(19) DANMARK

(10) **DK/EP 3602298 T3**

(12) Oversættelse af europæisk patentskrift

Patent- og Varemærkestyrelsen

(51) Int.Cl.: ***G 06 F 9/50 (2006.01)***

(45) Oversættelsen bekendtgjort den: **2025-03-24**

(80) Dato for Den Europæiske Patentmyndigheds bekendtgørelse om meddelelse af patentet: **2025-01-01**

(86) Europæisk ansøgning nr.: **18816396.8**

(86) Europæisk indleveringsdag: **2018-11-20**

(87) Den europæiske ansøgnings publiceringsdag: **2020-02-05**

(86) International ansøgning nr.: **US2018062111**

(87) Internationalt publikationsnr.: **WO2019104087**

(30) Prioritet: **2017-11-21 US 201762589535 P**

(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: **CHENG, Liqun, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA**
GOVINDARAJU, Rama Krishna, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA
ZHU, Haishan, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA
LO, David, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA
RANGANATHAN, Parthasarathy, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA
PATIL, Nishant, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA

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

(54) Benævnelse: **STYRING AF BEHANDLINGSSYSTEMERS EFFEKTIVITET**

(56) Fremdragne publikationer:
US-A1- 2009 276 781
US-A1- 2012 221 874
US-A1- 2013 117 521
US-A1- 2014 108 740
US-A1- 2015 067 691
US-A1- 2015 293 776
US-A1- 2016 103 715
KANG YUNJI ET AL: "Priority-driven spatial resource sharing scheduling for embedded graphics processing units", JOURNAL OF SYSTEMS ARCHITECTURE, vol. 76, 26 April 2017 (2017-04-26), pages 17 - 27, XP085042816, ISSN: 1383-7621, DOI: 10.1016/J.SYSARC.2017.04.002
YIN-CHI PENG ET AL: "Cross-layer dynamic prefetching allocation strategies for high-performance multicores", VLSI DESIGN, AUTOMATION, AND TEST (VLSI-DAT), 2013 INTERNATIONAL SYMPOSIUM ON, IEEE, 22 April 2013 (2013-04-22), pages 1 - 4, XP032427769, ISBN: 978-1-4673-4435-7, DOI: 10.1109/VLDI-DAT.2013.6533864

Fortsættes ...

DESCRIPTION

Description

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Patent Application No. 62/589,535, entitled "Managing Processing System Efficiency," which was filed on November 21, 2017.

BACKGROUND

[0002] This specification relates to improving accelerated resource-intensive computation efficiency.

[0003] Hardware accelerators, such as Graphical Processing Units (GPUs) or Tensor Processing Units (TPUs), have much greater computational capacity compared to general-purpose processors, e.g., traditional Central Processing Units (CPUs). As a result, accelerators have started to drive much of the improvement in performance for critical workloads. For example, accelerators are widely used for machine learning training and inference tasks. US2012221874A1 relates to thread contention in a multicore processor running high priority and low priority programs.

SUMMARY

[0004] The invention is defined by the independent claims. The dependent claims define embodiments thereof.

[0005] This specification describes a system implemented as computer programs on or more computers in one or more locations that manages the efficiency of a processing system that includes multiple general-purpose processing units.

[0006] The system splits a plurality of general-purpose processing units, e.g., CPU cores, into high-priority and low-priority domains. The general-purpose processing units in the high-priority domain are assigned to perform one or more tasks including one or more high-priority tasks, and the general processing units in the low-priority domain are assigned to perform one or more tasks including one or more low-priority tasks. Generally, the processing units in the low-priority domain are not assigned to perform any high-priority tasks. Moreover, the processing

system generally also includes one or more hardware accelerators that are assigned a resource-intensive workload, e.g., a machine learning workload, and the high-priority tasks are tasks that are associated with that resource-intensive workload, i.e., tasks that support the workload assigned to the hardware accelerators.

[0007] During runtime of the processing system, the system obtains memory usage measurements that characterize usage of system memory by the high-priority domain and the low-priority domain. Based on the memory usage measurements, the system adjusts a configuration of (i) the high-priority domain, (ii) the low-priority domain, or (iii) both to adjust utilization of the system memory by the general-purpose processing units.

[0008] The system repeatedly obtains the usage measurements and adjusts the configurations during runtime to increase the efficiency of the processing system.

[0009] Particular embodiments of the subject matter described in this specification can be implemented so as to realize one or more of the following advantages. While hardware accelerators are responsible for the most heavily computational tasks in resource-intensive computation, general purpose processors, e.g., CPUs, often perform various supporting roles. For example, in a large-scale distributed machine learning system, CPUs may perform the supporting role of collecting and synchronizing machine learning model parameters. The supporting role of CPUs, however, may degrade system performance efficiency by competing with accelerators for shared resources, such as system memory. The described technology splits a plurality of general-purpose processing units into high-priority and low-priority domains, memory requests within each subdomain are handled by the corresponding memory controller and enjoy both lower memory latency and cache latency. Moreover, by filling the high-priority domain with low-priority CPU tasks, lost throughput due to fragmentation in domain-partitioning can be regained. Furthermore, by comparing measurements from performance counters during runtime, the system can choose to boost, throttle, or keep the resource configuration to reduce resource contention within and between high-priority and low-priority domains.

[0010] The details of one or more embodiments of the subject matter of this specification are set forth in the accompanying drawing and description below. Other features, aspects, and advantages of the subject matter will become apparent from the description, the drawings, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011]

FIG. 1 is a diagram of an example processing system designed to manage resource-intensive computation.

