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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 May 2018 (11.05.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/084845 A1

(51) International Patent Classification:
G06F 9/46 (2006.01)

(21) International Application Number:
PCT/US2016/060339

(22) International Filing Date:
03 November 2016 (03.11.2016)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **CUMMINS INC.** [US/US]; 500 Jackson Street, Columbus, Indiana 47201 (US).

(72) Inventors: **CORONA, Michael Victor**; 3782 Sitka Circle, Apartment 1218, Columbus, Indiana 47201 (US). **PULKOWSKI III, Apolinary John**; 1460 N. National Road, Columbus, Indiana 47201 (US).

(74) Agent: **LUETTGEN, David G.** et al.; Foley & Lardner LLP, 3000 K St. NW., Washington, District of Columbia 20007 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

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

WO 2018/084845 A1

(54) Title: METHOD FOR EXPLICITLY SPLITTING SOFTWARE ELEMENTS ACROSS MULTIPLE EXECUTION CORES FOR A REAL TIME CONTROL SYSTEM

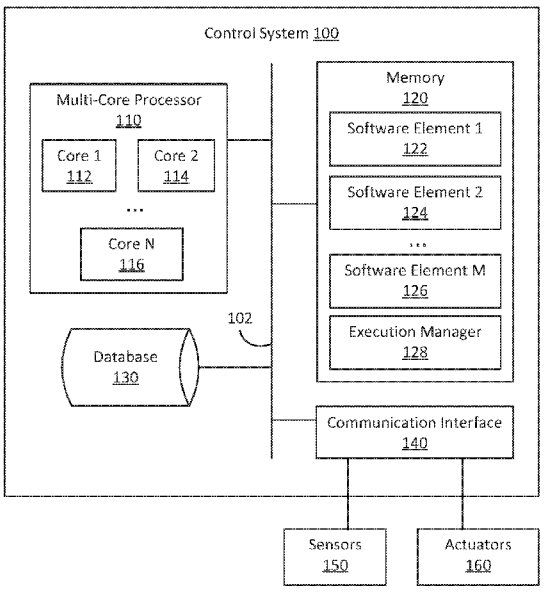

FIG. 1

(57) Abstract: Systems, apparatuses, and methods method disclosed provide for splitting a plurality of software elements across multiple cores of a multi-core processor for a real time control system. The method comprises determining user override for each of the plurality of software elements; grouping more than one software elements of the plurality of software elements as a group based on dependencies of the group of software elements upon each other; and statically determining a core from the multiple cores of the multi-core processor for each software element or group of software elements to execute on.

METHOD FOR EXPLICITLY SPLITTING SOFTWARE ELEMENTS ACROSS MULTIPLE EXECUTION CORES FOR A REAL TIME CONTROL SYSTEM

TECHNICAL FIELD

[0001] The present disclosure relates to apparatus and method for optimizing the performance of a control system with multiple execution cores, and more particularly, to an apparatus and method for splitting software elements across multiple execution cores for a real time control system.

BACKGROUND

[0002] Multi-core processors, which are becoming increasingly popular, provide advantages in terms of throughput, efficiency, power implementation, etc. comparing to single-core processors. In particular, for a single-core processor, as the clock speed increases, the temperature of the processor would increase accordingly. Thus, capabilities of the single-core processor are limited to a point where the processor can no longer be sped up without jeopardizing reliability. While for a multi-core processor, more central processing unit (CPU) cores can be integrated into the processor to improve performance and lower power implementation. The automotive industry is moving from single-core processors towards multi-core processors. Real time control system used on a vehicle controls components and sub-subsystems of the vehicle on a real time basis, which requires execution of software by strict deadlines, and thus presenting unique challenges when moving to multi-core processing systems. Software and hardware resources of a multi-core processing system need to be managed properly in order to implement reliable real time control and optimize performance at the same time.

SUMMARY

[0003] One embodiment relates to an apparatus for splitting a plurality of software elements across multiple cores of a multi-core processor. The apparatus comprises a user override analytics circuit structured to determine user override for each of the plurality of software elements; a dependencies analytics circuit structured to group more than one software elements

of the plurality of software elements as a group based on dependencies of the group of software elements upon each other; and a core determination circuit structured to statically determine a core from the multiple cores of the multi-core processor for each software element or group of software elements to execute on.

[0004] Another embodiment relates to a method for splitting a plurality of software elements across multiple cores of a multi-core processor. The method comprises determining, by an execution manager, user override for each of the plurality of software elements; grouping, by the execution manager, more than one software elements of the plurality of software elements as a group based on dependencies of the group of software elements upon each other; and determining, by the execution manager, a core from the multiple cores of the multi-core processor for each software element or group of software elements to execute on.

[0005] Yet another embodiment relates to a real time control system comprising a multi-core processor including multiple cores; a memory coupled to the multi-core processor and including a plurality of software elements; and an execution manager coupled to the multi-core processor and the memory. The execution manager is structured to determine user override for each of the plurality of software elements; group more than one software elements of the plurality of software elements as a group based on dependencies of the group of software elements upon each other; and statically determine a core from the multiple cores of the multi-core processor for each software element or group of software elements to execute on.

[0006] These and other features, together with the organization and manner of operation thereof, will become apparent from the following detailed description when taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE FIGURES

[0007] FIG. 1 is a schematic diagram of a control system using multiple execution cores, according to an example embodiment.

[0008] FIG. 2 is a block diagram of a multi-core processor used for the control system of FIG. 1, according to an example embodiment.

[0009] FIG. 3 is a schematic diagram of an execution manager used for the control system of FIG. 1, according to an example embodiment.

[0010] FIG. 4 is a flow diagram of a method of splitting software elements across multiple execution cores of a control system, according to an example embodiment.

DETAILED DESCRIPTION

[0011] For the purposes of promoting an understanding of the principles of the disclosure, reference will now be made to the embodiments illustrated in the drawings and specific language will be used to describe the same. It will nevertheless be understood that no limitation of the scope of the disclosure is thereby intended, any alterations and further modifications in the illustrated embodiments, and any further applications of the principles of the disclosure as illustrated therein as would normally occur to one skilled in the art to which the disclosure relates are contemplated herein.

[0012] The automotive industry is moving from single-core processors to multi-core processors for the advantages provided by multi-core processors in terms of throughput, efficiency, power implementation, and so on. As used herein, a "multi-core processor" refers to a processor that combines two or more central processing unit (CPU) cores into a single package. For example, a dual-core processor combines two CPU cores and a quad-core processor combines four CPU cores. As used herein, "core" is interchangeable with "CPU core" or "execution core." Risks regarding data validity ensue when processing systems are moving to multi-core processors. In particular, different cores may access the same data. If a first core is taking a few clock cycles to shift a global data value and a second core reads the global data value before the shifting is completed, the second core might have read invalid data. As a result, false faults regarding operation of the vehicle may be flagged, or parameters for controlling operation of the vehicle may be incorrectly computed, etc. To prevent this from happening, a lock flag may be set before the first core is writing a piece of data. When the first core is writing, the memory storing the data is flagged as locked so that other cores intending to access the data will wait until the lock flag is cleared. However, frequent data contention lowers the processing speed and efficiency. On the other hand, the control system used on a

vehicle controls components and sub-subsystems of the vehicle on a real time basis, which may require execution of software by strict deadlines. When the software is allocated to execute across multiple cores for the control system, the set deadlines may need to be met for reliable real time control.

