(12) STANDARD PATENT (11) Application No. **AU 2011314228 B2**
(19) AUSTRALIAN PATENT OFFICE

(54) Title
Entropy coder for image compression

(51) International Patent Classification(s)
G06F 9/44 (2006.01) ***G06F 17/00*** (2006.01)
G06F 13/14 (2006.01)

(21) Application No: **2011314228** (22) Date of Filing: **2011.09.14**

(87) WIPO No: **WO12/050722**

(30) Priority Data

(31) Number	(32) Date	(33) Country
12/894,793	**2010.09.30**	**US**

(43) Publication Date: **2012.04.19**
(44) Accepted Journal Date: **2014.07.31**

(71) Applicant(s)
Microsoft Corporation

(72) Inventor(s)
Abdo, Nadim Y.

(74) Agent / Attorney
Davies Collison Cave, Level 15 1 Nicholson Street, MELBOURNE, VIC, 3000

(56) Related Art
US 5689255 A1 (FRAZIER et al.) 18 November 1997
US 2010/0111410 A1 (LU et al.) 6 May 2010

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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 April 2012 (19.04.2012)

WIPO | PCT

(10) International Publication Number
WO 2012/050722 A3

(51) International Patent Classification:
G06F 17/00 (2006.01) *G06F 9/44* (2006.01)
G06F 13/14 (2006.01)

(21) International Application Number:
PCT/US2011/051660

(22) International Filing Date:
14 September 2011 (14.09.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
12/894,793 30 September 2010 (30.09.2010) US

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

(72) Inventor: **ABDO, Nadim Y.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US).

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

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

[Continued on next page]

(54) Title: ENTROPY CODER FOR IMAGE COMPRESSION

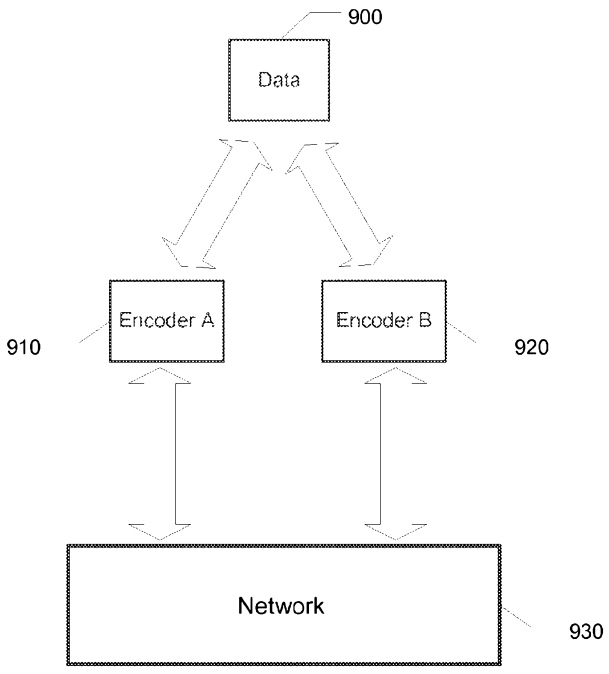

FIG. 9

(57) Abstract: Example embodiments of the present disclosure provide for a fast entropy coder / decoder for use in real time image compression. A method of processing graphics data for transmission to a remote computing device may comprise receiving graphics data representative of a client screen to be rendered, receiving information indicative of available bandwidth for transmission and, based on the information, determining that the available bandwidth meets a predetermined threshold, and entropy encoding the graphics data using a fixed bit size encoding stream, wherein runs of zeroes are encoded in a variable number of units of the fixed bit size, and literal values are encoded using one of an entry in a cache of recently used literal values or a variable number of units of the fixed bit size.

WO 2012/050722 A3

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

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

(88) Date of publication of the international search report:
19 July 2012

ENTROPY CODER FOR IMAGE COMPRESSION

BACKGROUND

[0001] One increasing popular form of networking may generally be referred to as remote presentation systems, which can use protocols such as Remote Desktop Protocol (RDP) and Independent Computing Architecture (ICA) to share a desktop and other applications executing on a server with a remote client. Such computing systems typically transmit the keyboard presses and mouse clicks or selections from the client to the server, relaying the screen updates back in the other direction over a network connection (e.g., the Internet). As such, the user has the experience as if his or her machine is operating entirely locally, when in reality the client device is only sent screenshots of the desktop or applications as they appear on the server side.

[0002] In the remote desktop environment, data representing graphics to be transmitted to the client are typically compressed by the server, transmitted from the server to the client through a network, and decompressed by the client and displayed on the local user display. The process of encoding the data typically requires significant processor computation cycles to compress and decompress the data. Such processing requirements may have a direct effect on the encoding and decoding latency from the server to the client and negatively impact the remote user's experience.

[0002A] It is desired to address or ameliorate one or more disadvantages or limitations associated with the prior art, or to at least provide a useful alternative

SUMMARY

[0003] In accordance with the present invention there is provided a method of processing graphics data for transmission to a remote computing device, the method comprising:

receiving graphics data representative of a client screen associated with a virtual machine session;

receiving information indicative of available bandwidth for said transmission and, based on the information, determining that the available bandwidth meets a predetermined threshold, and

2011314228 11 Jun 2014

entropy encoding coefficients of transformed graphics data using a compact stream of bit tokens that form groups that align to byte boundaries, wherein:

runs of zeroes are encoded in a variable number of multiples of a quantum size;

literal values are encoded using an entry in a cache of recently used literal values; and

other values are encoded using a minimum number of multiples of the quantum size.

