US009958932B2

(12) **United States Patent**
Williamson et al.

(10) **Patent No.: US 9,958,932 B2**
(45) **Date of Patent: May 1, 2018**

(54) **PROCESSOR INCLUDING MULTIPLE DISSIMILAR PROCESSOR CORES THAT IMPLEMENT DIFFERENT PORTIONS OF INSTRUCTION SET ARCHITECTURE**

(71) Applicant: **Apple Inc.**, Cupertino, CA (US)

(72) Inventors: **David J. Williamson**, Austin, TX (US); **Gerard R. Williams, III**, Los Altos, CA (US); **James N. Hardage, Jr.**, Austin, TX (US); **Richard F. Russo**, San Jose, CA (US)

(73) Assignee: **Apple Inc.**, Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 217 days.

(21) Appl. No.: **14/548,912**

(22) Filed: **Nov. 20, 2014**

(65) **Prior Publication Data**

US 2016/0147290 A1 May 26, 2016

(51) **Int. Cl.**
G06F 1/32 (2006.01)
G06F 9/50 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 1/3293*** (2013.01); ***G06F 1/3275*** (2013.01); ***G06F 1/3287*** (2013.01); ***G06F 9/5094*** (2013.01); *Y02B 60/121* (2013.01); *Y02B 60/1282* (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,281,055 B2 10/2007 Glasco
7,409,506 B2 8/2008 Kamigata
7,624,250 B2 11/2009 Lau et al.
8,301,932 B2 10/2012 Hay
(Continued)

FOREIGN PATENT DOCUMENTS

EP 2784674 10/2014
TW 201418972 A 5/2014
(Continued)

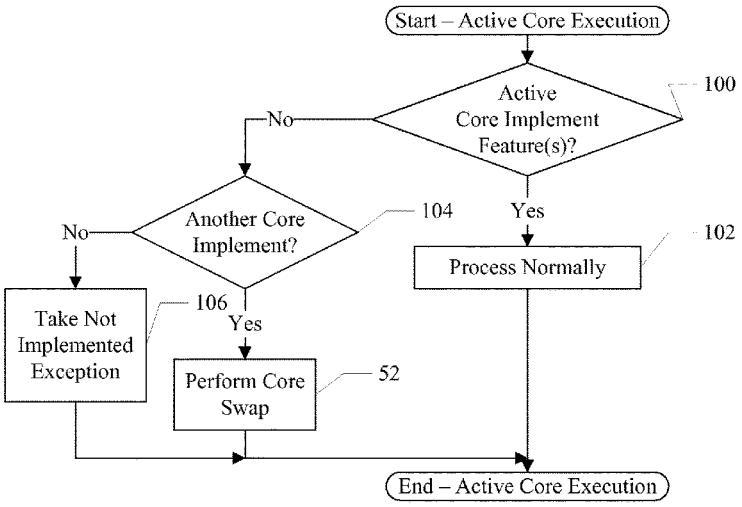
OTHER PUBLICATIONS

U.S. Appl. No. 14/844,212, filed Sep. 3, 2015, Hardage et al.
(Continued)

Primary Examiner — Zahid Choudhury
Assistant Examiner — Danny Chan
(74) *Attorney, Agent, or Firm* — Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.; Lawrence J. Merkel

(57) **ABSTRACT**

In an embodiment, an integrated circuit may include one or more processors. Each processor may include multiple processor cores, and each core has a different design/implementation and performance level. For example, a core may be implemented for high performance, and another core may be implemented at a lower maximum performance, but may be optimized for efficiency. Additionally, in some embodiments, some features of the instruction set architecture implemented by the processor may be implemented in only one of the cores that make up the processor. If such a feature is invoked by a code sequence while a different core is active, the processor may swap cores to the core the imple-
(Continued)



ments the feature. Alternatively, an exception may be taken and an exception handler may be executed to identify the feature and activate the corresponding core.

19 Claims, 8 Drawing Sheets

(56) **References Cited**

U.S. PATENT DOCUMENTS

8,533,505	B2	9/2013	Greenhalgh	
8,751,833	B2	6/2014	Hill	
8,793,686	B2	7/2014	Aasheim	
8,819,686	B2	8/2014	Memik et al.	
2006/0005200	A1*	1/2006	Vega	G06F 12/10 718/108
2008/0263324	A1*	10/2008	Sutardja	G06F 1/3203 712/43
2008/0276026	A1	11/2008	Branover	
2008/0307422	A1	12/2008	Kurland	
2009/0172713	A1*	7/2009	Kim	G06F 9/30145 719/331
2011/0213934	A1	9/2011	Greenhalgh	
2011/0213935	A1	9/2011	Greenhalgh	
2011/0265090	A1*	10/2011	Moyer	G06F 1/3203 718/103
2011/0271126	A1	11/2011	Hill	
2012/0233477	A1	9/2012	Wu	
2012/0254877	A1	10/2012	Comparan	
2013/0061237	A1*	3/2013	Zaarur	G06F 9/4856 718/105
2013/0268742	A1*	10/2013	Yamada	G06F 9/5044 712/226
2013/0346058	A1*	12/2013	Beckmann	G06F 9/4552 703/22
2014/0019723	A1*	1/2014	Yamada	G06F 1/3293 712/208
2014/0129808	A1	5/2014	Naveh et al.	
2014/0173311	A1	6/2014	Park	
2014/0196050	A1*	7/2014	Yu	G06F 9/5088 718/104
2014/0298060	A1*	10/2014	Hooker	G06F 1/3293 713/323
2015/0007196	A1	1/2015	Toll	
2015/0058650	A1	2/2015	Varma	
2015/0121105	A1	4/2015	Ahn et al.	
2015/0277867	A1*	10/2015	Hasabnis	G06F 8/433 717/164
2015/0281336	A1*	10/2015	Beale	G06F 9/4856 709/201
2015/0378417	A1*	12/2015	Lim	G06F 1/324 713/320
2016/0154649	A1	6/2016	Lin	

FOREIGN PATENT DOCUMENTS

TW	201437912 A	10/2014
WO	2009-029643	3/2009
WO	2013/036222	3/2013
WO	2013/100996	7/2013
WO	2013/101069	7/2013

OTHER PUBLICATIONS

"Emulating Asymmetric MPSoCs on the Intel SCC Many-core Processor", Roy Bakker, et al., Feb. 2014, pp. 1-8.

"Benefits of the big. LITTLE Architecutre", Hyun-Duk Cho, et al., Feb. 2012, Samsung Electronics, pp. 1-8.

"big.LITTLE Processing", ARM, accessed Jan. 14, 2014, pp. 1-4.

"Reflex: Using Low-Power Processors in Smartphones without Knowing Them", Felix Xiaozhu Lin, et al., Mar. 2012, pp. 1-12.

"Variable SMP—A Multi-Core CPU Architecture for Low Power and High Performance", nvidia, 2011, pp. 1-16.

"A HW/SW Co-designed Heterogeneous Multi-core Virtual Machine for Energy-Efficient General Purpose Computing", Youfeng Wu, et al., Apr. 2011, pp. 1-10.

"Distributed Thermal Management for Embedded Heterogeneous MPSoCs with Dedicated Hardware Accelerators", Yen-Kuan Wu, et al., Oct. 2011, pp. 1-7.

U.S. Appl. No. 14/548,872, filed Nov. 20, 2014, David J. Williamson.

International Search Report and Written Opinion in application No. PCT/US2015/054998 dated Feb. 2, 2016, 14 pages.

Office Action, ROC (Taiwan) Patent Application No. 104135132, dated Aug. 24, 2016, 8 pages.

Search Report, ROC Taiwan) Patent Application No. 104135132, dated Aug. 24, 2016, 1 page.

Office Action, U.S. Appl. No. 14/548,872, dated Dec. 1, 2016, 20 pages.

IPRP, Written Opinion, PCT/US2015/054998, dated Jun. 1, 2017, 10 pages.

Office Action, U.S. Appl. No. 14/844,212, dated Jun. 2, 2017, 18 pages.