FIG. 2 is a flowchart of an example process for managing resources on multiple processing

units.

FIG. 3 is a flowchart of an example software logic for configuring resources on multipe processing units.

[0012] Specific embodiments of the invention will now be described in detail with reference to the accompanying figures. Like elements in the various figures are denoted by like reference numerals for consistency.

DETAILED DESCRIPTION

[0013] FIG. 1 is a diagram of an example system 100 according to the invention that manages resource-intensive computation. The system 100 includes an accelerator package 103 designed to handle high-priority workloads such as machine learning tasks, and a processor package 102 designed to handle low-priority workloads such as CPU tasks. For example, a CPU task includes collecting the shared gradients from multiple accelerator packages. The accelerator package 103 is communicably coupled to the processor package 102 using one or more interfaces 112a and 112b. An optimization runtime system 120 manages the processor package 102 to improve its computation efficiency. For example, the optimization runtime system 120 is a set of computer programs running on a computer system including the processor package 102 and the accelerator package 103.

[0014] The accelerator package 103 includes an accelerator engine 114 that performs the intensive computation associated with high-priority workloads. For example, the accelerator engine 114 can be a TPU or a GPU and the computation in the high-priority workload involves the training of a deep neural network, e.g., to repeatedly compute gradients of an objective function being used to train the neural network, or performing inference using the deep neural network, i.e., generating outputs using the deep neural network after the neural network has been trained.

[0015] The processor package 102 includes 102 includes cores 104a-104d responsible for performing computations, last-level-caches (LLCs) 106a-106d that store data for the computations, an interconnect 108a that connects different processing cores and LLCs, and memory controllers 1lOa-l lOb. While the processor package 102 mostly handles low-priority workloads, part of the computation from the high-priority workloads, e.g., memory -intensive computations, still runs on the processor package 102. For example, the processor package 102 can play a supportive role of acting as a parameter server during the training of the neural network. As a parameter server, the processor package 102 during a machine learning task can collect shared gradients from multiple accelerator packages, can aggregate computed gradients, can update the parameters in real-time using the gradients, and then provide the updated parameter values to the accelerator packages. In another example, the processor

packages 102 can perform an in-feed operation, in which the processor package interprets and reshapes input data before sending the data to the accelerator package 103. In another example, the processor package 102 can handle irregular and complex supporting tasks such as beam search in machine translation applications.

[0016] As a result, in practice, the processor package 102 handles both low-priority tasks and certain parts of high-priority tasks. The low-priority tasks can interfere with the high-priority tasks by contending for shared resources such as in-pipeline resources, private caches shared through simultaneous multi-threading, last-level cache, and main memory bandwidth. To reduce performance bottlenecks, the optimization runtime system 120 splits the processor package 102 into a low-priority domain 126 and a high-priority domain 124.

[0017] Each domain has its dedicated processing units, memory, and memory controllers. For example, the high-priority domain 124 includes dedicated cores l04a and l04b, dedicated LLCs l06a and l06b, and a dedicated memory controller llOa. The low-priority domain 125 includes dedicated cores l04c and l04d, dedicated LLCs l06c and l06d, and a dedicated memory controller 1lOb. For example, the optimization runtime system 120 can use Non-uniform memory access (NUMA) subdomain performance isolation technique to split the processor package 102. As a result, the processor package 102 is exposed to an operating system running in a computer unit including the system 100 as two NUMA domains, e.g., the high-priority domain 124 and the low-priority domain 126. Example techniques to implement NUMA subdomain performance isolation include sub-NUMA Clustering (SNC), Cluster-on-Die (CoD), and so on. A control groups interface 122 monitors, controls, and manages different groups of processes and their resource usages in the subdomains. Memory controllers 1lOa and 1lOb handle memory requests within each NUMA subdomain respectively. As a result, local memory requests experience both lower LLC and memory latency.

[0018] In an implementation according to the invention although the high-priority domain 124 has been isolated from the low-priority domain 126, low-priority tasks can still interfere with the high-priority tasks due to a phenomenon called shared memory backpressure. Shared memory backpressure occurs when low-priority tasks in the low-priority domain 126 generate a large amount of memory traffic and saturate the corresponding memory controller 1lOb's bandwidth. In response, the memory controller 1lOb broadcasts a distress signal to all the cores 104a-104d across the processor package. When the cores 104a-104d receive the distress signal from the memory controller 1lO0b, they become throttled in order to avoid congesting the interconnect l08a. This mechanism is detrimental to the domain-splitting technique described above as each subdomain, e.g., the low-priority subdomain and the high-priority subdomain, already routes memory traffic internally. The memory saturation in the low-priority domain 126 itself has only minimal impact on the memory use in the high-priority domain 124, but the shared memory backpressure causes the cores 104a- 104b in the high-priority domain 124 to be throttled nevertheless. As a result, the shared memory backpressure reduces the effectiveness of the memory interference protection implemented by the domain-splitting technique.

[0019] In an implementation according to the invention to reduce the effect of shared memory backpressure, the optimization runtime system 120 repeatedly measures the level of memory saturation in the low-priority domain 126, the high-priority domain 124, and/or the processor package 102 and, when appropriate, performs some actions to reduce the undesirable effects.

[0020] The optimization runtime system 120 uses existing hardware performance monitoring infrastructure such as measurements from the performance event FAST_ASSERTED from the Intel Uncore LLC coherence engine. This performance event reports the number of cycles in which the distress signal is asserted. The optimization runtime system 120 quantifies the memory saturation by dividing this cycle number by the number of total elapsed cycles between two measurements. The optimization runtime system 120 then disables cache prefetching for low-priority tasks in the low-priority domain 126 to reduce memory traffic. This disabling causes performance loss of low-priority tasks, but maintains performance in the high-priority domain 124.