[0013] The apparatuses and methods disclosed herein provide flexibility and configurability to optimize system throughput, to reduce data contention, and to ensure event chain consistency for a real time control system with multiple execution cores. Referring to the Figures generally, the various embodiments disclosed herein relate to apparatuses and methods for splitting real time embedded software control system across multiple cores based on software characteristics. Each software element is analyzed to evaluate its dependency (e.g., order dependency and data dependency) on other software elements that have been assigned to execute on each of the multiple cores. The execution load of the software element and the load availability of each core are also evaluated. Based on the evaluation, the software element is allocated to run on a most suitable core. Since each software element is allocated appropriately, data contention can be minimized, throughput can be improved, and core utilization can be balanced.

[0014] Referring now to FIG. 1, a schematic diagram of a control system using multiple execution cores is shown, according to an example embodiment. The control system 100 may be a real time control system mounted on a vehicular electronic control unit as an example of an application. The vehicle (not shown in the present figure) may include, but is not limited to, line-haul trucks, mid-range trucks (e.g., pick-up truck), full electric vehicle, hybrid electric vehicle, etc. The electronic control unit may include, but is not limited to, an engine control unit for controlling operation of the engine, a hybrid control unit for coordinating operation of the engine and the electric motor, and/or a control unit for controlling operation of other components and/or sub-systems of the vehicle, such as the transmission system, the navigation system, the aftertreatment system, the air conditioning system, the on-board diagnostic (OBD) system, the telematics system, and so on.

[0015] The control system 100 comprises a multi-core processor 110, a memory 120, a database 130, and a communication interface 140 connected via an internal bus 102. The control system 100 is connected to sensors 150 and actuators 160 via the communication interface 140. The control system 100 can receive information indicative of operation conditions from the sensor 150 and send controlling instructions to actuators 160. In the vehicular application, the sensors 150 may include a variety of sensors such as, a fuel level sensor, an air flow sensor, a crank position sensor, an engine speed sensor, an electric motor current sensor, a rotor resolver, or the like. The actuators 160 may include a variety of actuators such as, a throttle valve, a fuel injection valve, an electric motor, or the like. The communication interface 140 may include any type and number of wired and wireless protocols (e.g., any standard under IEEE 802, etc.) to enable communication between the control system 100 and the sensors 150/actuators 160 via any type and number of wired and wireless connections. For example, a wired connection may include a serial cable, a fiber optic cable, an SAE J1939 bus, a CAT5 cable, or any other form of wired connection. In comparison, a wireless connection may include the Internet, Wi-Fi, Bluetooth, Zigbee, cellular, radio, etc. In one embodiment, a controller area network (CAN) bus including any number of wired and wireless connections provides the exchange of signals, information, and/or data between the real time control system 100 and the sensors 150/actuators 160. In other embodiments, a local area network (LAN), a wide area network (WAN), or an external computer (for example, through the Internet using an Internet Service Provider) may provide, facilitate, and support communication between the control system 100 and the sensors 150/actuators 160.

[0016] The multi-core processor 110 includes two or more central processing unit (CPU) cores in the same package, such as, the first core 112, the second core 114, the N-th core 116, etc. The multiple cores may share the same interconnect to the rest of the control system 100. In some embodiments, the cores 112, 114, and 116 are homogeneous, each having the same architecture, processing power, speed, and energy characteristics. In other embodiments, the cores 112, 114, and 116 are heterogeneous, having diverse characteristics.

[0017] FIG. 2 shows a top-level block diagram of an exemplary multi-core processor 200, which can be used in the control system 100 of FIG. 1. FIG. 2 illustrates the general

interconnection of functional modules through a crossbar switch 202. As shown, the processor 200 includes three CPU cores 210, 212, 214, each having a cache 211, 213, and 215 (e.g., 8K Cache, 16K Cache), and a memory management unit 220. The multi-core processor 200 further includes various types of memory 230 (e.g., static random access memory (SRAM) 232, flash memory 234, etc.). The cores 210, 212, and 214 can run independently and exchange data with each other using caches 211, 213, and 215 and/or memory 230. It should be understood that the illustrated processor 200 is one exemplary multi-core processor, multi-core processors with various numbers of cores and various components can also be used with the control system 100.

[0018] Referring back to FIG. 1, the memory 120 may include tangible, non-transient volatile memory, or non-volatile memory, such as NVRAM, RAM, ROM, flash memory, hard disk storage, etc. The memory 120 may further include database components, object code components, script components, or any other type of information structure. The memory 120 may store data and/or computer code for execution by multi-core processor 110, for example, a first software element 122, a second software element 124, an M-th software element 126, and an execution manager 128. As used herein, a "software element" refers to a software unit executable on a CPU core. Software elements 122, 124, and 126 can perform a variety of controlling processes. In the vehicular application, for example, the controlling processes may include controlling operation of various components and/or sub-systems of the vehicle, such as, the engine, the electric motor, the transmission system, the navigation system, the aftertreatment system, the air conditioning system, the on-board diagnostic (OBD) system, the telematics system, lights, lock/unlock, windshield wipers, and so on. The execution manager 128 manages the execution of the software elements, including, for example, allocating each of the software elements 122, 124, and 126 to a most suitable core 112, 114, or 116 of the multi-core processor 110 for execution based on evaluation of multiple factors. The execution manager 128 assigns each software element to a most suitable core while compiling a software program. The allocation or assignment is static, meaning that the allocation or assignment will not be changed when the program is running. The structure and function of the execution manager 128 will be discussed in further detail with reference to FIG. 3.

[0019] There are different types of executable software elements in terms of the function, including task, runnable, standard service, concurrent service, reentrant service, utility service, etc. As used herein, a "task" refers to a system event which triggers a number of runnables mapped to the task to be executed. For example, a task may be periodic interrupts, e.g., interrupts occurring every 5 ms or 10 ms. In another example, a task may be non-periodic events, e.g., pin voltage changes or engine speed changes. As used herein, a "runnable" refers to a function that is mapped to a task and executed in response to the task being triggered. For example, a runnable may be updating data on the CAN bus when the 5 ms periodic interrupt is triggered. In another example, a runnable may be processing data for controlling fuel injection when engine speed changes to certain extent. As used herein, a "standard service" refers to a function that is called to execute by the application code (i.e., not by the execution manager 128), reads or writes global data or physical resources (e.g., accessing the sensors 150 and/or the actuators 160), and cannot be called to execute concurrently from more than one cores. For example, a standard service may be a function for setting error information. When one core is writing error information, other cores would wait to access the information. As used herein, a "concurrent service" refers to a function called to execute by the application code, accesses global data, and can be called to execute concurrently from more than one cores. For example, a concurrent service may be a function for reading operating system (OS) time. As used herein, a "reentrant service" refers to a function that is called to execute by the application code, and does not access global data or physical resources. For example, a reentrant service may be a function for linearizing XYZ coordinates. As used herein, a "utility service" refers to a reentrant service which is used for generable purpose, and hence implemented in a reusable "utility" component. It shall be understood that the different types of executable software elements listed herein are for illustration, and not for limitation. The software elements can include other types of core-executable units. It shall be also understood that the software elements are not tied to any specific programming language. The software elements may be programmed using any suitable computer language, such as Java, C, C++, and so on.

[0020] Software elements may have parameters identifying and characterizing the elements, such as element identification, element type, core override, global data writes, global data

reads, number of instructions or execution time for minimum case, number of instructions or execution time for maximum case, periodic rate of execution, element order dependencies, and so on. The element identification is used to identify the software element for processing. The element type specifies which type the software element belongs to, such as task, runnable, standard service, concurrent service, reentrant service, utility service, etc., as discussed above. Core override specifies whether the software element is "hard coded" to a particular core to "override" any allocation by the execution manager 128. In other words, a user can set up the core override to assign the software element to execute on the particular core and the assignment would be not changed by the execution manager 128. For example, the user can have all software elements for transmission system control execute on a designated core. The global data writes specifies the global data and/or the physical resources the software element may write to. The global data reads specifies the global data and/or the physical resources the software element may read from. The parameters of global data writes/reads do not apply to all software element types. For example, a standard service and a concurrent service may access global data and/or physical resources during execution, but a reentrant service and a utility service do not access global data or physical resources during execution. The number of instructions or execution for minimum case specifies the minimum possible execution load the software element may have if allocated to a core. The number of instructions or execution time for maximum case specifies the maximum possible execution load the software element may have if allocated to a core. The periodic rate of execution specifies how often a software element may execute if allocated to a core. For example, a task triggered by periodic interrupts may have a periodic rate of execution of every 5 ms, 10 ms, etc., depending on the periodic rate of the interrupts. The element order dependencies specify a direct order the software element has with any other software elements. For example, a task and a runnable mapped to the task have direct order dependencies.