* cited by examiner

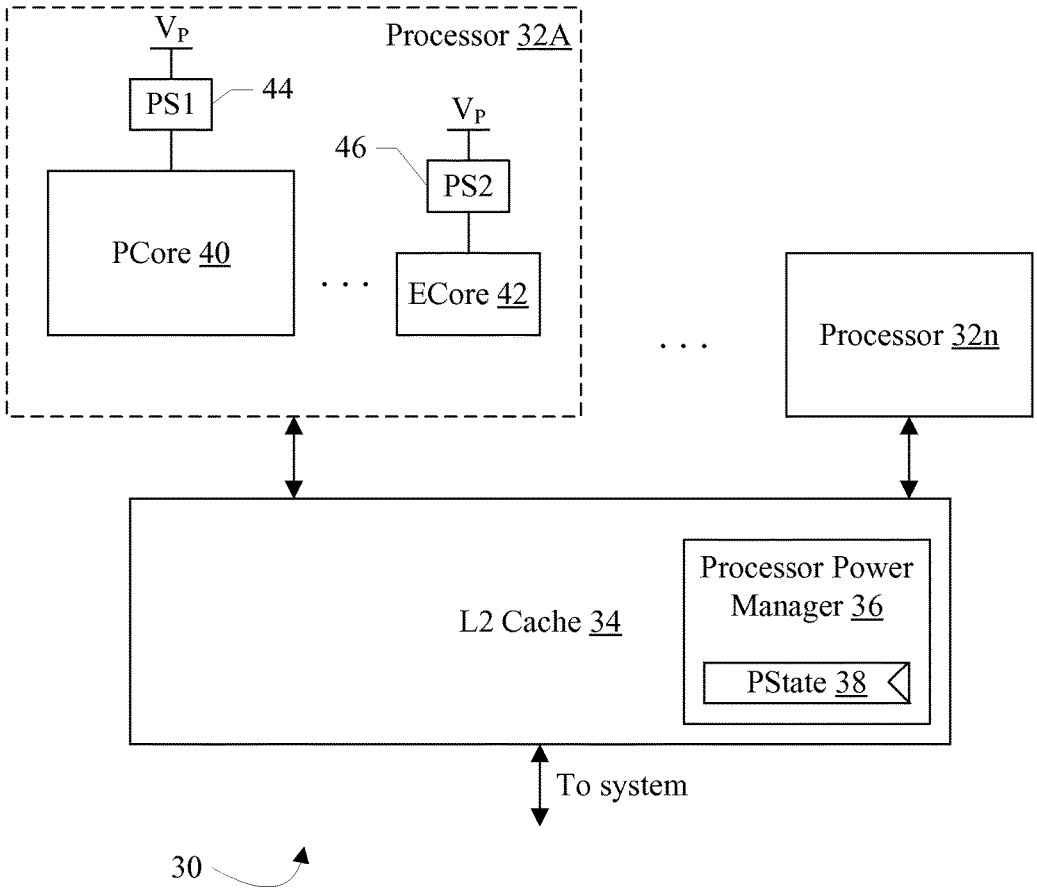

V_P
Processor 32A
PS1
44
V_P
46
PS2
PCore 40
ECore 42
Processor 32n
L2 Cache 34
Processor Power Manager 36
PState 38
To system
30

Fig. 1

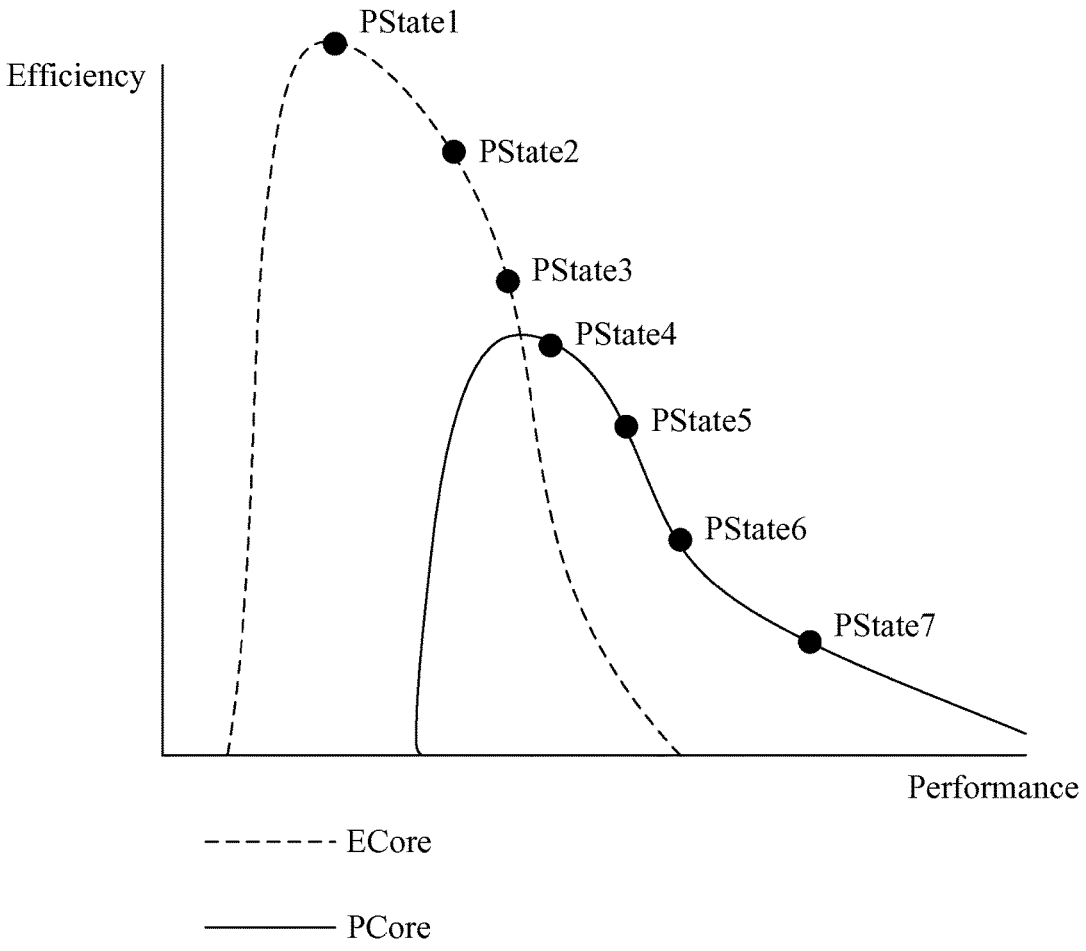

PState1
Efficiency
PState2
PState3
PState4
PState5
PState6
PState7
Performance
ECore
PCore

Fig. 2

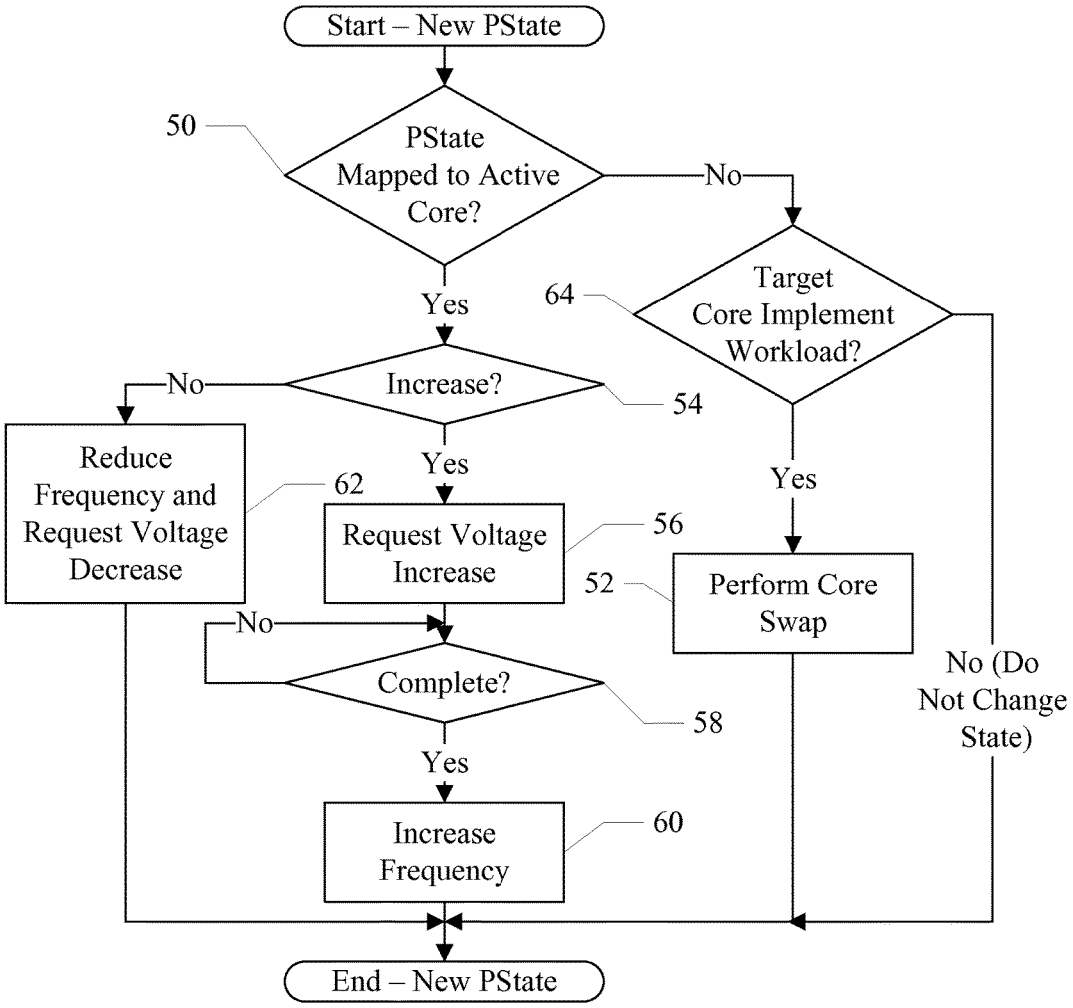

Start – New PState
50
PState Mapped to Active Core?
No
Yes
64
Target Core Implement Workload?
No
Increase?
54
Yes
Yes
Reduce Frequency and Request Voltage Decrease
62
Request Voltage Increase
56
52
Perform Core Swap
No
Complete?
58
Yes
No (Do Not Change State)
Increase Frequency
60
End – New PState

