(19) DANMARK

(10) **DK/EP 3382551 T3**

Patent- og Varemærkestyrelsen

(12) Oversættelse af europæisk patentskrift

(51) Int.Cl.: ***G 06 F 11/30 (2006.01)*** ***G 06 F 11/34 (2006.01)***

(45) Oversættelsen bekendtgjort den: **2023-12-04**

(80) Dato for Den Europæiske Patentmyndigheds bekendtgørelse om meddelelse af patentet: **2023-09-06**

(86) Europæisk ansøgning nr.: **17199337.1**

(86) Europæisk indleveringsdag: **2017-10-31**

(87) Den europæiske ansøgnings publiceringsdag: **2018-10-03**

(30) Prioritet: **2017-03-29 US 201715473101**

(84) Designerede stater: **AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR**

(73) Patenthaver: **Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA**

(72) Opfinder: **NORRIE, Thomas, GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, CA California 94043, USA**
**KUMAR, Naveen, GOOGLE LLC, 1600 Amphitheatre Parkway, Mountain View, CA California 94043, USA**

(74) Fuldmægtig i Danmark: **Ijon AB, Nordenskiöldsgatan 11A, 21119 Malmö, Sverige**

(54) Benævnelse: **SPORING AF FORDELT HARDWARE**

(56) Fremdragne publikationer:
**US-A- 6 134 676**
**US-A1- 2005 144 532**
**US-A1- 2010 106 678**
**US-A1- 2012 226 837**
**US-A1- 2012 324 290**

# DESCRIPTION

## BACKGROUND

**[0001]** This specification relates to analyzing execution of program code.

**[0002]** Effective performance analysis of distributed software executing within distributed hardware components can be a complex task. Distributed hardware components can be respective processor cores of two or more Central Processing Units (CPUs) (or Graphics Processing Units (GPUs)) that cooperate and interact to execute portions of a larger software program or program code.

**[0003]** From the hardware perspective (e.g., within the CPUs or GPUs), there are generally two types of information or features available for performance analysis: 1) hardware performance counters and 2) hardware event traces.

**[0004]** In United States Patent Application Publication US 2012/0226837 A1, there is described a bus monitoring and debugging system operating independently without impacting the normal operation of the CPU and without adding any overhead to the application being monitored. Users are alerted to timing problems as they occur, and bus statistics that are relevant to providing insight to system operation are automatically captured. Logging of relevant events may be enabled or disabled when a sliding time window expires, or alternatively by external trigger events.

**[0005]** In United States Patent Application Publication US 2005/0144532 A1, there is described a method and apparatus for time stamping events occurring on a large scale distributed network using a local counter associated with each processor of the distributed network.

**[0006]** In United States Patent Application Publication US 2012/0324290 A1, there is described an approach to trace a software program running in a multi-nodal complex computing environment.

**[0007]** In United States Patent Application Publication US 2010/0106678 A1, there is described methods, systems and computer-readable media for monitoring information passed from instances of role(s) of a service application installed on a distributed computing platform and for indexing and analyzing the information within a data store.

## SUMMARY

**[0008]** Various aspects and embodiments of the invention are set out in the appended claims.

**[0009]** The subject matter described in this specification can be implemented in particular embodiments so as to realize one or more of the following advantages. The described hardware tracing systems enable efficient correlation of hardware events that occur during execution of a distributed software program by distributed processing units including multi-node multi-core processors. The described hardware tracing system further includes mechanisms that enable collection and correlation of hardware events/trace data in multiple cross-node configurations. The described method is usable for analyzing performance of the program code being executed by at least the first processor component or the second processor component.

**[0010]** The hardware tracing system enhances computational efficiency by using dynamic triggers that execute through hardware knobs/features. Further, hardware events can be time-ordered in a sequenced manner with event descriptors such as unique trace identifiers, event timestamps, event source-address, and event destination-address. Such descriptors aid software programmers and processor design engineers with effective debugging and analysis of software and hardware performance issues that may arise during source code execution.

**[0011]** The details of one or more implementations of the subject matter described in this specification are set forth in the accompanying drawings and the description below. Other potential features, aspects, and advantages of the subject matter will become apparent from the description, the drawings, and the claims.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0012]**

FIG. 1 illustrates a block diagram of an example computing system for distributed hardware tracing.

FIG. 2 illustrates a block diagram of trace chains and respective nodes of an example computing system for distributed hardware tracing.

FIG. 3 illustrates a block diagram of an example trace mux design architecture and an example data structure.

FIG. 4 is a block diagram indicating trace activity for a direct memory access trace event executed by an example computing system for distributed hardware tracing.

FIG. 5 is a process flow diagram of an example process for distributed hardware tracing.

**[0013]** Like reference numbers and designations in the various drawings indicate like elements.

## DETAILED DESCRIPTION

**[0014]** The subject matter described in this specification generally relates to distributed hardware tracing. In particular, a computing system monitors execution of program code executed by one or more processor cores. For example, the computing system can monitor execution of program code executed by a first processor core and execution of program code executed by at least a second processor core. The computing system stores data identifying one or more hardware events in a memory buffer. The stored data identifying the events correspond to events that occur across distributed processor units that include at least the first and second processor cores.

**[0015]** For each hardware event, the stored data includes an event time stamp and metadata characterizing the hardware event. The system generates a data structure identifying the hardware events. The data structure arranges the events in a time ordered sequence and associates events with at least the first or second processor cores. The system stores the data structure in a memory bank of a host device and uses the data structure to analyze performance of the program code executed by the first or second processor cores.

**[0016]** FIG. 1 illustrates a block diagram of an example computing system 100 for distributed hardware tracing. As used in this specification, distributed hardware system tracing corresponds to storage of data identifying events that occur within components and sub-components of an example processor micro-chip. Further, as used herein, a distributed hardware system (or tracing system) corresponds to a collection of processor micro-chips or processing units that cooperate to execute respective portions of a software/program code configured for distributed execution amongst the collection of processor micro-chips or distributed processing units.

**[0017]** System 100 is a distributed processing system, having one or more processors or processing units that execute a software program in a distributed manner, i.e., by executing different parts or portions of the program code on different processing units of system 100. Processing units can include two or more processors, processor micro-chips, or processing units, e.g., at least a first processing unit and a second processing unit.

**[0018]** In some implementations, two or more processing units can be distributed processing units when the first processing unit receives and executes a first portion of program code of a distributed software program, and when the second processing unit receives and executes a second portion of program code of the same distributed software program.

**[0019]** In some implementations, different processor chips of system 100 can form respective nodes of the distributed hardware system. In alternative implementations, a single processor chip can include one or more processor cores and hardware features that can each form respective nodes of the processor chip.

**[0020]** For example, in the context of a central processing unit (CPU), a processor chip can include at least two nodes and each node can be a respective core of the CPU. Alternatively, in the context of a graphical processor unit (GPU), a processor chip can include at least two nodes and each node can be a respective streaming multiprocessor of the GPU. Computing system 100 can include multiple processor components. In some implementations, the processor components can be at least one of a processor chip, a processor core, a memory access engine, or at least one hardware component of the overall computing system 100.

**[0021]** In some instances, a processor component, such as a processor core, can be a fixed-function component configured to execute at least one specific operation based on at least one issued instruction of the executing program code. In other instances, a processor component, such as a memory access engine (MAE), can be configured to execute program code at a lower level of detail or granularity than program code executed by other processor components of system 100.

**[0022]** For example, program code executed by a processor core can cause an MAE descriptor to be generated and transmitted/sent to the MAE. After receipt of the descriptor, the MAE can execute a data transfer operation based on the MAE descriptor. In some implementations, data transfers executed by the MAE can include, for example, moving data to and from certain components of system 100 via certain data paths or interface components of the system, or issuing data requests onto an example configuration bus of system 100.

**[0023]** In some implementations, each tensor node of an example processor chip of system 100 can have at least two "front-ends" which can be hardware blocks/features that process program instructions. As discussed in more detail below, a first front-end can correspond to first processor core 104, while a second front-end can correspond to second processor core 106. Hence, the first and second processor cores may also be described herein as first front-end 104 and second front-end 106.

**[0024]** As used in this specification, a trace chain can be a specific physical data communication bus that trace entries can be put onto for transmission to an example chip manager within system 100. Received trace entries can be data words/structures including multiple bytes and multiple binary values or digits. Thus, the descriptor "word" indicates a fixed-sized piece of binary data that can be handled as a unit by hardware devices of an example processor core.

**[0025]** In some implementations, the processor chips of the distributed hardware tracing system are multi-core processors (i.e., having multiple cores) that each execute portions of program code in respective cores of the chip. In some implementations, portions of program code can correspond to vectorized computations for inference workloads of an example multi-layer neural network. While in alternative implementations, portions of program code can correspond generally to software modules associated with conventional programming languages.

**[0026]** Computing system 100 generally includes a node manager 102, a first processor core (FPC) 104, a second processor core (SPC) 106, a node fabric (NF) 110, a data router 112, and a host interface block (HIB) 114. In some implementations, system 100 can include a memory mux 108 that is configured to perform signal switching, multiplexing, and de-multiplexing functions. System 100 further includes a tensor core 116 that includes FPC 104 disposed therein. Tensor core 116 can be an example computational device configured to perform vectorized computations on multi-dimensional data arrays. Tensor core 116 can include a vector processing unit (VPU) 118, that interacts with a matrix unit (MXU) 120, transpose unit (XU) 122, and reduction and permutation unit (RPU) 124. In some implementations, computing system 100 can include one or more execution units of a conventional CPU or GPU, such as load/store units, arithmetic logic units (ALU's) and vector units.

