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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 June 2009 (18.06.2009)

PCT

(10) International Publication Number
WO 2009/076001 A2

(51) International Patent Classification:
G06F 9/46 (2006.01) *G06F 12/00* (2006.01)
G06F 9/30 (2006.01)

(21) International Application Number:
PCT/US2008/083530

(22) International Filing Date:
14 November 2008 (14.11.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/952,828 7 December 2007 (07.12.2007) US

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

(72) Inventors: **VAN DER HOEVEN, Arie**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US). **WALKER, Ellsworth D.**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US). **FOLTZ, Forrest C.**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US). **DENG, Zhong**; c/o Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, 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, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, 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))*

[Continued on next page]

(54) Title: KERNEL PROCESSOR GROUPING

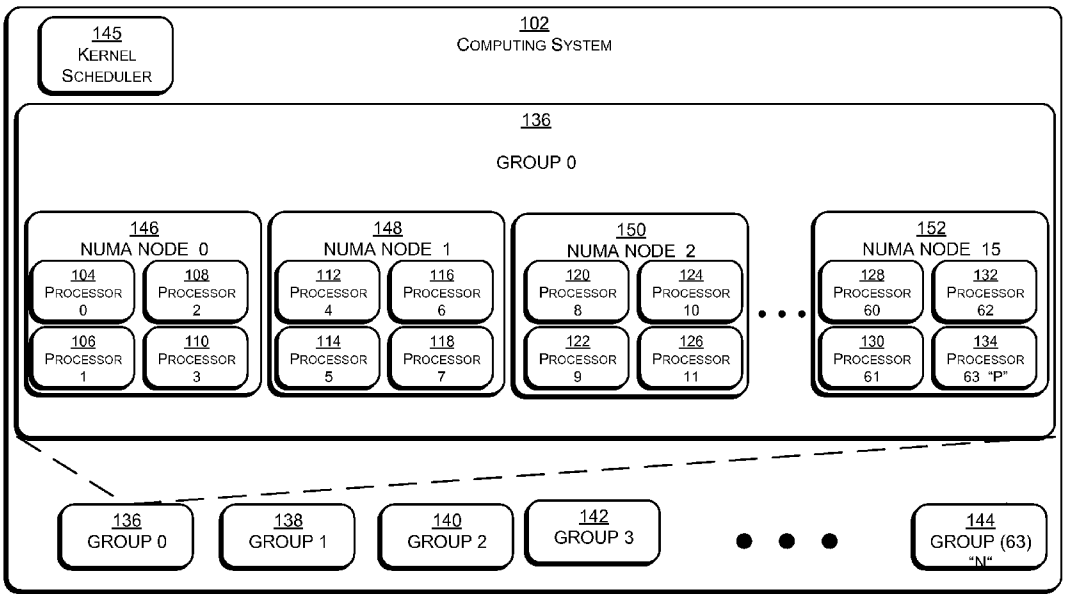

Fig. 1

(57) Abstract: Techniques for grouping individual processors into assignment entities are discussed. Statically grouping processors may permit threads to be assigned on a group basis. In this manner, the burden of scheduling threads for processing may be minimized, while the processor within the assignment entity may be selected based on the physical locality of the individual processors within the group. The groupings may permit a system to scale to meet the processing demands of various applications.

WO 2009/076001 A2

Published:

— *without international search report and to be republished upon receipt of that report*

KERNEL PROCESSOR GROUPING

BACKGROUND

[0001] As the number of processors in systems increase, the overall productivity of the processors may not match the system's expected processing capacity when processing applications which were designed to be processed on a system having less processors. For instance, bottlenecks may occur as individual threads for processing are distributed to the various processors. In other instances, some applications may limit the number of processors which may effectively process tasks for the application. For example, some applications may not be well suited for being processed by more processors that for which the application was designed. For example, while an application may operate as expected on a desktop system including two processors, an enterprise server having sixty-four or more processors may experience issues with the same application.

SUMMARY

[0002] Techniques for grouping individual processors into assignment entities are discussed. Statically grouping processors may permit threads to be assigned on a group basis. In this manner, the burden of scheduling threads for processing may be minimized, while the processor within the assignment entity may be selected based on the physical locality of the individual processors within the group. The groupings may permit a system to scale to meet the processing demands of various applications.