[0021] In some implementations, the optimization runtime system 120 backfills the high-priority domain 124 with low-priority tasks to improve system throughput. For example, the optimization runtime system 120 can be scheduled to run with the node-level scheduler runtime to gather necessary task information such as job priority and profile in both the high-priority domain 124 and the low-priority domain 126. The optimization runtime system 120 assigns both high-priority tasks and low-priority tasks to designated domains, with low-priority tasks prioritized to be assigned to the low-priority domain 126 and high-priority tasks exclusively assigned to the high-priority domain 124.

[0022] When a task is first scheduled on the processor package 102, the optimization runtime system 120 receives high and low watermarks for each measurement of the task. The optimization runtime system 120 makes different measurements at specified time intervals, including:

1. 1. Socket-level memory bandwidth
2. 2. Socket-level memory latency
3. 3. Socket-level memory saturation
4. 4. High-priority domain memory bandwidth

[0023] Where "socket-level" indicates that the measurements are taken across the entire processor package 102. By comparing the measurements with the watermarks specified in the task profile, the optimization runtime system 120 can choose to boost, throttle, or keep the resource configuration for low-priority tasks in each domain. FIG. 3 and the related descriptions explain in detail the node-level resource management logic used by the optimization runtime system 120.

[0024] In summary, the optimization runtime system 120 operates when the processor package 102 are assigned both high-priority tasks and low-priority tasks. The optimization

runtime system 120 improves the performance of the processor package 102 by redistributing computing resources between the high-priority tasks and the low-priority tasks. As a result, the high-priority tasks are isolated from interference by the low-priority tasks, e.g., such as memory interference.

[0025] FIG. 2 is a flowchart of an example process 200 according to the invention for managing resources on multiple processing units. For convenience, the process 200 will be described as being performed by a system, e.g., the optimization runtime system 120 of in FIG. 1.

[0026] The system can perform the process 200 to configure resources on multiple processing units, e.g., the processor package 102, to improve performance for both high-priority and low-priority tasks.

[0027] As the first step, the system splits the multiple processing units into a high-priority domain and a low-priority domain (210). As described in FIG. 1, the system assigns both high-priority tasks and low-priority tasks to the high-priority domain, and assigns only low-priority tasks to the low-priority domain. Low-priority tasks are prioritized to be assigned to the low-priority domain. Example high-priority tasks include machine learning tasks, and example low-priority tasks include CPU tasks.

[0028] The system then obtains shared system resource usage measurements across the high-priority and the low-priority domains (220). The system can make four types of measurements across the multiple processing units, including (1) socket-level memory bandwidth, (2) socket-level memory latency, (3) socket-level memory saturation, and (4) high-priority domain memory bandwidth. The system can take the measurement at a specified time interval to cause negligible performance overhead, e.g., every 10 seconds.

[0029] In some implementations, the system has previously collected a task profile when the task is first loaded onto the multiple processing units. The task profile includes high and low watermarks for each of the above-mentioned measurements.

[0030] By comparing the real-time measurement against the high and low watermarks, the system detects potential performance bottlenecks and configures the memory usage by the high-priority domain (230) and by the low priority domain (240). The system can disable or enable cache prefetching for processing cores in the low-priority domain, and can activate or deactivate processing cores in both domains. Configuring the high-priority and low-priority domain is described below with reference to FIG. 3.

[0031] The system repeatedly performs steps 220-240 during the performance of the task to improve overall system performance.

[0032] FIG. 3 is a flowchart of an example software logic 300 for configuring resource on multiple processing units. For convenience, the software logic 300 is described as being

performed by a system, e.g., the optimization runtime system 120 of FIG. 1.

[0033] As described in FIG. 2, after the system compares the real-time measurements against the high and low watermarks of the running tasks, the system configures resources on the processing unit to reduce performance bottlenecks.

[0034] The system measures socket-level memory latency, socket-level memory bandwidth, socket-level memory saturation, and high-priority domain memory bandwidth. By comparing the measurements against the high and low watermarks, the system determines whether the current measurements are "high" or "low." For example, the system can determine that a measured value being greater than 90% of the high watermark to be "high," and being smaller than 10% of the low watermark to be "low." The system configures the resources on the multiple processing units based on the following rules:

1. 1) If either the high-priority domain memory bandwidth or the socket-level memory latency is high, then the system throttles the high-priority domain (302).
2. 2) If both the high-priority domain memory bandwidth and the socket-level memory latency are low, then the system boosts the high-priority domain (304).
3. 3) If any of the three socket-level measurements is high, then the system throttles the low-priority domain (306).
4. 4) If all three socket-level measurements are low, then the system boosts the low-priority domain (308).

[0035] To throttle or boost the high-priority domain, the system increases or reduces the number of cores in the high-priority domain, respectively. To throttle or boost the low-priority domain, the system increases or reduces the number of cores in the low-priority domain, respectively, and increase or reduce the number of cores using prefetching in the low-priority domain, respectively.

[0036] To throttle the high-priority domain, the system checks if the number of cores operating in the high-priority domain is greater than a minimum number of cores, e.g., as defined in the corresponding task profile (303). If so, the system reduces the number of operating cores in the high-priority domain by one.

[0037] To boost the high-priority domain, the system checks if the number of cores operating in the high-priority domain is smaller than a maximum number of cores, e.g., as defined in the corresponding task profile (305). If so, the system increases the number of operating cores in the high-priority domain by one.