**[0027]** The components of system 100 collectively include a large set of hardware performance counters as well as support hardware that facilitates completion of tracing activity within the components. As described in more detail below, program code executed by respective processor cores of system 100 can include embedded triggers used to simultaneously enable multiple performance counters during code execution. In general, detected triggers cause trace data to be generated for one or more trace events. The trace data can correspond to incremental parameter counts that are stored in the counters and that can be analyzed to discern performance characteristics of the program code. Data for respective trace events can be stored in an example storage medium (e.g., a hardware buffer) and can include a timestamp that is generated responsive to detection of the trigger.

**[0028]** Further, trace data is generated for a variety of events occurring within hardware components of system 100. Example events can include inter-node and cross-node communication operations, such as direct memory access (DMA) operations and sync flag updates (each described in more detail below). In some implementations, system 100 can include a globally synchronous timestamp counter generally referred to as Global Time Counter ("GTC"). In other implementations, system 100 can include other types of global clocks, such as a Lamport Clock.

**[0029]** The GTC can be used for precise correlation of program code execution and performance of software/program code that executes in a distributed processing environment. Additionally, and related in part to the GTC, in some implementations system 100 can include one or more trigger mechanisms used by distributed software programs to start and stop data tracing in a distributed system in a highly coordinated manner.

**[0030]** In some implementations, a host system 126 compiles program code that can include embedded operands that trigger, upon detection, to cause capture and storage of trace data associated with hardware events. In some implementations, host system 126 provides the compiled program code to one or more processor chips of system 100. In alternative implementations, program code can be compiled (with embedded triggers) by an example external compiler and loaded to the to one or more processor chips of system 100. In some instances, the compiler can set one or more trace bits (discussed below) associated with

certain triggers that are embedded in portions of software instructions. The compiled program code can be a distributed software program that is executed by the one or more components of system 100.

**[0031]** Host system 126 can include a monitoring engine 128 configured to monitor execution of program code by one or more components of system 100. In some implementations, monitoring engine 128 enables host system 126 to monitor execution of program code executed by at least FPC 104 and SPC 106. For example, during code execution, host system 126 can monitor, via monitoring engine 128, performance of the executing code at least by receiving periodic timelines of hardware events based on generated trace data. Although a single block is shown for host system 126, in some implementations, system 126 can include multiple hosts (or host subsystems) that are associated with multiple processor chips or chip cores of system 100.

**[0032]** In other implementations, cross-node communications that involve at least three processor cores may cause host system 126 to monitor data traffic at one or more intermediate "hops" as data traffic traverses a communication path between FPC 104 and an example third processor core/node. For example, FPC 104 and the third processor core may be the only cores executing program code at given time period. Hence, a data transfer from FPC 104 to the third processor core can generate trace data for an intermediate hop at SPC 106 as data is transferred from FPC 104 to the third processor core. Stated another way, during data routing in system 100, data from a first processor chip going to a third processor chip may need to traverse a second processor chip, and so execution of the data routing operation may cause trace entries to be generated for routing activity in the second chip.

**[0033]** Upon execution of the compiled program code, the components of system 100 can interact to generate timelines of hardware events that occur in a distributed computer system. The hardware events can include intra-node and cross-node communication events. Example nodes of a distributed hardware system and their associated communications are described in more detail below with reference to FIG. 2. In some implementations, a data structure is generated that identifies a collection of hardware events for at least one hardware event timeline. The timeline enables reconstruction of events that occur in the distributed system. In some implementations, event reconstruction can include correct event ordering based on analysis of time stamps generated during occurrence of a particular event.

**[0034]** In general, an example distributed hardware tracing system can include the above described components of system 100 as well as at least one host controller associated with a host system 126. Performance or debugging of data obtained from a distributed tracing system can be useful when the event data is correlated in, for example, a time-ordered or sequenced manner. Data correlation occurs when multiple stored hardware events corresponding to connected software modules are stored and then sequenced for structured analysis by host system 126. For implementations including multiple host systems, correlation of data obtained via the different hosts may be performed, for example, by the host controller.

**[0035]** In some implementations, FPC 104 and SPC 106 are each distinct cores of a multi-core processor chip; while in other implementations, FPC and SPC 104, 106 are respective cores of distinct multi-core processor chips. As indicated above, system 100 can include distributed processor units having at least FPC 104 and SPC 106. Distributed processor units of system 100 include one or more hardware or software components configured to execute at least a portion of a larger distributed software program or program code.

**[0036]** Data router 112 is an inter-chip interconnect (ICI) providing data communication paths between the components of system 100. In particular, router 112 can provide communication coupling or connections between FPC 104 and SPC 106, and between the respective components associated with cores 104, 106. Node fabric 110 interacts with data router 112 to move data packets within the distributed hardware components and sub-components of system 100.

**[0037]** Node manager 102 is a high-level device that manages low-level node functions in multi-node processor chips. As discussed in more detail below, one or more nodes of a processor chip can include chip managers controlled by node manager 102 to manage and store hardware event data in local entry logs. Memory mux 108 is a multiplexing device that can perform switching, multiplexing, and de-multiplexing operations on data signals provided to an example external high bandwidth memory (HBM) or data signals received from the external HBM.

**[0038]** In some implementations, an example trace entry (described below) can be generated, by mux 108, when mux 108 switches between FPC 104 and SPC 106. Memory mux 108 can potentially impact performance of a particular processor core 104, 106 that is not able to access mux 108. Thus, trace entry data generated by mux 108 can aid in understanding resulting spikes in latencies of certain system activities associated with the respective cores 104, 106. In some implementations, hardware event data (e.g., trace points discussed below) originating within mux 108 can be grouped, in an example hardware event timeline, along with event data for node fabric 110. Event grouping can occur when certain tracing activity causes event data for multiple hardware components to be stored in an example hardware buffer (e.g., trace entry log 218, discussed below).

**[0039]** In system 100, performance analysis hardware encompasses FPC 104, SPC 106, mux 108, node fabric 110, data router 112, and HIB 114. Each of these hardware components or units include hardware performance counters as well as hardware event tracing facilities and functions. In some implementations, VPU 118, MXU 120, XU 122 and RPU 124 do not include their own dedicated performance hardware. Rather, in such implementations, FPC 104 can be configured to provide the necessary counters for VPU 118, MXU 120, XU 122 and RPU 124.

**[0040]** VPU 118 can include an internal design architecture that supports localized high bandwidth data processing and arithmetic operations associated with vector elements of an example matrix-vector processor. MXU 120 is a matrix multiplication unit configured to perform, for example, up to 128x128 matrix multiplies on vector data sets of multiplicands.

**[0041]** XU 122 is a transpose unit configured to perform, for example, up to 128x128 matrix transpose operations on vector data associated with the matrix multiply operations. RPU 124 can include a sigma unit and a permute unit. The sigma unit executes sequential reductions on vector data associated with the matrix multiply operations. The reductions can include sums and various types of compare operations. The permute unit can fully permute or replicate all elements of vector data associated with the matrix multiply operations.

**[0042]** In some implementations, program code executed by the components of system 100 can be representative of machine learning, neural network inference computations, and/or one or more direct memory access functions. Components of system 100 can be configured to execute one or more software programs including instructions that cause a processing unit(s) or device(s) of the system to execute one or more functions. The term "component" is intended to include any data processing device or storage device such as control status registers or any other device able to process and store data.

**[0043]** System 100 can generally include multiple processing units or devices that can include one or more processors (e.g., microprocessors or central processing units (CPUs)), graphics processing units (GPUs), application specific integrated circuits (ASICs), or a combination of different processors. In alternative embodiments, system 100 can each include other computing resources/devices (e.g., cloud-based servers) that provide additional processing options for performing computations related to hardware tracing functions described in this specification.

**[0044]** The processing units or devices further include one or more memory units or memory banks (e.g., registers/counters). The processing units execute programmed instructions stored in memory to devices of system 100 to perform one or more functions described in this specification. The memory units/banks can include one or more non-transitory machine-readable storage mediums. The non-transitory machine-readable storage medium can include solid-state memory, magnetic disk, and optical disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (e.g., EPROM, EEPROM, or Flash memory), or any other tangible medium capable of storing information.

**[0045]** FIG. 2 illustrates a block diagram of example trace chains and respective example nodes 200, 201 used for distributed hardware tracing executed by system 100. In some implementations, the nodes 200, 201 of system 100 can be different nodes within a single multi-core processor. In other implementations, node 200 can be a first node in a first multi-core processor chip and node 201 can be a second node in a second multi-core processor chip.

**[0046]** Although two nodes are depicted in the implementation of FIG. 2, in alternative implementations, system 100 can include multiple nodes. For implementations involving multiple nodes, cross-node data transfers can generate trace data at intermediate hops along an example data path that traverse multiple nodes. For example, intermediate hops can

correspond to data transfers which pass through distinct nodes in a particular data transfer path. In some instances, trace data associated with ICI traces/hardware events can be generated for one or more intermediate hops that occur during cross-node data transfers which pass through one or more nodes.

**[0047]** In some implementations, node 0 and node 1 are tensor nodes used for vectorized computations associated with portions of program code for inference workloads. As used in this specification, a tensor is a multi-dimensional geometric object and example multi-dimensional geometric objects include matrices and data arrays.

**[0048]** As shown in the implementation of FIG. 2, node 200 includes a trace chain 203 that interacts with at least a subset of the components of system 100. Likewise, node 201 includes a trace chain 205 that interacts with at least a subset of the components of system 100. In some implementations, nodes 200, 201 are example nodes of the same subset of components, while in other implementations, nodes 200, 201 are respective nodes of distinct component subsets. Data router/ICI 112 includes a trace chain 207 that generally converges with trace chains 203 and 205 to provide trace data to chip manager 216.