[0004] The present invention also provides a system for processing graphics data for transmission to a remote computing device, comprising:

a computing device comprising at least one processor;

a memory communicatively coupled to said processor when said system is operational; said memory having stored therein computer instructions that upon execution by the at least one processor cause:

receiving graphics data representing a client screen associated with a virtual machine session;

dividing said graphics data into data tiles;

entropy encoding coefficients of transformed data tiles using a stream of bit tokens that form groups that align to byte boundaries, wherein:

runs of zeroes are encoded in a variable number of multiples of a quantum size;

literal values are encoded using an entry in a cache of recently used literal values; and

other values are encoded using a minimum number of units of the quantum size.

[0004A] The present invention also provides a computer readable storage device storing thereon computer executable instructions for processing graphics data for transmission to a client computer, said instructions for:

receiving graphics data representative of a client screen associated with a virtual machine session; and

entropy encoding coefficients of transformed graphics data using a compact stream of bit tokens that form groups that align to byte boundaries such that the encoded data can be decoded using a byte-based decoding process, wherein:

runs of zeroes are encoded in a variable number of multiples of a nibble:

literal values are encoded using an entry in a cache of recently used literal values; and

other values are encoded using a minimum number of multiples of a nibble.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] Preferred embodiments of the present invention are hereinafter described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

[0006] FIGs. 1 and 2 depict an example computer system wherein aspects of the present disclosure can be implemented.

[0007] FIG. 3 depicts an operational environment for practicing aspects of the present disclosure.

[0008] FIG. 4 depicts an operational environment for practicing aspects of the present disclosure.

[0009FIG. 5 illustrates a computer system including circuitry for effectuating remote desktop services.

[0010 FIG. 6 illustrates a computer system including circuitry for effectuating remote services.

[0011 FIG. 7 illustrates an example of a decoding process.

[0012 FIG. 8 illustrates an example of an encoding process.

[0013 FIG. 9 illustrates an example of an operational procedure for processing graphics data for transmission to a client computer.

[0014 FIG. 10 illustrates an example of an operational procedure for processing graphics data for transmission to a client computer.

[0015FIG. 11 illustrates an example system for processing graphics data for transmission to a client computer.

DETAILED DESCRIPTION

[0016] One problem with remote presentation systems is that such systems tend to favor data compression at the expense of processor performance. Many systems assume that bandwidth is more likely to be limited and thus sacrifice processor performance in order to

achieve higher levels of data compression and thus reduce the amount of data that needs to be transmitted over the limited bandwidth data link. However, many remote presentation clients today are lower end devices that may use lower speed processors but that may have access to abundant bandwidth. In such cases overall performance and user experience may be improved by using a simpler compressor and less computationally demanding compression techniques even if it means that the compression is reduced.

[0016A] In various embodiments, methods and systems are disclosed for a fast entropy coder / decoder for use in real time image compression. For example, a method of processing graphics data for transmission to a remote computing device may comprise receiving graphics data representative of a client screen to be rendered, receiving information indicative of available bandwidth for transmission and, based on the information, determining that the available bandwidth meets a predetermined threshold, and entropy encoding the graphics data using a fixed bit size encoding stream, wherein runs of zeroes are encoded in a variable number of units of the fixed bit size, and literal values are encoded using one of an entry in a cache of recently used literal values or a variable number of units of the fixed bit size.

[0017] Systems, methods, and computer readable media for graphics data for transmission to a remote computing device, and systems, methods, and computer readable media for altering a view perspective within a virtual environment, are described herein.

Computing Environments In General Terms

[0017A] Certain specific details are set forth in the following description and figures to provide a thorough understanding of various embodiments of the disclosure. Certain well-

known details often associated with computing and software technology are not set forth in the following disclosure to avoid unnecessarily obscuring the various embodiments of the disclosure. Further, those of ordinary skill in the relevant art will understand that they can practice other embodiments of the disclosure without one or more of the details described below. Finally, while various methods are described with reference to steps and sequences in the following disclosure, the description as such is for providing a clear implementation of embodiments of the disclosure, and the steps and sequences of steps should not be taken as required to practice this disclosure.

[0018] It should be understood that the various techniques described herein may be implemented in connection with hardware or software or, where appropriate, with a combination of both. Thus, the methods and apparatus of the disclosure, or certain aspects or portions thereof, may take the form of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other machine-readable storage medium wherein, when the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the disclosure. In the case of program code execution on programmable computers, the computing device generally includes a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. One or more programs that may implement or utilize the processes described in connection with the disclosure, e.g., through the use of an application programming interface (API), reusable controls, or the like. Such programs are preferably implemented in a high level procedural or object oriented programming language to communicate with a computer system. However, the program(s) can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language, and combined with hardware implementations.

[0019] A remote desktop system is a computer system that maintains applications that can be remotely executed by client computer systems. Input is entered at a client computer system and transferred over a network (e.g., using protocols based on the International Telecommunications Union (ITU) T.120 family of protocols such as Remote Desktop Protocol (RDP)) to an application on a terminal server. The application processes the input as if the input were entered at the terminal server. The application generates output in response to the received input and the output is transferred over the network to the client

[0020] Embodiments may execute on one or more computers. FIGs. 1 and 2 and the following discussion are intended to provide a brief general description of a suitable computing environment in which the disclosure may be implemented. One skilled in the art can appreciate that computer systems 200, 300 can have some or all of the components described with respect to computer 100 of FIG. 1 and 2.

