference to the system 100 of Fig. 1.

[0027] As shown in Fig. 2, a core parking mask is created in the system memory at 202. For example, the kernel power manager 114 may create the core parking mask at 202, which resides in the operating system 108. An illustrative core parking mask ("bit mask" or simply "mask") 204 may provide a cell representing a corresponding core. As shown in Fig. 2, the illustrative system includes eight

cores, however more or fewer cores may be used. The bit mask 204 includes a bit value in each cell, where "1" represents a parked core and "0" represents an unparked core. A parked core is a core that is placed into a low power sleep state. In some embodiments, a parked core has no power consumption, thus uses zero watts. In some embodiments, cores have dependencies such as shared hardware circuitry. If both cores can be put into a low power state, dependencies can also be implicitly placed into a low power state. Thus a core parking mask can be selected which will maximize power savings compared to another mask. For example, turning off all cores in a single processor socket might save more power than turning off half the cores in two processor sockets. The bit mask 204 includes four parked cores, numbered (right to left, zero to seven): 3, 5, 6, and 7. It follows that cores 0, 1, 2, and 4 are unparked cores.

[0028] In accordance with one or more embodiments, the bit mask 204 may be inverted at 206 to create the inverted bit mask 208. For example, the kernel power manager 114 may create the inverted bit mask 208. The inverted bit mask includes an inverted bit value for each cell (i.e., core). Accordingly, cores designated with a "1" may be preferred to process data while cores designated with a "0" may not be preferred to process application threads.

[0029] At 210, an application schedule is determined, such as by the kernel scheduler 116. For example, the computing device 102 may be running one or more of the program modules 110. Some of the program modules 110 may include single threaded programs while other program modules may include multi-threaded modules. Typically, the kernel scheduler 116 schedules each thread to an available core, based on a number of factors such as priority, core availability, affinity (scheduling restrictions), and other factors. When the threads outnumber the available cores, then the kernel scheduler 116 alternates threads or otherwise schedules threads to ensure that the threads eventually make progress (i.e., are executed by the processor). The processors 104 may alternate threads on a single

core many times per second, thus providing opportunities for the kernel scheduler 116 to effectively schedule threads to available cores.

[0030] While many program modules do not assign threads to specific processors, some complex program modules may require a thread to be executed by a specific processor, referred to as setting the thread's processor affinity. Program module thread affinity masks 212 represent the cores requested for processing the threads, as determined by the program modules 110. For example, a first program module may have a first affinity mask 212(1) which indicates that threads must be scheduled by the kernel scheduler 116 on cores 0 and 1. A second affinity mask 212(2) associated with a second program module may indicate that threads may be scheduled on cores 2 and 3 while another program module may include an affinity mask 212(P) indicating that threads can be scheduled on any of the available cores (all cores are shown as selected). It should be noted that the affinity mask 212(P) is a special case in that it effectively includes no constraints for allocation of threads on the cores.

[0031] In some embodiments, at 214 the program module affinity masks 212 are combined, one at a time, with the inverted bit mask 208 using an "AND" operator 216 to determine the set of eligible processors on for an available processor set 218. At 220, the first affinity mask 212(1) is used to create a first available processor set 218(1). The process 200 may include an iterative process of operations 210, 214, and 220 for each program module (i.e., for each combination of the inverted bit mask 208 and the affinity mask 212). Thus, the second affinity mask 212(2) is used to create a second available processor set 218(2) during a second iteration of the operations 210, 214, and 220, and so forth.

[0032] As discussed above, the bit values for each core (e.g., core 0, ..., core 7) are used to determine the available processor set 218 for scheduling threads. The "AND" operator 216 returns a core bit value of "1" where both of the operands (i.e., combined bit values for a core) include a "1" representing a thread affinitized to a specific core. For example, when the first affinity mask 212(1) is combined

with the inverted bit mask 208, cores 0 and 1 both are active cores and will return a core bit value of "1" while the remaining cores 2-7 include a core bit value of "0," as illustratively shown in the first available processor set 218(1).

[0033] The second affinity mask 212(2) includes a core value of "1" at core 3, while the inverted bit mask 208 indicates core 3 is parked. The kernel scheduler 116 may choose to override the inverted bit mask to accommodate the second affinity mask 212(2), which is represented in the second available processor set 218(2) where core 3 includes a core value of "1," therefore shifting work to a core designated as parked in the inverted bit mask 208 (which may be subsequently unparked). In some embodiments, the thread may be scheduled using any number of heuristics. An optimal core in the thread's affinity may be used while ignoring the inverted bit mask 208. If the optimal core is parked, a fallback may include selecting a processor in the same NUMA (non-uniform memory access) node as the preferred core. The scheduler treats the core parking inverted mask as a hint of preferred locations to run the thread, but it will choose amongst the hard limitation (the hard affinity) of what it believes to be the most performant option.

[0034] Where all cores are designated as available, such as in the affinity mask 212(P), the bit value may be ignored because the program module is indicated that it allows the threads to be executed by any core. The affinity mask 212(P) may include scheduled cores at any of the cores indicated by the inverted bit mask 208, such as cores 0, 1, 2, and 4, as represented by the available processor set 218(P) by core values showing "1/0" (either "1" or "0", while at least one core must have a core value of "1" to enable scheduling the work of the affinity mask 212(P)). In some embodiments, the available processor set 218(P) may select cores that are unparked and idle when determining an allocation of work to available cores. Ideally, work shifting may allocate work to cores 0, 1, 2, and 3, thus leaving core 4 unused and possibly parked in a subsequent action. Other considerations, as discussed below, may determine which cores are allocated work in response to the

affinity mask 212(P) to create a preferred location. For example, a preferred location may be based on factors such as optional memory access performance.

[0035] The operations 210, 214, and 220 may be used to schedule threads, as described above with reference to Fig. 2. In addition, other work may be performed in the process 200 by the operations 210, 214, and 220 such as a deferred procedure call (DPC), timers, processing interrupts, or other processor work.

[0036] In accordance with one or more embodiments, Fig. 2 illustrates an example core usage 222 for a time slice. For example, a combination of the cores that are active from the available processor set 218 may result in the core usage 222 during a given time check interval, such as without limitations 100 milliseconds. The core 4 in the core usage 222 may or may not indicated as used depending on whether work is scheduled to core 4 in the available processor set 218(P), as discussed above. From the perspective of the kernel power manager 114, the core usage 222 will ideally include a core value of "0" for core 4, thus minimizing the number of unparked cores and resulting in a reduction of power consumption. Regardless of the core usage 222, a new core parking mask may be created for the next time slice, which may use information from the core usage 222 to determine the new core parking mask. In some instances, core 3 may not be required because core 2 may have enough processing utilization to satisfy the second affinity mask 212(2) and is still an unparked core.

[0037] Figs. 3A and 3B show illustrative core utilizations in accordance with at least one embodiment of the disclosure. More specifically, Fig. 3B shows an illustrative revision of the core utilizations in response to an illustrative system core utilization as shown in Fig. 3A.

[0038] In accordance with one or more embodiments, Fig. 3A shows a system 300 including a number of cores 302, 304, 306, and 308, although more or fewer cores may be included in alternative embodiments of the system 300. The kernel power manager 114 may control the cores using a power policy. The power policy may determine the number of active cores, and influence how the kernel power

manager 114 may calculate the set of available cores. The power policy may be used to restrict the number of available cores, or to allow the kernel power manager 114 to scale the number of available cores. The number of cores supported by a platform may vary across different systems, thus a flexible scheme is necessary to allow the number of cores to be specified by the kernel power manager without knowing in advance how many cores are available. Therefore, in some embodiments, the number of cores to be used may be expressed as a percentage of maximum core utilization. Accordingly, an implementation of the cores may require rounding up the percentage to the next available number that represents the number of cores. For example, the core usage percent may be calculated as 60 percent. If the system includes four cores, the percent may be rounded up to 75% and three cores will be unparked while one core is parked.