**[0049]** In the implementation of FIG. 2, nodes 200, 201 can each include respective component subsets having at least FPC 104, SPC 106, node fabric 110, and HIB 114. Each component of nodes 200, 201 includes one or more trace muxes configured to group trace points (described below) generated by a particular component of the node. FPC 104 includes a trace mux 204, node fabric 110 includes trace muxes 210a/b, SPC 106 includes trace muxes 206a/b/c/d, HIB 214 includes trace mux 214, and ICI 212 includes trace mux 212. In some implementations, a trace control register for each trace mux allows individual trace points to be enabled and disabled. In some instances, for one or more trace muxes, their corresponding trace control registers can include individual enable bits as well as broader trace mux controls.

**[0050]** In general, the trace control registers can be conventional control status registers (CSR) that receive and store trace instruction data. Regarding the broader trace mux controls, in some implementations, tracing can be enabled and disabled based on CSR writes executed by system 100. In some implementations, tracing can be dynamically started and stopped, by system 100, based on the value of a global time counter (GTC), the value of an example trace-mark register in the FPC 104 (or core 116), or based on the value of a step mark in SPC 106.

**[0051]** Details and descriptions relating to computing systems and computer-implemented methods for dynamically starting and stopping tracing activity as well as for synchronized hardware event collection are described in related U.S. Patent Application US2018285233 entitled "Synchronous Hardware Event Collection," filed on March 29, 2017.

**[0052]** In some implementations, for core 116, FPC 104 can use a trace control parameter to define a trace window associated with event activity occurring within core 116. The trace control parameter allows the trace window to be defined in terms of lower and upper bounds for the GTC as well as lower and upper bounds for the trace-mark register.

**[0053]** In some implementations, system 100 can include functions that enable reduction of the number of trace entries that are generated, such as trace event filtering features. For example, FPC 104 and SPC 106 can each include filtering features which limit the rate at which each core sets a trace bit in an example generated trace descriptor (described below). HIB 114 can include similar filtering features such as an example DMA rate limiter that limits trace bits associated with capture of certain DMA trace events. Additionally, HIB 114 can include controls (e.g., via an enable bit) for limiting which queues source DMA trace entries.

**[0054]** In some implementations, a descriptor for a DMA operation can have a trace bit that is set by an example compiler of host system 126. When the trace bit is set, hardware features/knobs that determine and generate trace data are used to complete an example trace event. In some instances, a final trace bit in the DMA can be a logical OR operation between a trace bit that is statically inserted by the compiler and a trace bit that is dynamically determined by a particular hardware component. Hence, in some instances, the compiler generated trace bit can provide a mechanism, apart from filtering, to reduce an overall amount of trace data that is generated.

**[0055]** For example, a compiler of host system 126 may decide to only set trace bits for one or more remote DMA operations (e.g., a DMA across at least two nodes) and clear trace bits for one or more local DMA operations (e.g., a DMA within a particular tensor node, such as node 200). In this manner, an amount of trace data that is generated can be reduced based on tracing activity being limited to cross-node (i.e., remote) DMA operations, rather than tracing activity that includes both cross-node and local DMA operations.

**[0056]** In some implementations, at least one trace event initiated by system 100 can be associated with a memory access operation that includes multiple intermediate operations occurring across system 100. A descriptor (e.g., an MAE descriptor) for the memory access operation can include a trace bit that causes data associated with the multiple intermediate operations to be stored in one or more memory buffers. Thus, the trace bit can be used to "tag" intermediate memory operations and generate multiple trace events at intermediate hops of the DMA operation as data packets traverse system 100.

**[0057]** In some implementations, ICI 112 can include a set of enable bits and a set of packet filters that provide control functionality for each ingress and egress port of a particular component of node 200, 201. These enable bits and packet filters allow ICI 112 to enable and disable trace points associated with particular components of nodes 200, 201. In addition to enabling and disabling trace points, ICI 112 can be configured to filter trace data based on event source, event destination, and trace event packet type.

**[0058]** In some implementations, in addition to using step markers, GTC, or trace markers, each trace control register for processor cores 104, 106 and HIB 114 can also include an "everyone" trace mode. This "everyone" trace mode can enable tracing across an entire processor chip to be controlled by either trace mux 204 or trace mux 206a. While in the

everyone trace mode, traces muxes 204 and 206a can send an "in-window" trace control signal that specifies whether or not that particular trace mux, either mux 204 or mux 206a, is in a trace window.

**[0059]** The in-window trace control signal can be broadcast or universally transmitted to all other trace muxes, for example, within a processor chip or across multiple processor chips. The broadcast to the other trace muxes can cause all tracing to be enabled when either mux 204 or mux 206a is executing tracing activity. In some implementations, trace muxes associated with processor cores 104, 106, and HIB 114 each include a trace window control register that specifies when and/or how the "everyone trace" control signal is generated.

**[0060]** In some implementations, tracing activity in trace muxes 210a/b and trace mux 212, is generally enabled based on whether a trace bit is set in data words for DMA operations or control messages that traverses ICI/data router 112. DMA operations or control messages can be fixed-size binary data structures that can have a trace bit within the binary data packets set based on certain circumstances or software conditions.

**[0061]** For example, when a DMA operation is initiated in FPC 104 (or SPC 106) with a trace-type DMA instruction and the initiator (processor cores 104 or 106) is in a trace window, the trace bit will be set in that particular DMA. In another example, for FPC 104, control messages for data writes to another component within system 100 will have the trace bit set if FPC 104 is in a trace window and a trace point that causes trace data to be stored is enabled.

**[0062]** In some implementations, zero-length DMA operations provide an example of a broader DMA implementation within system 100. For example, some DMA operations can produce non-DMA activity within system 100. Execution of the non-DMA activity can also be traced (e.g., generate trace data) as if the non-DMA activity were a DMA operation (e.g., DMA activity including non-zero-length operations). For example, a DMA operation initiated at a source location but without any data (e.g., zero-length) to be sent or transferred could instead send a control message to the destination location. The control message will indicate that there is no data to be received, or worked with, at the destination, and the control message itself would be traced by system 100 as a non-zero-length DMA operation would be traced.

**[0063]** In some instances, for SPC 106, zero-length DMA operations can generate a control message, and a trace bit associated with the message is set only if the DMA would have had the trace bit set, i.e., had the control message not had a zero-length. In general, DMA operations initiated from host system 126 will have the trace bit set if HIB 114 is in a trace window.

**[0064]** In the implementation of FIG. 2, trace chain 203 receives trace entry data for the component subset that aligns with node 0, while trace chain 205 receives trace entry data for the component subset that aligns with node 1. Each trace chain 203, 205, 207 are distinct data communication paths used by respective nodes 200, 201 and ICI 112 to provide trace entry data to an example trace entry data log 218 of a chip manager 216. Thus, the endpoint of

trace chains 203, 205, 207 is chip manager 216 where trace events can be stored in example memory units.

**[0065]** In some implementations, at least one memory unit of chip manager 216 can be 128-bit wide and can have a memory depth of at least 20,000 trace entries. In alternative implementations, at least one memory unit can have a larger or smaller bit-width and can have a memory depth capable of storing more or fewer entries.

**[0066]** In some implementations, chip manager 216 can include at least one processing device executing instructions to manage received trace entry data. For example, chip manager 216 can execute instructions to scan/analyze time stamp data for respective hardware events of trace data received via trace chains 203, 205, 207. Based on the analysis, chip manager 216 can populate trace entry log 218 to include data that can be used to identify (or generate) a time-ordered sequence of hardware trace events. The hardware trace events can correspond to movement of data packets occurring at the component and sub-component level when processing units of system 100 execute an example distributed software program.

**[0067]** In some implementations, hardware units of system 100 may generate trace entries (and corresponding timestamps) that populate an example hardware trace buffer in a non-time-ordered manner (i.e., out-of-order). For example, chip manager 216 can cause multiple trace entries, having generated time-stamps, to be inserted into entry log 218. Respective trace entries, of the multiple inserted trace entries, may not be time-ordered relative to one another. In this implementation, non-time-ordered trace entries can be received by an example host buffer of host system 126. Upon receipt by the host buffer, host system 126 can execute instructions relating to performance analysis/monitoring software to scan/analyze time stamp data for the respective trace entries. The executed instructions can be used to sort the trace entries and to construct/generate a timeline of hardware trace events.

**[0068]** In some implementations, trace entries can be removed from entry log 218 during a tracing session via a host DMA operation. In some instances, host system 126 may not DMA entries out of trace entry log 218 as quickly as they are added to the log. In other implementations, entry log 218 can include a predefined memory depth. If the memory depth limit of entry log 218 is reached, additional trace entries may be lost. In order to control which trace entries are lost, entry log 218 can operate in first-in-first-out (FIFO) mode, or, alternatively, in an overwrite recording mode.

**[0069]** In some implementations, the overwrite recording mode can be used, by system 100, to support performance analysis associated with post-mortem debugging. For example, program code can be executed for a certain time-period with tracing activity enabled and overwrite recording mode enabled. In response to a post-mortem software event (e.g., a program crash) within system 100, monitoring software executed by host system 126 can analyze the data contents of an example hardware trace buffer to gain insight into hardware events that occurred before the program crash. As used in this specification, post-mortem debugging relates to analysis or debugging of program code after the code has crashed or has generally

failed to execute/operate as intended.

**[0070]** In FIFO mode, if entry log 218 is full, and if host system 126 does remove saved log entries within a certain timeframe, to conserve memory resources, new trace entries may not be saved to a memory unit of chip manager 216. While in the overwrite recording mode, if entry log 218 is full because host system 126 does remove saved log entries within a certain timeframe, to conserve memory resources new trace entries can overwrite the oldest trace entry stored within entry log 218. In some implementations, trace entries are moved to a memory of host system 126 in response to a DMA operation using processing features of HIB 114.