[0003] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] 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 use of the same reference numbers in different instances in the description and the figures may indicate similar or identical items.

[0005] FIG. 1 illustrates an environment in exemplary implementations that may use kernel level group processing.

[0006] FIG. 2 is a flow diagram depicting a procedure in exemplary implementations in which processor assignment entity static grouping is used.

DETAILED DESCRIPTION

[0007] **Overview**

Accordingly, techniques are described which may provide kernel level processor grouping. For instance, individual processors may be statically configured into kernel level groupings based on the locality of the individual processors so that threads or discrete application tasks for processing may be scheduled and processed on a per group basis. Grouping may permit the OS to interact on a group basis rather than interacting with individual processors. As a result, the OS may be simplified in comparison to an operating system which distributes processing tasks on per processor basis. The individual threads may be assigned to a kernel group for processing. Statically grouping processors, and assigning processing tasks on a per group basis, may minimize the burden associated with scheduling individual processors in systems including a large number of processors. For example, a enterprise server having one hundred twenty eight processors may be configured to handle several applications which were designed to operate effectively in a two processor desktop type system.

[0008] For applications which may experience synchronization or correctness issues, if processing is carried out by more processors than was anticipated for processing the application, the threads associated with the application may be processed in a single kernel grouping so that processing may be isolated (e.g., occur as if the processors in the kernel group were the only processors in the system). In this manner, a first kernel group may service a first application while other applications, if applicable, may be processed by other kernel groups. The static

processor grouping may promote efficient processing of applications which were designed for processing on a limited number of processors, while supporting overall scalability for the applications running on the computing system.

[0009] **Exemplary Environment**

FIG. 1 illustrates an environment 100 in exemplary implementations that permits kernel grouping. For example, a computing system 102 having sixty four or more processors (104- 134 will be referenced) may be configured so that the processors are configured into groups (GROUPS "0-N" will be referenced, respectively 136-144) on the kernel level. An OS operating on the computing system may be configured on the kernel level to cause threads or discrete tasks to be processed on particular group of processors, in the computing system. Kernel level processor grouping may relieve applications or other level from having to account for the number of processor included in the system while considering the physical locality of the individual processors. This is to say, that by accounting for multiple processor on the kernel level, issues associated with conducting processing on numerous processors, in comparison to that for which an application was configured, may be minimized. For example, software application modules may be isolated or otherwise partitioned to make use of the available processing resources. As a result, applications which are not readily scaleable may be processed without overly consuming computing system resources in comparison to an anticipated processor use. For instance, an application may consume more processing capacity when processed on large number of processors in comparison to the same application when run on the number of processors for which the application was

designed. Performing operations on a per-group basis may minimize scheduling burden on the OS when distributing threads for processing. For example, by distributing individual threads to a kernel group for processing, the burden on a kernel scheduler 145 may minimized in comparison to distributing tasks to individual processors. For instance, minimizing the number of kernel group in the computing system may allow the kernel scheduler 145 to implement a relatively simpler, and potentially faster, algorithm.

[0010] While physical embodiments are discussed, virtualized implementations are contemplated as well. For example, applications may be run in a virtual environment, or in a combined environment, and so on. For example, an application may be executed in a virtual environment on a computing system which is physically partitioned as well.

[0011] Although, a 64 (sixty-four) processor computing system is referenced, the techniques and principles discussed herein may be scaled, such as for "P" processors (such as in a particular GROUP) and "N" kernel groups (including "P" individual processors), as desired based on computing system resources, such as a one hundred twenty eight processor system, hardware/software, and so on. For example, the sixty-four processors or more processors may be configured into "N" kernel level groupings with individual kernel groupings having "P" processors. In implementations, a kernel group may include a single processor, two processor and so on. In another example, a sixty-four processor system may be configured as a single group as a sixty-four bit bitmask may effectively address the grouping. In

further examples, a system having sixty-four or more processors may be configured into approximately sixty-four processor per group.