[0021] The term circuitry used throughout the disclosure can include hardware components such as hardware interrupt controllers, hard drives, network adaptors, graphics processors, hardware based video/audio codecs, and the firmware/software used to operate such hardware. The term circuitry can also include microprocessors configured to perform function(s) by firmware or by switches set in a certain way or one or more logical processors, e.g., one or more cores of a multi-core general processing unit. The logical processor(s) in this example can be configured by software instructions embodying logic operable to perform function(s) that are loaded from memory, e.g., RAM, ROM, firmware, and/or virtual memory. In example embodiments where circuitry includes a combination of hardware and software an implementer may write source code embodying logic that is subsequently compiled into machine readable code that can be executed by a logical processor. Since one skilled in the art can appreciate that the state of the art has evolved to a point where there is little difference between hardware, software, or a combination of hardware/software, the selection of hardware versus software to effectuate functions is merely a design choice. Thus, since one of skill in the art can appreciate that a software process can be transformed into an equivalent hardware structure, and a hardware structure can itself be transformed into an equivalent software process, the selection of a hardware implementation versus a software implementation is trivial and left to an implementer.

[0022] FIG. 1 depicts an example of a computing system which is configured to with aspects of the disclosure. The computing system can include a computer 20 or the like, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the processing unit 21. The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system 26 (BIOS), containing the basic routines that help to transfer information between elements within the computer 20, such as during start up, is stored in ROM 24. The computer 20 may further include a hard disk

drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media. In some example embodiments, computer executable instructions embodying aspects of the disclosure may be stored in ROM 24, hard disk (not shown), RAM 25, removable magnetic disk 29, optical disk 31, and/or a cache of processing unit 21. The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer readable media provide non volatile storage of computer readable instructions, data structures, program modules and other data for the computer 20. Although the environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs) and the like may also be used in the operating environment.

[0023] A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37 and program data 38. A user may enter commands and information into the computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite disk, scanner or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A display 47 or other type of display device can also be connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the display 47, computers typically include other peripheral output devices (not shown), such as speakers and printers. The system of FIG. 1 also includes a host adapter 55, Small Computer System Interface (SCSI) bus 56, and an external storage device 62 connected to the SCSI bus 56.

[0024] The computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. The remote computer 49 may be another computer, a server, a router, a network PC, a peer device or

other common network node, a virtual machine, and typically can include many or all of the elements described above relative to the computer 20, although only a memory storage device 50 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 can include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise wide computer networks, intranets and the Internet.

[0025] When used in a LAN networking environment, the computer 20 can be connected to the LAN 51 through a network interface or adapter 53. When used in a WAN networking environment, the computer 20 can typically include a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, can be connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are examples and other means of establishing a communications link between the computers may be used. Moreover, while it is envisioned that numerous embodiments of the disclosure are particularly well-suited for computer systems, nothing in this document is intended to limit the disclosure to such embodiments.

[0026] Referring now to FIG. 2, another embodiment of an exemplary computing system 100 is depicted. Computer system 100 can include a logical processor 102, e.g., an execution core. While one logical processor 102 is illustrated, in other embodiments computer system 100 may have multiple logical processors, e.g., multiple execution cores per processor substrate and/or multiple processor substrates that could each have multiple execution cores. As shown by the figure, various computer readable storage media 110 can be interconnected by one or more system busses which couples various system components to the logical processor 102. The system buses may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. In example embodiments the computer readable storage media 110 can include for example, random access memory (RAM) 104, storage device 106, e.g., electromechanical hard drive, solid state hard drive, etc., firmware 108, e.g., FLASH RAM or ROM, and removable storage devices 118 such as, for example, CD-ROMs, floppy disks, DVDs, FLASH drives, external storage devices, etc. It should be appreciated by those skilled in the art that other types of computer

readable storage media can be used such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges.

[0027] The computer readable storage media provide non volatile storage of processor executable instructions 122, data structures, program modules and other data for the computer 100. A basic input/output system (BIOS) 120, containing the basic routines that help to transfer information between elements within the computer system 100, such as during start up, can be stored in firmware 108. A number of programs may be stored on firmware 108, storage device 106, RAM 104, and/or removable storage devices 118, and executed by logical processor 102 including an operating system and/or application programs.

[0028] Commands and information may be received by computer 100 through input devices 116 which can include, but are not limited to, a keyboard and pointing device. Other input devices may include a microphone, joystick, game pad, scanner or the like. These and other input devices are often connected to the logical processor 102 through a serial port interface that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A display or other type of display device can also be connected to the system bus via an interface, such as a video adapter which can be part of, or connected to, a graphics processor 112. In addition to the display, computers typically include other peripheral output devices (not shown), such as speakers and printers. The exemplary system of FIG. 1 can also include a host adapter, Small Computer System Interface (SCSI) bus, and an external storage device connected to the SCSI bus.

[0029] Computer system 100 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer. The remote computer may be another computer, a server, a router, a network PC, a peer device or other common network node, and typically can include many or all of the elements described above relative to computer system 100.

[0030] When used in a LAN or WAN networking environment, computer system 100 can be connected to the LAN or WAN through a network interface card 114. The NIC 114, which may be internal or external, can be connected to the system bus. In a networked environment, program modules depicted relative to the computer system 100, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections described here are exemplary and other means of establishing a communications link between the computers may be used. Moreover, while it is

envisioned that numerous embodiments of the present disclosure are particularly well-suited for computerized systems, nothing in this document is intended to limit the disclosure to such embodiments.