**[0071]** As used in this specification, a trace point is the generator of a trace entry and data associated with the trace entry received by chip manager 216 and stored in trace entry log 218. In some implementations, a multi-core multi-node processor microchip can include three trace chains within the chip such that a first trace chain receives trace entries from a chip node 0, a second trace chain receives trace entries from a chip node 1, and a third trace chain receives trace entries from an ICI router of the chip.

**[0072]** Each trace point has a unique trace identification number, within its trace chain, that it inserts into the header of the trace entry. In some implementations, each trace entry identifies the trace chain it originated from in a header indicated by one or more bytes/bits of the data word. For example, each trace entry can include a data structure having defined field formats (e.g., header, payload, etc.) that convey information about a particular trace event. Each field in a trace entry corresponds to useful data applicable to the trace point that generated the trace entry.

**[0073]** As indicated above, each trace entry can be written to, or stored within, a memory unit of chip manager 216 associated with trace entry log 218. In some implementations, trace points can be enabled or disabled individually and multiple trace points can generate the same type of trace entry although with different trace point identifiers.

**[0074]** In some implementations, each trace entry type can include a trace name, trace description, and a header that identifies encodings for particular fields and/or a collection of fields within the trace entry. The name, description, and header collectively provide a description of what the trace entry represents. From the perspective of chip manager 216, this description can also identify the particular trace chain 203, 205, 207 that a specific trace entry came in on within a particular processor chip. Thus, fields within a trace entry represent pieces of data (e.g., in bytes/bits) relevant to the description and can be a trace entry identifier used to determine which trace point generated a particular trace entry.

**[0075]** In some implementations, trace entry data associated with one or more of the stored hardware events can correspond, in part, to data communications that occur: a) between at least a node 0 and node 1; b) between at least components within node 0; and c) between at least components within node 1. For example, stored hardware events can correspond, in part,

to data communications that occur between at least one of: 1) FPC 104 of node 0 and FPC 104 of node 1; FPC 104 of node 0 and SPC 106 of node 0; 2) SPC 106 of node 1 and SPC 106 of node 1.

**[0076]** FIG. 3 illustrates a block diagram of an example trace mux design architecture 300 and an example data structure 320. Trace mux design 300 generally includes a trace bus input 302, a bus arbiter 304, and a local trace point arbiter 306, a bus FIFO 308, at least one local trace event queue 310, a shared trace event FIFO 312, and a trace bus out 314.

**[0077]** Mux design 300 corresponds to an example trace mux disposed within a component of system 100. Mux design 300 can include the following functionality. Bus in 302 can relate to local trace point data that is temporarily stored within bus FIFO 308 until such that time arbitration logic (e.g., arbiter 304) can cause the trace data to be placed unto an example trace chain. One or more trace points for a component can insert trace event data into at least one local trace event queue 310. Arbiter 306 provides first level arbitration and enables selection of events from among the local trace events stored within queue 310. Selected events are place in shared trace event FIFO 312 which also functions as a storage queue.

**[0078]** Arbiter 304 provides second level arbitration that receives local trace events from FIFO queue 312 and merges the local trace events onto a particular trace chain 203, 205, 207 via trace bus out 314. In some implementations, trace entries may be pushed into local queues 310 faster than they can be merged to shared FIFO 312, or, alternatively, trace entries may be pushed into shared FIFO 312 faster than they can be merged onto trace bus 314. When these scenarios occur, the respective queues 310 and 312 will become full with trace data.

**[0079]** In some implementations, when either queue 310 or 312 becomes full with trace data, system 100 can be configured so that the newest trace entries are dropped and not stored to, or merged to, a particular queue. In other implementations, rather than dropping trace entries when certain queues fill up (e.g., queues 310, 312), system 100 can be configured to stall an example processing pipeline until queues that are filled once again have available queue space to receive entries.

**[0080]** For example, a processing pipeline that uses queues 310, 312 can be stalled until a sufficient, or threshold, number of trace entries are merged onto trace bus 314. The sufficient or threshold number can correspond to a particular number merged trace entries that result in available queue space for one or more trace entries to be received by queues 310, 312. Implementations in which processing pipelines are stalled, until downstream queue space becomes available, can provide higher-fidelity trace data based on certain trace entries being retained rather than dropped.

**[0081]** In some implementations, local trace queues are as wide as required by the trace entry, such that each trace entry takes only one spot in local queue 310. However, shared trace FIFO queue 312 can use a unique trace entry line encoding such that some trace entries can occupy two locations in shared queue 312. In some implementations, when any data of a trace packet

is dropped, the full packet is dropped so that no partial packets appear in trace entry log 218.

**[0082]** In general, a trace is a timeline of activities or hardware events associated with a particular component of system 100. Unlike performance counters (described below), which are aggregate data, traces contain detailed event data that provide insight into hardware activity occurring during a specified trace window. The described hardware system enables extensive support for distributed hardware tracing, including generation of trace entries, temporary storage of trace entries in hardware managed buffer, static and dynamic enabling of one or more trace types, and streaming of trace entry data to host system 126.

**[0083]** Traces are generated for hardware events executed by components of system 100, such as, generating a DMA operation, executing a DMA operation, issuing/execution of certain instructions, or updating sync flags. In some instances, tracing activity can be used to track DMAs through the system, or to track instructions executing on a particular processor core.

**[0084]** System 100 is configured to generate at least one data structure 320 that identifies one or more hardware events 322, 324 from a timeline of hardware events. Data structure 320 arranges one or more hardware events 322, 324 in a time ordered sequence of events that are associated with at least FPC 104 and SPC 106. System 100 stores data structure 320 in a memory bank of a host control device of host system 126. Data structure 320 can be used to assess performance of program code executed by at least processor cores 104 and 106.

**[0085]** As shown by hardware events 324, a particular trace identification (ID) number (e.g., trace ID `003) is associated with multiple hardware events that occur across the distributed processor units. The multiple hardware events correspond to a particular memory access operation (e.g., a DMA), and the particular trace ID number is used to correlate one or more hardware events.

**[0086]** As indicated by event 324, a single trace ID for a DMA operation includes multiple time steps corresponding to multiple different points in the DMA. In some instances, trace ID `003 has an "issued" event, an "executed" event, and a "completed" event that are identified as being some time apart relative to each other. Hence, in this regard, the trace ID is further used for determining a latency attribute of the memory access operation based on the correlation and with reference to the time steps.

**[0087]** In some implementations, generating data structure 320 can include, for example, system 100 comparing event time stamps of respective events in a first subset of hardware events with event time stamps of respective events in a second subset of hardware events. Generating data structure 320 can further include, system 100 providing, for presentation in the data structure, a correlated set of hardware events based, in part, on the comparison between the first subset of events and the second subset of events.

**[0088]** As shown in FIG. 3, data structure 320 can identify at least one parameter that indicates a latency attribute of a particular hardware event 322, 324. The latency attribute can

indicate at least a duration of the particular hardware event. In some implementations, data structure 320 is generated by software instructions executed by a control device of host system 126. In some instances, structure 320 can be generated responsive to the control device storing trace entry data to a memory disk/unit of host system 126.

**[0089]** FIG. 4 is a block diagram 400 indicating example trace activity for a direct memory access (DMA) trace event executed by system 100. For DMA tracing, data for an example DMA operation originating from a first processor node to a second processor node can travel via ICI 112 and can generate intermediate ICI/router hops along the data path. The DMA operation will generate trace entries at each node within a processor chip, and along each hop, as the DMA operation traverses ICI 112. Information is captured by each of these generated trace entries to reconstruct a temporal progression of the DMA operations along the nodes and hops.

**[0090]** An example DMA operation can be associated with the process steps depicted in the implementation of FIG. 4. For this operation, a local DMA transfers data from a virtual memory 402 (vmem 402) associated with at least one of processor cores 104, 106 to HBM 108. The numbering depicted in diagram 400 correspond to the steps of table 404 and generally represents activities in node fabric 110 or activities initiated by node fabric 110.

**[0091]** The steps of table 404 generally describe associated trace points. The example operation will generate six trace entries for this DMA. Step one includes the initial DMA request from the processor core to node fabric 110 which generates a trace point in the node fabric. Step two includes a read command in which node fabric 110 asks the processor core to transfer data which generates another trace point in node fabric 110. The example operation does not have a trace entry for step three when vmem 402 completes a read of node fabric 110.

**[0092]** Step four includes node fabric 110 performing a read resource update to cause a sync flag update in the processor core which generates a trace point in the processor core. Step five includes a write command in which node fabric 110 notifies memory mux 108 of the forthcoming data to be written to the HBM. The notification via the write command generates a trace point in node fabric 110, while at step six, completion of the write to HBM also generates a trace point in node fabric 110. At step seven, node fabric 110 performs a write resource update to cause a sync flag update in the processor core which generates a trace point in the processor core (e.g., in FPC 104). In addition to the write resource update, node fabric 110 can perform an acknowledge update ("ack update") where data completion for the DMA operation is signaled back to the processor core. The ack update can generate trace entries that are similar to trace entries generated by the write resource update.

**[0093]** In another example DMA operation, a first trace entry is generated when a DMA instruction is issued in a node fabric 110 of the originating node. Additional trace entries can be generated in node fabric 110 to capture time used to read data for the DMA and writing the data to outgoing queues. In some implementations, node fabric 110 can packetize DMA data

into smaller chunks of data. For data packetized into smaller chunks, read and write trace entries can be produced for a first data chunk and a last data chunk. Optionally, in addition to the first and last data chunks, all data chunks can be set to generate trace entries.