[0038] To throttle the low-priority domain, the system checks if the number of cores using prefetching in the low-priority domain is greater than zero (307a). If so, the system closes half of the prefetching cores in the low-priority domain. Furthermore, if the number of operating cores in the low-priority domain is greater than a minimum number of cores (307b), the system

reduces the number of operating cores in the low-priority domain by one.

[0039] To boost the low-priority domain, the system checks if the number of prefetching cores is smaller than the number of operating cores in the low-priority domain (309a). If so, the system increases the number of prefetching cores in the low-priority domain by one. Furthermore, the system checks if the number of operating cores is smaller than the maximum number of cores in the low-priority domain (309b). If so, the system increases the number of operating cores in the low-priority domain by one.

[0040] The system is more aggressive in disabling prefetching cores (closing half of the cores in throttle mode but only increase one core in boost mode) in order to prioritize high-priority task performance.

[0041] . This specification uses the term "configured" in connection with systems and computer program components. For a system of one or more computers to be configured to perform particular operations or actions means that the system has installed on its software, firmware, hardware, or a combination of them that in operation cause the system to perform the operations or actions. For one or more computer programs to be configured to perform particular operations or actions means that the one or more programs include instructions that, when executed by data processing apparatus, cause the apparatus to perform the operations or actions.

[0042] 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 nontransitory storage medium for execution by, or to control the operation of, 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. 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, that is generated to encode information for transmission to suitable receiver apparatus for execution by a data processing apparatus.

[0043] The term "data processing apparatus" refers to data processing hardware and encompasses all kinds of apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can also be, or further include, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit). The apparatus can optionally include, in addition to hardware, code that creates an execution environment for computer programs, e.g., code that constitutes processor firmware, a protocol

stack, a database management system, an operating system, or a combination of one or more of them.

[0044] A computer program, which may also be referred to or described as a program, software, a software application, an app, a module, a software module, a script, or code, can be written in any form of programming language, including compiled or interpreted languages, or declarative or procedural languages; and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or another unit suitable for use in a computing environment. A program may, but need not, correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data, e.g., one or more scripts stored in a markup language document, in a single file dedicated to the program in question, or in multiple coordinated files, e.g., files that store one or more modules, subprograms, or portions of code. A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a data communication network.

[0045] In this specification, the term "database" is used broadly to refer to any collection of data: the data does not need to be structured in any particular way, or structured at all, and it can be stored on storage devices in one or more locations. Thus, for example, the index database can include multiple collections of data, each of which may be organized and accessed differently.

[0046] Similarly, in this specification, the term "engine" is used broadly to refer to a software-based system, subsystem, or process that is programmed to perform one or more specific functions. Generally, an engine will be implemented as one or more software modules or components, installed on one or more computers in one or more locations. In some cases, one or more computers will be dedicated to a particular engine; in other cases, multiple engines can be installed and running on the same computer or computers.

[0047] The processes and logic flow 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. The processes and logic flows can also be performed by special purpose logic circuitry, e.g., an FPGA or an ASIC, or by a combination of special purpose logic circuitry and one or more programmed computers.

[0048] Computers suitable for the execution of a computer program 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. The central processing unit and the memory can be supplemented by, or incorporated in, special purpose logic circuitry. 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. Moreover, a computer can be embedded in another device, e.g., a mobile telephone, a personal digital assistant (PDA), a mobile audio or video player, a game console, a Global Positioning System (GPS) receiver, or a portable storage device, e.g., a universal serial bus (USB) flash drive, to name just a few.

[0049] 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; magneto-optical disks; and CD ROM and DVD-ROM disks.

[0050] To provide for interaction with a user, embodiments of the subject matter described in this specification can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input. In addition, a computer can interact with a user by sending documents to and receiving documents from a device that is used by the user; for example, by sending web pages to a web browser on a user's device in response to requests received from the web browser. Also, a computer can interact with a user by sending text messages or other forms of a message to a personal device, e.g., a smartphone that is running a messaging application and receiving responsive messages from the user in return.

[0051] Data processing apparatus for implementing machine learning models can also include, for example, special-purpose hardware accelerator units for processing common and compute-intensive parts of machine learning training or production, i.e., inference, workloads.

[0052] Machine learning models can be implemented and deployed using a machine learning framework, e.g., a TensorFlow framework, a Microsoft Cognitive Toolkit framework, an Apache Singa framework, or an Apache MXNet framework.

[0053] Embodiments of the subject matter described in this specification can be implemented in a computing system that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface, a web browser, or an app through which a user can interact with an implementation of the subject matter described in this specification, or any combination of one or more such back-end, middleware, or front-end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network (LAN) and a wide area network (WAN), e.g., the Internet.

[0054] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other. In some embodiments, a server transmits data, e.g., an HTML page, to a user device, e.g., for purposes of displaying data to and receiving user input from a user interacting with the device, which acts as a client. Data generated at the user device, e.g., a result of the user interaction, can be received at the server from the device.

[0055] While this specification contains many specific implementation details, these should not be construed as limitations on the scope of any invention or on the scope of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular inventions. Certain features that are described in this specification in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination.

[0056] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system modules and components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

[0057] Particular embodiments of the subject matter have been described.

[0058] As one example, the processes depicted in the accompanying figures do not necessarily require the particular order shown, or sequential order, to achieve desirable results. In some cases, multitasking and parallel processing may be advantageous.