[0021] The real time control system 100 further includes the database 130 that may receive, store, hold, and otherwise serve as a repository for information relating to the cores 112, 114, and 116, for example, which software elements have been assigned to each of the cores 112, 114, and 116, parameters related to the assigned software elements, characteristics and load

availability of each core, etc. The database 130 may also include one or more classification and/or categorization functions (e.g., logic processing, etc.). The classification function may sort, categorized, or otherwise classify each piece of information for each core 112, 114, and 116 of the multi-core processor 110.

[0022] Referring now to FIG. 3, a schematic diagram of an execution manager used for the real time control system of FIG. 1 is shown, according to an example embodiment. The execution manager 300 includes various circuits for completing the activities described herein. In some embodiments, the circuits of the execution manager 300 may utilize the multi-core processor 310 that contains multiple CPU cores and/or the memory 320 that stores the software elements to accomplish, perform, or otherwise implement various actions described herein with respect to each particular circuit. In other embodiments, the circuits (or at least one of the circuits) may include their own dedicated processing circuit having a processor and a memory device. In the latter embodiments, the circuit may be structured as an integrated circuit or an otherwise integrated processing component. While various circuits with particular functionality are shown in FIG. 3, it should be understood that the execution manager 300 may include any number of circuits for completing the functions and activities described herein. For example, the activities of multiple circuits may be combined as a single circuit, as an additional circuit(s) with additional functionality, etc. Further, it should be understood that the execution manager 300 may further control other activity beyond the scope of the present disclosure.

[0023] Certain operations of the execution manager 300 described herein include operations to interpret and/or to determine one or more parameters. Interpreting or determining, as utilized herein, includes receiving values by any method known in the art, including at least receiving values from a datalink or network communication, receiving a computer generated parameter indicative of the value, reading the value from a memory location on a non-transient computer readable storage medium, receiving the value as a run-time parameter by any means known in the art, and/or by receiving a value by which the interpreted parameter can be calculated, and/or by referencing a default value that is interpreted to be the parameter value.

[0024] As shown, the execution manager 300 includes a user override analytics circuit 301, a dependencies analytics circuit 302, and a core determination circuit 303 which includes an element order dependencies analytics circuit 304, an element data dependencies analytics circuit 305, and an execution load analytics circuit 306. Through the circuit 301 – 306, the execution manager 300 can allocate each of the software elements stored in the memory 320 (which may correspond to the memory 120 of FIG. 1) to execute on a particular core of the multi-core processor 310 (which may correspond to the multi-core processor 110 of FIG. 1) using information stored in the database 330 (which may correspond to the database 130 of FIG. 1).

[0025] The user override analytics circuit 301 is structured to analyze user override data to determine whether a software element has been assigned to execute on a particular core by a user. As discussed above, a parameter core override related to a software element specifies whether the software element is "hard coded" to a particular core to "override" any allocation by the execution manager 300. The user can set up the core override parameter of the software element to assign the software element to execute on the particular core and the execution manager 300 will not change the assignment. The user override analytics circuit 301 may read the core override parameter to determine any user override for the software elements. If the software is assigned to a particular core, the execution manager 300 will so allocate. In some embodiments, the user may have a software element execute on a particular core having the architecture, processing power, speed, and/or cache size that particularly fits the execution of the software elements. In some embodiments, the user may have can have all software elements for a particular system (e.g., transmission system) control execute on a core designated for transmission system control.

[0026] The dependencies analytics circuit 302 is structured to analyze dependencies among software elements to be assigned and group more than one software elements based on the dependencies upon each other as one element for allocation to execute on a same core. As used herein, "dependencies" refer to the relationships among software elements. In the control system where multiple CPU cores are mounted, the control system can allocate a plurality of software elements to each core while allowing temporal overlap of execution, thus improving

the processing efficiency. The respective cores are capable of performing the software elements independently in parallel. However, some software elements may be so closely related to (*i.e.*, "dependent upon") each other that it is infeasible and/or inefficient to allocate them to execute on different cores. The dependencies may include two aspects, order dependencies and data dependencies. As used herein, "order dependencies" refer to the direct order a software element has with any other software elements. As used herein, "data dependencies" refer to the overlap of global data and/or physical resource that a software element and any other software elements access.

[0027] There are situations where software elements need to execute in a particular order. For example, a task and a runnable mapped to the task have direct order dependencies. The runnable is executed in response to the task being triggered. In an example, the runnable updates data on the CAN bus when the task of 5 ms periodic interrupt is triggered. In another example, the runnable processes data for controlling fuel injection when the task of engine speed changes is triggered. In addition, for series of vehicular controls, a software element needs to use an execution result of a previous software element. For example, a software element that determines the amount of current supplied to the electric motor to reach a target speed of the electric motor uses the previous amount of current for feedback controls. Distributing the software elements having order dependencies to different cores might not be efficient. In particular, if two software elements that need to execute successively are allocated to two cores, the core to which the latter software element is allocated might idle for relatively long time until the preceding software element executes on the other core.

[0028] In another aspect, different software elements may need to access the same global data and/or physical resources frequently. If a first core is taking a few clock cycles to shift a global data value and a second core reads the global data value before the shifting is completed, the second core may have read invalid data. As a result, false faults regarding operation of the vehicle may be flagged, or parameters for controlling operation of the vehicle may be incorrectly computed, etc. To prevent this from happening, a lock flag may be set before the first core is writing a piece of data. When the first core is writing, the memory storing the data is flagged as locked so that other cores intending to access the data will wait until the lock flag

is cleared. If two software elements that access the same global data frequently are allocated to run on different cores, a significant amount of overhead might be introduced due to locking and cache inefficiencies. Locking inefficiencies arises from the two software elements concurrently attempting to access the same locked structure employed to protect the global data. Cache inefficiencies arise because the data required by the reading software element must be first written into the cache memory associated with core to which the writing software element is assigned, and thereafter transferred over the system bus into the cache memory associated with the core to which the reading software element is assigned.

[0029] The dependencies of software elements upon each other can be determined based on the order dependencies, the data dependencies, or any combination of order and data dependencies. In some embodiments, the dependencies analytics circuit 302 determines that certain software elements are grouped as one if the software elements are strictly restricted to execute in a particular order. The dependencies analytics circuit 302 may obtain the information of order dependencies by analyzing the parameter of element order dependencies associated with the software elements. In some embodiments, the dependencies analytics circuit 302 determines that certain software elements are grouped as one if the number of global data and/or physical resources accessed by the software elements and/or the frequency of accessing exceed a predefined threshold number and/or a predefined threshold frequency. In particular, the dependencies analytics circuit 302 may analyze the parameter of element type associated with each software element to determine whether the data dependencies analysis is needed. As discussed above, a standard service and a concurrent service may access global data and/or physical resources during execution, but a reentrant service and a utility service do not access global data or physical resources during execution. For element types that indicate global data and/or physical resources would be accessed, the dependencies analytics circuit 302 further analyzes the parameter of global data writes and global data reads associated with the software elements to determine overlap of the global data accessed and whether to group the software elements as one. In some embodiments, the dependencies analytics circuit 302 determines that certain software elements are grouped as one if both direct order dependencies and certain extent of data dependencies exist. It should be understood that the examples of

dependencies analysis given here are not for limitation; various algorithms can be used to determine dependencies among software elements. The dependencies analytics circuit 302 may then group the software elements having close dependencies upon each other as one element and the execution manager 300 would allocate the group to execute on the same core.