[0031] A remote desktop system is a computer system that maintains applications that can be remotely executed by client computer systems. Input is entered at a client computer system and transferred over a network (e.g., using protocols based on the International Telecommunications Union (ITU) T.120 family of protocols such as Remote Desktop Protocol (RDP)) to an application on a terminal server. The application processes the input as if the input were entered at the terminal server. The application generates output in response to the received input and the output is transferred over the network to the client computer system. The client computer system presents the output data. Thus, input is received and output presented at the client computer system, while processing actually occurs at the terminal server. A session can include a shell and a user interface such as a desktop, the subsystems that track mouse movement within the desktop, the subsystems that translate a mouse click on an icon into commands that effectuate an instance of a program, etc. In another example embodiment the session can include an application. In this example while an application is rendered, a desktop environment may still be generated and hidden from the user. It should be understood that the foregoing discussion is exemplary and that the presently disclosed subject matter may be implemented in various client/server environments and not limited to a particular terminal services product.

[0032] In most, if not all remote desktop environments, input data (entered at a client computer system) typically includes mouse and keyboard data representing commands to an application and output data (generated by an application at the terminal server) typically includes video data for display on a video output device. Many remote desktop environments also include functionality that extend to transfer other types of data.

[0033] Communications channels can be used to extend the RDP protocol by allowing plug-ins to transfer data over an RDP connection. Many such extensions exist. Features such as printer redirection, clipboard redirection, port redirection, etc., use communications channel technology. Thus, in addition to input and output data, there may be many communications channels that need to transfer data. Accordingly, there may be occasional requests to transfer output data and one or more channel requests to transfer other data contending for available network bandwidth.

[0034] Referring now to FIG. 3 and 4, depicted are high level block diagrams of computer systems configured to effectuate virtual machines. As shown in the figures, computer system 100 can include elements described in FIGs. 1 and 2 and components operable to effectuate virtual machines. One such component is a hypervisor 202 that may also be referred to in the art as a virtual machine monitor. The hypervisor 202 in the depicted embodiment can be configured to control and arbitrate access to the hardware of computer system 100. Broadly stated, the hypervisor 202 can generate execution environments called partitions such as child partition 1 through child partition N (where N is an integer greater than or equal to 1). In embodiments a child partition can be considered the basic unit of isolation supported by the hypervisor 202, that is, each child partition can be mapped to a set of hardware resources, e.g., memory, devices, logical processor cycles, etc., that is under control of the hypervisor 202 and/or the parent partition and hypervisor 202 can isolate one partition from accessing another partition's resources. In embodiments the hypervisor 202 can be a stand-alone software product, a part of an operating system, embedded within firmware of the motherboard, specialized integrated circuits, or a combination thereof.

[0035] In the above example, computer system 100 includes a parent partition 204 that can also be thought of as domain 0 in the open source community. Parent partition 204 can be configured to provide resources to guest operating systems executing in child partitions 1-N by using virtualization service providers 228 (VSPs) that are also known as back-end drivers in the open source community. In this example architecture the parent partition 204 can gate access to the underlying hardware. The VSPs 228 can be used to multiplex the interfaces to the hardware resources by way of virtualization service clients (VSCs) that are also known as front-end drivers in the open source community. Each child partition can include one or more virtual processors such as virtual processors 230 through 232 that guest operating systems 220 through 222 can manage and schedule threads to execute thereon. Generally, the virtual processors 230 through 232 are executable instructions and associated state information that provide a representation of a physical processor with a specific architecture. For example, one virtual machine may have a virtual processor having characteristics of an Intel x86 processor, whereas another virtual processor may have the characteristics of a PowerPC processor. The virtual processors in this example can be mapped to logical processors of the computer system such that the instructions that effectuate the virtual processors will be backed by logical processors. Thus, in these example embodiments, multiple virtual processors can be

simultaneously executing while, for example, another logical processor is executing hypervisor instructions. Generally speaking, and as illustrated by the figures, the combination of virtual processors, various VSCs, and memory in a partition can be considered a virtual machine such as virtual machine 240 or 242.

[0036] Generally, guest operating systems 220 through 222 can include any operating system such as, for example, operating systems from Microsoft®, Apple®, the open source community, etc. The guest operating systems can include user/kernel modes of operation and can have kernels that can include schedulers, memory managers, etc. A kernel mode can include an execution mode in a logical processor that grants access to at least privileged processor instructions. Each guest operating system 220 through 222 can have associated file systems that can have applications stored thereon such as terminal servers, e-commerce servers, email servers, etc., and the guest operating systems themselves. The guest operating systems 220-222 can schedule threads to execute on the virtual processors 230-232 and instances of such applications can be effectuated.

[0037] Referring now to FIG. 4, illustrated is an alternative architecture that can be used to effectuate virtual machines. FIG. 4 depicts similar components to those of FIG. 3, however in this example embodiment the hypervisor 202 can include the virtualization service providers 228 and device drivers 224, and parent partition 204 may contain configuration utilities 236. In this architecture, hypervisor 202 can perform the same or similar functions as the hypervisor 202 of FIG. 2. The hypervisor 202 of FIG. 4 can be a stand alone software product, a part of an operating system, embedded within firmware of the motherboard or a portion of hypervisor 202 can be effectuated by specialized integrated circuits. In this example parent partition 204 may have instructions that can be used to configure hypervisor 202 however hardware access requests may be handled by hypervisor 202 instead of being passed to parent partition 204.