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

- US62589535 [0001]
- US2012221874A1 [0003]

Patentkrav

1. Fremgangsmåde (200), der omfatter:
opnåelse (210) af data, der opsplitter en flerhed af generelformålsbehandlingsenheder (104a-104d) i en processorpakke (102) i et behandlingssystem (100) i et højprioritetsdomæne (124) og et lavprioritetsdomæne (126), hvor hvert domæne er associeret med en tilhørende hukommelsesstyreenhed (110a, 110b),
hvor behandlingssystemet (100) yderligere omfatter en acceleratorpakke (103), der omfatter en flerhed af specialformåls-hardware-acceleratorer, der er tildelt udførelse af en accelereret arbejdsbelastning (105), hvor
generelformålsbehandlingsenhederne (104a-104d) i højprioritetsdomænet (124) er tildelt udførelse af opgaver, der omfatter én eller flere højprioritetsopgaver og én eller flere lavprioritetsopgaver,
generelformålsbehandlingsenhederne (104a-104d) i lavprioritetsdomænet (126) kun er tildelt udførelse af lavprioritetsopgaver, og
hvor højprioritetsopgaverne er opgaver, der understøtter den accelererede arbejdsbelastning (105);
opnåelse (220), under behandlingssystemets (100) kørselstid, af målinger af brug af delt hukommelse, som kendetegner anvendelse af systemhukommelse, som deles af højprioritetsdomænet (124) og lavprioritetsdomænet (126); og
justering (230, 240), baseret på målingerne af brug af delt hukommelse, af en konfiguration af (i) højprioritetsdomænet, (ii) lavprioritetsdomænet eller (iii) begge med henblik på at justere udnyttelse af systemhukommelsen, der deles af generelformålsbehandlingsenhederne, som kører lavprioritetsopgaver;
hvor målingerne af brug af delt hukommelse omfatter en måling af systemhukommelsesmætning, herunder:
udsendelse af et nødsignal til alle generelformålsbehandlingsenhederne (104a-104d) i processorpakken (102), når lavprioritetsopgaver i lavprioritetsdomænet genererer hukommelsestrafik, der mætter den tilsvarende hukommelsesstyreenheds (110b) båndbredde;
anvendelse af en performance-monitoreringsinfrastruktur til at rapportere et antal cykler, i hvilke nødsignalet gøres gældende;
kvantificering af systemhukommelsesmætningen ved division af antallet af cykler med et samlet antal cykler, der er forløbet mellem to målinger;

bestemmelse af, at systemhukommelsesmætningen er høj, når målingerne af brug af delt hukommelse overstiger et vandmærke for høj systemhukommelsesmætning; og reduktion (306), som reaktion på bestemmelse af, at systemhukommelsesmætningen er høj, af et antal af aktive generelformålsprocessorer under anvendelse af cache-prefetching i lavprioritetsdomænet (126).

2. Fremgangsmåde ifølge krav 1, hvor den accelererede arbejdsbelastning (105) er en maskinindlæringsarbejdsbelastning.

3. Fremgangsmåde ifølge krav 1, hvor målingerne af brug af delt hukommelse omfatter en måling af systemhukommelsesbåndbredde.

4. Fremgangsmåde ifølge krav 3, hvor justeringen omfatter:
bestemmelse af, hvorvidt målingen af systemhukommelsesbåndbredde er høj eller lav, ved sammenligning af målingen af systemhukommelsesbåndbredde med vandmærker for høj og lav systemhukommelsesbåndbredde;
reduktion af antallet af aktive generelformålsprocessorer i lavprioritetsdomænet (126), når målingen af systemhukommelsesbåndbredde er høj; eller
forøgelse af antallet af aktive generelformålsprocessorer i lavprioritetsdomænet (126), når målingen af systemhukommelsesbåndbredde er lav.

5. Fremgangsmåde ifølge et hvilket som helst af kravene 1-4, hvor målingerne af brug af delt hukommelse omfatter en måling af systemhukommelsesventetid.

6. Fremgangsmåde ifølge krav 5, hvor justeringen omfatter:
bestemmelse af, hvorvidt en måling af systemhukommelsesventetid er høj eller lav, ved sammenligning af målingen af systemhukommelsesventetiden med vandmærker for høj og lav systemhukommelsesventetid;
reduktion af antallet af aktive generelformålsprocessorer i lavprioritetsdomænet (126), når målingen af systemhukommelsesventetid er høj; eller
forøgelse af antallet af aktive generelformålsprocessorer i lavprioritetsdomænet (126), når målingen af systemhukommelsesventetid er lav.

7. Fremgangsmåde ifølge krav 1-6, hvor justeringen omfatter:

bestemmelse af, at systemhukommelsesmætningen er lav, når målingerne af brug af delt hukommelse er under et vandmærke for lav systemhukommelsemætning;
forøgelse, som reaktion på bestemmelse af, at systemhukommelsesmætningen er lav, af antallet af aktive generelformålsprocessorer i lavprioritetsdomænet (126), når systemhukommelsesmætningen er lav.

8. Fremgangsmåde ifølge et hvilket som helst af kravene 1-7, hvor målingerne af brug af delt hukommelse omfatter en måling af højprioritetsdomænehukommelsesbåndbredde.

9. Fremgangsmåde ifølge krav 8, hvor justeringen omfatter:
bestemmelse af, hvorvidt målingen af højprioritetsdomænehukommelsesbåndbredde er høj eller lav, ved sammenligning af målingen af højprioritetsdomænehukommelsesbåndbredde med vandmærker for høj og lav højprioritetsdomænehukommelsesbåndbredde;
reduktion af antallet af aktive generelformålsprocessorer i højprioritetsdomænet (124), når målingen af højprioritetsdomænehukommelsesbåndbredde er høj; eller
forøgelse af antallet af aktive generelformålsprocessorer i højprioritetsdomænet (124), når målingen af højprioritetsdomænehukommelsesbåndbredde er lav.