Fig. 3

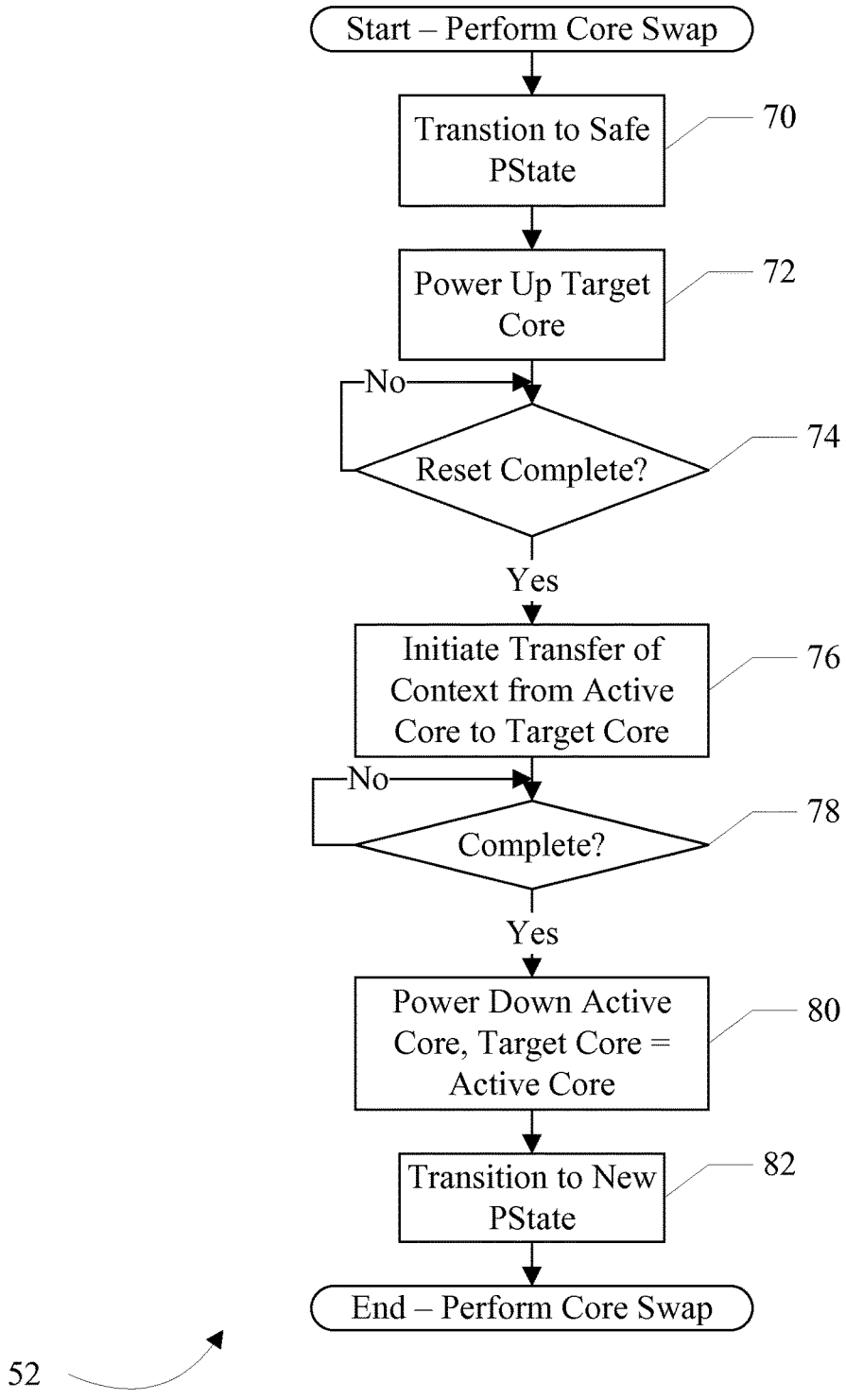

Start – Perform Core Swap
Transtion to Safe PState
70
Power Up Target Core
72
No
Reset Complete?
74
Yes
Initiate Transfer of Context from Active Core to Target Core
76
No
Complete?
78
Yes
Power Down Active Core, Target Core = Active Core
80
Transition to New PState
82
End – Perform Core Swap
52

Fig. 4

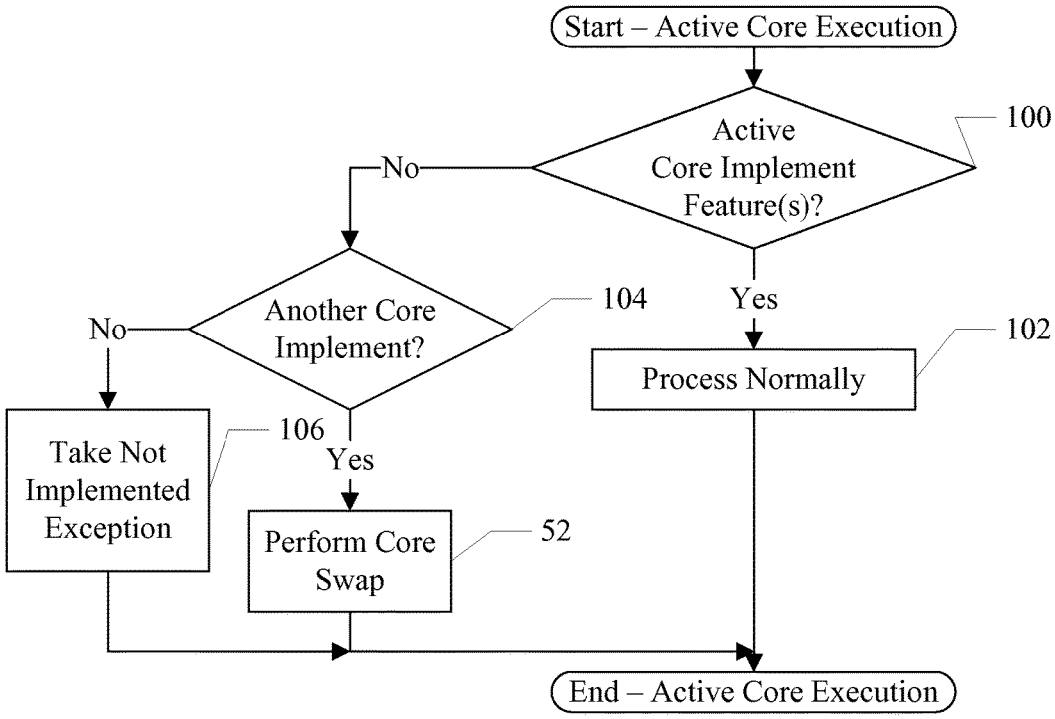

Start – Active Core Execution
Active Core Implement Feature(s)?
100
No
Yes
Another Core Implement?
104
No
Yes
Process Normally
102
Take Not Implemented Exception
106
Perform Core Swap
52
End – Active Core Execution

Fig. 5

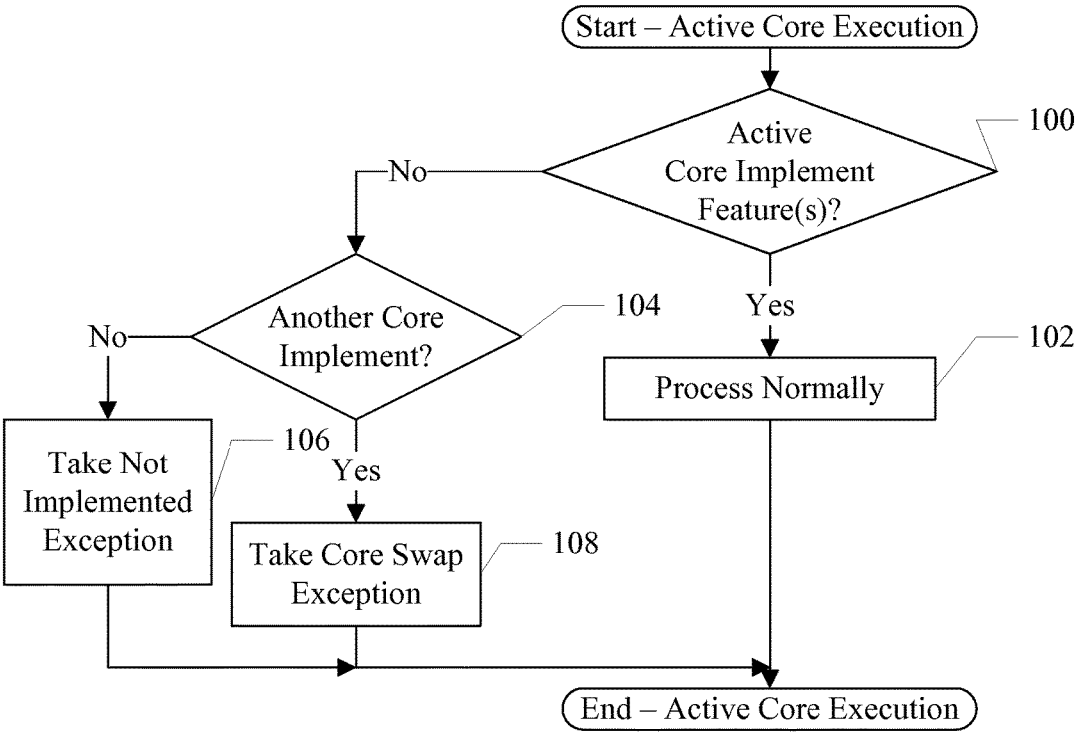

Start – Active Core Execution
Active Core Implement Feature(s)?
100
No
Yes
Another Core Implement?
104
No
Yes
Process Normally
102
Take Not Implemented Exception
106
Take Core Swap Exception
108
End – Active Core Execution