[0012] The precise number of kernel groups and individual processors within individual kernel groups may vary. For example, processors may be hot-swapped into groups as desired, and so on. For example, while a computing system may include sixty-four or more physical processors, some groups may be dedicated to specific tasks, or processors may be reserved for subsequent group assignment and so on. The burden of scheduling tasks may be limited by minimizing the number of kernel groups. For example, the number of individual processors within a group may be selected based on the number of processors an applications is designed to be processed on. In this manner, a computing system may support applications which do not lend themselves to processing by a larger number of processors. For example, a kernel group may include two processors if an application, which is expected to be routinely process, may make efficient use of two processors. This is to say, the number of individual processors in a kernel group may be assigned based on application parameters or for applications which are expected to be processed. For instance, if an enterprise server is anticipated to run at least one application which was designed for processing on four processors, four processors may be statically assigned to a kernel group so that additional processors beyond that which may be used effectively are not included in the kernel group.

[0013] Other considerations may include, grouping a sufficient number of processors together in order to efficiently process a thread. In addition to scheduling application threads to dedicated groups, the kernel scheduler 145 may

assign tasks on a round-robin basis as kernel groups become available for processing.

[0014] Individual processors within kernel groups may be selected based on the locality of the processor with respect to the other processors within the kernel group. In implementations, individual processors within non-uniform memory access (NUMA) nodes may be included in kernel groups. For example, one or more individual processors assigned to a local memory node may be included in a kernel group. As a result, the processors within the NUMA node may have efficient access to local memory resources when processing a thread assigned to the kernel group which includes the processors in the NUMS node. In-turn, the processors included in the NUMA nodes (for reference NUMA nodes 0-15, respectively 146-152) may be assigned to particular kernel groupings. Including locality adjacent processors within a kernel grouping, whether in the same NUMA node or not, may improve overall processing while various portions of the system are used by different applications. Other factors for deciding groupings may be used in conjunction with locality or may be implemented, such processing core configurations or other factors as desired. For example, a kernel grouping configuration may be based on the configuration of a processing core and the core's socket configuration.

[0015] Kernel level processor grouping may prevent erratic application performance, correctness issues, synchronization issues and so on for computing systems having large numbers of processors, in comparison to a low processor system for which an application was designed. In instances, a computing system

having a large number of processors (such as one hundred and twenty-eight) may potentially experience the above issues, while a lower resource system running the same application may not. Grouping the processor according to the techniques herein may cause the application and/or the system to effectively minimize the potential for erratic behavior.

[0016] In implementations, the number of processors within a kernel may permit a common bitmask of a designated size to be used. For example, a sixty-four bit bitmask may be managed in an efficient manner, while accommodating the grouped processor configuration. Other exemplary situations may implement a 32-bit bitmask (thirty-two bit bitmask) configuration.

[0017] By using kernel groupings, OS assigned threads available for processing, may make efficient use of computing system resources thereby avoiding potential issues which may occur in server having large numbers of processors. In a large scale processor system, a kernel scheduler 145 may assign individual application threads for processing to particular kernel groups so that multiple application, which are suited to a lesser number of processors than what are included in the system, may be processed in a generally contemporaneous manner thereby making more efficient used of the system's processor than if the processors were handled on an individual basis.

[0018] If a particular application has synchronization, correctness or other multiple processor issues (if executed on a computing system with multiple processors), the kernel scheduler 145 may schedule the application's threads to a single group to avoid or minimize these potential issues. For example, if an application is not

readily scalable, the kernel scheduler 145 may direct the application's processing task to a single group. For example, if a computer system having sixty-four or more processors is to process a task which is multiple processor sensitive, the kernel scheduler 145 may direct the application threads to GROUP 0 (zero) which may isolate the application. In this manner, the threads may be processed as if the processors in the group were the system's processing resources. The level of isolation may vary from a physical or virtual partition type isolation to lesser forms of isolation as desired.

[0019] In contrast, if an application is configured for multiple processor processing, threads for processing may be individually scheduled for processing among GROUP 1, GROUP 2 and GROUP 3, (which individually may include multiple processor akin to the configuration of GROUP 0) to take advantage of the computing system's processor resources.

[0020] In implementations, applications and drivers may be given visibility to the entire system. For example, a driver may be made aware of the kernel level structure so that the driver may support a component having access to the system. In this manner, the computing system may obtain group processing benefits while applications and drivers may be made aware of the systems processor groupings.