**[0094]** For remote/non-local DMA operations that may require ICI hops, the first data and the last data chunk can generate additional trace entries at ingress and egress points in each intermediate hop along ICI/router 112. When DMA data arrives at a destination node, trace entries similar to the previous node fabric 110 entries are generated (e.g., read/write of first and last data chunks) at the destination node. In some implementations, a final step of the DMA operation can include executed instructions associated with the DMA causing an update to a sync flag at the destination node. When the sync flag is updated a trace entry can be generated indicating completion of the DMA operation.

**[0095]** In some implementations, DMA tracing is initiated by FPC 104, SPC 106, or HIB 114 when in each component is in trace mode such that trace points can be executed. Components of system 100 can enter trace mode based on global controls in FPC 104 or SPC 106 via a trigger mechanism. The trace points trigger in response to the occurrence of a specific action or condition associated with execution of program code by the components of system 100. For example, portions of the program code can include embedded trigger functions that are detectable by at least one hardware component of system 100.

**[0096]** The components of system 100 can be configured to detect a trigger function associated with portions of program code executed by at least one of FPC 104 or SPC 106. In some instances, the trigger function can correspond to at least one of: 1) a particular sequence step in a portion or module of the executed program code; or 2) a particular time parameter indicated by the GTC used by the distributed processor units of system 100.

**[0097]** Responsive to detecting the trigger function, a particular component of system 100 can initiate, trigger, or execute at least one trace point (e.g., a trace event) that causes trace entry data associated with one or more hardware events to be stored in at least one memory buffer of the hardware component. As noted above, stored trace data can then be provided to chip manager 216 by way of at least one trace chain 203, 205, 207.

**[0098]** FIG. 5 is a process flow diagram of an example process 500 for distributed hardware tracing using component features of system 100 and the one or more nodes 200, 201 of system 100. Thus, process 500 can be implemented using one or more of the above-mentioned computing resources of systems 100 including nodes 200, 201.

**[0099]** Process 500 begins at block 502 and includes computing system 100 monitoring execution of program code executed by one or more processor components (including at least FPC 104 and SPC 106). In some implementations, execution of program code that generates tracing activities can be monitored, at least in part, by multiple host systems, or subsystems of a single host system. Hence, in these implementations, system 100 can perform multiple processes 500 relating to analysis of tracing activities for hardware events occurring across

distributed processing units.

**[0100]** A first processor component is configured to execute at least a first portion of the program code that is monitored. At block 504, process 500 includes computing system 100 monitoring execution of program code executed by a second processor component. The second processor component is configured to execute at least a second portion of the program code that is monitored.

**[0101]** Components of computing system 100 can each include at least one memory buffer. Block 506 of process 500 includes system 100 storing data identifying one or more hardware events in the at least one memory buffer of a particular component. In some implementations, the hardware events occur across distributed processor units that include at least the first processor component and the second processor component. The stored data identifying the hardware events can each include a hardware event time stamp and metadata characterizing the hardware event. In some implementations, a collection of hardware events corresponds to a timeline events.

**[0102]** For example, system 100 can store data identifying one or more hardware events that correspond, in part, to movement of data packets between a source hardware component within system 100 and a destination hardware component within system 100. In some implementations, the stored metadata characterizing the hardware event can correspond to at least one of: 1) a source memory address, 2) a destination memory address, 3) a unique trace identification number relating to a trace entry that causes the hardware event to be stored, or 4) a size parameter associated with a direct memory access (DMA) trace entry.

**[0103]** In some implementations, storing data that identifies a collection of hardware events includes storing event data in a memory buffer of FPC 104 and/or SPC 106 that corresponds, for example, to at least one local trace event queue 310. The stored event data can indicate subsets of hardware event data that can be used to generate a larger timeline of hardware events. In some implementations, storing of event data occurs in response to at least one of FPC 104 or SPC 106 executing hardware trace instructions associated with portions of program code executed by components of system 100.

**[0104]** At block 508 of process 500, system 100 generates a data structure, such as structure 320, that identifies one or more hardware events from the collection of hardware events. The data structure arranges the one or more hardware events in a time ordered sequence of events that are associated with at least the first processor component and the second processor component. In some implementations, the data structure identifies a hardware event time stamp for a particular trace event, a source address associated with the trace event, or a memory address associated with the trace event.

**[0105]** At block 510 of process 500, system 100 stores the generated data structure in a memory bank of a host device associated with host system 126. In some implementations, the stored data structure can be used by host system 126 to analyze performance of the program

code executed by at least the first processor component or the second processor component. Likewise, the stored data structure can be used by host system 126 to analyze performance of at least one component of system 100.

**[0106]** For example, the user, or host system 126, can analyze the data structure to detect or determine if there is a performance issue associated with execution of a particular software module within the program code. An example issue can include the software module not completing execution within an allotted execution time window.

**[0107]** Further, the user, or host device 126, can detect or determine if a particular component of system 100 is operating above or below a threshold performance level. An example issue relating to component performance can include a particular hardware component executing certain events but generating result data that is outside acceptable parameter ranges for result data. In some implementations, the result data may not be consistent with result data generated by other related components of system 100 that execute substantially similar operations.

**[0108]** For example, during execution of the program code, a first component of system 100 can be required to complete an operation and to generate a result. Likewise, a second component of system 100 can be required to complete a substantially similar operation and to generate a substantially similar result. Analysis of the generated data structure can indicate that the second component generated a result that is drastically different than the result generated by the first component. Likewise, the data structure may indicate a result parameter value of the second component that is noticeably outside a range of acceptable result parameters. These results can likely indicate a potential performance issue with the second component of system 100.

**[0109]** Embodiments of the subject matter and the functional operations described in this specification can be implemented in digital electronic circuitry, in tangibly-embodied computer software or firmware, in computer hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Embodiments of the subject matter described in this specification can be implemented as one or more computer programs, i.e., one or more modules of computer program instructions encoded on a tangible non transitory program carrier for execution by, or to control the operation of, data processing apparatus. Alternatively, or in addition, the program instructions can be encoded on an artificially generated propagated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal, which is generated to encode information for transmission to suitable receiver apparatus for execution by a data processing apparatus. The computer storage medium can be a machine-readable storage device, a machine-readable storage substrate, a random or serial access memory device, or a combination of one or more of them.

**[0110]** The processes and logic flows described in this specification can be performed by one or more programmable computers executing one or more computer programs to perform

functions by operating on input data and generating output(s). The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array), an ASIC (application specific integrated circuit), or a GPGPU (General purpose graphics processing unit).

**[0111]** Computers suitable for the execution of a computer program include, by way of example, can be based on general or special purpose microprocessors or both, or any other kind of central processing unit. Generally, a central processing unit will receive instructions and data from a read only memory or a random access memory or both. The essential elements of a computer are a central processing unit for performing or executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices.

**[0112]** Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

**[0113]** Particular embodiments of the subject matter have been described. Other embodiments are within the scope of the following claims.

# REFERENCES CITED IN THE DESCRIPTION

Cited references

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

**Patent documents cited in the description**


- US20120226837A1 **[0004]**
- US20050144532A1 **[0005]**
- US20120324290A1 **[0006]**

- US20100106678A1 [0007]
- US2018285233A [0051]

Patentkrav

1. Computerimplementeret fremgangsmåde, der udføres af et computersystem med én eller flere processorer, hvilken fremgangsmåde omfatter:

overvågning af udførelse af programkode ved hjælp af en første processorkomponent (104), idet den første processorkomponent er konfigureret til at udføre i det mindste en første del af programkoden

overvågning af udførelsen af programkoden ved hjælp af en anden processorkomponent (106), idet den anden processorkomponent er konfigureret til at udføre mindst en anden del af programkoden

lagring ved hjælp af computersystemet af data, der identificerer én eller flere hardwarehændelser (322, 324), som finder sted tværs over processorenheder, der omfatter den første processorkomponent og den anden processorkomponent, idet hver hardwarehændelse angiver mindst ét af: datakommunikationer, der er knyttet til en hukommelsesadgangsoperation af programkoden, en udstedt instruktion af programkoden eller en udført instruktion af programkoden, hvor de data, der identificerer hver af den ene eller de flere hardwarehændelser, omfatter et hardwarehændelses-tidsstempel og metadata, der karakteriserer hardwarehændelsen

generering ved hjælp af computersystemet af en datastruktur (320), der identificerer den ene eller de flere hardwarehændelser, idet datastrukturen er konfigureret til at arrangere den ene eller de flere hardwarehændelser i en tidsordnet sekvens af hændelser, der er knyttet til i det mindste den første processorkomponent og den anden processorkomponent, og

lagring ved hjælp af computersystemet af den genererede datastruktur i en hukommelsesbank i en værtsenhed,

kendetegnet ved, at:

datastrukturen omfatter en særlig sporingsidentifikator, ID, der er knyttet til multiple hardwarehændelser, svarende til en særlig hukommelsesadgangsoperation, der

finder sted tværs over processorenhederne, og hvor det særlige sporings-ID anvendes til at korrelere én eller flere hardwarehændelser af de multiple hardwarehændelser.

2. Fremgangsmåde ifølge krav 1, der endvidere omfatter:

detektering ved hjælp af computersystemet af en aktiveringsfunktion, som er knyttet til dele af programkoden, der udføres af den første processorkomponent og/eller den anden processorkomponent, og

som reaktion på detektering af aktiveringsfunktionen, igangsættelse ved hjælp af computersystemet af mindst én sporingshændelse, som medfører, at data, der er knyttet til den ene eller de flere hardwarehændelser, lagres i mindst én hukommelsesbuffer.