Fig. 6

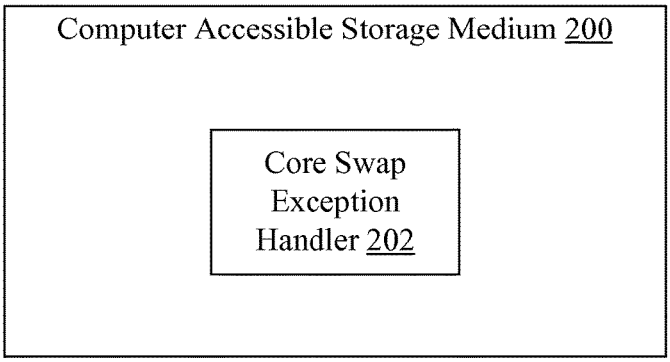

Computer Accessible Storage Medium 200
Core Swap Exception Handler 202

Fig. 7

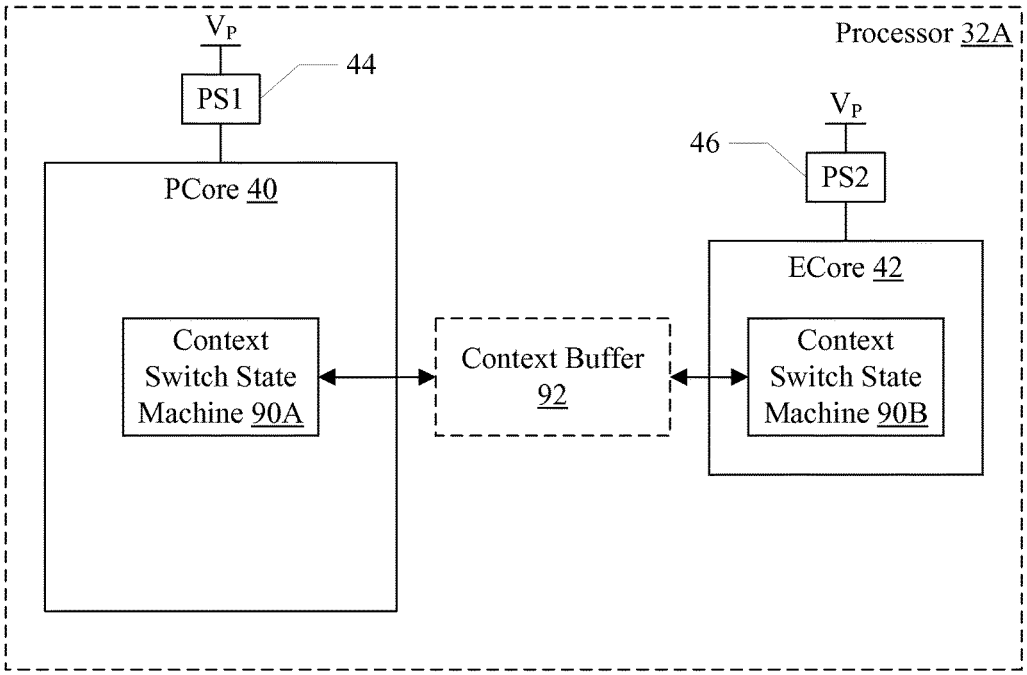

V_P
44
PS1
PCore 40
Context Switch State Machine 90A
Context Buffer 92
Processor 32A
V_P
46
PS2
ECore 42
Context Switch State Machine 90B

Fig. 8

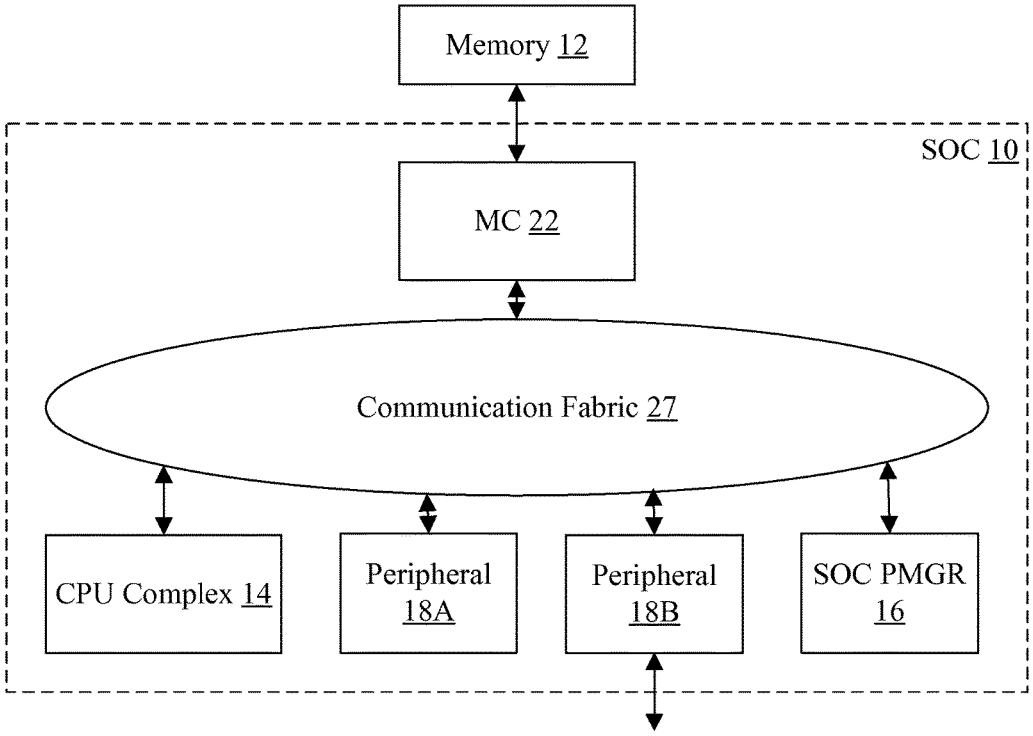

Memory 12
SOC 10
MC 22
Communication Fabric 27
CPU Complex 14
Peripheral 18A
Peripheral 18B
SOC PMGR 16

Fig. 9

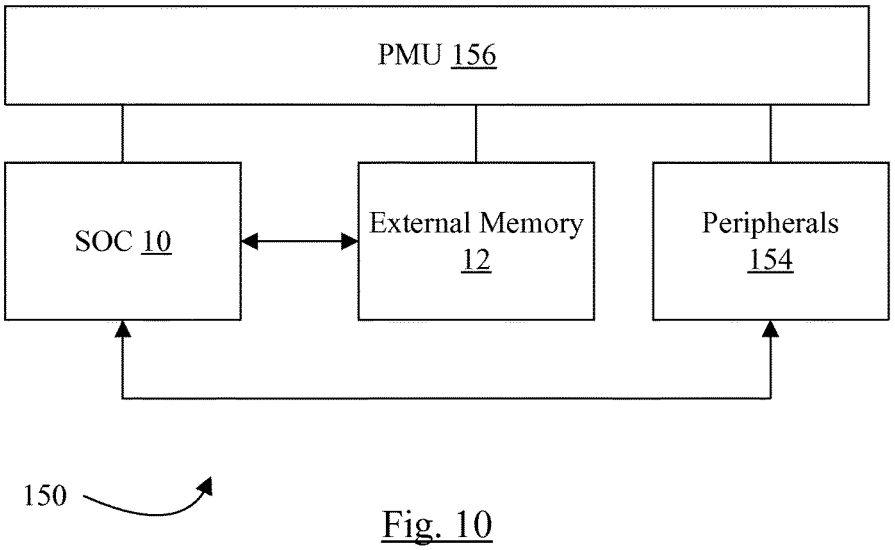

PMU 156
SOC 10
External Memory 12
Peripherals 154
150

Fig. 10

PROCESSOR INCLUDING MULTIPLE DISSIMILAR PROCESSOR CORES THAT IMPLEMENT DIFFERENT PORTIONS OF INSTRUCTION SET ARCHITECTURE

BACKGROUND

Technical Field

Embodiments described herein are related to processors and, more particularly, to multiple processor cores forming a processor.

Description of the Related Art

Various processors are included in electronic systems to execute software providing some amount of user functionality. The processors may include the central processing units (CPUs) in the system, as well as special purpose processors dedicated to specific tasks such as graphics, media processing, etc. Generally, the processors are designed to operate at multiple operating points (settings of supply voltage magnitude and clock frequency). Lower operating points consume less power but also offer limited performance compared to higher operating points. For some workloads, the limited performance is sufficient and the lower operating points can be used. For other workloads, the higher operating points are needed to provide sufficient performance.

In some systems, a wide diversity of workloads are experienced. Designing a processor that can provide the performance needed by the most demanding workloads while also supporting the lowest possible operating point that would provide sufficient performance for many frequently-executed workloads has become a challenge. Processors that operate at high operating points may only support a reduction in supply voltage to a certain level before circuitry ceases to function correctly. Compromises must be made, and typically the lowest operating point is increased until the design can meet the desired high end operating point. As the high end operating points continue to increase, more and more workloads are executable at the lowest operating point (and many could be executed at even lower operating points). Power is expended unnecessarily for such workloads, which can be a critical factor in mobile systems that frequently operate on a limited energy source such as a battery.

SUMMARY

In an embodiment, an integrated circuit may include one or more processors. Each processor may include multiple processor cores, and each core has a different design/implementation and performance level. For example, a core may be implemented for high performance, but may have higher minimum voltage at which it operates correctly. Another core may be implemented at a lower maximum performance, but may be optimized for efficiency and may operate correctly at a lower minimum voltage. Additionally, in some embodiments, some features of the instruction set architecture employed by the processor may be implemented in only one of the cores that make up the processor (or may be implemented by a subset of the cores that excludes at least one core). If such a feature is invoked by a code sequence while a different core is active, the processor may swap cores to one of the cores the implements the feature. Alternatively, an exception may be taken and an exception handler may be executed to identify the feature and activate the corresponding core.

In some embodiments, limiting certain features to one core or, at least, to less than all the cores may provide an area efficient implementation by eliminating duplicative circuitry in the cores to process the same instruction types. Features that will only likely be used in high performance code, for example, may be implemented only in the high performance core since that core is the most likely to execute the high performance code. Features which are unlikely to be used (e.g. features provided for backwards compatibility but which are not used by newer code) may be implemented in one core and thus may be supported efficiently from an area standpoint.