[0038] Referring now to FIG. 5, computer 100 may include circuitry configured to provide remote desktop services to connecting clients. In an example embodiment, the depicted operating system 400 may execute directly on the hardware or a guest operating system 220 or 222 may be effectuated by a virtual machine such as VM 216 or VM 218. The underlying hardware 208, 210, 234, 212, and 214 is indicated in the illustrated type of dashed lines to identify that the hardware can be virtualized.

[0039] Remote services can be provided to at least one client such as client 401 (while one client is depicted remote services can be provided to more clients.) The example client 401 can include a computer terminal that is effectuated by hardware configured to

direct user input to a remote server session and display user interface information generated by the session. In another embodiment, client 401 can be effectuated by a computer that includes similar elements as those of computer 100 FIG. 1b. In this embodiment, client 401 can include circuitry configured to effect operating systems and circuitry configured to emulate the functionality of terminals, e.g., a remote desktop client application that can be executed by one or more logical processors 102. One skilled in the art can appreciate that the circuitry configured to effectuate the operating system can also include circuitry configured to emulate a terminal.

[0040] Each connecting client can have a session (such as session 404) which allows the client to access data and applications stored on computer 100. Generally, applications and certain operating system components can be loaded into a region of memory assigned to a session. Thus, in certain instances some OS components can be spawned N times (where N represents the number of current sessions). These various OS components can request services from the operating system kernel 418 which can, for example, manage memory; facilitate disk reads/writes; and configure threads from each session to execute on the logical processor 102. Some example subsystems that can be loaded into session space can include the subsystems that generates desktop environments, the subsystems that track mouse movement within the desktop, the subsystems that translate mouse clicks on icons into commands that effectuate an instance of a program, etc. The processes that effectuate these services, e.g., tracking mouse movement, are tagged with an identifier associated with the session and are loaded into a region of memory that is allocated to the session.

[0041] A session can be generated by a session manager 416, e.g., a process. For example, the session manager 416 can initialize and manage each remote session by generating a session identifier for a session space; assigning memory to the session space; and generating system environment variables and instances of subsystem processes in memory assigned to the session space. The session manager 416 can be invoked when a request for a remote desktop session is received by the operating system 400.

[0042] A connection request can first be handled by a transport stack 410, e.g., a remote desktop protocol (RDP) stack. The transport stack 410 instructions can configure logical processor 102 to listen for connection messages on a certain port and forward them to the session manager 416. When sessions are generated the transport stack 410 can instantiate a remote desktop protocol stack instance for each session. Stack instance 414 is an example stack instance that can be generated for session 404. Generally, each remote

desktop protocol stack instance can be configured to route output to an associated client and route client input to an environment subsystem 444 for the appropriate remote session.

[0043] As shown by the figure, in an embodiment an application 448 (while one is shown others can also execute) can execute and generate an array of bits. The array can be processed by a graphics interface 446 which in turn can render bitmaps, e.g., arrays of pixel values, that can be stored in memory. As shown by the figure, a remote display subsystem 420 can be instantiated which can capture rendering calls and send the calls over the network to client 401 via the stack instance 414 for the session.

[0044] In addition to remoting graphics and audio, a plug and play redirector 458 can also be instantiated in order to remote diverse devices such as printers, mp3 players, client file systems, CD ROM drives, etc. The plug and play redirector 458 can receive information from a client side component which identifies the peripheral devices coupled to the client 401. The plug and play redirector 458 can then configure the operating system 400 to load redirecting device drivers for the peripheral devices of the client 401. The redirecting device drivers can receive calls from the operating system 400 to access the peripherals and send the calls over the network to the client 401.

[0045] As discussed above, clients may use a protocol for providing remote presentation services such as Remote Desktop Protocol (RDP) to connect to a resource using terminal services. When a remote desktop client connects to a terminal server via a terminal server gateway, the gateway may open a socket connection with the terminal server and redirect client traffic on the remote presentation port or a port dedicated to remote access services. The gateway may also perform certain gateway specific exchanges with the client using a terminal server gateway protocol transmitted over HTTPS.

[0046] Turning to FIG. 6, depicted is a computer system 100 including circuitry for effectuating remote services and for incorporating aspects of the present disclosure. As shown by the figure, in an embodiment a computer system 100 can include components similar to those described in FIG. 2 and FIG. 5, and can effectuate a remote presentation session. In an embodiment of the present disclosure a remote presentation session can include aspects of a console session, e.g., a session spawned for a user using the computer system, and a remote session. Similar to that described above, the session manager 416 can initialize and manage the remote presentation session by enabling/disabling components in order to effectuate a remote presentation session.

[0047] One set of components that can be loaded in a remote presentation session are the console components that enable high fidelity remoting, namely, the components that take advantage of 3D graphics and 2D graphics rendered by 3D hardware.

[0048] 3D/2D graphics rendered by 3D hardware can be accessed using a driver model that includes a user mode driver 522, an API 520, a graphics kernel 524, and a kernel mode driver 530. An application 448 (or any other process such as a user interface that generates 3D graphics) can generate API constructs and send them to an application programming interface 520 (API) such as Direct3D from Microsoft®. The API 520 in turn can communicate with a user mode driver 522 which can generates primitives, e.g., the fundamental geometric shapes used in computer graphics represented as vertices and constants which are used as building blocks for other shapes, and stores them in buffers, e.g., pages of memory. In one embodiment the application 448 can declare how it is going to use the buffer, e.g., what type of data it is going to store in the buffer. An application, such as a videogame, may use a dynamic buffer to store primitives for an avatar and a static buffer for storing data that will not change often such as data that represents a building or a forest.