[0039] Each core includes a core utilization ("core utility") 310 that represents the workload of that core, expressed as the percentage of a core's run time out of the total time, independent of the core's performance state. For example, the core 0 302 may have a core utility of 80% indicating that the core is doing 20% less work than the workload maximum capacity of core "0." Accordingly, a core utility of 100% represents a core working a maximum capacity while a core having a 0% core utility represents an unused core. In some embodiments, the kernel power manager 114 may monitor the core utility 310.

[0040] In addition, each core may include a performance state (advanced configuration and power interface (ACPI) p-state) 312. The p-state 312 is a core frequency and voltage setting and is controlled by the kernel power manager 114. The p-state 312 is analogous to a throttle control of a motor. A p-state 312 of 100% represents a maximum performance state of a core while a p-state of 50% represents a core at half of the maximum frequency with a corresponding reduced core voltage level. It should be noted that the actual power consumption of a core may not coincide or be proportional with the p-state 312. For example, doubling the p-state 312 of a core may not double the power consumption of the core

because of other factors, such as core power leakage, core base power consumption, and/or other factors. In some embodiments, the kernel power manager 114 may determine and/or control the p-state 312, such as by referencing the power policy.

[0041] Each core includes an output utilization value ("output utility") 314 that represents the workload of the core in relation to the total workload capacity. For example, the output utility 314 may have a scale of 0-10,000 where 0 represents no utilization and 10,000 represents maximum utilization. The output utility 314 may be calculated by multiplying the core utility 310 and the p-state 312. For example, the core 0 302 includes a core utility of 80% and a p-state of 80%, therefore the output utility is 6,400. In some embodiments, the output utility 314 is used by the kernel power manager 114 to determine core parking decisions and/or determine p-state 312 settings, such as with reference to the power policy.

[0042] In some embodiments, the system 300 may include a core block 316, such as a first core block 316(1) and a second core block 316(2), however more or fewer core blocks may be implemented in the system 300. The core block 316 may represent a platform having multiple cores with a single circuit, such as a dual-core or multi-core processor. Each core block 316 may include unique power consumption characteristics. For example, a core may include active level power consumption, core leakage, or other power dissipations which occur when either of the cores in the block is unparked. For example, if both cores in the first core block 316(1) are unparked and have the output utility of 10,000, the combined power consumption may be $2x$ watts. If the core 0 302 in the first core block 316(1) is subsequently parked (e.g., output utilization is 0) and the core 1 304 remains unchanged, the combined power consumption may be greater than x watts because of factors associated with the core block 316 such as power leakage, active power consumption, and/or other factors. When the core 1 is subsequently parked, the resultant power consumption may be 0 watts. Therefore, it may be advantageous to

park cores such that entire core blocks become parked before other cores are subsequently parked, thus maximizing power savings.

[0043] As shown in Fig. 3A, an illustrative implementation of the system 300 includes the core 3 308 being parked while the other cores are unparked (i.e., active). Although core 3 includes a p-state of 100%, this may not indicate that power is supplied to the core 3. Stated another way, the kernel power manager 114 may park a core while leaving a p-state at a level greater than 0%.

[0044] In an example utilization scenario, the kernel power manager 114 may calculate the output of system 300 to determine a total system utility of 12,600 (i.e., 6,400 + 3,200 + 3,000 = 12,600) of a maximum total system utility of 30,000 (i.e., 3 unparked cores x 10,000 = 30,000). The utilization numbers referenced above are intended to be explanatory in nature of calculations that may be performed using the total system utility, and thus are not limiting to the disclosure.

[0045] Fig. 3B shows an illustrative revision of the core utilizations in a revised system 318 in response to the total system utilization as shown in Fig. 3A. The revise system 318 includes a revision in the parked/unparked status of the cores 302, 304, 306, and 308. As noted above, with reference to the system 300, the total system utility was calculated to be 12,600 in an example. Therefore, the kernel power manager 114 may park an additional core without reducing the ability of the system to meet the current workload demands because the total system utility is less than the maximum utilization capacity of two cores (i.e., 12,600 < 20,000). In accordance with some embodiments, the kernel power manager 114 may select a core to park, perhaps based on the power policy or other factors, which may be completely transparent to a user of the system. The user transparency includes no user perceivable system or application performance impact except for its principle goal of better power efficiency. Any changes required to implement core parking may be constrained to very low level operation system internal components, and accordingly there may be no behavioral or experiential change for end users associated with core parking.

[0046] When a second core is parked, the maximum total system utility of the revised system will drop to 20,000 (2 cores x 10,000). The kernel power manager 114 may park any of the cores which were active in the previous state (as shown in Fig. 3A). As discussed above, it may be advantageous to park core 2 306 to completely park the second core block 316(2), resulting in a parked core block 320. Accordingly, the parked core block 320 may increase power saving as compared to parking the core 0 302 or the core 1 304 instead of the core 2 306.

[0047] In order to accommodate a transparent change to any users of the system 318, the output utility of the parked core (core 2) must be absorbed or reallocated to the remaining unparked cores (i.e., core 0 and core 1). From Fig. 3A, the output utility of core 2 was 3,000. Therefore, in one instance, the unparked cores may equally share the burden by having each core have a modified output utility 322 that assumes an additional output utility of 1,500 for each core. In some instances, other divisions of a parked core's total utility may be used when allocating the utility to unparked cores. For example, an unparked core may be running near capacity (total utility near 10,000, which is the maximum utility). In such an instance, cores with greater bandwidth may absorb more of the parked core's total utility.

[0048] In one or more embodiments, the kernel power manager 114 may adjust the core utility 310 of the core 0 302 and the core 1 304 to 79% and 47%, respectively. In addition, the kernel power manager 114 may increase the p-state 312 to 100% for both of the unparked cores. Therefore, the total system utility of the revised system 318 remains equal to the total system utility of the system 300 at 12,600.

[0049] The revised system 318 illustrates one possible revision of the parked/unparked status, the core utility 310, and/or the p-state 312 to accommodate a core revision that is transparent to users while resulting in reduced power consumption for the revised system 318. However, many other revisions may be made in system 318 which result in reduced power consumption and that are

transparent to users. For example, the core utility 310 of core 1 304 may be increase to 94% while the p-state is reduced to 50%, resulting in a total utility of 4700. The revision strategy may be determined by the power policy which may take into account the competing interests of the kernel power manager 114 and the kernel scheduler 116.

[0050] Other considerations may be implemented in the power policy that may affect the revised system 318. For example, threads that are not time sensitive (e.g., background threads) may have less impact on the core parking decision while keeping changes transparent to the users. More specifically, by including the distribution of average runtimes across thread priorities executed on a given core, the kernel power manager 114 may scale a core's calculated utilization such that low priority threads and workloads do not count as much as high priority threads and workloads.

[0051] Fig. 4 shows a flow diagram of at least one embodiment of an illustrative process 400 of evaluating thread scheduling and dynamic use of processors and determining a revised configuration for the processors. The process 400 may be implemented by the kernel power manager 114 in addition to other modules in the operating system 108 and/or residing on the system memory 106.

[0052] In accordance with one or more embodiments, the kernel power manager 114 evaluates a time period for monitoring the output utility 314 (among other factors including the core utility 310, the p-state 312, etc.). The time period may be selected that is equal to, or longer than, a time slice available for processing each thread. At a given frequency, the kernel power manager 114 may initiate the monitoring of the cores.

[0053] At 404, the kernel power manager 114 may calculate the total system utilization and the maximum total system utilization. The kernel power manager 114 may review the power policy at 406 to determine how to adjust the core usage

to balance the power savings needs of the kernel power manager 114 and the core availability (processing performance) needs of the kernel scheduler 116. At 408, other factors may be used to determine whether to adjust the core usage, and if so, how to adjust the usage to achieve system goals such as a transparent change for the users, accommodate thermo requirements, and/or accommodate other constraints.