The processor may support multiple processor states (PStates). Each PState may specify an operating point (e.g. a combination of supply voltage magnitude and clock frequency), and each PState may be mapped to one of the processor cores. During operation, one of the cores is active: the core to which the current PState is mapped. If a new PState is selected and is mapped to a different core, the processor may automatically context switch the processor state to the newly-selected core and may begin execution on that core. In an embodiment, the processor may detect whether or not the newly-selected core supports the features in use by the current workload and may take corrective action if not supported.

BRIEF DESCRIPTION OF THE DRAWINGS

The following detailed description makes reference to the accompanying drawings, which are now briefly described.

FIG. **1** is a block diagram of one embodiment of a processor cluster.

FIG. **2** is a graph illustrating efficiency versus performance for a PCore and an ECore as illustrated in FIG. **1** for one embodiment.

FIG. **3** is a flowchart illustrating operation of one embodiment of a processor power management unit to change processor states.

FIG. **4** is a flowchart illustrating operation of one embodiment of the processor power management unit to swap cores.

FIG. **5** is a flowchart illustrating operation on one embodiment of an active core during execution of instructions.

FIG. **6** is a flowchart illustrating operation on another embodiment of an active core during execution of instructions.

FIG. **7** is a block diagram of a computer accessible storage medium.

FIG. **8** is a block diagram of one embodiment of context switching hardware for a core swap.

FIG. **9** is a block diagram of one embodiment of a system on a chip (SOC) including one embodiment of the processor cluster shown in FIG. **1**.

FIG. **10** is a block diagram of one embodiment of a system.

While embodiments described in this disclosure may be susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the embodiments to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the appended claims. The headings used herein are for organizational purposes only and are not meant to be used to limit the scope of the description. As used throughout this

application, the word "may" is used in a permissive sense (i.e., meaning having the potential to), rather than the mandatory sense (i.e., meaning must). Similarly, the words "include", "including", and "includes" mean including, but not limited to.

Various units, circuits, or other components may be described as "configured to" perform a task or tasks. In such contexts, "configured to" is a broad recitation of structure generally meaning "having circuitry that" performs the task or tasks during operation. As such, the unit/circuit/component can be configured to perform the task even when the unit/circuit/component is not currently on. In general, the circuitry that forms the structure corresponding to "configured to" may include hardware circuits and/or memory storing program instructions executable to implement the operation. The memory can include volatile memory such as static or dynamic random access memory and/or nonvolatile memory such as optical or magnetic disk storage, flash memory, programmable read-only memories, etc. Similarly, various units/circuits/components may be described as performing a task or tasks, for convenience in the description. Such descriptions should be interpreted as including the phrase "configured to." Reciting a unit/circuit/component that is configured to perform one or more tasks is expressly intended not to invoke 35 U.S.C. § 112(f) interpretation for that unit/circuit/component.

This specification includes references to "one embodiment" or "an embodiment." The appearances of the phrases "in one embodiment" or "in an embodiment" do not necessarily refer to the same embodiment, although embodiments that include any combination of the features are generally contemplated, unless expressly disclaimed herein. Particular features, structures, or characteristics may be combined in any suitable manner consistent with this disclosure.

DETAILED DESCRIPTION OF EMBODIMENTS

FIG. **1** is a block diagram of one embodiment of a processor cluster **30**. In the illustrated embodiment, multiple processors **32**A-**32***n* and a level 2 (L2) cache **34** are included. The processors **32**A-**32***n* are coupled to the L2 cache **34**, which is further coupled to communicate with other elements of a system that includes the cluster **30**. In the illustrated embodiment, the L2 cache **34** includes a processor power manager **36** that includes a PState register **38** storing a PState for the processors **32**A-**32***n*. Each processor **32**A-**32***n* may have its own independent PState, groups of processors **32**A-**32***n* may share a PState, or the cluster **30** may have a shared PState for the processors **32**A-**32***n*, in various embodiments. Processor **32**A is shown in more detail in FIG. **1** to include at least two processor cores, a performance core (PCore) **40** and an efficient core (ECore) **42**. Other embodiments may include additional cores. Each core **40** and **42** is coupled to a power supply rail (V_P) through respective power switches **44** and **46**. Thus, each core **40** and **42** may be independently powered up or down. Other processors, such as the processor **32***n*, may be similar to the processor **32**A.

Each processor **32**A-**32***n* may be an entity to which software executing in the system may assign code to execute. For example, the software may be part of an operating system (OS) that controls the hardware in the system. The software may be a thread or task scheduler which schedules code to be executed. The OS may also assign a PState to the processors **32**A-**32***n*, based on the performance demands of the code being executed. The OS may track the behavior of the code to determine PStates, may statically record information for each thread/task to select a PState, etc., or any combination thereof. In addition or alternatively, the PState may be affected by other conditions in the system (thermal limits, battery power available, etc.).

The PCore **40** and the ECore **42** may be different designs, different implementations of an instruction set architecture (ISA) employed by the processors **32**A-**32***n*. Viewed in another way, the PCore **40** and the ECore **42** may implement different microarchitectures. The PCore **40** may be an aggressive design that attempts to maximize performance with power conservation as a less-emphasized design goal. The circuitry in the PCore **40** may be aggressive, which may prevent the minimum supply voltage at which the PCore **40** may operate from being as low as may be desired in some of the PStates. On the other hand, the ECore **42** may implement a more conservative design, and thus may operate correctly at lower minimum voltages than the PCore **40**. The performance of the ECore **42** may be lower than the PCore **40** at a given operating point, and power conservation may be a more highly-emphasized goal for the ECore **42**. The semiconductor area occupied by the ECore **42** may be less than that of the PCore **40** as well.

More particularly, in an embodiment, the ECore **42** and/or the PCore **40** may implement a subset of the ISA employed by the processors **32**A-**32***n*, where one or more features of the ISA are not included in the subset. In an embodiment, the PCore **40** may implement an entirety of the ISA and the ECore **42** may implement a subset. In another embodiment, the PCore **40** and the ECore **42** may each implement a different subset. The subsets may partially overlap (e.g. commonly used instructions, such as the integer instructions, may be part of each subset).

Various embodiments may select the features that are excluded from a particular subset in a variety of ways. For the ECore **42**, a feature that is both infrequently used and expensive to implement (e.g. in terms of semiconductor substrate area occupied, power consumption, etc.) may be excluded. For example, in an embodiment, the ISA may define multiple operand sizes of a given operand type. In an embodiment, the operand type may be integer and the sizes may include 32-bit and 64-bit. Modern code appears to be trending toward 64-bit integer code. On the other hand, the hardware to support both 64-bit and 32-bit integer operand sizes may be area-consuming and may pressure timing, which may result in a higher power implementation. Accordingly, the ECore **42** may implement 64-bit integer hardware and may exclude support for 32-bit integer code. Other ISA features may similarly be excluded. For example, ISAs often include vector instruction sets that perform single-instruction, multiple data (SIMD) processing on a vector of operands. The vector implementations may be high power and/or area-consuming. A more recent ISA introduction is the predicated vector instruction set to facilitate loop vectorization. Such a feature may also be eliminated from the ECore **42**. In general, an ISA feature may include an instruction, a set of instructions, an operand type or size, a mode, etc.

Each possible PState may specify an operating point for a processor **32**A-**32***n*. For example, the operating point may include a supply voltage magnitude for V_P and a clock frequency for the clocks in the processor **32**A-**32***n*. Other embodiments may define the operating point in other fashions, but generally the operating point may indicate the performance and power consumption of the processor. In an embodiment, the PState may be a pair of values that are directly used as the supply voltage magnitude and the clock

frequency. In other embodiments, the PState may be a value that is used to obtain the supply voltage magnitude and the clock frequency (e.g. an index into a table of values).

As illustrated in FIG. **1**, the processor **32**A includes the PCore **40** and the ECore **42**. Each PState that is supported by the processor **32**A is mapped to one of the cores **40** and **42**. Each core **40** and **42** may have more than one PState mapped to it.

As the code being executed by a processor **32**A-**32***n* changes and/or other system considerations warrant a change in the PState, the PState register **38** may be updated (e.g. by the OS). If the PState is changed from a current PState that is mapped to one of the cores **40** and **42** (the "active core") to a new PState that is mapped to another one of the cores **40** and **42** (the "target core"), the cluster **30** may automatically, in hardware, transfer the processor context of the processor **32**A from the active core to the target core. The target core may be powered off at the time the PState is changed. The process of transferring the context may include powering on the target core, resetting and initializing the target core, transferring the processor context, and powering off the active core (making the target core the active core). Execution may continue on the target core (now active core). Accordingly, switching between cores may be invisible to software. In fact, software may not even be "aware" that there are multiple cores in the processor **32**A-**32***n*.

While the example illustrated in FIG. **1** includes two cores in the processor **32**A, other embodiments may include more than two cores. One core may be the most efficient core operating at the lowest PStates, and other cores may be optimized for other points along the performance/efficiency spectrum until yet another core is the highest performance core of the multiple cores in the processor. Any number of cores may be used in various embodiments.

Generally, a processor may be any circuitry configured to implement a defined instruction set architecture (ISA). Various ISAs exist and may be used in various embodiments, such as the x86 architecture (also known as APX), the ARM architecture, the MIPS architecture, PowerPC (now simply Power), etc. A variety of microarchitectural techniques may be employed by the processor, including the multiple core approach described above. Each core may implement various microarchitectural techniques as well. Generally, the microarchitecture may refer to the organization of execution units and other circuitry that is used to implement the ISA. Examples may include in-order versus out-of-order execution, speculative execution, branch prediction, superscalar, superpipelined, etc. Embodiments may implement microcoding techniques in addition to various other techniques.