[0030] The core determination circuit 303 is structured to determine a particular core to which a software element or a group of software elements is allocated based upon multiple factors. The core determination circuit 303 may further comprises the element order dependencies analytics circuit 304, the element data dependencies analytics circuit 305, and the execution load analytics circuit 306. The element order dependencies analytics circuit 304 is structured to evaluate the order dependencies of a software element (or a group of software elements) to be allocated upon software elements that have been allocated to execute on each of the cores. In particular, the element order dependencies analytics circuit 304 may obtain information of the software elements that have been allocated to each of the cores from the database 330 and obtain the parameter of element order dependencies associated with the software element to be allocated. If the parameter of element order dependencies indicates that the software element has order dependencies upon an allocated software element of a core, the element order dependencies analytics circuit 304 may increase a value of order dependencies for the core. The more allocated software elements of a core indicated by the parameter of element order dependencies associated with the software element to be allocated, the more value of order dependencies are determined for the core. In some embodiments, different values are determined for different element types. For example, if the software element to be allocated is a standard service, which cannot be called concurrently from multiple cores, and the software element that has been allocated to a core is a standard service, the order dependencies of these two software elements add the most to the value of order dependencies. If the software element to be allocated is a concurrent service, which can be called concurrently from multiple cores, the software element that has been allocated to a core is a standard service, the order dependencies of these two software elements add the second most to the value of order dependencies. If the software element to be allocated is a concurrent service, and the software element that has been allocated to a core is a concurrent service, the order

dependencies of these two software elements adds the least to the value of order dependencies. The element order dependencies analytics circuit 304 may communicate the value of order dependencies for each of the multiple cores to the core determination circuit 303.

[0031] The element data dependencies analytics circuit 305 is structured to evaluate the data dependencies of a software element (or a group of software elements) to be allocated upon software elements that have been allocated to execute on each of the cores. In particular, the element data dependencies analytics circuit 305 may first analyze the parameter of element type associated with the software element to determine whether the data dependencies analysis is needed. As discussed above, a standard service and a concurrent service may access global data and/or physical resources during execution, but a reentrant service and a utility service do not access global data or physical resources during execution. If the software element is a standard service or a concurrent service, the element data dependencies analytics circuit 305 further analyzes data dependencies for the software element. The element data dependencies analytics circuit 305 may obtain information of the software elements that have been allocated to each of the cores from the database 330 and obtain the parameters of global data writes and global data reads associated with the software element to be allocated. The element data dependencies analytics circuit 305 may determine a value of data dependencies for each core based on the overlap of global data accessed by the software element to be allocated with global data accessed by the software elements that have been allocated to the core. In some embodiments, the more overlapped global data accessed by the software element to be allocated and the software elements that have been allocated to the core, the larger value of data dependencies is determined for the core. In some embodiments, the more frequently the overlapped global data is accessed by the software element to be allocated and the software elements that have been allocated to the core, the larger value of data dependencies is determined for the core. In some embodiments, different values of data dependencies are determined for different element types. For example, if the software element to be allocated is a standard service, which cannot be called concurrently from multiple cores, and the software element that has been allocated to a core is a standard service, the overlap of global data accessed by these two software elements adds the most to the value of data dependencies. If

the software element to be allocated is a concurrent service, which can be called concurrently from multiple cores, the software element that has been allocated to a core is a standard service, the overlap of global data accessed by these two software elements adds the second most to the value of data dependencies. If the software element to be allocated is a concurrent service, and the software element that has been allocated to a core is a concurrent service, the overlap of global data accessed by these two software elements adds the least to the value of data dependencies. The element data dependencies analytics circuit 305 may communicate the value of data dependencies for each of the multiple cores to the core determination circuit 303.

[0032] The execution load analytics circuit 306 is structured to evaluate the execution load of a software element (or a group of software elements) to be allocated and the load availability of each of the cores. The execution manager 300 may decide software element allocation to maintain good load balance and/or optimal system resource utilization. In particular, the execution load analytics circuit 306 may obtain information of the load availability of each of the cores from the database 330, including for example, throughput capacities, clock speed, processing power of the core, load of the software elements that have been assigned to each core, etc. The execution load analytics circuit 306 may obtain the information of the execution load of the software element to be allocated based on parameters of number of instructions or execution time for minimum case, number of instructions or execution time for maximum case, periodic rate of execution, etc. In particular, the number of instructions or execution for minimum case specifies the minimum possible execution load the software element may have if allocated to a core. The number of instructions or execution time for maximum case specifies the maximum possible execution load the software element may have if allocated to a core. The periodic rate of execution specifies how often the software element may execute if allocated to a core. The execution load analytics circuit 306 may use these parameters to estimate the predicted minimal load and maximum load of each core. The execution load analytics circuit 306 may then determine a value of execution load for each core based on the load availability of the core and the predicted load of the software elements to be allocated. There are various ways to balance the load among multiple cores. In some embodiments, the higher processing speed a core has, the larger value of execution load is determined for the

core. In some embodiments, the more cache is available, the larger value of execution load is determined for the core. In some embodiments, the core with the minimum predicted load has the largest value. It shall be understood that other characteristic and/or combinations of characteristics can be used to determine the value of execution load for each of the cores. The execution load analytics circuit 306 may communicate the value of execution load for each of the multiple cores to the core determination circuit 303.

[0033] The core determination circuit 303, upon obtaining the value of order dependencies from the element order dependencies analytics circuit 304, the value of data dependencies from the element data dependencies analytics circuit 305, and the value of execution load from the execution load analytics circuit 306, determines a most suitable core to which the software element or a group of software elements with hard dependencies is allocated. In some embodiments, the values of order dependencies, data dependencies, and execution load are added with a corresponding weight given to each other. The core with the highest weighted sum for the software element is the most suitable core where the software is assigned for execution. Different sets of weights can be given depending on requirements of the real time control system and characteristic of the multi-core processor. For example, if the multi-core processor on which the real time control system runs is known to take extended time to fetch data from memory, the data dependencies weight can be increased. Since each software element is allocated to execute on the most suitable core, data contention can be minimized, throughput can be improved, and core utilization can be balanced for the real time control system.

[0034] Referring now to FIG. 4, a flow diagram of a method of splitting software elements across multiple execution cores for a control system is shown, according to an example embodiment. The method 400 may be implemented with the execution manager 300 and in the control system 100.

[0035] At process 401, user override is determined for each of a plurality of software elements. Each software element has an associated parameter core override which specifies whether the software element is “hard coded” to a particular core to “override” any later

allocation by the execution manager 300. The user can set up the core override parameter of the software element to assign the software element to execute on the particular core and the assignment will not be changed by the real time control system. The core override parameter can be read to determine any user override for the software elements. If the software is assigned to a particular core, the real time control system will so allocate.