3. Fremgangsmåde ifølge krav 2, hvorved aktiveringsfunktionen svarer til et særligt sekvenstrin i programkoden og/eller en særlig tidsparameter, som angives af et globalt tidsur, der anvendes af processorenhederne, og

hvorved igangsættelsen af den mindst ene sporingshændelse omfatter bestemmelse af, at en sporingsbit er sat til en særlig værdi, idet den mindst ene sporingshændelse er knyttet til en hukommelsesadgangsoperation, der omfatter multiple mellemoperationer, som finder sted tværs over processorenhederne, og hvor data, der er knyttet til de multiple mellemoperationer, lagres i én eller flere hukommelsesbuffere som reaktion på bestemmelse af, at sporingsbitten er sat til den særlige værdi.

4. Fremgangsmåde ifølge et hvilket som helst af de foregående krav, hvor lagring af data, der identificerer den ene eller de flere hardwarehændelser, endvidere omfatter:

lagring i en første hukommelsesbuffer i den første processorkomponent af et første undersæt af data, der identificerer hardwarehændelser af den ene eller de flere hardwarehændelser, hvor lagringen finder sted som reaktion på, at den første processorkomponent udfører en

hardwaresporingsinstruktion, der er knyttet til i det mindste den første del af programkoden.

5. Fremgangsmåde ifølge krav 4, hvor lagring af data, der identificerer den ene eller de flere hardwarehændelser, endvidere omfatter:

lagring i en anden hukommelsesbuffer i den anden processorkomponent af et andet undersæt af data, der identificerer hardwarehændelser af den ene eller de flere hardwarehændelser, hvor lagringen finder sted som reaktion på, at den første processorkomponent udfører en hardwaresporingsinstruktion, der er knyttet til i det mindste den anden del af programkoden.

6. Fremgangsmåde ifølge krav 5, hvorved generering af datastrukturen endvidere omfatter:

sammenligning ved hjælp af computersystemet af i det mindste hardwarehændelses-tidsstempler for respektive hændelser i det første undersæt af data, der identificerer hardwarehændelser, med i det mindste hardwarehændelses-tidsstempler for respektive hændelser i det andet undersæt af data, der identificerer hardwarehændelser, og

tilvejebringelse ved hjælp af computersystemet og til præsentation i datastrukturen af et korreleret sæt af hardwarehændelser, der delvist er baseret på sammenligningen mellem de respektive hændelser i det første undersæt og de respektive hændelser i det andet undersæt.

7. System (100) til sporing af fordelt hardware, hvilket system omfatter:

én eller flere processorer (104, 106), der omfatter én eller flere processorkerner

én eller flere maskinlæsbare lagerenheder til at lagre instruktioner, som kan eksekveres af den ene eller de flere processorer til at udføre operationer, der omfatter:

overvågning af udførelse af programkode ved hjælp af en første processorkomponent (104), idet den første processorkomponent er konfigureret til at udføre i det mindste en første del af programkoden

overvågning af udførelsen af programkoden ved hjælp af en anden processorkomponent (106), idet den anden processorkomponent er konfigureret til at udføre mindst en anden del af programkoden

lagring ved hjælp af computersystemet af data, der identificerer én eller flere hardwarehændelser (322, 324), som finder sted tværs over processorenheder, der omfatter den første processorkomponent og den anden processorkomponent, idet hver hardwarehændelse angiver mindst ét af: datakommunikationer, der er knyttet til en hukommelsesadgangsoperation af programkoden, en udstedt instruktion af programkoden eller en udført instruktion af programkoden, hvor de data, der identificerer hver af den ene eller de flere hardwarehændelser, omfatter et hardwarehændelses-tidsstempel og metadata, der karakteriserer hardwarehændelsen

generering ved hjælp af computersystemet af en datastruktur (320), der identificerer den ene eller de flere hardwarehændelser, idet datastrukturen er konfigureret til at arrangere den ene eller de flere hardwarehændelser i en tidsordnet sekvens af hændelser, der er knyttet til i det mindste den første processorkomponent og den anden processorkomponent

lagring ved hjælp af computersystemet af den genererede datastruktur i en hukommelsesbank i en værtsenhed, kendetegnet ved, at:

de instruktioner, der genererer datastrukturen, genererer datastrukturen således, at datastrukturen omfatter en særlig sporingsidentifikator, ID, der er knyttet til multiple hardwarehændelser, svarende til en bestemt hukommelsesadgangsoperation, der finder sted tværs over processorenhederne, og hvor det særlige sporings-ID anvendes til at korrelere én eller flere hardwarehændelser af de multiple hardwarehændelser.

8. System til sporing af fordelt hardware ifølge krav 7, hvor operationerne endvidere omfatter:

detektering ved hjælp af computersystemet af en aktiveringsfunktion, som er knyttet til dele af programkoden, der udføres af den første processorkomponent og/eller den anden processorkomponent, og

som reaktion på detektering af aktiveringsfunktionen, igangsættelse ved hjælp af computersystemet af mindst én sporingshændelse, som medfører, at data, der er knyttet til den ene eller de flere hardwarehændelser, lagres i mindst én hukommelsesbuffer.

9. System til sporing af fordelt hardware ifølge krav 8, hvor aktiveringsfunktionen svarer til mindst ét af: et særligt sekvenstrin i programkoden eller en særlig tidsparameter, som angives af et globalt tidsur, der anvendes af processorenhederne, og

hvorved igangsættelsen af den mindst ene sporingshændelse omfatter bestemmelse af, at en sporingsbit er sat til en særlig værdi, idet den mindst ene sporingshændelse er knyttet til en hukommelsesadgangsoperation, der omfatter multiple mellemoperationer, som finder sted tværs over processorenhederne, og hvor data, der er knyttet til de multiple mellemoperationer, lagres i én eller flere hukommelsesbuffere som reaktion på bestemmelse af, at sporingsbitten er sat til den særlige værdi.

10. System til sporing af fordelt hardware ifølge et hvilket som helst af kravene 7 til 9, hvor lagring af data, der identificerer den ene eller de flere hardwarehændelser, endvidere omfatter:

lagring i en første hukommelsesbuffer i den første processorkomponent af et første undersæt af data, der identificerer hardwarehændelser af den ene eller de flere hardwarehændelser, hvor lagringen finder sted som reaktion på, at den første processorkomponent udfører en

hardwaresporingsinstruktion, der er knyttet til i det mindste den første del af programkoden.

11. System til sporing af fordelt hardware ifølge krav 10, hvor lagring af data, der identificerer den ene eller de flere hardwarehændelser, endvidere omfatter:

lagring i en anden hukommelsesbuffer i den anden processorkomponent af et andet undersæt af data, der identificerer hardwarehændelser af den ene eller de flere hardwarehændelser, hvor lagringen finder sted som reaktion på, at den anden processorkomponent udfører en hardwaresporingsinstruktion, der er knyttet til i det mindste den anden del af programkoden.

12. System til sporing af fordelt hardware ifølge krav 11, hvor generering af datastrukturen endvidere omfatter:

sammenligning ved hjælp af computersystemet af i det mindste hardwarehændelses-tidsstempler for respektive hændelser i det første undersæt af data, der identificerer hardwarehændelser, med i det mindste hardwarehændelses-tidsstempler for respektive hændelser i det andet undersæt af data, der identificerer hardwarehændelser, og

tilvejebringelse ved hjælp af computersystemet og til præsentation i datastrukturen af et korreleret sæt af hardwarehændelser, der delvist er baseret på sammenligningen mellem de respektive hændelser i det første undersæt og de respektive hændelser i det andet undersæt.

13. System til sporing af fordelt hardware ifølge et hvilket som helst af kravene 7 til 12, hvor den genererede datastruktur identificerer mindst én parameter, der angiver en latensattribut for en særlig hardwarehændelse, hvor latensattributten i det mindste angiver en varighed af den bestemte hardwarehændelse.

14. System til sporing af fordelt hardware ifølge et hvilket

som helst af kravene 7 til 13, hvor mindst én processor er en multi-core, multi-node processor med én eller flere behandlingskomponenter, og den ene eller de flere hardwarehændelser delvist svarer til datakommunikation, der finder sted mellem i det mindste den første processorkomponent i en første node og den anden processorkomponent i en anden node.

15. System til sporing af fordelt hardware ifølge et hvilket som helst af kravene 7 til 13, hvor den første processorkomponent og den anden processorkomponent er ét af: en processor, en processorkerne, en hukommelsesadgangsmaskine eller en hardwareegenskab ved computersystemet,

hvor den ene eller de flere hardwarehændelser delvist svarer til bevægelse af datapakker mellem en kilde og en destination, og

hvor metadata, der karakteriserer hardwarehændelsen, svarer til mindst ét af: en kildehukommelsesadresse, en destinationshukommelsesadresse, en entydig sporingshændelsesidentifikations-, ID, -nummer eller en størrelsesparameter, der er knyttet til en anmodning om direkte hukommelsesadgangssporing.

16. Ikke-flygtig computerlagerenhed, der er anbragt i en databehandlingsenhed, og som er kodet med et computerprogram, idet programmet omfatter instruktioner, som, når de udføres af én eller flere processorer, får den ene eller flere processorer til at udføre trinnene i fremgangsmåden ifølge et hvilket som helst af kravene 1 til 6.