The processors **32**A-**32***n* and/or the processor complex **30** may be used as any processors in a system. For example, the processors may be central processing units (CPU) that execute the OS to control other hardware in the system and schedule application code to be executed. The CPU may execute the application code as well, etc. The processors may be special purpose processors such as graphics processing units (GPU) optimized for graphics manipulations, digital signal processors (DSPs) optimized for signal processing, embedded processors performing software execution in various peripheral components, etc.

In an embodiment, at most one of the cores **40** and **42** forming a processor **32**A-**32***n* may be powered on during execution, except for times when the processor context is being transferred. A given processor **32**A-**32***n* may be completely off (all cores powered down). The processor power manager **36** may be configured to control the powering on/up of the processor cores and powering off/down of the processor cores using the power switches **44** and **46**.

In some embodiments, the cores **40** and/or **42** may implement data caches that may store modified data (i.e. data that has been written in the cache, e.g. responsive to stores in the processor code being executed, but that has not been written to memory yet such that the data in memory is no longer the correct data). In addition to transferring processor context, the modified data may be flushed from the data cache. Particularly, the data may be flushed to the L2 cache **34**, but may remain stored in the L2 cache **34** unless normal operation of the L2 cache **34** causes the data to be evicted. Once the newly-active core is executing, modified data may be a hit in the L2 cache **34** and may be moved into the newly-active core's cache with relatively low latency.

The processor power manager **36** may be configured to manage PState transitions within the processor cluster **30**. The processor power manager **36** may be configured to communicate supply voltage magnitude transitions to a system level power manager or directly to a power management unit (PMU) that supplies the voltages to the system. The processor power manager **36** may be configured to interact with the clock generation hardware (not shown in FIG. **1**) such as a phase lock loop (PLL) or the like.

The processor context may generally include any software-visible processor state. The state may typically be stored in registers accessible as operands of various instructions defined in the ISA. The state may include architected registers such as the operand registers of various types (integer, floating point, vector, etc.). The registers may also include processor control registers such as status registers, processor mode registers, etc. The registers may also include special purpose registers defined to include specific content for a specific unit. The registers may further include model specific registers, whose existence may be architecturally specified but whose contents may vary from implementation to implementation.

The L2 cache **34** may have any capacity and configuration. The L2 cache **34** may be inclusive of caches in the processors **32**A-**32***n*, exclusive of the caches, or non-inclusive.

FIG. **2** is a graph illustrating efficiency versus performance for one embodiment of the cores **40** and **42**. The dotted curve corresponds to the ECore **42** and the solid line corresponds to the PCore **40**. Efficiency is graphed on the vertical axis and performance on the horizontal axis. Efficiency may be measured in a variety of ways (e.g. performance/watt). Performance may be measured using various benchmark programs such as Specint, SpecFlt, Dhrystone, etc. Various PStates for the processor **32**A are illustrated along the curves in FIG. **2**. PStates that corresponding to higher performance are on the PCore curve, since the PCore is optimized for performance, wherein the PStates corresponding to lower performance/higher energy conservation are on the ECore curve, which is more efficient at lower performance levels but less performant at higher performance levels.

Accordingly, in the example of FIG. **2**, the PStates **1**, **2**, and **3** are mapped to the ECore **42** and the PStates **4**, **5**, **6**, and **7** are mapped to the PCore **40**. Any number of PStates may be supported and any number may be mapped to various cores. In another embodiment, continuous PState settings may be supported. In such an embodiment, a break over point where the curves intersect in FIG. **2** may be defined at which a core switch may occur.

FIG. **3** is a flowchart illustrating one embodiment of operation of the processor power manager **36** in response to

a new PState written to the PState register **38**. While the blocks are shown in a particular order for ease of understanding, other orders may be used. Blocks may be performed in parallel in combinatorial logic in the processor power manager **36**. Blocks, combinations of blocks, and/or the flowchart as a whole may be pipelined over multiple clock cycles. The processor power state manager **36** may be configured to implement the operation shown in FIG. **3**.

The active core may be the core **40**/**42** which is currently executing code. In some embodiments, the active core may be the only core that is powered on during steady state operation at the current PState. If the new PState is not mapped to the active core (decision block **50**, "no" leg), the processor power state manager **36** may consider any information regarding the workload and the target core to determine if the target core supports the workload (decision block **64**). For example, some code may include descriptors when stored in non-volatile memory such as a Flash memory, and the descriptors may indicate which ISA features are used by the code. The processor power state manager **36** may determine the features used by the code from the descriptors. Alternatively, the processors **32**A-**32***n* may track ISA features that are implemented by fewer than all the cores. The tracked state may be used to determine if the target core supports the features that are currently in use.

If the code being executed uses features that are not implemented on the target core (decision block **64**, "no" leg), the processor power state manager **36** may not perform the state change. In an embodiment, the processor power state manager **36** may record the lack of state change in a register or other software-readable location so that software may determine that the state change did not occur. Other indications may be used as well (e.g. an interrupt or other signalling mechanism) when the state change is not performed. In some embodiments, the attempt to prevent state change may not be performed and decision block **64** may be omitted. Instead, unsupported features may be detected while the code is executed on the target core. If the code being executed uses only features that are implemented on the target core (decision block **64**, "yes" leg), the processor power state manager **36** may perform a "core swap" to the core to which the new PState is mapped (block **52**).

If the new PState is mapped to the active core (decision block **50**, "yes" leg), the active core may remain active and execution may continue while the PState is changed. If the new PState is an increase from the current PState (decision block **54**, "yes" leg), the supply voltage magnitude may be increased first to support the increased frequency. Thus, the processor power state manager **36** may request the voltage increase (block **56**) and wait for the voltage increase to complete (decision block **58**, "yes" leg). The processor power state manager **36** may determine that voltage increase is complete by waiting for a specified period of time, or may receive a communication that indicates when the voltage increase is complete. In an embodiment, the processor power state manager **36** may transmit the voltage increase request to another power manager (e.g. an SOC level power manager shown in FIG. **9**, in one embodiment) or may transmit the voltage request directly to a PMU that supplies the voltage. Once the voltage increase is complete, the processor power manager **36** may increase the frequency of the clock (block **60**). On the other hand, if the new PState is a decrease from the current PState, the current supply voltage may support the new (lower) frequency. Thus (decision block **54**, "no" leg), the processor power manager **36** may update the clock frequency and request the new supply voltage without waiting for the voltage change to complete (block **62**).

FIG. **4** is a flowchart illustrating one embodiment of operation of the processor power manager **36** to perform a core swap (block **52** from FIG. **3**). While the blocks are shown in a particular order for ease of understanding, other orders may be used. Blocks may be performed in parallel in combinatorial logic in the processor power manager **36**. Blocks, combinations of blocks, and/or the flowchart as a whole may be pipelined over multiple clock cycles. The processor power state manager **36** may be configured to implement the operation shown in FIG. **4**.

The processor power manager **36** may transition the active core to a "safe" PState (block **70**). The safe PState may be a state at which both the active core and the target core operate correctly. In this context, the target core may be the core to which the new PState is mapped. In embodiments in which there are more than two cores, the safe PState may be different depending on which cores are the active and target cores. The safe PState need not be a PState that is selectable in the PState register **38**. That is, the combination of supply voltage and frequency may not be one of the supported combinations that are mapped to the cores. For example, the PCore may be capable of running at a higher frequency given the supply voltage magnitude in the safe PState. However, the ECore may not be capable of running at the higher frequency with the given supply voltage magnitude. Thus, a safe PState could include the current supply voltage magnitude but a lower clock frequency. Alternatively, the target core may not support the current supply voltage, and the safe PState may include different supply voltage magnitude and clock frequency. Transitioning to the safe PState may be similar to blocks **54**, **56**, **58**, **60**, and **62** in FIG. **3**.

In some embodiments, the operation of FIGS. **3** and **4** may be implemented in hardware circuitry. In other embodiments, the operation may be implemented in a combination of hardware and software stored on a computer accessible storage medium and executed by the processors **32**A-**32***n*, or completely in software.

The processor power manager **36** may power up the target core (block **72**). For example, in the embodiment of FIG. **1**, the processor power manager **36** may close the power switches to the target core, allowing power to flow to the target core. The target core may be reset after power has stabilized. In some embodiments, the target core may initialize after reset is complete. Once reset (and initialization, if applicable) is complete (decision block **74**, "yes" leg), the processor power manager **36** may initiate a transfer of the processor context from the active core to the target core (block **76**). In an embodiment, the cores may include circuitry configured to transmit/receive the processor context. In another embodiment, the circuitry may be in the processor power manager **36**. As mentioned previously, the cores may also be configured to flush the caches during the context transfer. Once the context transfer is complete (decision block **78**, "yes" leg), the processor power manager may power down the (previously) active core and the target core may become the active core (block **80**). The powering down may be accomplished, e.g. by opening the power switches to the previously active core. The processor power manager **36** may transition the active core to the new PState (block **82**). Transitioning to the new PState may be similar to blocks **54**, **56**, **58**, **60**, and **62** in FIG. **3**.

FIG. **5** is a flowchart illustrating one embodiment of operation of a given processor **32**A-**32***n* (and more particu-

larly the active core **40**/**42**) during code execution. While the blocks are shown in a particular order for ease of understanding, other orders may be used. Blocks may be performed in parallel in combinatorial logic within the processor **32**A-**32***n*. Blocks, combinations of blocks, and/or the flowchart as a whole may be pipelined over multiple clock cycles.