[0054] At 410, the kernel power manager 114 may calculate new configurations for the cores. For example, the kernel power manager may calculate a new bit mask 202 as shown in Fig. 2. In one or more embodiments, the bit mask may be adjusted to create the available processor set 218 at 410. At 412, the kernel power manager 114 may implement the configurations from 410. In some embodiments, the process 400 may repeat to create a dynamic core allocation, such as by repeating at a predetermined frequency. Alternatively, the process 400 may create a static core allocation over a given period of time.

Illustrative Operation

[0055] Fig. 5 shows a flow diagram of at least one embodiment of an illustrative process 500 of providing power-aware thread scheduling and dynamic use of processors. The process 500 shall be construed similarly as the process 200 of Fig. 2 regarding ordering and implementation of the process. For example, the order in which the operations in the process 500 are described is not intended to be construed as a limitation, and any number of the described blocks can be combined in any order and/or in parallel to implement the process. For discussion purposes, the process 500 is described with reference to the system 100 of Fig. 1.

[0056] In accordance with one or more embodiments, a "TimeCheck" periodic evaluation routine may begin at 502. For example, a deferred procedure call (DPC) may begin at 502. In some embodiments, a state machine is entered on each core via the DPC running at a fixed periodic rate configured by a power policy parameter "TimeCheck" for a time value, such as 100 ms, 50 ms, or another time value. At 504, the kernel power manager 114 may gather metrics for the cores. For example, the DPC is queued to each currently active core to snap metrics for the active cores. The metrics may include core utilization, thread priority distribution, an average wait time for ready threads for each core, and/or success and failure metrics for idle state residency, among other possible metrics.

[0057] At 506, the kernel power manager 114 may calculate a bit mask, such as the core parking mask 204. For example, a new value may be calculated for the target number of active cores based on utilization thresholds, the power policy, and/or any dependency relationships. At 508, an active set may be updated, such as by implementing portions of the process 200 governed by the kernel power manager 114 to create the available processor set 218.

[0058] At 510, the active set may be implemented by the operating system 108. The kernel power manager 114 may determine if cores have been added (unparked) at 512. If cores are unparked, the kernel scheduler 116 may be notified at 514 and may begin using the unparked cores to schedule threads. These unparked cores can either be the target of remote thread scheduling (i.e., from a different processor), or can proactively choose to select threads from other processors. In some embodiments, threads may be reassigned to run on the unparked cores, thereby reducing workload from other unparked cores.

[0059] At 516, the expected core output utilization may be calculated by the kernel power manager 114 and may include unparked cores from 506. Thus, a new

value for the number of active cores determined at 506 is used to calculate expected processor utilization. At 518, the kernel power manager 114 may calculate a new value for the p-state. In some embodiments, a DPC is scheduled on each core to update its target p-state. In an example, if the expected utilization increases, the p-state value may also increase if no cores are unparked at 512. However, if cores are unparked at 512, the p-state may increase or decrease to balance power savings needs of the kernel power manager 114 with processing needs of the kernel scheduler 116.

[0060] At 520, the kernel power manager 114 determines if the p-state and/or t-state (linear throttle state) current values should be revised based on the results from the calculation at 518. If the p-state and/or t-state are modified at 520, the kernel power manager 114 may queue transition DPC's to the affected cores at 522. Thus, the threads scheduled by the kernel scheduler 116 may be scheduled to cores running at the new p-state and/or t-state as implemented at 520.

[0061] At 524, the kernel power manager 114 determines if cores have been parked in 506. For example, the expected core output utilization from 516 may be less than the current utilization. If cores have been parked at 524, the kernel power manager 114 may notify the kernel scheduler 116 at 526 to terminate scheduling of threads to the newly parked cores. For any cores added or removed to the active core mask, a DPC is scheduled for that core. As described with reference to Figs. 3A and 3B, the work from a parked core may be reallocated to one or more unparked cores. In some embodiments, newly parked cores may be placed into the deepest c-state that is available. Finally, at 528 the process 500 may be repeated.

[0062] Fig. 6 shows another flow diagram of at least one embodiment of an illustrative process 600 of providing power-aware thread scheduling and dynamic use of processors, further including domain idle accounting. The process 600

includes many of the sub-processes as described in Fig. 5, and therefore those sub-processes from Fig. 5 will not be described again.

[0063] At 602, the kernel power manager 114 may determine whether domain idle accounting is enabled. If domain idle accounting is enabled, a domain master snaps metrics for the domain (e.g., the core block 316 or all cores) at 604.

[0064] In some embodiments, the domain idle accounting may initiate another decision at 606. If domain idle accounting is enabled, the kernel power manager 114 may calculate the domain target p-state at 608. Finally, at 610 the process 600 may be repeated.

[0065] Fig. 7 shows a flow diagram of at least one embodiment of a process 700 of providing additional power policy settings and other inputs for power-aware thread scheduling and dynamic use of processors. In some embodiments, core parking may be implemented as an enhancement to the existing state machine that calculates the target state to be used for processor performance states. The correct number of cores to be used at any given time will be determined based a number of factors as described below with reference to the process 700.

[0066] In accordance with one or more embodiments, the current power policy may be used to set the number of cores to be utilized at 702. At 704, the minimum or maximum number of cores may be set. The kernel power manager 114 may calculate the number of active cores required to complete a given workload in an energy efficient manner. In some instances, running the minimum number of cores may be beneficial to the power savings and benefit the kernel power manager 114. Conversely, running the maximum number of cores may provide the highest level of performance, thus benefiting the kernel scheduler 116.

[0067] At 706, additional power policy parameters for core parking may be implemented by the kernel power manager 114, using one or more of sub-processes 706(1), ..., 706(4). At 706(1), the required time interval for parking and/or unparking cores may be adjusted. For example, the frequency of parking and/or unparking cores may be manipulated by changing an interval. In some embodiments, cores may be parked at a first interval and unparked at a second interval. For example, a policy favoring power savings may park cores as frequently as every 100 ms, but may only unpark cores every 500 ms.

[0068] At 706(2), the kernel power manager 114 may implement an increase and/or decrease policy. For example, a first policy option may only park a set number of cores at a time, such as one core at a time. A second policy option may park or unpark cores to achieve an ideal core utilization, thus parking and/or unparking multiple cores at a time. A third policy may go to one extreme or the other (either park as many as possible or unpark as many as possible).

[0069] At 706(3), the required utilization threshold may be increased or decreased based on the busyness of the processors. For example, a processor may not undergo a change in a status of parked or unparked until the processor (or other processors) include a state of busyness for a given period of time. This may reduce a processor from flipping between a parked and unparked state in a rapid succession.

[0070] Finally, at 706(4), the policy for scaling unparked cores may be implemented. For example, the kernel power manager 114 may calculate the ideal target processor performance state based on the number of processor cores in the currently active set. To provide the best tradeoffs between power savings, performance, and responsiveness to specific workloads, the kernel power manager 114 may advantageously run a smaller number of processors in a higher

performance state, or conversely, run a larger number of cores in a lower performance state.

[0071] At 708, core and or system heuristics may be implemented by the kernel power manager 114, using one or more of sub-processes 708(1) and 708(2). At 708(1), the kernel power manager 114 may calculate the number of active cores required and the optimal performance state of the cores in active use based on the successful use of deep processor idle power states (sleep states). This may allow the kernel power manager to detect when the deeper idle states are not being efficiently used across the set of active (unparked) cores. To conserve power and still provide performance, it may be beneficial to place more cores into the parked state, and increase the performance state of the remaining active cores to ensure work gets executed efficiently.