[0036] At process 402, software elements are grouped as one element based on dependencies upon each other. The dependencies can be determined based on the order dependencies, the data dependencies, or any combination of order and data dependencies. In some embodiments, the software elements are grouped as one if the software elements are strictly restricted to execute in a particular order. The information of order dependencies may be obtained by analyzing the parameter of element order dependencies associated with the software elements. In some embodiments, the software elements are grouped as one if the number of global data and/or physical resources accessed by the software elements and/or the frequency of accessing exceeds a predefined threshold number and/or a predefined threshold frequency. First, the parameter of element type associated with each software element is analyzed to determine whether the data dependencies analysis is needed. As discussed above, a standard service and a concurrent service may access global data and/or physical resources during execution, but a reentrant service and a utility service do not access global data or physical resources during execution. For element types that indicate global data and/or physical resources would be accessed, the parameter of global data writes and global data reads associated with the software elements may be further analyzed to determine overlap of the global data accessed and thus whether hard dependencies exist. In some embodiments, the software elements are grouped as one if both direct order dependencies and certain extent of data dependencies exist. It should be understood that the examples of dependencies analysis given here are not for limitation; various algorithms can be used to determine dependencies of software elements. The software elements having close dependencies upon each other are then grouped as one element and will be allocate to execute on the same core.

[0037] At process 403, a core is determined to which a software element or a group of software elements is allocated. In some embodiments, the determination is made based upon

multiple factors, such as order dependencies, data dependencies, and execution load. In some embodiments, the process 403 further includes processes 404, 405, and 406.

[0038] At process 404, the order dependencies of a software element (or a group of software elements) to be allocated upon software elements that have been allocated to execute on each of the cores are evaluated. The parameter of element order dependencies associated with the software element to be allocated can be obtained. If the parameter of element order dependencies indicates that the software element has order dependencies upon an allocated software element of a core, a value of order dependencies for the core can increase. The more allocated software elements of a core indicated by the parameter of element order dependencies associated with the software element to be allocated, the more value of order dependencies are determined for the core. In some embodiments, different values are determined for different element types. For example, if the software element to be allocated is a standard service, which cannot be called concurrently from multiple cores, and the software element that has been allocated to a core is a standard service, the order dependencies of these two software elements add the most to the value of order dependencies. If the software element to be allocated is a concurrent service, which can be called concurrently from multiple cores, the software element that has been allocated to a core is a standard service, the order dependencies of these two software elements add the second most to the value of order dependencies. If the software element to be allocated is a concurrent service, and the software element that has been allocated to a core is a concurrent service, the order dependencies of these two software elements adds the least to the value of order dependencies.

[0039] At process 405, the data dependencies of a software element (or a group of software elements) to be allocated upon software elements that have been allocated to execute on each of the cores are evaluated. The parameter of element type associated with the software element can be first analyzed to determine whether the data dependencies analysis is needed. As discussed above, a standard service and a concurrent service may access global data and/or physical resources during execution, but a reentrant service and a utility service do not access global data or physical resources during execution. If the software element is a standard service or a concurrent service, data dependencies for the software element may be further

analyzed. The parameters of global data writes and/or global data reads associated with the software element to be allocated may be obtained. A value of data dependencies for each core is determined based on the overlap of global data accessed by the software element to be allocated with global data accessed by the software elements that have been allocated to the core. In some embodiments, the more overlapped global data accessed by the software element to be allocated and the software elements that have been allocated to the core, the larger value of data dependencies is determined for the core. In some embodiments, the more frequently the overlapped global data is accessed by the software element to be allocated and the software elements that have been allocated to the core, the larger value of data dependencies is determined for the core. In some embodiments, different values of data dependencies are determined for different element types. For example, if the software element to be allocated is a standard service, which cannot be called concurrently from multiple cores, and the software element that has been allocated to a core is a standard service, the overlap of global data accessed by these two software elements adds the most to the value of data dependencies. If the software element to be allocated is a concurrent service, which can be called concurrently from multiple cores, the software element that has been allocated to a core is a standard service, the overlap of global data accessed by these two software elements adds the second most to the value of data dependencies. If the software element to be allocated is a concurrent service, and the software element that has been allocated to a core is a concurrent service, the overlap of global data accessed by these two software elements adds the least to the value of data dependencies.

[0040] At process 406, execution load of a software element (or a group of software elements) to be allocated and the load availability of each of the cores are evaluated. Information of the load availability of each of the cores may be obtained, including for example, capacity of each core, clock speed, processing power, load of the software elements that have been assigned to each core, etc. Information of the execution load of the software element to be allocated based on parameters of number of instructions or execution time for minimum case, number of instructions or execution time for maximum case, periodic rate of execution, etc. These parameters may be used to estimate the predicted minimal load and

maximum load of each core. Then a value of execution load for each core is determined based on the load availability of the core and the predicted load of the software elements to be allocated. There are various ways to balance the load among multiple cores. In some embodiments, the higher processing speed a core has, the larger value of execution load is determined for the core. In some embodiments, the more cache is available, the larger value of execution load is determined for the core. In some embodiments, the core with the minimum predicted load has the largest value.

[0041] A most suitable core is determined to which the software element or a group of software elements with hard dependencies is allocated based on the order dependencies, data dependencies, and execution load. In some embodiments, the values of order dependencies, data dependencies, and execution load are added with a corresponding weight given to each other. The core with the highest weighted sum for the software element is the most suitable core where the software is assigned for execution. Different sets of weights can be given depending on the requirement of the real time control system and characteristic of the multi-core processor. For example, if the multi-core processor on which the real time control system runs is known to take extended time to fetch data from memory, the data dependencies weight can be increased.

[0042] It should be understood that no claim element herein is to be construed under the provisions of 35 U.S.C. § 112(f), unless the element is expressly recited using the phrase "means for." The schematic flow chart diagrams and method schematic diagrams described above are generally set forth as logical flow chart diagrams. As such, the depicted order and labeled steps are indicative of representative embodiments. Other steps, orderings and methods may be conceived that are equivalent in function, logic, or effect to one or more steps, or portions thereof, of the methods illustrated in the schematic diagrams. Further, reference throughout this specification to "one embodiment", "an embodiment", "an example embodiment", or similar language means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, appearances of the phrases "in one embodiment", "in an

embodiment", "in an example embodiment", and similar language throughout this specification may, but do not necessarily, all refer to the same embodiment.

[0043] Additionally, the format and symbols employed are provided to explain the logical steps of the schematic diagrams and are understood not to limit the scope of the methods illustrated by the diagrams. Although various arrow types and line types may be employed in the schematic diagrams, they are understood not to limit the scope of the corresponding methods. Indeed, some arrows or other connectors may be used to indicate only the logical flow of a method. For instance, an arrow may indicate a waiting or monitoring period of unspecified duration between enumerated steps of a depicted method. Additionally, the order in which a particular method occurs may or may not strictly adhere to the order of the corresponding steps shown. It will also be noted that each block of the block diagrams and/or flowchart diagrams, and combinations of blocks in the block diagrams and/or flowchart diagrams, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and program code.

[0044] Many of the functional units described in this specification have been labeled as circuits, in order to more particularly emphasize their implementation independence. For example, a circuit may be implemented as a hardware circuit comprising custom very-large-scale integration (VLSI) circuits or gate arrays, off-the-shelf semiconductors such as logic chips, transistors, or other discrete components. A circuit may also be implemented in programmable hardware devices such as field programmable gate arrays, programmable array logic, programmable logic devices or the like.

[0045] As mentioned above, circuits may also be implemented in machine-readable medium for execution by various types of processors. An identified circuit of executable code may, for instance, comprise one or more physical or logical blocks of computer instructions, which may, for instance, be organized as an object, procedure, or function. Nevertheless, the executables of an identified circuit need not be physically located together, but may comprise disparate instructions stored in different locations which, when joined logically together, comprise the circuit and achieve the stated purpose for the circuit. Indeed, a circuit of computer readable

program code may be a single instruction, or many instructions, and may even be distributed over several different code segments, among different programs, and across several memory devices. Similarly, operational data may be identified and illustrated herein within circuits, and may be embodied in any suitable form and organized within any suitable type of data structure. The operational data may be collected as a single data set, or may be distributed over different locations including over different storage devices, and may exist, at least partially, merely as electronic signals on a system or network.