Each instruction in the code may be an ISA feature and/or may make use of one or more ISA features. If the ISA features used for a given instruction are implemented by the active core (decision block **100**, "yes" leg), the instruction may be processed normally (block **102**). On the other hand, if at least one feature is not implemented by the active core (decision block **100**, "no" leg) but another core does implement the feature (decision block **104**, "yes" leg), a core swap may be performed to the core that does implement the features (block **52**). If none of the cores implement the feature (decision blocks **100** and **104**, "no" legs), a "not implemented" exception may be taken so that software may handle the error (block **106**).

Generally, the operation illustrated in FIG. **5** (and FIG. **6** discussed below) may be performed for each instruction as it is processed through the processor pipeline. Various features may be detected at different states. Thus, the flowcharts of FIGS. **5** and **6** may be implemented by the processors **32**A-**32***n* in parallel for each instruction in the code sequence being executed.

FIG. **6** is another embodiment of operation of a given processor **32**A-**32***n* (and more particularly the active core **40**/**42**) during code execution. Similar to the embodiment of FIG. **5**, the embodiment of FIG. **6** may determine whether or not the active core implements the ISA features used by the code (decision block **100**), process the code normally if so (block **102**), determine whether or not another core implements the feature (decision block **104**), and take the not implemented exception if not implemented on any core (block **106**). However, in this embodiment, if another core does implement the features (decision block **104**), a core swap exception may be taken (block **108**). The core swap exception may be different from the not implemented exception and other exceptions implemented by the cores **40**/**42**. The core swap exception may cause the processor to execute a core swap exception handler, which may perform the core swap **52** mentioned previously. Similarly, the core swap exception handler may be used at other times that the core swap **52** is performed in some embodiments.

FIG. **7** is a block diagram of one embodiment of a computer accessible storage medium **200**. Generally speaking, a computer accessible storage medium may include any storage media accessible by a computer during use to provide instructions and/or data to the computer. For example, a computer accessible storage medium may include storage media such as magnetic or optical media, e.g., disk (fixed or removable), tape, CD-ROM, DVD-ROM, CD-R, CD-RW, DVD-R, DVD-RW, or Blu-Ray. Storage media may further include volatile or non-volatile memory media such as RAM (e.g. synchronous dynamic RAM (SDRAM), Rambus DRAM (RDRAM), static RAM (SRAM), etc.), ROM, or Flash memory. The storage media may be physically included within the computer to which the storage media provides instructions/data. Alternatively, the storage media may be connected to the computer. For example, the storage media may be connected to the computer over a network or wireless link, such as network attached storage. The storage media may be connected through a peripheral interface such as the Universal Serial Bus (USB). Generally, the computer accessible storage medium **200** may store data in a non-transitory manner, where non-transitory in this context may refer to not transmitting the instructions/data on a signal. For example, non-transitory storage may be volatile (and may lose the stored instructions/data in response to a power down) or non-volatile.

The computer accessible storage medium **200** in FIG. **7** may store code forming the core swap exception handler **202**. The core swap exception handler **202** may include instructions which, when executed by a processor **32**A-**32***n*, implements the operation described above for the core swap exception handler (for example, block **108** in FIG. **6** and the blocks of FIG. **4**). A carrier medium may include computer accessible storage media as well as transmission media such as wired or wireless transmission.

FIG. **8** is a block diagram of one embodiment of the processor **32**A in greater detail. In the illustrated embodiment, the PCore **40** and ECore **42** are shown including instances of a context state machine **90** (i.e. **90**A and **90**B in FIG. **8**). The implementations of the state machine **90** in the cores **40** and **42** may differ, but they may logically operate in a similar fashion. Generally, the state machine **90** in the active core may cause register state to be output by the active core to a context buffer **92** to which the state machines **90** are coupled. The order of the registers in the state may be fixed, so that the receiving state machine may simply read the data and write it to the correct registers. In another implementation, the order may be arbitrary and each register may be assigned an identifier which may be written, with the register contents, to the context buffer **92** and used by the receiving state machine to write the correct register within the receiving core.

The state machine may be implemented in a variety of fashions: fixed function circuitry (e.g. a finite state machine), microcode executed by the processor, in the processor power manager **36** (e.g. transmitting commands to the cores to transfer various registers), etc. Additionally, the state machine **90** in the active processor may flush the data cache(s) to the L2 cache **34**, as mentioned above.

The context buffer **92** may be a first in, first out buffer (FIFO) to capture context state from one core to another. The context buffer **92** may provide elasticity, handle clock domain crossings, etc. In an embodiment, the context buffer **92** may be part of the processor power manager **36** and thus is shown in dotted lines in FIG. **8**. The state machines **90** may also be implemented in the processor power manager **36** in another embodiment. In such embodiments, the processor power manager **36** may have access to the register state in the cores **40** and **42**, or may cause instructions to be executed to perform the register reads/writes to perform the transmission of the register states.

FIG. **9** is a block diagram of one embodiment of an SOC **10** coupled to a memory **12**. As implied by the name, the components of the SOC **10** may be integrated onto a single semiconductor substrate as an integrated circuit "chip." In some embodiments, the components may be implemented on two or more discrete chips in a system. However, the SOC **10** will be used as an example herein. In the illustrated embodiment, the components of the SOC **10** include a central processing unit (CPU) complex **14** (which may be implemented by the processor cluster **30** shown in FIG. **1**), peripheral components **18**A-**18**B (more briefly, "peripherals" **18**), a memory controller **22**, an SOC power manager (PMGR) **16**, and a communication fabric **27**. The components **14**, **16**, **18**A-**18**B, and **22** may all be coupled to the communication fabric **27**. The memory controller **22** may be coupled to the memory **12** during use.

The memory controller **22** may generally include the circuitry for receiving memory operations from the other components of the SOC **10** and for accessing the memory **12** to complete the memory operations. The memory controller **22** may be configured to access any type of memory **12**. For example, the memory **12** may be static random access memory (SRAM), dynamic RAM (DRAM) such as synchronous DRAM (SDRAM) including double data rate (DDR, DDR2, DDR3, DDR4, etc.) DRAM. Low power/mobile versions of the DDR DRAM may be supported (e.g. LPDDR, mDDR, etc.). The memory controller **22** may include queues for memory operations, for ordering (and potentially reordering) the operations and presenting the operations to the memory **12**. The memory controller **22** may further include data buffers to store write data awaiting write to memory and read data awaiting return to the source of the memory operation. In some embodiments, the memory controller **22** may include a memory cache to store recently accessed memory data. In SOC implementations, for example, the memory cache may reduce power consumption in the SOC by avoiding reaccess of data from the memory **12** if it is expected to be accessed again soon. In some cases, the memory cache may also be referred to as a system cache, as opposed to private caches such as the L2 cache or caches in the processors, which serve only certain components. Additionally, in some embodiments, a system cache need not be located within the memory controller **22**.

The peripherals **18**A-**18**B may be any set of additional hardware functionality included in the SOC **10**. For example, the peripherals **18**A-**18**B may include video peripherals such as an image signal processor configured to process image capture data from a camera or other image sensor, display controllers configured to display video data on one or more display devices, graphics processing units (GPUs), video encoder/decoders, scalers, rotators, blenders, etc. The peripherals may include audio peripherals such as microphones, speakers, interfaces to microphones and speakers, audio processors, digital signal processors, mixers, etc. The peripherals may include interface controllers for various interfaces external to the SOC **10** (e.g. the peripheral **18**B) including interfaces such as Universal Serial Bus (USB), peripheral component interconnect (PCI) including PCI Express (PCIe), serial and parallel ports, etc. The peripherals may include networking peripherals such as media access controllers (MACs). Any set of hardware may be included.

The communication fabric **27** may be any communication interconnect and protocol for communicating among the components of the SOC **10**. The communication fabric **27** may be bus-based, including shared bus configurations, cross bar configurations, and hierarchical buses with bridges. The communication fabric **27** may also be packet-based, and may be hierarchical with bridges, cross bar, point-to-point, or other interconnects.

The SOC PMGR **16** may be configured to control the supply voltage magnitudes requested from the PMU in the system. There may be multiple supply voltages generated by the PMU for the SOC **10**. For example, the V_P voltage may be generated for the processors **32**A-**32***n* in the CPU complex **14**, and a V_{SOC} voltage may be generated for other components in the SOC **10**. In an embodiment, V_{SOC} may serve the memory controller **22**, the peripherals **18**, the SOC PMGR **16**, and the other components of the SOC **10** and power gating may be employed based on power domains. There may be multiple supply voltages for the rest of the SOC **10**, in some embodiments. In some embodiments, there may also be a memory supply voltage for various memory arrays in the CPU complex **14** and/or the SOC **10**. The memory supply voltage may be used with the voltage supplied to the logic circuitry (e.g. V_P or V_{SOC}), which may have a lower voltage magnitude than that required to ensure robust memory operation. The SOC PMGR **16** may be under direct software control (e.g. software may directly request the power up and/or power down of components) and/or may be configured to monitor the SOC **10** and determine when various components are to be powered up or powered down. For the CPU complex **14**, the voltage requests for V_P may be provided to the SOC PMGR **16**, which may communicate the requests to the PMU to effect the change in supply voltage magnitudes.

Generally, a component may be referred to as powered on or powered off. The component may be powered on if it is receiving supply voltage so that it may operate as designed. If the component is powered off, then it is not receiving the supply voltage and is not in operation. The component may also be referred to as powered up if it is powered on, and powered down if it is powered off. Powering up a component may refer to supplying the supply voltage to a component that is powered off, and powering down the component may refer to terminating the supply of the supply voltage to the component. Similarly, any subcomponent and/or the SOC **10** as a whole may be referred to as powered up/down, etc. A component may be a predefined block of circuitry which provides a specified function within the SOC **10** and which has a specific interface to the rest of the SOC **10**. Thus, the peripherals **18**A-**18**B, the CPU complex **14**, the memory controller **22**, and the SOC PMGR **16** may each be examples of a component.