[0021] Generally, any of the functions described herein can be implemented using software, firmware, hardware (e.g., fixed logic circuitry), manual processing, or a combination of these implementations. The terms "module," "functionality," and "logic" as used herein generally represent software, firmware, hardware, or a combination thereof. In the case of a software implementation, for instance, the

module, functionality, or logic represents program code that performs specified tasks when executed on a processor (e.g., CPU or CPUs). The program code can be stored in one or more computer readable memory devices, e.g., tangible memory and so on.

[0022] The following discussion describes transformation techniques that may be implemented utilizing the previously described systems and devices. Aspects of each of the procedures may be implemented in hardware, firmware, or software, or a combination thereof. The procedures are shown as a set of blocks that specify operations performed by one or more devices and are not necessarily limited to the orders shown for performing the operations by the respective blocks.

[0023] **Exemplary Procedures**

The following discussion describes a methodology that may be implemented utilizing the previously described systems and devices. Aspects of each of the procedures may be implemented in hardware, firmware, or software, or a combination thereof. The procedures are shown as a set of blocks that specify operations performed by one or more devices and are not necessarily limited to the orders shown for performing the operations by the respective blocks. A variety of other examples are also contemplated.

[0024] FIG. 2 discloses exemplary procedures for statically grouping processors. For example, a computing system's OS kernel level may be configured to group individual processor so applications which may be sensitive to large scale processor environments may be processed.

[0025] The techniques discussed herein may permit processing of threads assigned by the OS in an isolated manner on a group basis. These techniques may minimize the overall complexity of the OS as processing may be consider on a group basis rather than parsing out tasks on a individual processor basis.

[0026] For application designed for large scale processing the individual groups may isolate the individual task in one assignment entity from other assignment entities within computing system.

[0027] The individual processor may be grouped 202 into an assignment entity for processing threads. For example, the kernel scheduler may assign a particular application to processors statically grouped at startup. For example, threads from a first application may be scheduled to a first assignment entity, while other application tasks are assigned to a second assignment entity and so on. For example, a first application may be assigned to a first static kernel grouping of two processors, which may be physically local to the other processor included in the group, while a second application is assigned to a second static kernel group having four processors. The foregoing may permit the first and second applications to be processed more efficiently (make more efficient use of the processors) than if the OS interacted with the processors on an individual basis.

[0028] In implementations, the number of processors included in an assignment entity may be commensurate with the number of processor for which an application was configured. In this way, processing resources may not be dedicated to a particular application which may not be able to make effective use of the number of processors assigned to the group. Other assignment entities may be similarly

configured so that the individual groups may be assigned tasks individually from different applications.

[0029] The number of processors included in an assignment entity may be determined based on the bitmask used in conjunction with the processors. Thus, the number of processors within a group may be, for example, sixty-four or less in order to make use of a sixty-four bit bitmask. In this manner, the system may use lower bit bitmask configuration, which may be of a simpler configuration, while accommodating a system having a processors beyond that which the bitmask may effectively address. In instances, the number of processors assigned into processing groups may be less than that of the bitmask as some processor may be withheld hot-swapping and so on.

[0030] Using a grouped processor configuration (on the kernel level) may minimize the impact of a multi-processor environment on applications executing at a higher level. Thus, while the kernel level may be configured for controlling multiple processors as a entity, such as in a real environment, a virtualized environment or a combination thereof, a lower bit bitmask configuration may be used without having to reconfigure the bitmask for the additional processors beyond that for which the bitmask is configured.

[0031] In implementations, the number of assignment entities within an OS/computing system may be minimized in order to minimize the scheduling burden associated with dispersing application tasks (e.g., potential bottlenecks). This may permit kernel level scheduling using simpler algorithms in comparison to a system which individually addresses the processors. The precise number of

assignment entities and the number of individual processors within the assignment entities may be based on a number of factors including the expected application processing demands, the number of processors which may be implemented with an application before the application experiences processing issues, minimize scaling bottlenecks and so on.

[0032] Individual processors may be included in a particular assignment entity to maximize the locality of the processors within the assignment entity. For example, the processors within an assignment entity may be physically adjacent to the other individual processors within the assignment entity.