[0046] The computer readable medium (also referred to herein as machine-readable media or machine-readable content) may be a tangible computer readable storage medium storing the computer readable program code. The computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, holographic, micromechanical, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. As alluded to above, examples of the computer readable storage medium may include but are not limited to a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a portable compact disc read-only memory (CD-ROM), a digital versatile disc (DVD), an optical storage device, a magnetic storage device, a holographic storage medium, a micromechanical storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, and/or store computer readable program code for use by and/or in connection with an instruction execution system, apparatus, or device.

[0047] Computer readable program code for carrying out operations for aspects of the present invention may be written in any combination of one or more programming languages, including an object oriented programming language such as Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the "C" programming language or similar programming languages.

[0048] The program code may also be stored in a computer readable medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a

particular manner, such that the instructions stored in the computer readable medium produce an article of manufacture including instructions which implement the function/act specified in the schematic flowchart diagrams and/or schematic block diagrams block or blocks.

[0049] Accordingly, the present disclosure may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the disclosure is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is Claimed is:

1. An apparatus for splitting a plurality of software elements across multiple cores of a multi-core processor, the apparatus comprising:

a user override analytics circuit structured to determine user override for each of the plurality of software elements;

a dependencies analytics circuit structured to group more than one software elements of the plurality of software elements as a group based on dependencies of the group of software elements upon each other; and

a core determination circuit structured to statically determine a core from the multiple cores of the multi-core processor for each software element or group of software elements to execute on.

2. The apparatus of claim 1, wherein the core determination circuit further comprises an element order dependencies analytics circuit structured to evaluate each of the multiple cores based on order dependencies of each software element or group of software elements upon each of the multiple cores.

3. The apparatus of claim 2, wherein the element order dependencies analytics circuit is further structured to determine a value of order dependencies for each of the multiple cores based on the evaluation.

4. The apparatus of claim 1, wherein the core determination circuit further comprises an element data dependencies analytics circuit structured to evaluate each of the multiple cores based on data dependencies of each software element or group of software elements upon each of the multiple cores.

5. The apparatus of claim 4, wherein the element data dependencies analytics circuit is further structured to determine a value of data dependencies for each of the multiple cores based on the evaluation.

6. The apparatus of claim 1, wherein the core determination circuit further comprises an execution load analytics circuit structured to evaluate each of the multiple cores based on execution load of each software element or group of software elements and execution load availability of each of the multiple cores.

7. The apparatus of claim 6, wherein the element data dependencies analytics circuit is further structured to determine a value of execution load for each of the multiple cores based on the evaluation.

8. The apparatus of claim 1, wherein each software elements is one of a task which triggers a number of runnables mapped to the task, a runnable which is mapped to a task and executed in response to the task being triggered, a standard service which reads or writes global data or physical resources and cannot be called to execute concurrently from more than one cores, a concurrent service which accesses global data and can be called to execute concurrently from more than one cores, a reentrant service which does not access global data or physical resources, or a utility service refers which is implemented in a reusable utility component.

9. The apparatus of claim 1, wherein the core determined for each software element or group of software elements to execute on is the core with the highest weighed sum of multiple factors for the software element or group of software elements.

10. The apparatus of claim 1, wherein the multiple factors includes element order dependencies, element data dependencies, and execution load.

11. The apparatus of claim 1, wherein the core determination circuit is further structured to assign each software element or group of software element to execute on the determined core.

12. A method for splitting a plurality of software elements across multiple cores of a multi-core processor, comprising:

determining, by an execution manager, user override for each of the plurality of software elements;

grouping, by the execution manager, more than one software elements of the plurality of software elements as a group based on dependencies of the group of software elements upon each other; and

determining statically, by the execution manager, a core from the multiple cores of the multi-core processor for each software element or group of software elements to execute on.

13. The method of claim 12, further comprising determining a value of order dependencies for each of the multiple cores indicative of order dependencies of each software element or group of software elements upon each of the multiple cores.

14. The method of claim 12, further comprising determining a value of data dependencies for each of the multiple cores indicative of data dependencies of each software element or group of software elements upon each of the multiple cores.

15. The method of claim 12, further comprising determining a value of execution load for each of the multiple cores indicative of execution load availability of each of the multiple cores with respect to execution load of each software element or group of software elements.

16. The method of claim 12, wherein the core determined for each software element or group of software elements to execute on is the core with the highest weighed sum of multiple factors for the software element or group of software elements.

17. The method of claim 16, wherein the multiple factors includes element order dependencies, element data dependencies, and execution load.

18. The method of claim 12, further comprising assigning each software element or group of software element to execute on the determined core.

19. A control system, comprising:

a multi-core processor including multiple cores;

a memory coupled to the multi-core processor and including a plurality of software elements; and

an execution manager coupled to the multi-core processor and the memory, the execution manager structured to:

determine user override for each of the plurality of software elements;

group more than one software elements of the plurality of software elements as a group based on dependencies of the group of software elements upon each other; and

statically determine a core from the multiple cores of the multi-core processor for each software element or group of software elements to execute on.

20. The real time control system of claim 19, wherein the execution manager is further structured to determine a value of order dependencies for each of the multiple cores indicative of order dependencies of each software element or group of software elements upon each of the multiple cores.

21. The real time control system of claim 19, wherein the execution manager is further structured to determine a value of data dependencies for each of the multiple cores indicative of data dependencies of each software element or group of software elements upon each of the multiple cores.

22. The real time control system of claim 19, wherein the execution manager is further structured to determine a value of execution load for each of the multiple cores indicative of execution load availability of each of the multiple cores with respect to execution load of each software element or group of software elements.

23. The real time control system of claim 19, wherein the core determined for each software element or group of software elements to execute on is the core with the highest weighed sum of multiple factors for the software element or group of software elements.

24. The real time control system of claim 23, wherein the multiple factors includes element order dependencies, element data dependencies, and execution load.

25. The real time control system of claim 19, further comprising assigning each software element or group of software element to execute on the determined core.