It is noted that the number of components of the SOC **10** (and the number of subcomponents for those shown in FIG. **1**, such as within the CPU complex **14**) may vary from embodiment to embodiment. There may be more or fewer of each component/subcomponent than the number shown in FIG. **1**.

Turning next to FIG. **10**, a block diagram of one embodiment of a system **150** is shown. In the illustrated embodiment, the system **150** includes at least one instance of the SOC **10** coupled to one or more peripherals **154** and the external memory **12**. The PMU **156** is provided which supplies the supply voltages to the SOC **10** as well as one or more supply voltages to the memory **12** and/or the peripherals **154**. In some embodiments, more than one instance of the SOC **10** may be included (and more than one memory **12** may be included as well).

The PMU **156** may generally include the circuitry to generate supply voltages and to provide those supply voltages to other components of the system such as the SOC **10**, the memory **12**, various off-chip peripheral components **154** such as display devices, image sensors, user interface devices, etc. The PMU **156** may thus include programmable voltage regulators, logic to interface to the SOC **10** and more particularly the SOC PMGR **16** to receive voltage requests, etc.

The peripherals **154** may include any desired circuitry, depending on the type of system **150**. For example, in one embodiment, the system **150** may be a mobile device (e.g. personal digital assistant (PDA), smart phone, etc.) and the peripherals **154** may include devices for various types of wireless communication, such as wifi, Bluetooth, cellular, global positioning system, etc. The peripherals **154** may also include additional storage, including RAM storage, solid state storage, or disk storage. The peripherals **154** may include user interface devices such as a display screen, including touch display screens or multitouch display

screens, keyboard or other input devices, microphones, speakers, etc. In other embodiments, the system **150** may be any type of computing system (e.g. desktop personal computer, laptop, workstation, net top etc.).

The external memory **12** may include any type of memory. For example, the external memory **12** may be SRAM, dynamic RAM (DRAM) such as synchronous DRAM (SDRAM), double data rate (DDR, DDR2, DDR3, etc.) SDRAM, RAMBUS DRAM, low power versions of the DDR DRAM (e.g. LPDDR, mDDR, etc.), etc. The external memory **12** may include one or more memory modules to which the memory devices are mounted, such as single inline memory modules (SIMMs), dual inline memory modules (DIMM5), etc. Alternatively, the external memory **12** may include one or more memory devices that are mounted on the SOC **10** in a chip-on-chip or package-on-package implementation.

Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

What is claimed is:

1. A processor apparatus comprising:

a plurality of processor cores, wherein:

the processor cores implement at least a portion of an instruction set architecture employed by the processor apparatus;

the instruction set architecture specifies a plurality of operand sizes for a first operand type;

a first processor core of the plurality of processor cores implements only a first operand size of the plurality of operand sizes, wherein a first instruction is executable by the first processor core in the event that the first instruction uses the first operand size and the first instruction is not executable by the first processor core in the event that the first instruction uses a different operand size of the plurality of operand sizes;

a second processor core of the plurality of processor cores implements the plurality of operand sizes, wherein the first instruction is executable by the second processor core with any of the plurality of operand sizes; and

at most one of the plurality of processor cores is active at a given point in time, except during a context switch between two of the plurality of processor cores; and

a processor power manager coupled to the plurality of processor cores, wherein the processor power manager is programmable with a plurality of processor states, wherein each of the plurality of processor states maps to one of the plurality of processor cores, and wherein the processor power manager is configured to:

detect that the processor power manager has been programmed to change a current processor state mapped to the second processor core to a requested processor state mapped to the first processor core during a time that code is being executed by the second processor core;

cause the change to the requested power state and a transfer of a processor context from the second processor core to the first processor core in response to being programmed to change from the current processor state to the requested processor state and further in response to detecting that the code uses only the first operand size, wherein the first processor core is configured to continue execution of the code responsive to the transfer; and

prevent the change to the requested power state and prevent the transfer to the first processor core and continue execution of the code the second processor core responsive to detecting that the code uses one or more operand sizes of the plurality of operand sizes that are different from the first operand size.

2. The processor apparatus as recited in claim **1**, wherein the processor power manager is configured to:

activate the second processor core and deactivate the first processor core responsive to the code being executed on the first processor core and the code using the one or more operand sizes.

3. The processor apparatus as recited in claim **1**, wherein:

the first processor core is configured to detect the use of the one or more operand sizes during execution of the code and to signal an exception in response to detecting the use; and

the exception indicates to the processor power manager that the code uses the one or more operand sizes.

4. The processor apparatus as recited in claim **3**, wherein the processor power manager is configured to cause the transfer of the processor context from the first processor core to the second processor core responsive to the exception.

5. The processor apparatus as recited in claim **4**, wherein the processor power manager comprises a non-transitory computer accessible storage medium storing a plurality of instructions executable by the processor apparatus.

6. The processor apparatus as recited in claim **3**, wherein:

the first processor core is configured to detect that a different feature of the instruction set architecture is not implemented by any core of the plurality of cores; and

the first processor core is configured to signal a different exception in response to detecting use of the different feature.

7. The processor apparatus as recited in claim **1**, wherein the processor power manager is configured to:

power on the second processor core; and

power down the first processor core responsive to transferring the processor context to the second processor core.

8. The processor apparatus as recited in claim **1** wherein the instruction set architecture further specifies a vector instruction set that is not implemented by the first processor core.

9. The processor apparatus as recited in claim **1** wherein the instruction set architecture further specifies a predicated vector instruction set that is not implemented by the first processor core.

10. The processor apparatus as recited in claim **1** wherein the first operand size is a largest operand size of the plurality of operand sizes.

11. A processor apparatus employing an instruction set architecture, the processor apparatus comprising:

a plurality of processor cores, wherein:

a first processor core of the plurality of processor cores implements a subset of the instruction set architecture;

a second processor core of the plurality of processor cores implements an entirety of the instruction set architecture;

the instruction set architecture specifies a plurality of operand sizes for a first operand type;

the subset implemented by the first processor core is the instructions that employ a first operand size of the plurality of operand sizes; and

at most one of the plurality of processor cores is active at a given point in time, except during a context switch between two of the plurality of processor cores; and

a processor power manager coupled to the plurality of processor cores, wherein:

the processor power manager is configured to activate the second processor core and deactivate the first processor core responsive to code that uses one or more operand sizes of the plurality of operand sizes that are different from the first operand size;

the processor power manager is programmable with a plurality of processor states, wherein each of the plurality of processor states maps to one of the plurality of processor cores;

the processor power manager is configured to detect that the processor power manager has been programmed to change a current processor state mapped to the second processor core to a requested processor state mapped to the first processor core during a time that code is being executed by the second processor core;

cause the change to the requested power state and a transfer of a processor context from the second processor core to the first processor core in response to being programmed to change from the current processor state to the requested processor state and further in response detecting that the code uses only the first operand size, wherein the first processor core is configured to continue execution of the code responsive to the transfer; and

the processor power manager is configured to prevent the change to the requested power state and prevent the transfer to the first processor core and continue execution of the code the second processor core responsive to the code using the one or more operand sizes.

12. The processor apparatus as recited in claim **11** wherein the first processor core is configured to detect the one or more operand sizes during execution of the code and indicate the detection to the processor power manager.

13. The processor apparatus as recited in claim **12** wherein the indication is an exception.

14. The processor apparatus as recited in claim **11** wherein the processor power manager is configured to cause the transfer of the processor context from the first processor to the second processor core to activate the second processor core and deactivate the first processor core.

15. The processor apparatus as recited in claim **14** wherein the processor power manager is configured to cause a power on of the second processor core prior to the transfer of the processor context; and cause a power off of the first processor core responsive to completing the transfer of the processor context.

16. A method comprising:

executing first code on a first processor core of a plurality of processor cores forming a processor, wherein the first code is coded to an instruction set architecture that specifies a plurality of operand sizes for a first operand type, and wherein the first processor core implements only a first operand size of the plurality of operand sizes, wherein a first instruction is executable by the first processor core in the event that the first instruction uses the first operand size and the first instruction is not executable by the first processor core in the event that the first instruction uses a different operand size of the plurality of operand sizes;

determining that the first code uses one or more operand sizes of the plurality of operand sizes that are different from the first operand size, wherein the one or more operand sizes are implemented on a second processor core of the plurality of cores, and wherein the first instruction is executable by the second processor core with any of the plurality of operand sizes;

transferring a processor context from the first processor core to the second processor core responsive to the determining;

executing the first code on the second processor core;

operating the second processor core responsive to a first operating point;

detecting a change to a second operating point of the processor during a time that the first code is executing on the second processor core, wherein the second operating point is mapped to the first processor core; and

responsive to determining that the first code uses the one or more operand sizes:

preventing the change to the second operating point;

preventing a transfer of the processor context to the first processor core; and

continuing execution of the first code on the second processor at the first operating point.

17. The method as recited in claim **16** further comprising:

powering on the second processor core responsive to the determining; and

powering off the first processor core subsequent to the transferring.

18. The method as recited in claim **16** further comprising:

operating the first processor core responsive to a third operating point;

detecting a change to a fourth operating point of the first processor, wherein the fourth operating point is mapped to the second processor core; and

transferring the processor context to the second processor core responding to the detecting.

19. The method as recited in claim **16** wherein the second processor core is a higher performance processor core than the first processor core and wherein the second processor core implements an entirety of the instruction set architecture.

* * * * *