[0072] At 708(2), an average wait time may be used for threads ready to be allocated (assigned) to a core. For example, the kernel power manager 114 may calculate the number of processor cores required by using the distribution of average wait times for threads in the ready state, which allows the kernel power manager to scale the number of cores in use to reduce the latency before threads in the ready state are able to run, thus increasing performance and responsiveness. In an example, when a large number of threads need to be run, it may be advantageous to unpark cores while reducing the p-state of cores because each core can only process one thread at a time. Thus, more core availability will enable processing the large number of threads in some instances.

[0073] At 710, idle state dependencies may be used by the kernel power manager 114 to adjust a core parking implementation. When the kernel power manager 114 selects which specific cores should be parked or unparked, it will first examine the idle state dependency relationships of cores to determine which cores might share

power or clock resources, and choose to park or unpark cores in the most power efficient manner based on shared controls.

[0074] At 712, performance and throttle state relationships may be considered when determining whether to park or unpark one or more cores. Cores that share performance state or throttling controls may be parked or unparked together to realize greater power efficiencies.

[0075] At 714, the core package (block) relationships may be considered, such as dependencies described in Figs. 3A and 3B regarding the efficiencies of a core block, and more specifically parking a core block before parking another core in a new core block. Finally, at 716, memory locality may be used by the kernel power manager 114 when implementing core parking considerations. For example, two or more cores may have package relationships such as the cores sharing a physical processor package having a shared memory bank (e.g., NUMA (non-uniform memory access) node). The shared memory bank may enable the cores to have reduced memory access time as compared to cores that do not share the shared memory bank.

Conclusion

[0076] The above-described techniques, systems and apparatuses pertain to providing power-aware thread scheduling and dynamic use of processors. Although the techniques, systems and apparatuses have been described in language specific to structural features and/or methodological acts, it is to be understood that the appended claims are not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing such techniques and apparatuses.

CLAIMS

What is claimed is:

1. A method for balancing performance and power savings of a computing device having multiple cores, comprising:

defining which cores (202, 204) are used to process work;

determining a power policy (210, 214, 406) to initiate a performance and power savings plan for both the defined and undefined cores; and

parking at least one of the undefined cores (222) based on the power policy.

2. The method of claim 1, wherein the determining a power policy further includes:

creating a core parking mask (202);

determining thread processor affinities (210);

providing at least a portion of the performance and power savings plan for the cores by combining the core parking mask and the thread processor affinity masks (214) to create an available processor set (220); and

calculating which cores are parked or unparked based on the available processer set and the cores that are actively processing work (222).

3. The method of claim 2, wherein the determining a power policy occurs dynamically as an iterative process (210, 214, 220).

4. The method of claim 1, further comprising scaling at least one of the defined cores based on the power policy (512).

5. The method of claim 4, wherein the scaling at least one of the defined cores includes adjusting at least one of the core utility or the power state (p-state) of a core to increase power savings (512).

6. The method of claim 1, wherein the parking at least one of the undefined cores includes:

determining if an unparked core block includes a parked core (316(2)); and

if the unparked core block having the parked core is determined, parking at least one unparked core in the unparked core block (320).

7. The method of claim 1, further comprising modifying the power policy using at least one of:

core and system heuristics (708);

processor dependency relationship (710, 714, 718); and

core policy parameters for core parking (706).

8. A method, comprising:

monitoring core activity in a multi-core system (402);

retrieving a power policy for the multi-core system (406), the power policy balancing power savings and processing performance of each core in the multi-core system; and

parking at least one core in response to the core activity based on the power policy (412).

9. The method of claim 8, wherein the parking a core includes reducing the core power state to minimal power processor idle power state (C-state) (524).

10. The method of claim 8, further comprising adjusting the power state of at least one unparked core in the multi-core system (520).

11. The method of claim 8, wherein the parking at least one core in response to the core activity is dynamically initiated at a predetermined frequency (502, 524).

12. The method of claim 8, wherein the monitoring core activity in a multi-core system occurs for a predetermined time period (402), and wherein the parking at least one core in response to the core activity occurs dynamically in response a power savings opportunity detected by the monitored core activity (210, 214, 220).

13. The method of claim 8, wherein the parking at least one core in response to the core activity includes work shifting to an unparked core, as allowed by an affinity mask associated with a program schedule (712, 714).

14. The method of claim 8, further comprising

revising the monitored core activity to create a revised core activity (508);

unparking at least one core in response to the revised core activity based on the power policy (512); and

reassigning at least one thread to the at least one unparked core (514).

15. A multiple logical processor system, comprising:

a plurality of processors (104); and

a controller (106) coupled to the plurality of processors, the controller to:

implement a performance schedule;

implement a power savings policy; and

balance the performance schedule and power savings policy by parking one or more of the plurality of processors.

16. The system of claim 15, wherein the controller receives instructions from a kernel power manager (114) residing in system memory to implement the power savings policy.

17. The system of claim 15, wherein the power savings policy (702) includes a core parking mask (202), the core parking mask combined with program

module schedules to determine a resulting set of processors on which to schedule work (116, 220).

18. The system of claim 15, wherein the plurality of processors include at least one core block having multiple cores (300).

19. The system of claim 18, wherein balancing the performance schedule and power savings policy includes creating a core parking prioritization for an unparked core in the core block having at least one parked core (318, 714).

20. The system of claim 15, further comprising adjusting the power state of at least one unparked core (520).

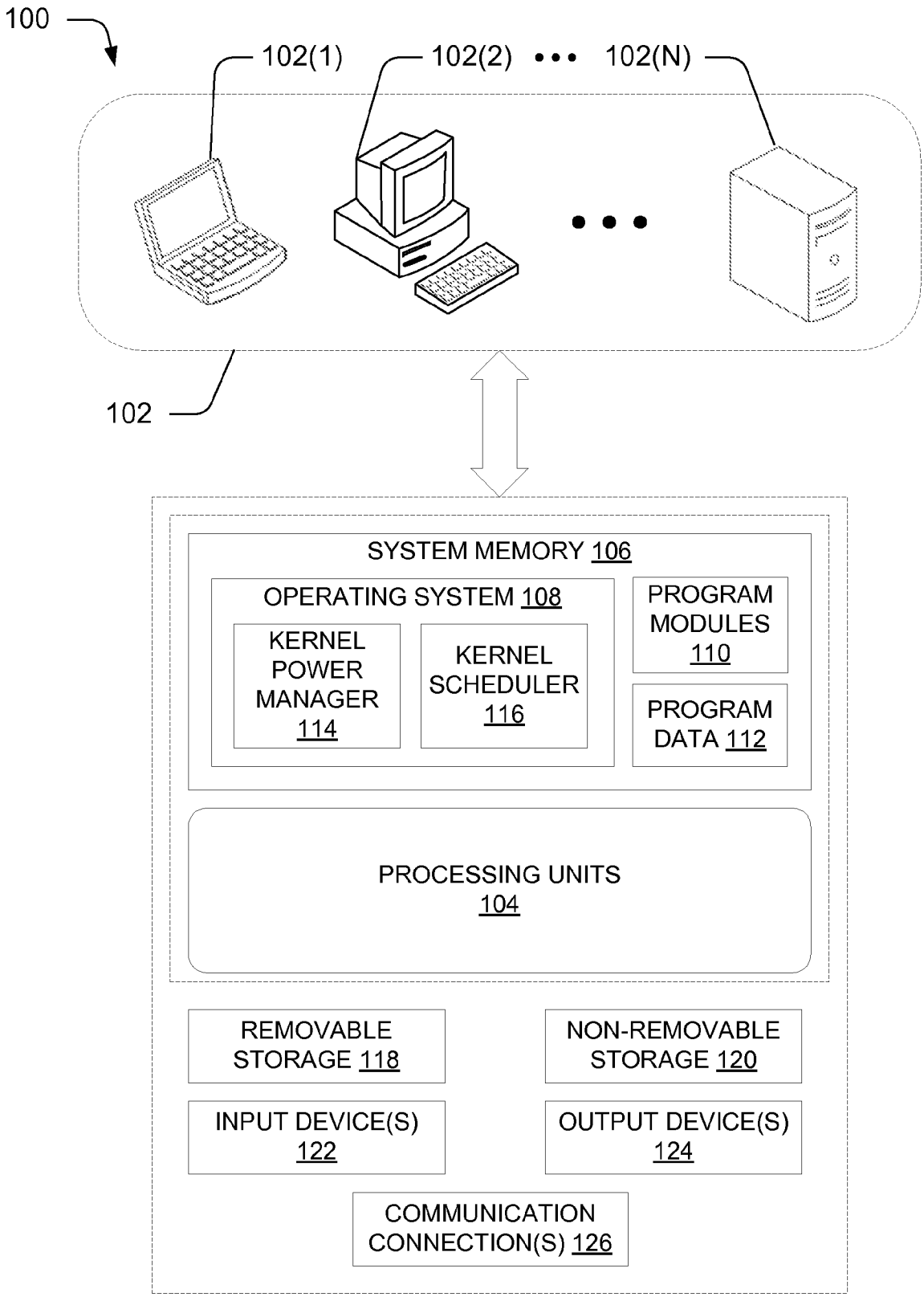
100
102(1)
102(2)
102(N)
102
SYSTEM MEMORY 106
OPERATING SYSTEM 108
KERNEL POWER MANAGER 114
KERNEL SCHEDULER 116
PROGRAM MODULES 110
PROGRAM DATA 112
PROCESSING UNITS 104
REMOVABLE STORAGE 118
NON-REMOVABLE STORAGE 120
INPUT DEVICE(S) 122
OUTPUT DEVICE(S) 124
COMMUNICATION CONNECTION(S) 126