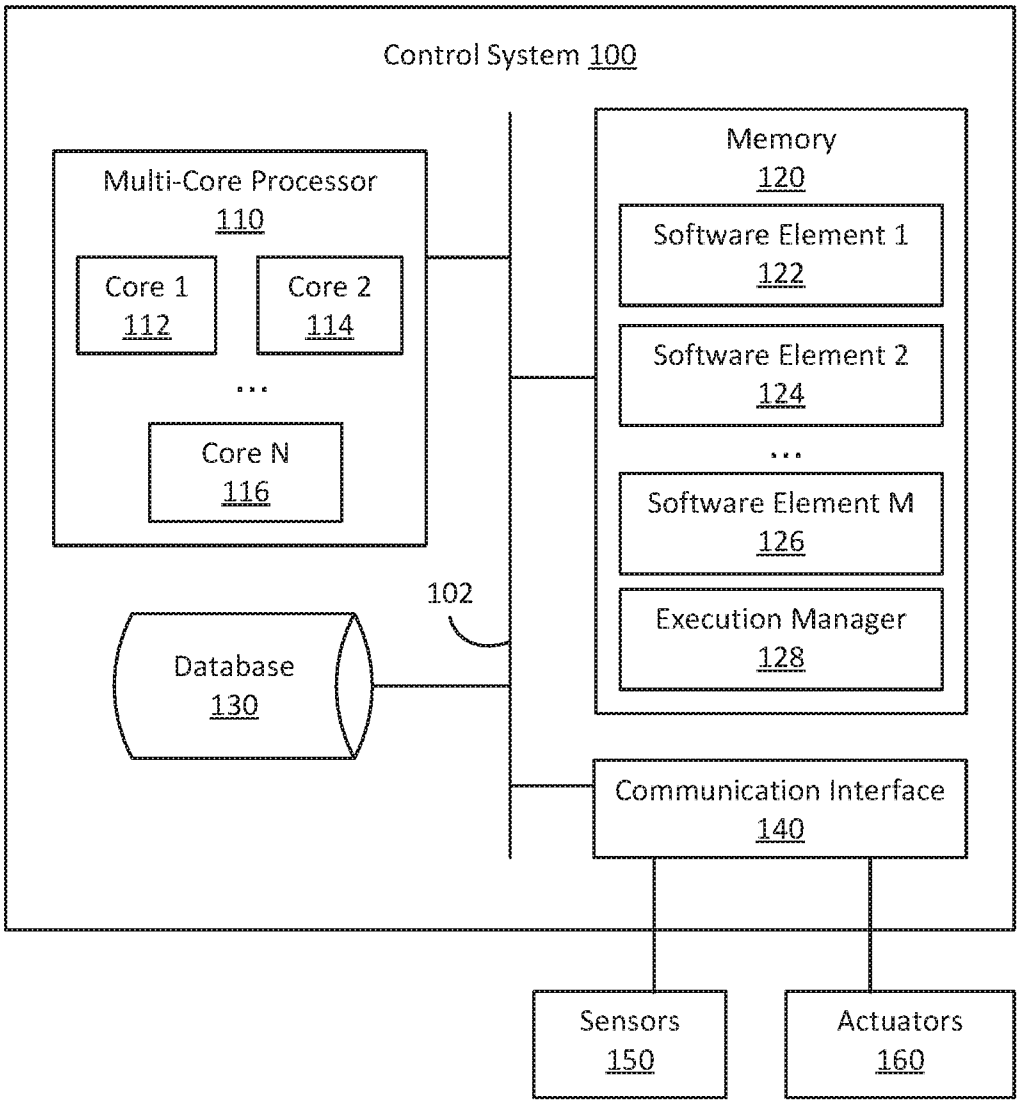
Control System 100
Multi-Core Processor
110
Core 1
112
Core 2
114
...
Core N
116
Memory
120
Software Element 1
122
Software Element 2
124
...
Software Element M
126
Execution Manager
128
102
Database
130
Communication Interface
140
Sensors
150
Actuators
160

FIG. 1

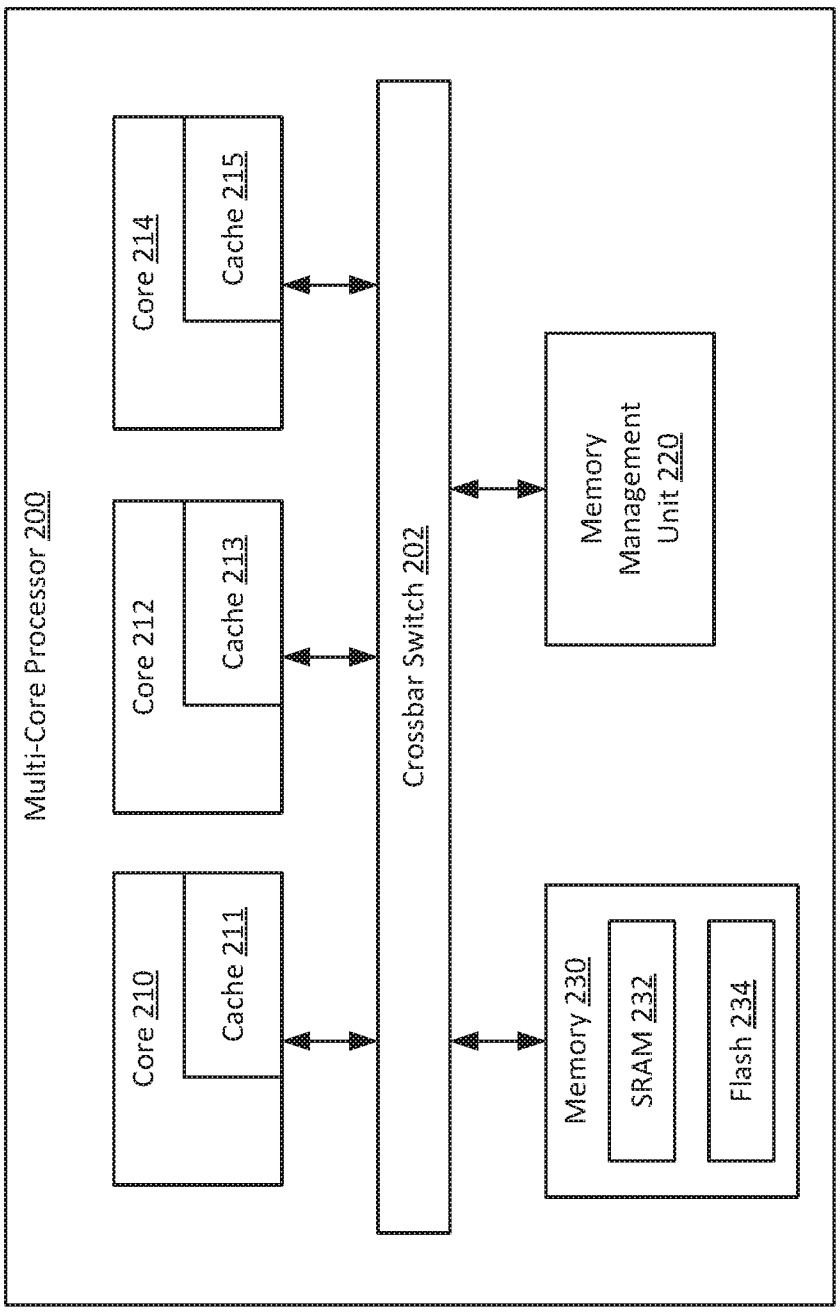
Multi-Core Processor 200
Core 210
Cache 211
Core 212
Cache 213
Core 214
Cache 215
Crossbar Switch 202
Memory Management Unit 220
Memory 230
SRAM 232
Flash 234