[0033] The individual processors within a non-uniform memory access node (NUMA) node may be grouped into an assignment entity. Thus, individual processors assigned 204 in a NUMA node may be grouped into a particular kernel level processor group. In the previous manner, the individual processors included in the assignment entity and the NUMA node may have rapid access the local memory (in comparison to individual processors in which the preceding does not apply). While NUMA nodes may not be not tied to particular kernel groups, in implementations nodes may be mapped 206 to kernel groups in order to closely affinitize specific tasks. For example, an application program interface may be configured to tie individual physical processors with the kernel groupings. Closely mapping physical resources with the kernel level assignment entities may permit assigning closely associated tasks to particular kernel groupings for processing.

[0034] For applications suitable for scaling 208, the kernel scheduler may assign the threads as desired. For applications designed for a limited number of processors,

the threads may be directed to a single kernel group. For example, an application may have its processing tasks directed to GROUP 0, while other application may be directed to GROUP 1.

[0035] Applications and drivers may be given visibility to the system's grouping. For example, a driver may be made aware of the kernel level grouping structure so that the driver may support a component having access to the overall system. In this manner, a computing system operating in conformance with the techniques discussed herein may obtain group processing benefits while applications and drivers may be made aware of the systems processor groupings as desired.

[0036] **Conclusion**

Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.

CLAIMS

What is claimed is:

1. A method comprising:

statically grouping individual processors into an assignment entity, the individual processors being grouped based on the physical locality of the individual processors with respect to other individual processors within the group; and

scaling one or more assignment entities to handle individual threads so that individual threads are assigned on a per assignment entity basis.

2. The method as described in claim 1 wherein the individual processors are statically grouped on a kernel level.

3. The method as described in claim 1 wherein the assignment entity is configured to isolate individual threads, associated with an application, within the assignment entity.

4. The method as described in claim 1 wherein a single thread is assigned to a single individual assignment entity at a time.

5. The method as described in claim 1 wherein individual processors within the assignment entity are configured into a non-uniform memory

access (NUMA) node with other individual processors in the assignment entity.

6. The method as described in claim 1 wherein individual processor are grouped at start-up.

7. The method as described in claim 1 further comprising partially populating a non-uniform memory access (NUMA) node at start-up to permit hot-adding of processors.

8. The method as described in claim 1 wherein threads for a single application are processed in a single assignment entity.

9. One or more computer-readable media comprising computer-executable instructions that, when executed, direct a computing system to:

assign locality related individual processors to non-uniform memory access (NUMA) nodes; and

group the locality related individual processors statically at startup into a kernel group which is configured to process an individual thread at a time.

10. The one or more computer-readable media as described in claim 9 further comprising schedule the individual thread on a per kernel group basis.

11. The one or more computer-readable media as described in claim 9 wherein application threads are all scheduled to single kernel group for processing.

12. The one or more computer-readable media as described in claim 9 further comprising add driver model extensions for hardware configured for scalable processing.

13. The one or more computer-readable media as described in claim 9 further comprising scale additional kernel groups for processing threads from applications supporting distributing of threads among different kernel groups.

14. The one or more computer-readable media as described in claim 9 wherein individual processors in the kernel group are isolated from processors outside the kernel group.

15. The one or more computer-readable media as described in claim 9 wherein a group includes approximately sixty-four individual processors.

16. A system comprising:

a plurality of processors, statically configured in a kernel group such that individual threads for processing are assigned on a kernel group basis; the individual processors being physically local to the other processors included in the kernel group.

17. The system of claim 16 further comprising a kernel scheduler configured to distribute individual threads on a kernel group basis.

18. The system of claim 17 wherein the kernel scheduler assigns all threads for a particular application to a single kernel group.

19. The system of claim 16 wherein individual processors, in the plurality of processors, are physically located locally to other processors included in the kernel group.

20. The system of claim 16 wherein the plurality of processors are assigned to a kernel group upon starting up the system.