Fig. 1

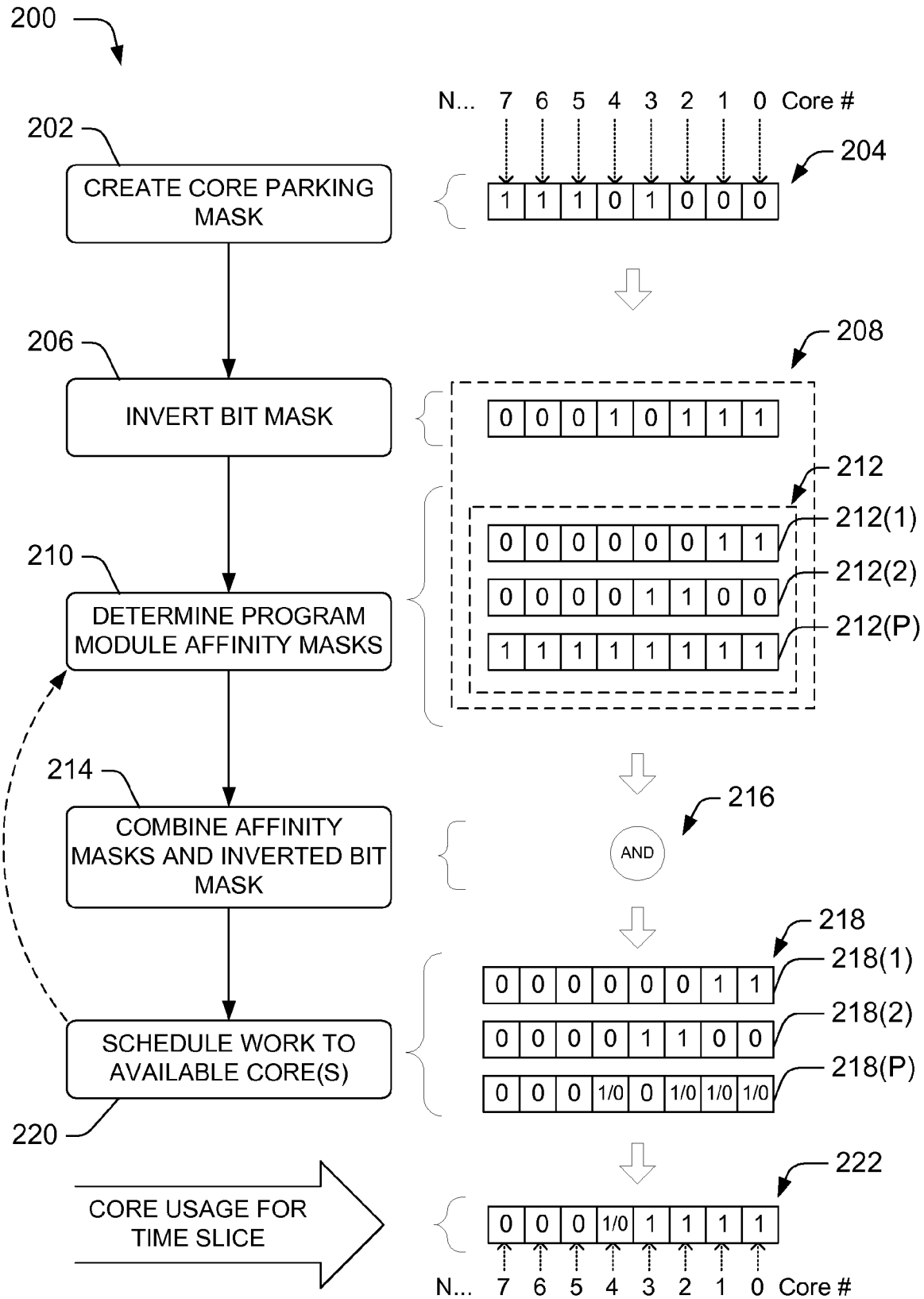
200
202
CREATE CORE PARKING MASK
206
INVERT BIT MASK
210
DETERMINE PROGRAM MODULE AFFINITY MASKS
214
COMBINE AFFINITY MASKS AND INVERTED BIT MASK
SCHEDULE WORK TO AVAILABLE CORE(S)
220
CORE USAGE FOR TIME SLICE
N... 7 6 5 4 3 2 1 0 Core #
204
1 1 1 0 1 0 0 0
208
0 0 0 1 0 1 1 1
212
212(1)
0 0 0 0 0 0 1 1
212(2)
0 0 0 0 1 1 0 0
212(P)
1 1 1 1 1 1 1 1
216
AND
218
218(1)
0 0 0 0 0 0 1 1
218(2)
0 0 0 0 1 1 0 0
218(P)
0 0 0 1/0 0 1/0 1/0 1/0
222
0 0 0 1/0 1 1 1 1
N... 7 6 5 4 3 2 1 0 Core #

Fig. 2

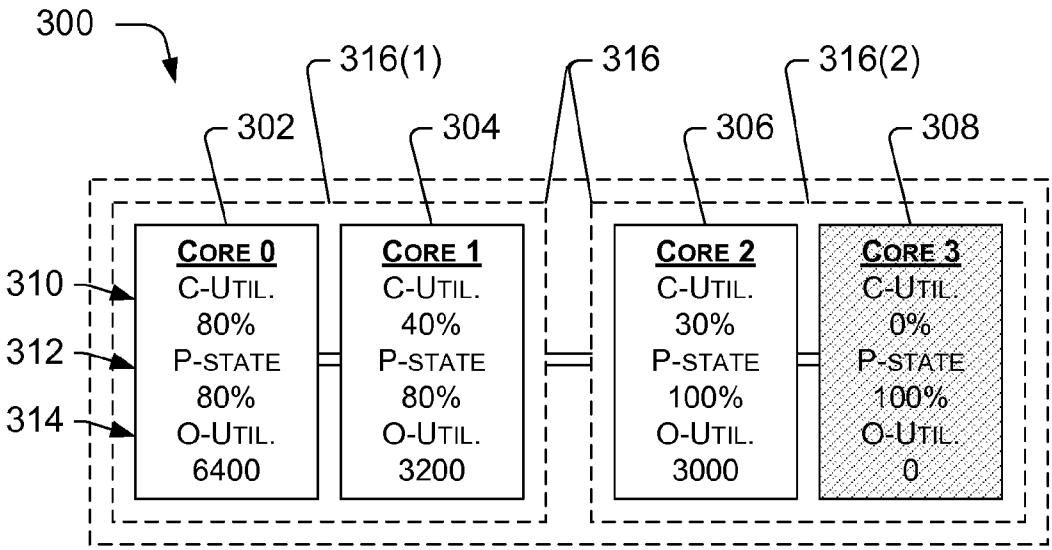
300
316(1)
316
316(2)
302
304
306
308
310
312
314
CORE 0
C-UTIL.
80%
P-STATE
80%
O-UTIL.
6400
CORE 1
C-UTIL.
40%
P-STATE
80%
O-UTIL.
3200
CORE 2
C-UTIL.
30%
P-STATE
100%
O-UTIL.
3000
CORE 3
C-UTIL.
0%
P-STATE
100%
O-UTIL.
0