FIG. 2

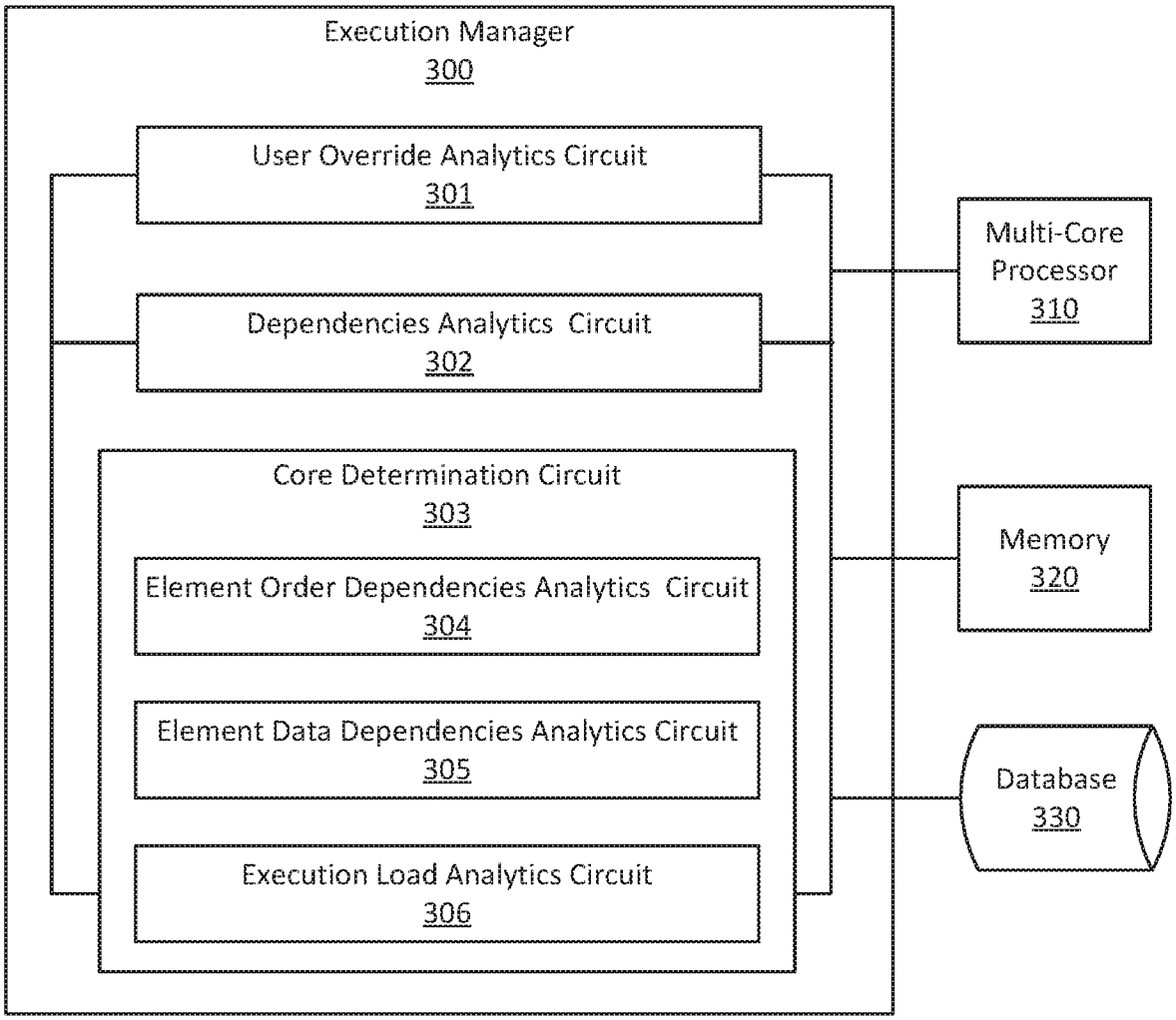
Execution Manager
300
User Override Analytics Circuit
301
Dependencies Analytics Circuit
302
Core Determination Circuit
303
Element Order Dependencies Analytics Circuit
304
Element Data Dependencies Analytics Circuit
305
Execution Load Analytics Circuit
306
Multi-Core Processor
310
Memory
320
Database
330

FIG. 3

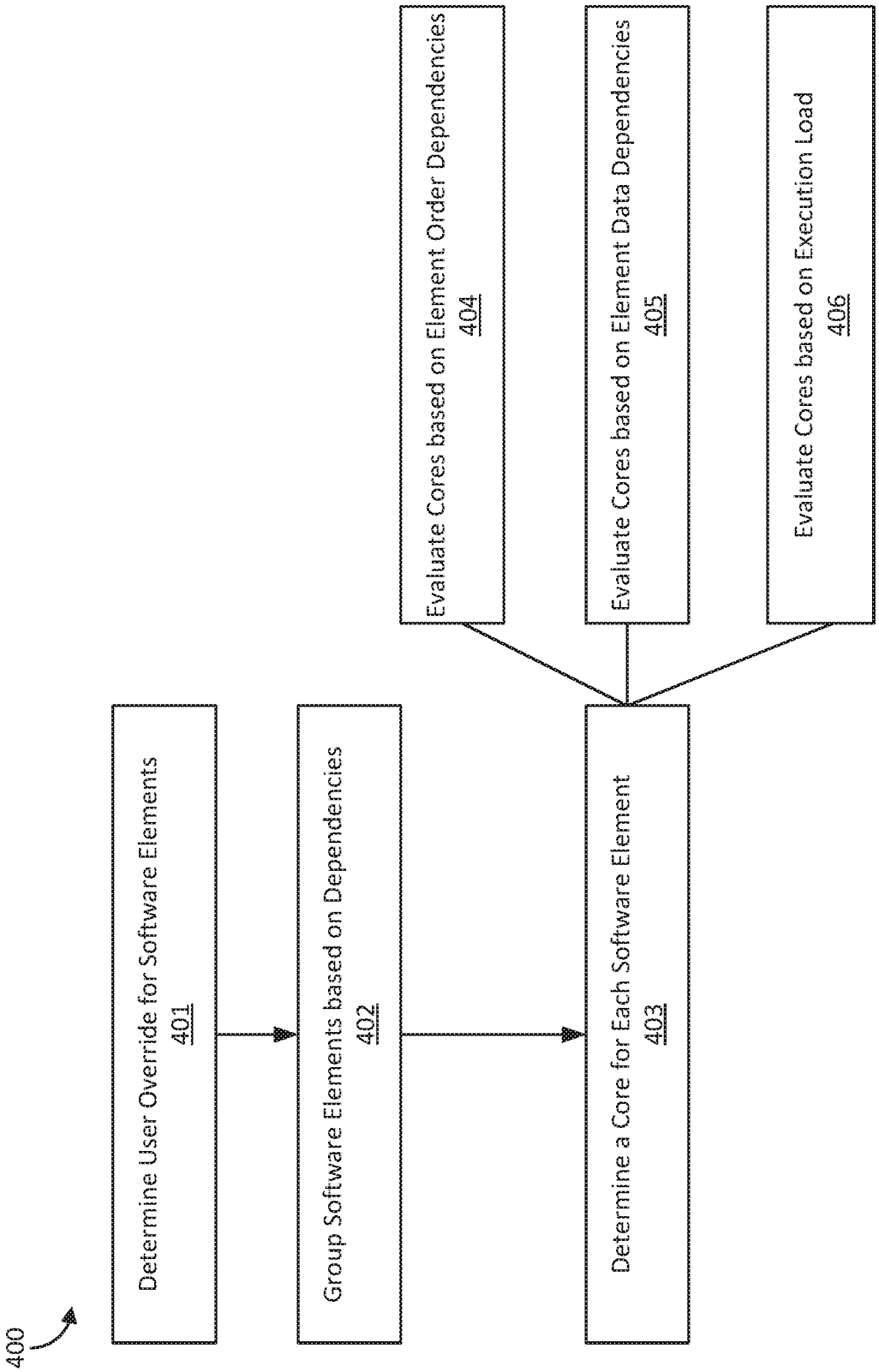
400
Determine User Override for Software Elements
401
Group Software Elements based on Dependencies
402
Determine a Core for Each Software Element
403
Evaluate Cores based on Element Order Dependencies
404
Evaluate Cores based on Element Data Dependencies
405
Evaluate Cores based on Execution Load
406

FIG. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 16/60339

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 9/46 (2016.01)
CPC - G06F 9/52
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 9/46 (2016.01)
CPC: G06F 9/52

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 718/100 or 718/104
IPC(8): G06F 9/46 (2016.01); CPC: G06F 9/52, G06F 9/5016, G06F 9/5077, G06F 9/505, G06F 9/5011

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Patbase; Google Scholar; core, multi-core, load, override, task, software element, execute

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US 2011/0078652 A1 (Mani et al.) 31 March 2011 (31.03.2011) entire document (especially para [0066]-[0069], [0074], [0079]-[0081], [0097]-[0098], [0103], [0106], [0133], [0161]-[0162])	1-5, 8-9, 11-14, 16, 18-21, 23, and 25 --- 6-7, 10, 15, 17, 22, 24,
Y	US 2013/0339977 A1 (Dennis et al.) 19 December 2013 (19.12.2013) (para [0014], [0023]-[0024])	6-7, 10, 15, 17, 22, 24,

☐ Further documents are listed in the continuation of Box C. ☐

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

Date of the actual completion of the international search	Date of mailing of the international search report
29 December 2016 (29.12.2016)	23 JAN 2017

Name and mailing address of the ISA/US	Authorized officer:
Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-8300	Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

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