# DRAWINGS

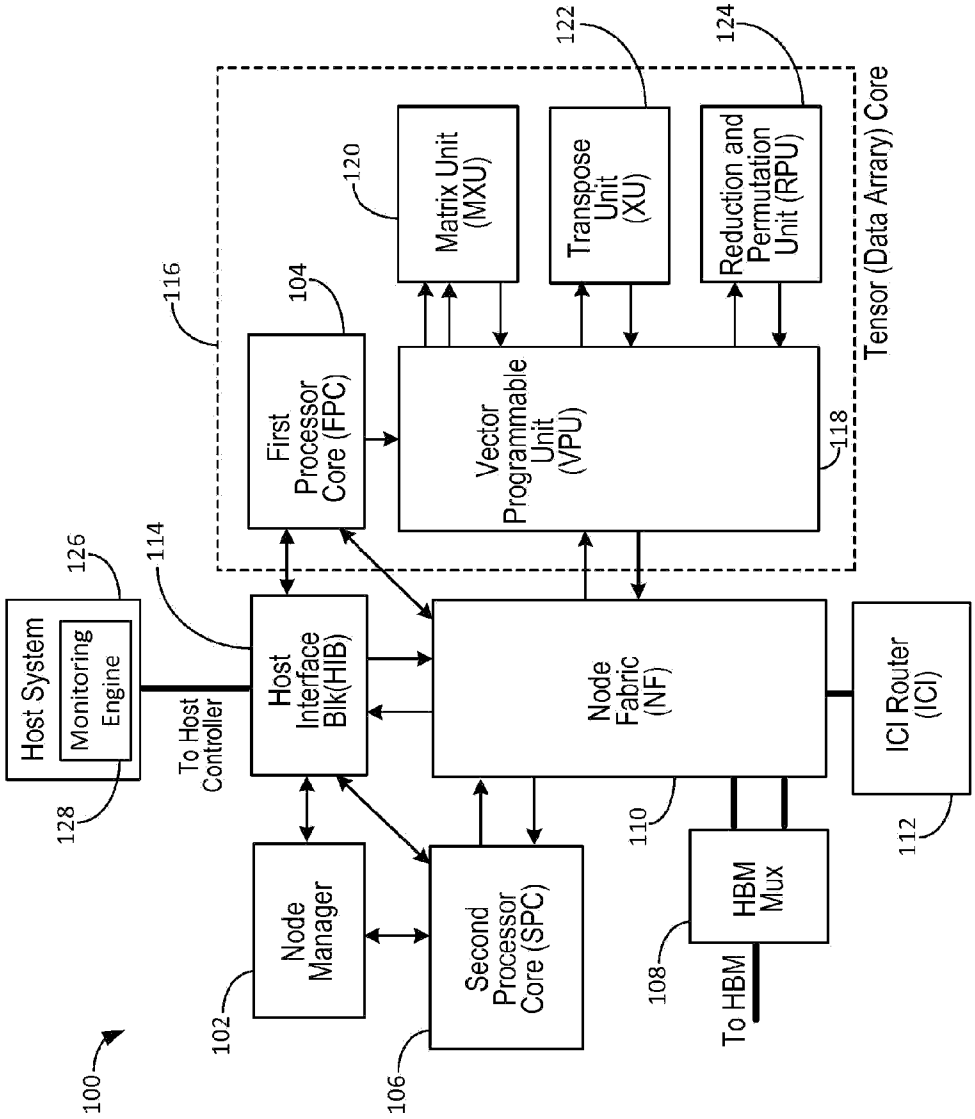



FIG. 1

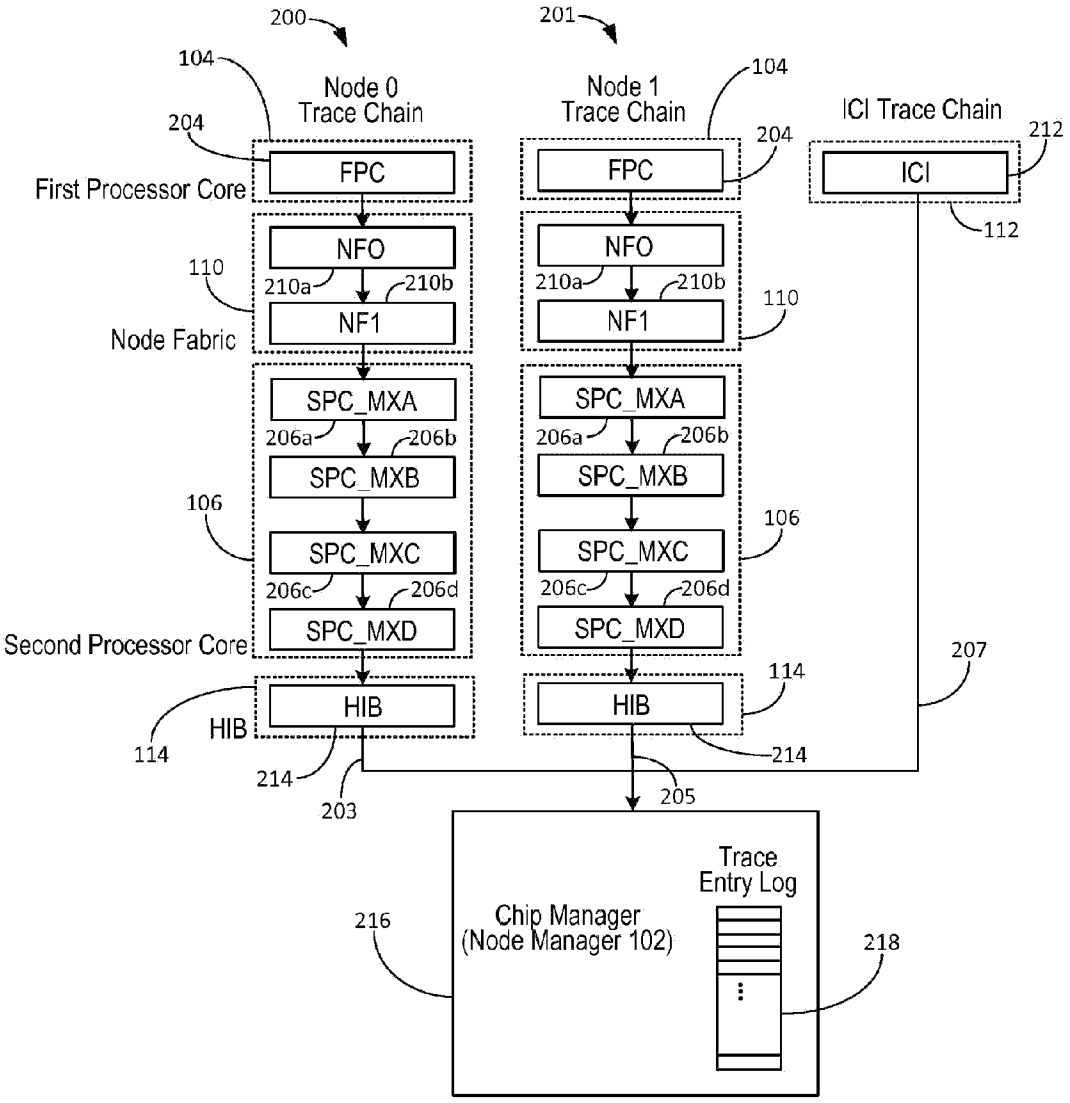
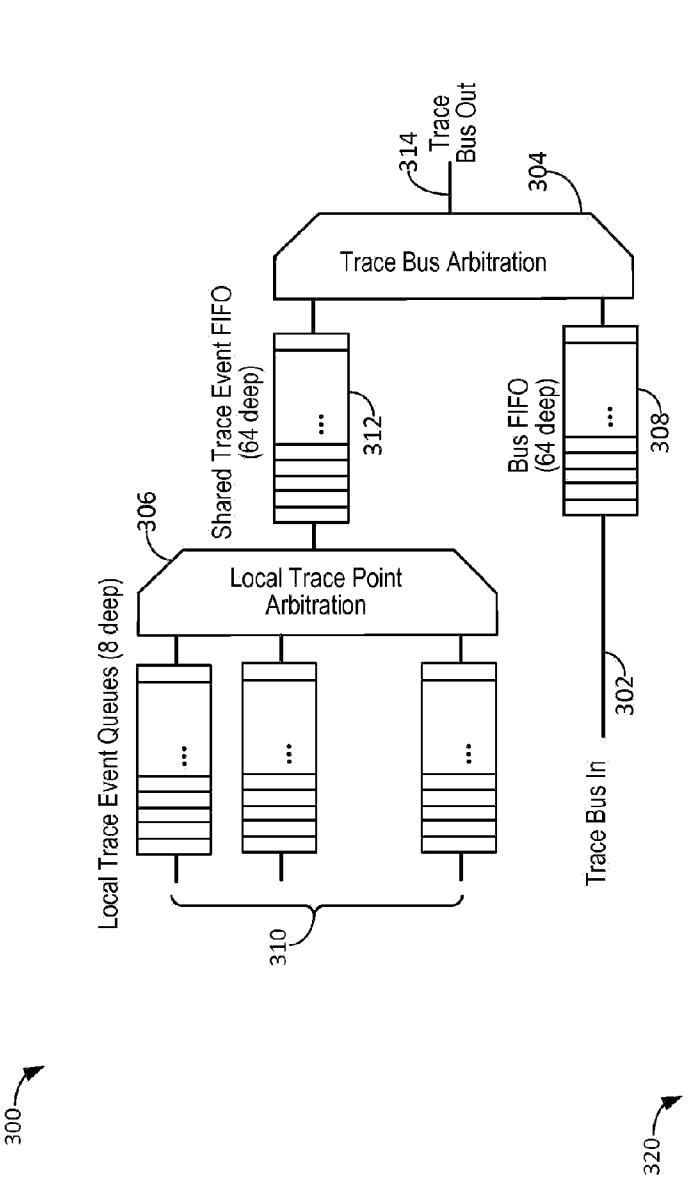