Fig. 3A

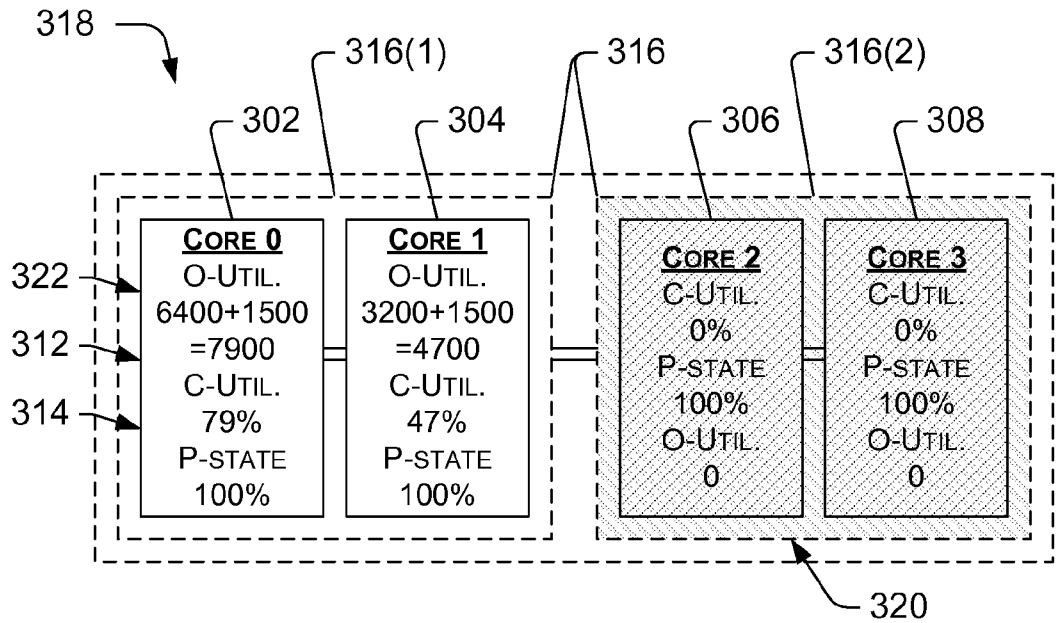
318
316(1)
316
316(2)
302
304
306
308
322
312
314
CORE 0
O-UTIL.
6400+1500
=7900
C-UTIL.
79%
P-STATE
100%
CORE 1
O-UTIL.
3200+1500
=4700
C-UTIL.
47%
P-STATE
100%
CORE 2
C-UTIL.
0%
P-STATE
100%
O-UTIL.
0
CORE 3
C-UTIL.
0%
P-STATE
100%
O-UTIL.
0
320

Fig. 3B

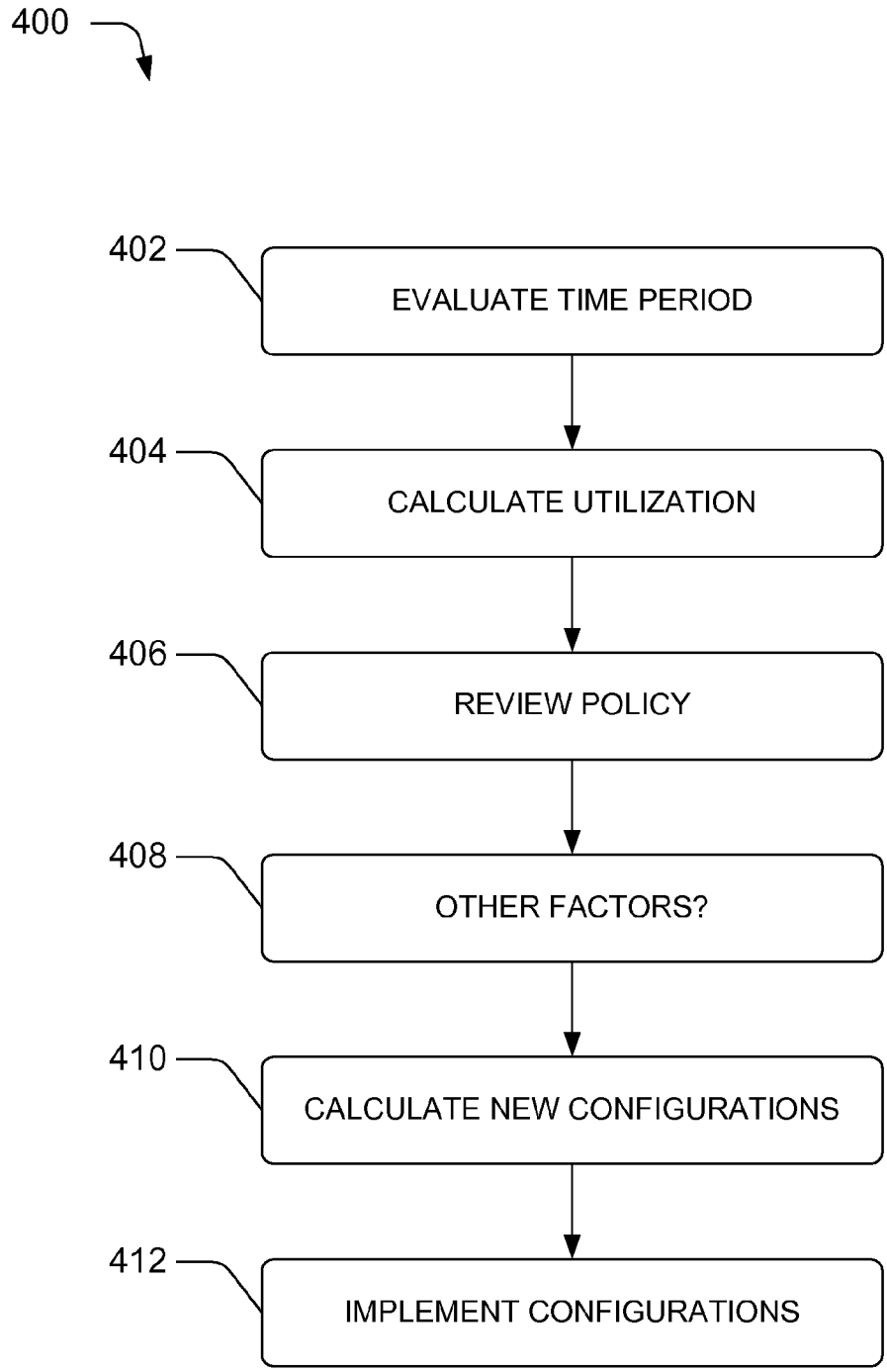
400
402
EVALUATE TIME PERIOD
404
CALCULATE UTILIZATION
406
REVIEW POLICY
408
OTHER FACTORS?
410
CALCULATE NEW CONFIGURATIONS
412
IMPLEMENT CONFIGURATIONS