[0049] Continuing with the description of the driver model, the application can fill the buffers with primitives and issue execute commands. When the application issues an execute command the buffer can be appended to a run list by the kernel mode driver 530 and scheduled by the graphics kernel scheduler 528. Each graphics source, e.g., application or user interface, can have a context and its own run list. The graphics kernel 524 can be configured to schedule various contexts to execute on the graphics processing unit 112. The GPU scheduler 528 can be executed by logical processor 102 and the scheduler 528 can issue a command to the kernel mode driver 530 to render the contents of the buffer. The stack instance 414 can be configured to receive the command and send the contents of the buffer over the network to the client 401 where the buffer can be processed by the GPU of the client.

[0050] Illustrated now is an example of the operation of a virtualized GPU as used in conjunction with an application that calls for remote presentation services. Referring to FIG. 6, in an embodiment a virtual machine session can be generated by a computer 100. For example, a session manager 416 can be executed by a logical processor 102 and a remote session that includes certain remote components can be initialized. In this example the spawned session can include a kernel 418, a graphics kernel 524, a user mode display driver 522, and a kernel mode display driver 530. The user mode driver 522 can generate

graphics primitives that can be stored in memory. For example, the API 520 can include interfaces that can be exposed to processes such as a user interface for the operating system 400 or an application 448. The process can send high level API commands such as such as Point Lists, Line Lists, Line Strips, Triangle Lists, Triangle Strips, or Triangle Fans, to the API 420. The API 520 can receive these commands and translate them into commands for the user mode driver 522 which can then generate vertices and store them in one or more buffers. The GPU scheduler 528 can run and determine to render the contents of the buffer. In this example the command to the graphics processing unit 112 of the server can be captured and the content of the buffer (primitives) can be sent to client 401 via network interface card 114. In an embodiment, an API can be exposed by the session manager 416 that components can interface with in order to determine whether a virtual GPU is available.

[0051] In an embodiment a virtual machine such as virtual machine 240 of FIG. 3 or 4 can be instantiated and the virtual machine can serve as a platform for execution for the operating system 400. Guest operating system 220 can embody operating system 400 in this example. A virtual machine may be instantiated when a connection request is received over the network. For example, the parent partition 204 may include an instance of the transport stack 410 and may be configured to receive connection requests. The parent partition 204 may initialize a virtual machine in response to a connection request along with a guest operating system including the capabilities to effectuate remote sessions. The connection request can then be passed to the transport stack 410 of the guest operating system 220. In this example each remote session may be instantiated on an operating system that is executed by its own virtual machine.

[0052] In one embodiment a virtual machine can be instantiated and a guest operating system 220 embodying operating system 400 can be executed. Similar to that described above, a virtual machine may be instantiated when a connection request is received over the network. Remote sessions may be generated by an operating system. The session manager 416 can be configured to determine that the request is for a session that supports 3D graphics rendering and the session manager 416 can load a console session. In addition to loading the console session the session manager 416 can load a stack instance 414' for the session and configure system to capture primitives generated by a user mode display driver 522.

[0053] The user mode driver 522 may generate graphics primitives that can be captured and stored in buffers accessible to the transport stack 410. A kernel mode driver 530 can

append the buffers to a run list for the application and a GPU scheduler 528 can run and determine when to issue render commands for the buffers. When the scheduler 528 issues a render command the command can be captured by, for example, the kernel mode driver 530 and sent to the client 401 via the stack instance 414.

[0054] The GPU scheduler 528 may execute and determine to issue an instruction to render the content of the buffer. In this example the graphics primitives associated with the instruction to render can be sent to client 401 via network interface card 114.

[0055] In an embodiment, at least one kernel mode process can be executed by at least one logical processor 112 and the at least one logical processor 112 can synchronize rendering vertices stored in different buffers. For example, a graphics processing scheduler 528, which can operate similarly to an operating system scheduler, can schedule GPU operations. The GPU scheduler 528 can merge separate buffers of vertices into the correct execution order such that the graphics processing unit of the client 401 executes the commands in an order that allows them to be rendered correctly.

[0056] One or more threads of a process such as a videogame may map multiple buffers and each thread may issue a draw command. Identification information for the vertices, e.g., information generated per buffer, per vertex, or per batch of vertices in a buffer, can be sent to the GPU scheduler 528. The information may be stored in a table along with identification information associated with vertices from the same, or other processes and used to synchronize rendering of the various buffers.

[0057] An application such as a word processing program may execute and declare , for example, two buffers - one for storing vertices for generating 3D menus and the other one storing commands for generating letters that will populate the menus. The application may map the buffer and; issue draw commands. The GPU scheduler 528 may determine the order for executing the two buffers such that the menus are rendered along with the letters in a way that it would be pleasing to look at. For example, other processes may issue draw commands at the same or a substantially similar time and if the vertices were not synchronized vertices from different threads of different processes could be rendered asynchronously on the client 401 thereby making the final image displayed seem chaotic or jumbled.

[0058] A bulk compressor 450 can be used to compress the graphics primitives prior to sending the stream of data to the client 401. In an embodiment the bulk compressor 450 can be a user mode (not shown) or kernel mode component of the stack instance 414 and can be configured to look for similar patterns within the stream of data that is being sent to

the client 401. In this embodiment, since the bulk compressor 450 receives a stream of vertices, instead of receiving multiple API constructs, from multiple applications, the bulk compressor 450 has a larger data set of vertices to sift through in order to find opportunities to compress. That is, since the vertices for a plurality of processes are being remoted, instead of diverse API calls, there is a larger chance that the bulk compressor 450 will be able to find similar patterns in a given stream.