10. Fremgangsmåde ifølge et hvilket som helst af kravene 1-9, hvor justeringen omfatter:
bestemmelse af, hvorvidt en måling af systemhukommelsesbåndbredde er høj eller lav, ved sammenligning af målingen af systemhukommelsesbåndbredde med vandmærker for høj og lav systemhukommelsesbåndbredde;
bestemmelse af, hvorvidt en måling af systemhukommelsesventetid er høj eller lav, ved sammenligning af målingen af systemhukommelsesventetiden med vandmærker for høj og lav systemhukommelsesventetid;
reduktion af antallet af aktive generelformålsprocessorer i lavprioritetsdomænet (126), når målinger af systemhukommelsesbåndbredde eller systemhukommelsesventetid er høj, og
forøgelse af antallet af aktive generelformålsprocessorer i lavprioritetsdomænet (126), når målingerne af systemhukommelsesbåndbredde og systemhukommelsesventetid er lav.

11. Fremgangsmåde ifølge et hvilket som helst af kravene 1-10, hvor justeringen omfatter:
bestemmelse af, hvorvidt en måling af systemhukommelsesbåndbredde er høj eller lav, ved sammenligning af målingen af systemhukommelsesbåndbredde med vandmærker for høj og lav systemhukommelsesbåndbredde;
bestemmelse af, hvorvidt en måling af systemhukommelsesventetid er høj eller lav, ved sammenligning af målingen af systemhukommelsesventetiden med vandmærker for høj og lav systemhukommelsesventetid;
reduktion af antallet af aktive generelformålsprocessorer under anvendelse af cache-prefetching i lavprioritetsdomænet (126), når målingerne af systemhukommelsesbåndbredde eller systemhukommelsesventetid er høj, og
forøgelse af antallet af aktive generelformålsprocessorer under anvendelse af cache-prefetching i lavprioritetsdomænet (126), når målingerne af systemhukommelsesbåndbredde og systemhukommelsesventetid er lav.

12. Fremgangsmåde ifølge et hvilket som helst af kravene 1-11, hvor justeringen omfatter:
bestemmelse af, hvorvidt en måling af højprioritetsdomænehukommelsesbåndbredde er høj eller lav, ved sammenligning af målingen af højprioritetsdomænehukommelsesbåndbredde med vandmærker for høj og lav højprioritetsdomænehukommelsesbåndbredde;
bestemmelse af, hvorvidt en måling af systemhukommelsesventetid er høj eller lav, ved sammenligning af målingen af systemhukommelsesventetiden med vandmærker for høj og lav systemhukommelsesventetid;
reduktion af antallet af aktive generelformålsprocessorer i højprioritetsdomænet (124), når målingerne af højprioritetsdomænehukommelsesbåndbredde eller systemhukommelsesventetid er høj, og
forøgelse af antallet af aktive generelformålsprocessorer i højprioritetsdomænet (124), når højprioritetsdomænehukommelsesbåndbredde og systemhukommelsesventetid er lav.

13. System (120), der omfatter
en flerhed af generelformålsbehandlingsenheder (104a-104d) i en processorpakke i

et behandlingssystem,
en acceleratorpakke (103), der omfatter en flerhed af specialformåls-hardware-acceleratorer, og
én eller flere lagerenheder, der lagrer instruktioner, som ved afvikling får flerheden af generelformålsbehandlingsenheder (104a-104d) og acceleratorpakken (103) til at udføre operationerne ved den tilhørende fremgangsmåde ifølge et hvilket som helst af kravene 1-12.

14. Én eller flere computerlagermedier, der lagrer instruktioner, som ved afvikling med én eller flere computere, hvor den ene eller flere computere omfatter en flerhed af generelformålsbehandlingsenheder (104a-104d) i en processorpakke i et behandlingssystem og en acceleratorpakke (103), der omfatter en flerhed af specialformåls-hardware-acceleratorer, får den ene eller flere computere til at udføre operationerne ved den tilhørende fremgangsmåde ifølge et hvilket som helst af kravene 1-12.