Fig. 4

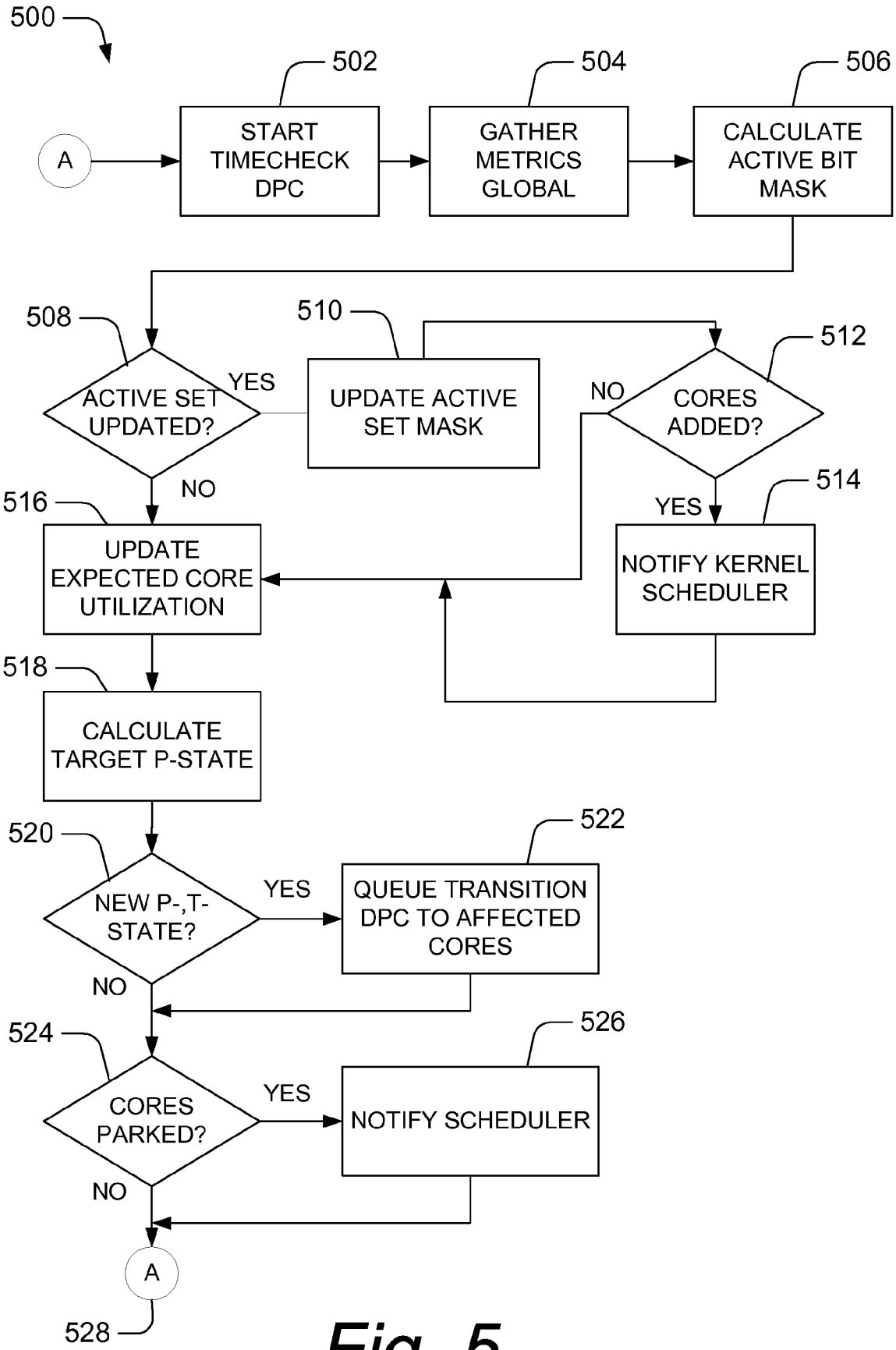
500
502
504
506
A
START TIMECHECK DPC
GATHER METRICS GLOBAL
CALCULATE ACTIVE BIT MASK
508
510
512
ACTIVE SET UPDATED?
YES
UPDATE ACTIVE SET MASK
NO
CORES ADDED?
NO
YES
514
516
UPDATE EXPECTED CORE UTILIZATION
NOTIFY KERNEL SCHEDULER
518
CALCULATE TARGET P-STATE
520
522
NEW P-,T-STATE?
YES
QUEUE TRANSITION DPC TO AFFECTED CORES
NO
524
526
CORES PARKED?
YES
NOTIFY SCHEDULER
NO
A
528

Fig. 5

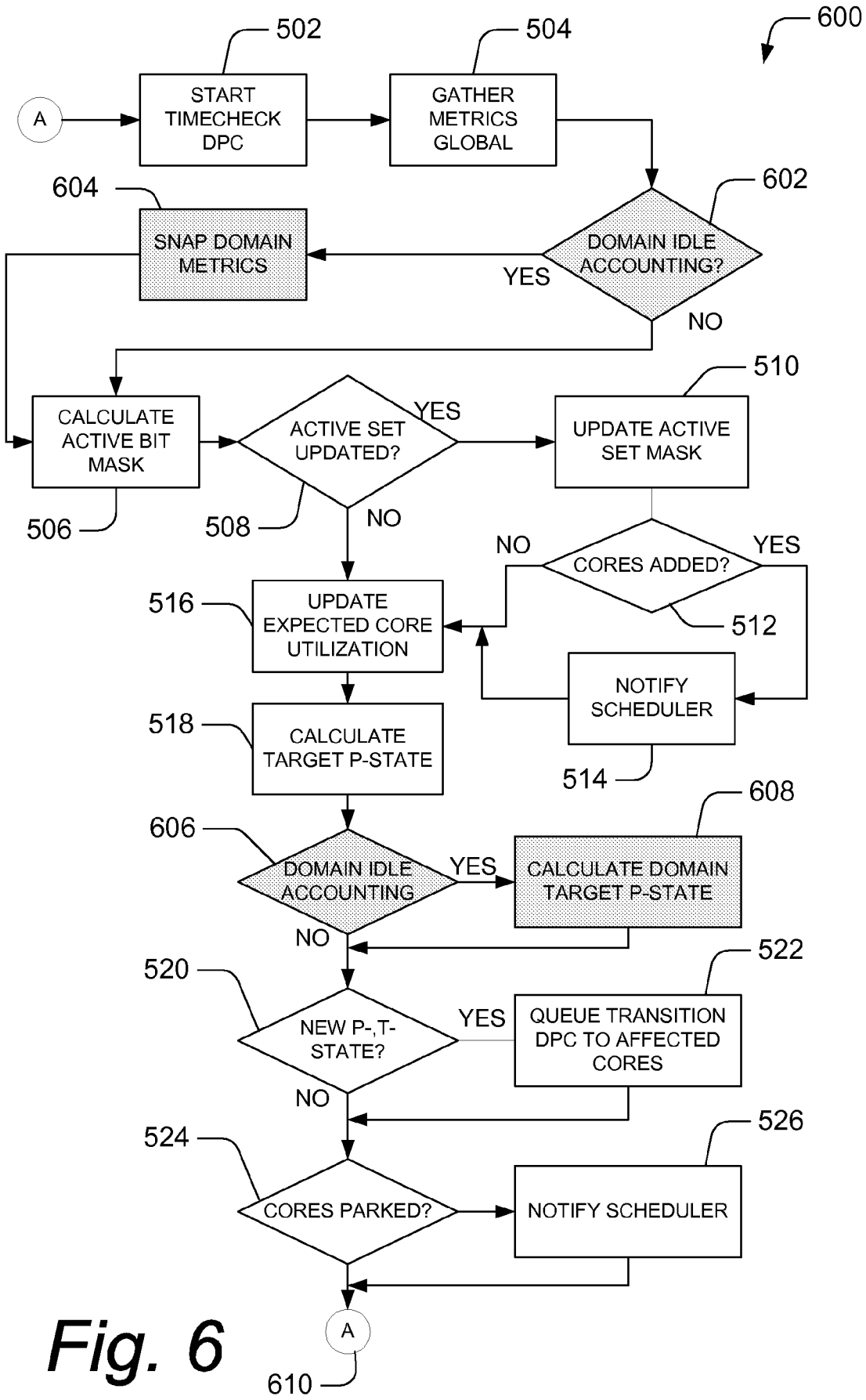
600
502
504
A
START TIMECHECK DPC
GATHER METRICS GLOBAL
604
602
SNAP DOMAIN METRICS
DOMAIN IDLE ACCOUNTING?
YES
NO
510
CALCULATE ACTIVE BIT MASK
ACTIVE SET UPDATED?
YES
UPDATE ACTIVE SET MASK
506
508
NO
NO
CORES ADDED?
YES
516
UPDATE EXPECTED CORE UTILIZATION
512
NOTIFY SCHEDULER
518
CALCULATE TARGET P-STATE
514
608
606
DOMAIN IDLE ACCOUNTING
YES
CALCULATE DOMAIN TARGET P-STATE
NO
522
520
NEW P-,T-STATE?
YES
QUEUE TRANSITION DPC TO AFFECTED CORES
NO
526
524
CORES PARKED?
NOTIFY SCHEDULER
A
610