[0059] In an embodiment, the graphics processing unit 112 may be configured to use virtual addressing instead of physical addresses for memory. Thus, the pages of memory used as buffers can be paged to system RAM or to disk from video memory. The stack instance 414' can be configured to obtain the virtual addresses of the buffers and send the contents from the virtual addresses when a render command from the graphics kernel 528 is captured.

[0060] An operating system 400 may be configured, e.g., various subsystems and drivers can be loaded to capture primitives and send them to a remote computer such as client 401. Similar to that described above, a session manager 416 can be executed by a logical processor 102 and a session that includes certain remote components can be initialized. In this example the spawned session can include a kernel 418, a graphics kernel 524, a user mode display driver 522, and a kernel mode display driver 530.

[0061] A graphics kernel may schedule GPU operations. The GPU scheduler 528 can merge separate buffers of vertices into the correct execution order such that the graphics processing unit of the client 401 executes the commands in an order that allows them to be rendered correctly.

[0062] Referring to Figure 7, a block diagram illustrating a decoding process is shown, in accordance with one embodiment of the present disclosure. The encoding process is shown in Figure 8. The encoded tile may be first run through an RLGR decoder 900 to generate a quantized tile coefficient. This may be performed on the CPU.

[0063] Dequantization 705 may be implemented on the CPU using SSE2 instructions. After dequantization, the ten subbands of the three components of the tile may be copied into three Direct3D texture buffers of format L16, one for each of Y, U and V. These three textures may be uploaded onto the GPU and can be used by the Inverse DWT stage 710 as input.

[0064] All of these variations for implementing the above mentioned partitions are just exemplary implementations, and nothing herein should be interpreted as limiting the disclosure to any particular virtualization aspect.

Entropy Encoder

[0065] In a virtual desktop or remote presentation session, the user graphics and video may be rendered at the server for each user. The resulting bitmaps may then be sent to the client for display and interaction. To reduce the bandwidth requirements on the network, bitmaps may be compressed before sending to the client. It is desirable that the compression technique be efficient with low latency.

[0066] Described herein is a system and method for encoding and decoding bitmaps and other graphics data. The encoding system may include a tiling system with a tiling module that initially divides source image data into data tiles. A frame differencing module may then output only altered data tiles to various processing modules that convert the altered data tiles into corresponding tile components. In an embodiment, a quantizer may perform a compression procedure upon the tile components to generate compressed data according to an adjustable quantization parameter. An adaptive entropy encoder selector may then select one of a plurality of entropy encoders to perform an entropy encoding procedure to thereby produce encoded data. The entropy encoder may also utilize a feedback loop to adjust the quantization parameter in light of current transmission bandwidth characteristics. The process of compressing, encoding and decoding graphics data as referred to herein may generally use one or more methods and systems described in commonly assigned U.S. Patent Number 7,460,725 entitled "System And Method For Effectively Encoding And Decoding Electronic Information" and U.S. Application No. 12/399,302 entitled "Frame Capture, Encoding, And Transmission Management" filed on March 6, 2009, hereby incorporated by reference in their entirety.

[0067] In various methods and systems disclosed herein, improvements to the processing and handling of the various processes described above may be used to provide more efficient processing and thus a more timely and rich user experience. The methods and systems also provide for improvements in providing such graphics support when the network and/or system resources are providing adequate bandwidth and/or the client device has slower processing speed or resources. The embodiments disclosed herein for rendering, encoding and transmitting graphics data may be implemented using various combinations of hardware and software processes. In some embodiments, functions may be executed entirely in hardware. In other embodiments, functions may be performed entirely in software. In yet further embodiments, functions may be implemented using a combination of hardware and software processes. Such processes may further be

implemented using one or more CPUs and/or one or more specialized processors such as a graphics processing unit (GPU) or other dedicated graphics rendering devices.

[0068] Furthermore, while the following descriptions are provided in the context of remote presentation systems, it should be understood that the disclosed embodiments may be implemented in any type of system in which graphics data is encoded and compressed for delivery over a network.

[0069] Various embodiments may incorporate the use of the discrete wavelet transform (DWT) function for transforming individual YUV components of the tiles into corresponding YUV tile subbands. A quantizer function may perform a quantization procedure by utilizing appropriate quantization techniques to compress the tile subbands. The quantizer function may produce compressed image data by reducing the bit rate of the tiles according to a particular compression ratio that may be specified by an adaptive quantization parameter received via a feedback loop from an entropy encoder.

[0070] In one embodiment, a GPU may be provided a bitmap with changed rectangles that need to be compressed. The bitmap may be further split into logical tiles and only tiles that change within the changed rectangle are encoded and compressed. In this manner, the process effectively implements a caching scheme in concert with the client where the resulting decoded image is maintained and displayed.

[0071] Remote presentation compression algorithms are employed to reduce the bandwidth of the display stream to levels that are acceptable for transmission over local area networks, wide area networks, and low-bandwidth networks. Such algorithms typically trade off CPU time on the server side for a lower desired bandwidth.

[0072] Image compressors may be used that may employ a phase called an entropy coder. An entropy encoder function may perform an entropy encoding procedure to generate encoded data. In certain embodiments, the entropy encoding procedure further reduces the bit rate of the compressed image data by substituting appropriate codes for corresponding bit patterns in the compressed image data received from the quantizer.