200
201
104
Node 0
Trace Chain
Node 1
Trace Chain
ICI Trace Chain
212
204
FPC
First Processor Core
ICI
112
NFO
110
210a
210b
NF1
Node Fabric
SPC_MXA
206a
206b
SPC_MXB
106
SPC_MXC
206c
206d
SPC_MXD
Second Processor Core
207
114
HIB
214
203
205
Trace
Entry Log
Chip Manager
(Node Manager 102)
216
218

FIG. 2



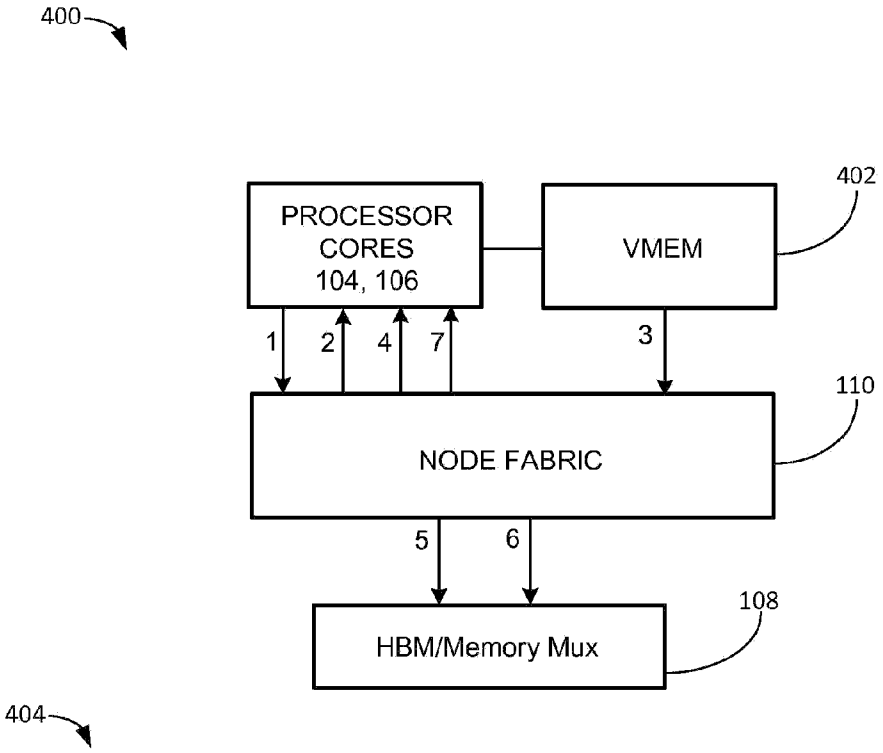

320

| Time | Trace ID | Source | Destination | Component(s) | Trace Event | Size | Latency |
|---|---|---|---|---|---|---|---|
| 09:01.13 | 000001 | 0x00F100 | 0x00F200 | FPC → FPC | DMA | 128 bytes | 9µs |
| 09:05.27 | 000002 | 0x00F300 | 0x00F400 | FPC → SPC | DMA | 256 bytes | .023ms |
| 09:12.35 | 000003 | 0x00F500 | 0x00F600 | SPC → NF | DMA (issued) | 512 bytes | .xxx secs |
| 09:12.49 | 000003 | 0x00F600 | 0x00F700 | NF → HBM | DMA (executed) | 512 bytes | .xxx secs |
| 09:12.58 | 000003 | 0x00F700 | 0x00F800 | NF → FPC | DMA (complt) | 512 bytes | .xxx secs |

322

324

FIG. 3




| STEP | OPERATION |
|---|---|
| 1 | INITIAL DMA REQUEST; TRACEPOINT IN NODE FABRIC |
| 2 | READ CMD: NF ASKS CORE TO TRANSFER DATA; TRACEPOINT IN NF |
| 3 | READ COMPLETE: NO TRACEPOINT IN NF HERE! |
| 4 | READ RESOURCE UPDATE: SYNC FLAG UPDATE IN CORE; TRACEPOINT IN FPC |
| 5 | WRITE CMD: NF NOTIFIES HBM; TRACEPOINT IN NF |
| 6 | WRITE COMPLETE: TRACEPOINT IN NF |
| 7 | WRITE RESOURCE UPDATE: SYNCH FLAG UPDATES IN FPC; TRACEPOINT IN FPC |

FIG. 4

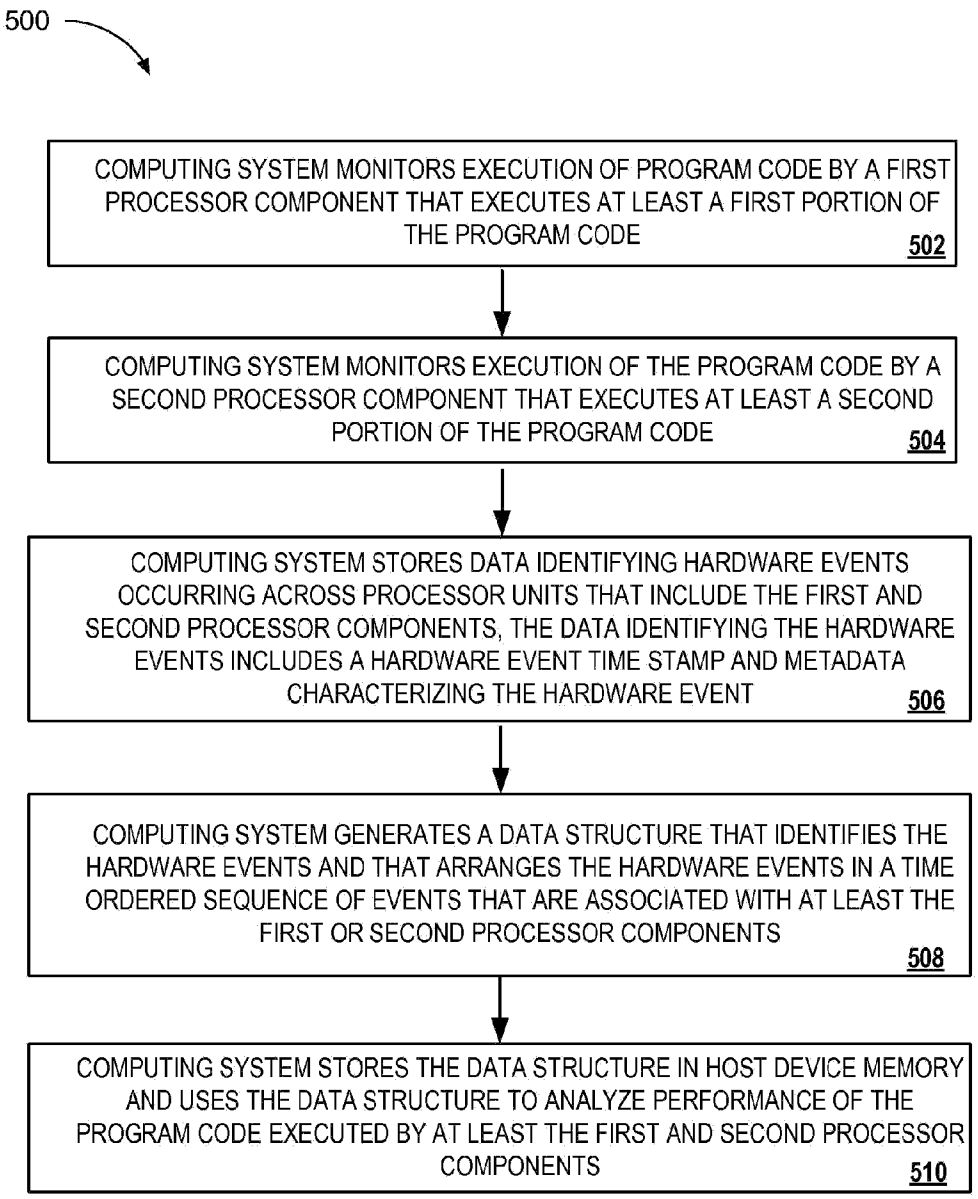

500
COMPUTING SYSTEM MONITORS EXECUTION OF PROGRAM CODE BY A FIRST PROCESSOR COMPONENT THAT EXECUTES AT LEAST A FIRST PORTION OF THE PROGRAM CODE
502
COMPUTING SYSTEM MONITORS EXECUTION OF THE PROGRAM CODE BY A SECOND PROCESSOR COMPONENT THAT EXECUTES AT LEAST A SECOND PORTION OF THE PROGRAM CODE
504
COMPUTING SYSTEM STORES DATA IDENTIFYING HARDWARE EVENTS OCCURRING ACROSS PROCESSOR UNITS THAT INCLUDE THE FIRST AND SECOND PROCESSOR COMPONENTS, THE DATA IDENTIFYING THE HARDWARE EVENTS INCLUDES A HARDWARE EVENT TIME STAMP AND METADATA CHARACTERIZING THE HARDWARE EVENT
506
COMPUTING SYSTEM GENERATES A DATA STRUCTURE THAT IDENTIFIES THE HARDWARE EVENTS AND THAT ARRANGES THE HARDWARE EVENTS IN A TIME ORDERED SEQUENCE OF EVENTS THAT ARE ASSOCIATED WITH AT LEAST THE FIRST OR SECOND PROCESSOR COMPONENTS
508
COMPUTING SYSTEM STORES THE DATA STRUCTURE IN HOST DEVICE MEMORY AND USES THE DATA STRUCTURE TO ANALYZE PERFORMANCE OF THE PROGRAM CODE EXECUTED BY AT LEAST THE FIRST AND SECOND PROCESSOR COMPONENTS
510

FIG. 5