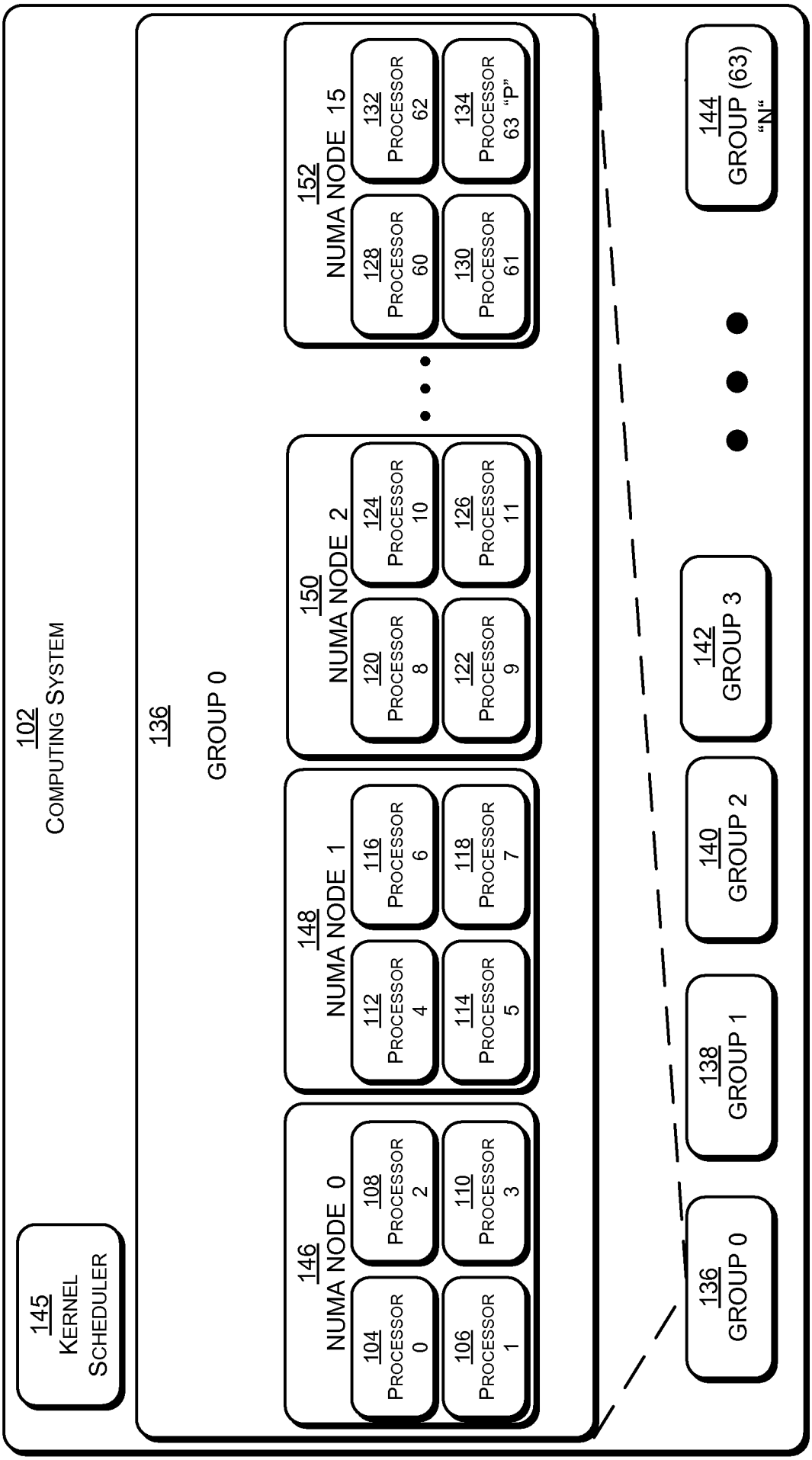
102
COMPUTING SYSTEM
145
KERNEL SCHEDULER
136
GROUP 0
146
NUMA NODE 0
104
PROCESSOR 0
106
PROCESSOR 1
108
PROCESSOR 2
110
PROCESSOR 3
148
NUMA NODE 1
112
PROCESSOR 4
114
PROCESSOR 5
116
PROCESSOR 6
118
PROCESSOR 7
150
NUMA NODE 2
120
PROCESSOR 8
122
PROCESSOR 9
124
PROCESSOR 10
126
PROCESSOR 11
152
NUMA NODE 15
128
PROCESSOR 60
130
PROCESSOR 61
132
PROCESSOR 62
134
PROCESSOR 63 "P"
136
GROUP 0
138
GROUP 1
140
GROUP 2
142
GROUP 3
144
GROUP (63) "N"