Fig. 6

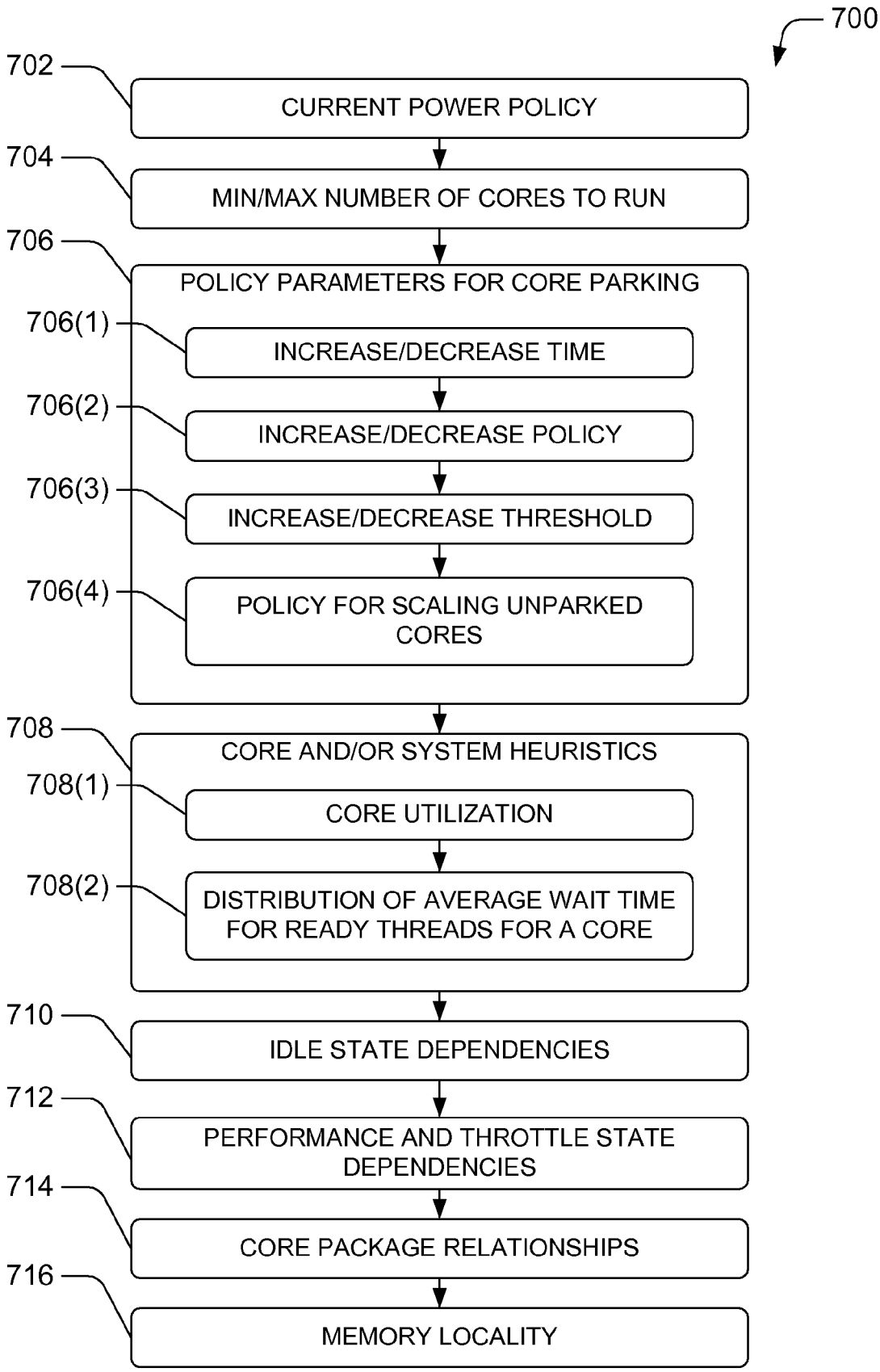
700
702
CURRENT POWER POLICY
704
MIN/MAX NUMBER OF CORES TO RUN
706
POLICY PARAMETERS FOR CORE PARKING
706(1)
INCREASE/DECREASE TIME
706(2)
INCREASE/DECREASE POLICY
706(3)
INCREASE/DECREASE THRESHOLD
706(4)
POLICY FOR SCALING UNPARKED CORES
708
CORE AND/OR SYSTEM HEURISTICS
708(1)
CORE UTILIZATION
708(2)
DISTRIBUTION OF AVERAGE WAIT TIME FOR READY THREADS FOR A CORE
710
IDLE STATE DEPENDENCIES
712
PERFORMANCE AND THROTTLE STATE DEPENDENCIES
714
CORE PACKAGE RELATIONSHIPS
716
MEMORY LOCALITY

Fig. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2009/034209

A. CLASSIFICATION OF SUBJECT MATTER

G06F 1/26(2006.01)i, G06F 9/46(2006.01)i, G06F 1/32(2006.01)i, G06F 12/00(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)

IPC 8 : G06F

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 since 1975

Japanese Utility models and application for Utility models since 1975

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) "multi-core processor" "scheduling"

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2006/0090161 A1 (DEVADATTA V. BODAS, JUN NAKAJIMA) 27 April 2006 See [0015] - [0030].	1-20
A	US 2006/0123251 A1 (JUN NAKAJIMA, DEVADATTA V. BODAS) 9 June 2006 See [0011] - [0027].	1-20
A	US 2007/0204268 A1 (ULRICH DREPPER) 30 August 2007 See [0024] - [0061].	1-20
A	US 2006/0004988 A1 (PAUL J. JORDAN) 5 January 2006 See [0021] - [0072].	1-20

☐ 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 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
31 AUGUST 2009 (31.08.2009)	**01 SEPTEMBER 2009 (01.09.2009)**

Name and mailing address of the ISA/KR

Korean Intellectual Property Office
Government Complex-Daejeon, 139 Seonsa-ro, Seo-gu, Daejeon 302-701, Republic of Korea

Facsimile No. 82-42-472-7140

Authorized officer

KWON, Oh Seong

Telephone No. 82-42-481-8526

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2009/034209

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2006-0090161 A1	27.04.2006	None	
US 2006-0123251 A1	08.06.2006	None	
US 2007-0204268 A1	30.08.2007	None	
US 2006-0004988 A1	05.01.2006	EP 1766511 A1 WO 2006-004827 A1	28.03.2007 12.01.2006