DRAWINGS

Drawing

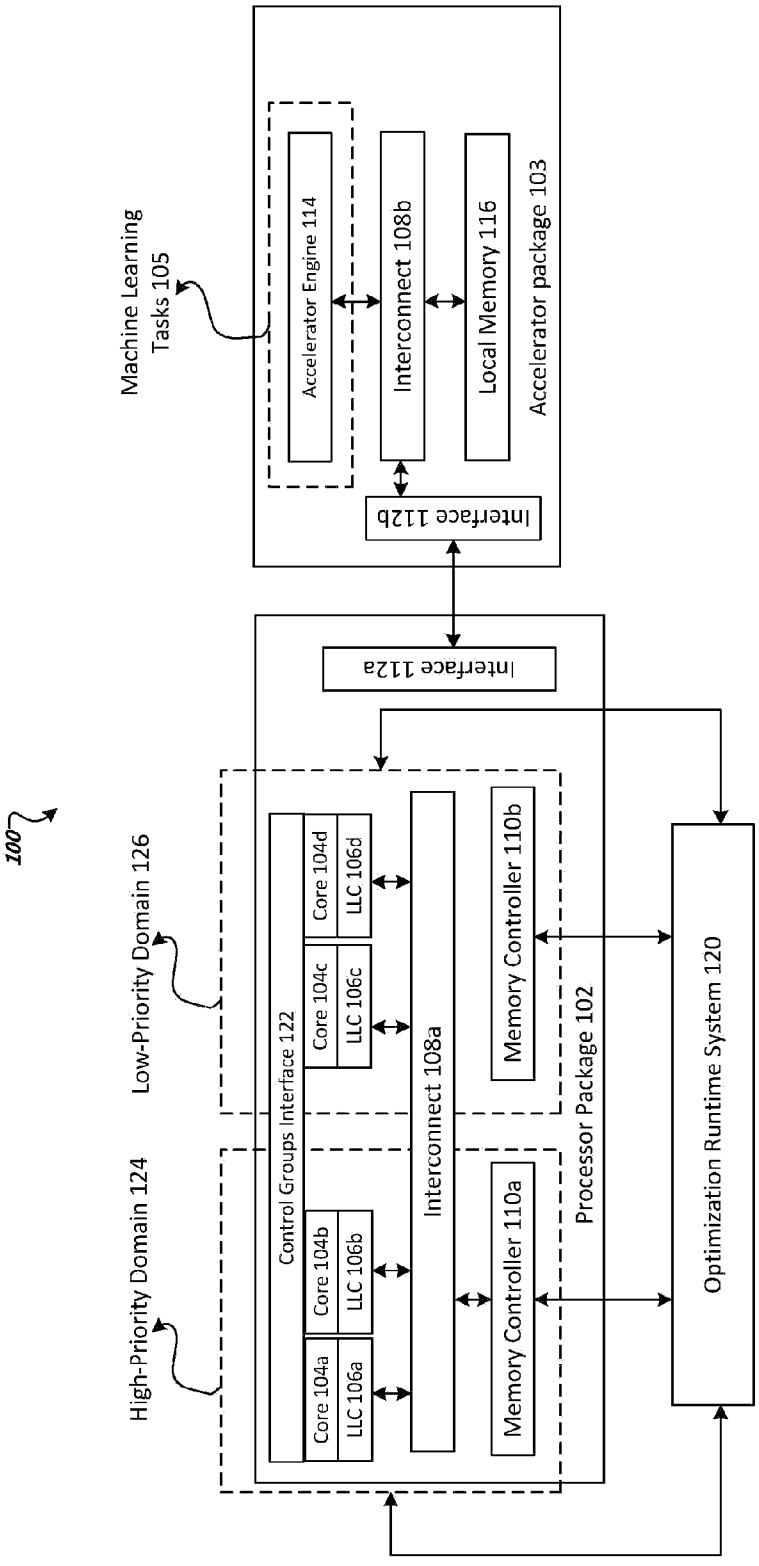

FIG. 1

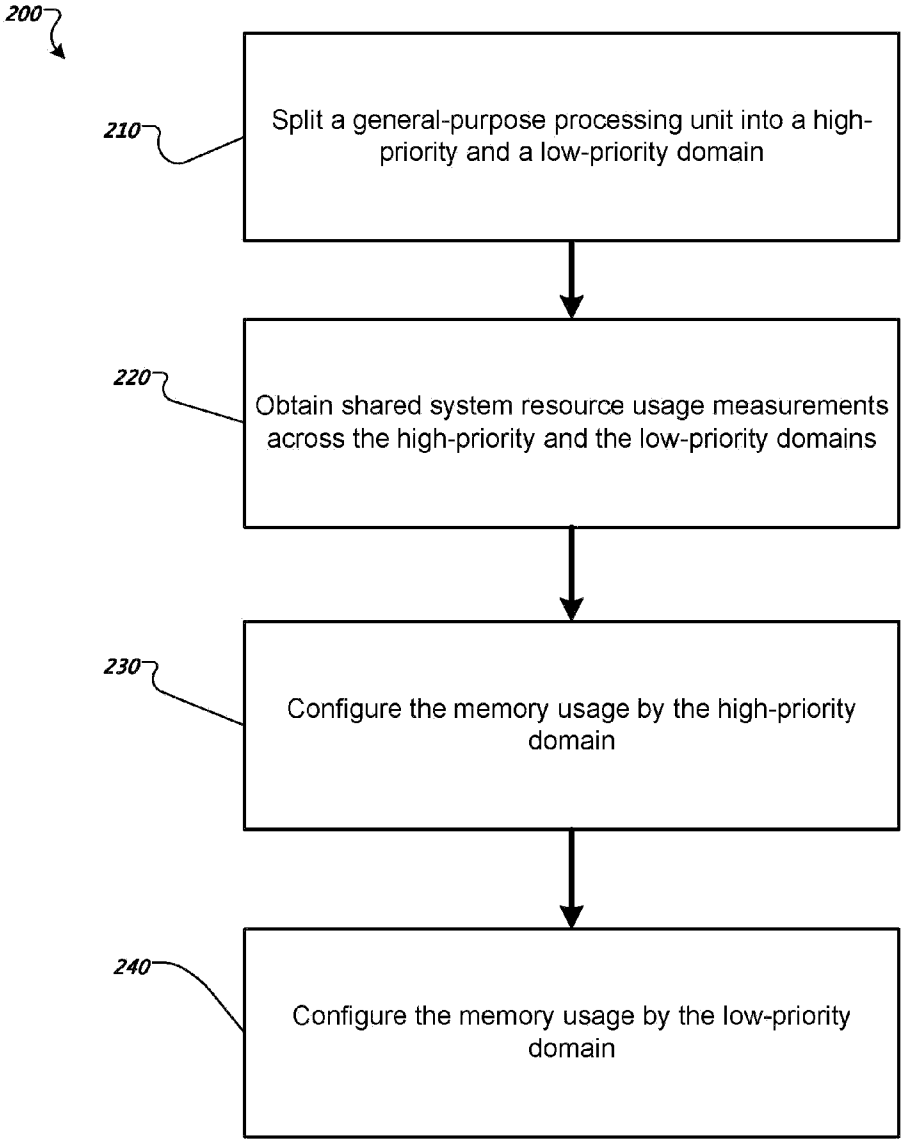
200
210
Split a general-purpose processing unit into a high-priority and a low-priority domain
220
Obtain shared system resource usage measurements across the high-priority and the low-priority domains
230
Configure the memory usage by the high-priority domain
240
Configure the memory usage by the low-priority domain