Fig. 1

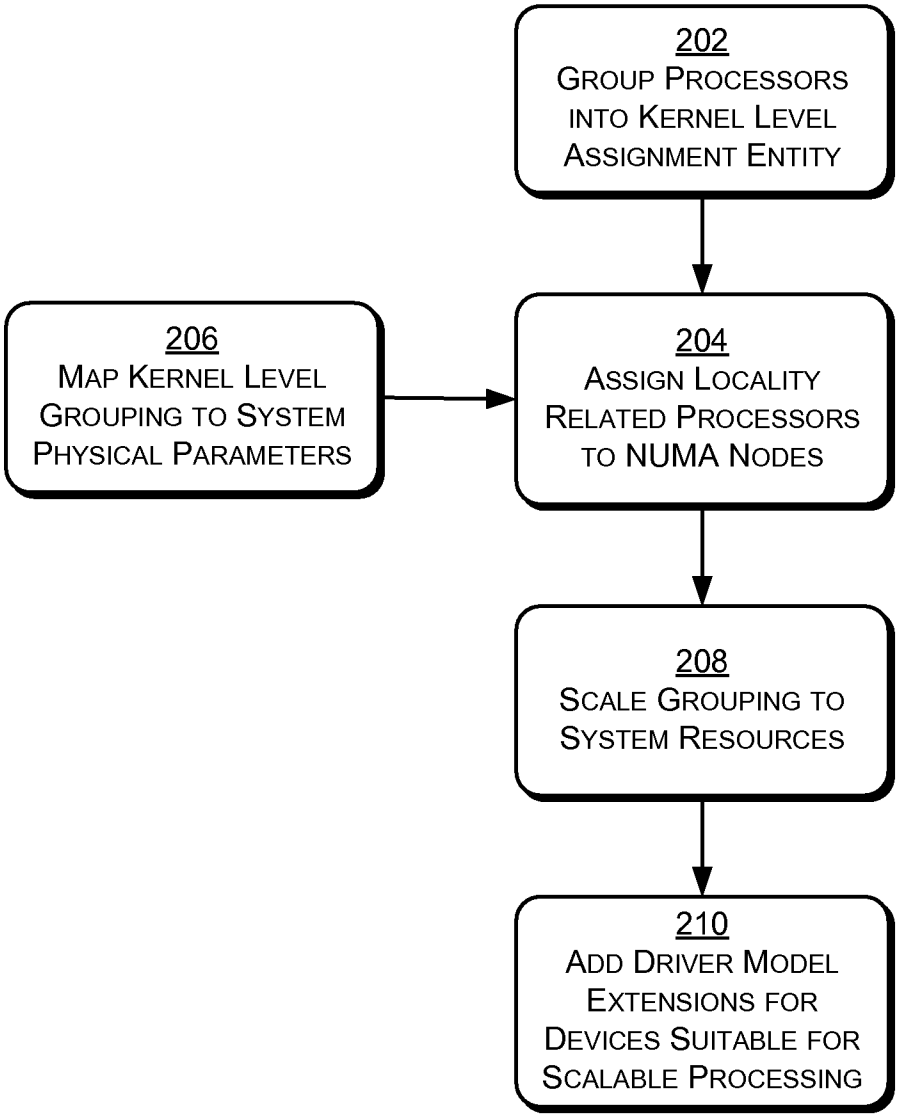
202
GROUP PROCESSORS INTO KERNEL LEVEL ASSIGNMENT ENTITY
206
MAP KERNEL LEVEL GROUPING TO SYSTEM PHYSICAL PARAMETERS
204
ASSIGN LOCALITY RELATED PROCESSORS TO NUMA NODES
208
SCALE GROUPING TO SYSTEM RESOURCES
210
ADD DRIVER MODEL EXTENSIONS FOR DEVICES SUITABLE FOR SCALABLE PROCESSING

FIG. 2