FIG. 2

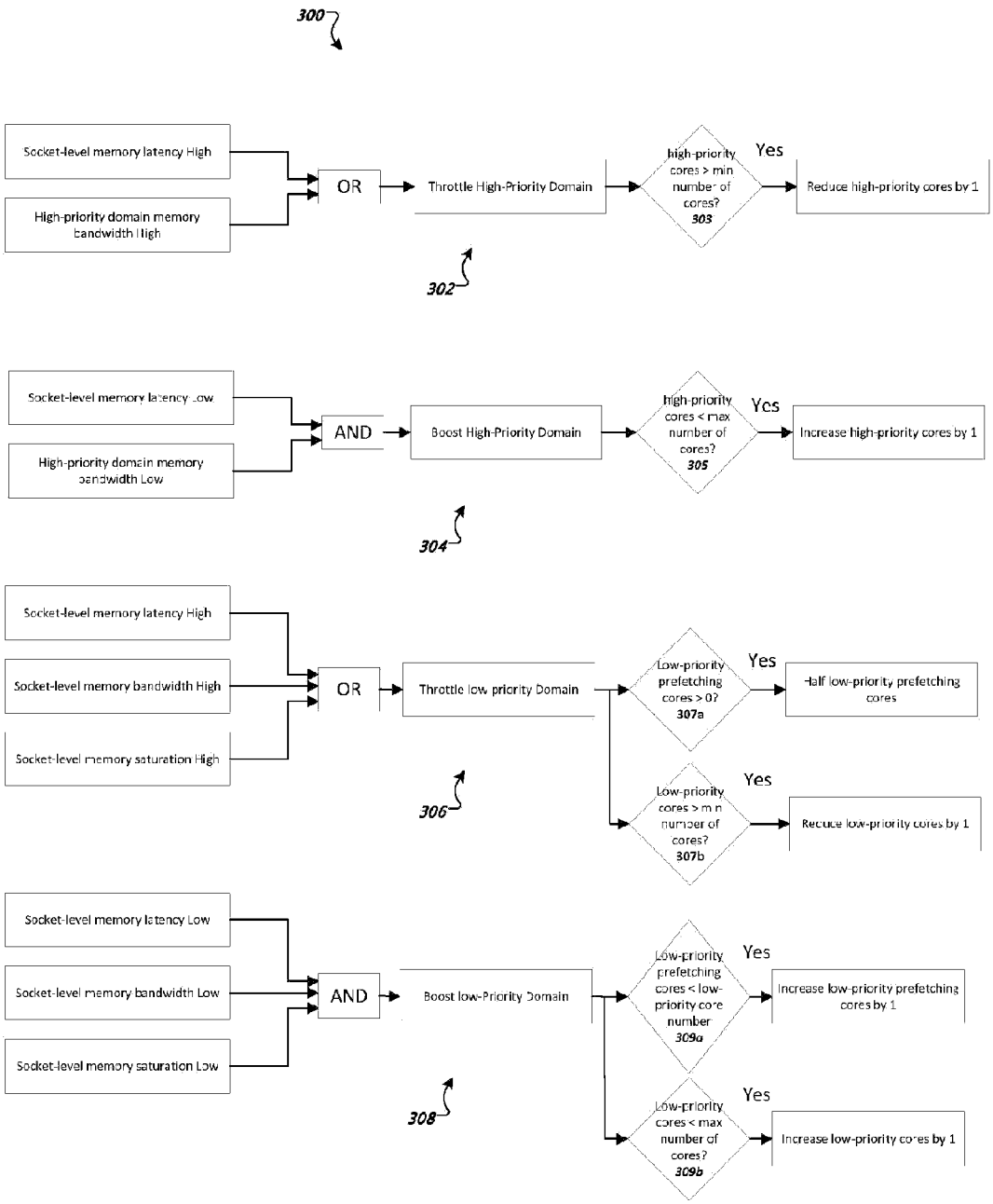
300
Socket-level memory latency High
High-priority domain memory bandwidth High
OR
Throttle High-Priority Domain
high-priority cores > min number of cores?
303
Yes
Reduce high-priority cores by 1
302
Socket-level memory latency Low
High-priority domain memory bandwidth Low
AND
Boost High-Priority Domain
high-priority cores < max number of cores?
305
Yes
Increase high-priority cores by 1
304
Socket-level memory latency High
Socket-level memory bandwidth High
Socket-level memory saturation High
OR
Throttle low priority Domain
Low-priority prefetching cores > 0?
307a
Yes
Half low-priority prefetching cores
Low-priority cores > min number of cores?
307b
Yes
Reduce low-priority cores by 1
306
Socket-level memory latency Low
Socket-level memory bandwidth Low
Socket-level memory saturation Low
AND
Boost low-Priority Domain
Low-priority prefetching cores < low-priority core number
309a
Yes
Increase low-priority prefetching cores by 1
Low-priority cores < max number of cores?
309b
Yes
Increase low-priority cores by 1
308

FIG. 3