[0073] The entropy encoding employed in a remote presentation system generally balances CPU performance (i.e. speed) with compression ratio. Entropy coders may be tuned for good compression at reasonable CPU speed. Typical entropy coders include Run-Length, Huffman, arithmetic, and variations of Golomb-Rice coders. One of the main problems in designing efficient entropy coders for remote presentation applications is that typically there are large variations in the statistics of blocks of integers to be encoded. Studies have shown that in most cases the data prior to quantization have probability

distributions that can be significantly more concentrated near zero than a Gaussian distribution.

[0074] The present disclosure is concerned with implementing a simplified entropy coder that is configured to improve encoding and decoding speed at the potential expense of loss of compressibility. However, in many situations this tradeoff may be acceptable and actually more desirable in scenarios which are limited by low-speed CPUs rather than bandwidth. The net result is that an encoder/decoder may be provided that is two or three times faster than current encoders/decoders at a loss of compressibility on the order of 10% to 20%.

[0075] Such an encoder/decoder may be useful because it enables scenarios in which optimizing for processor speed is a higher priority than saving every last bit of bandwidth. For example, lower end client devices may achieve better performance with a simpler compressor to allow for faster processing. Remote presentations systems are typically optimized to reduce bandwidth regardless of CPU cost and capability. In many systems today, bandwidth may be plentiful while the client devices may be simpler devices such as set top boxes or thin clients.

[0076] In one embodiment an entropy encoder may be configured to avoid using a variable bit-stream format. Employing a variable bit-stream is invariably slower to encode and decode efficiently. In an embodiment, an encoder may be configured to use nibble-sized (aka quad-bit) codes to achieve a regular-size of fixed-size encoding stream. By using such a stream, the stream may be faster to decode and can be decoded securely (with full overflow checks) at much less CPU cost.

[0077] In one example scheme the encoder encodes the following types of operations:

1) Run's of 0's (which are common inputs to the entropy encoder) - Run's of 0's are encoded in a variable number of quads matched to the statistical observation that the majority of runs are very short

2) Literal values - Literal values are encoded either as an Least Recently Used (LRU) hit in a table (cache) of the most recent literal values seen or as a variable number of quads again benefiting from the statistical properties that smaller values are more likely to occur

[0078] In both cases there are effectively two streams: (a) the quad-stream of op-codes and (b) the large value stream. Certain op-codes for either a run-length or a literal length simply indicate "get the next value in the large value stream."

[0079] The large value stream may be encoded with a basic multi-byte encoding scheme that uses less bytes for small values than large values, but at the same time is guaranteed to operate only on fixed-byte boundaries. By using such an encoding scheme, both the quad-stream and value-stream can be decoded without the need for complex bit-shifting or variable-bit decoding schemes, thus allowing for much faster performance than the more general/complex entropy coders (e.g., RLGR or various Huffman based schemes). Such a simplified encoder may be configured such that any number of bits, for example from 1 to 32 bits, can be encoded. In more complex encoding, decoding becomes computationally challenging because of the variable bits and the necessary processing which typically requires many coding branches and a significant amount of bookkeeping during processing. Using this simplified scheme allows for minimizing such complexity by using a regular size structure (e.g., quads). In such a scheme, no output symbol is more than four bits and the data are in multiples of a byte with no shifts or rolls. A byte may contain two codes that can be processed in parallel if desired. Furthermore, if the number of quads are known then buffer overruns can be avoided. In trials using typical remote presentation scenarios a loss of only 10-20% compressibility was measured at a gain of 2-3 times performance using currently available CPU's.

[0080] In some embodiments, logic may be provided for switching between a more complex/slower entropy coder and a simpler encoder as described herein. For example, referring to Fig. 9, a remote presentation system may provide at least two encoders 910 and 920. Encoder 910 may be a complex entropy encoder such as one that implements RLGR. Encoder 920 may be a simplified encoder in accordance with the present disclosure. Depending on the conditions of network 930, the system may choose one of the encoders 910 or 920 to encode data 900. For example, if network conditions indicate that the network is congested and that available bandwidth is limited, complex encoder 910 may be selected for encoding data 900 in order to minimize the amount of data to be transmitted over network 930. Similarly, if network conditions indicate that the network is not congested, simplified encoder 910 may be selected for encoding data 900 in order to provide faster processor performance at the client.

[0081] Appendix A provides an example implementation of a simplified encoder in accordance with this disclosure.

[0082] Figure 10 depicts an exemplary operational procedure for processing graphics data for transmission to a client computer including operations 1000, 1002, 1004, and 1006. Referring to Figure 10, operation 1000 begins the operational procedure and

operation 1002 illustrates receiving graphics data representative of a client screen associated with a virtual machine session. Operation 1004 illustrates receiving information indicative of available bandwidth for said transmission and, based on the information, determining that the available bandwidth meets a predetermined threshold. Operation 1006 illustrates entropy encoding coefficients of transformed graphics data using a compact stream of bit tokens aligned to byte boundaries. In an embodiment, runs of zeroes are encoded in a variable number of multiples of a quantum size, literal values are encoded using an entry in a cache of recently used literal values, and other values are encoded using a minimum number of multiples of the quantum size. A bit token may be a string of bits that define a unit of data. For example, in a nibble based system a four bit token is used.

[0083] In various embodiments, the quantum size may be a nibble. In some embodiments, the operational procedure may include generating a stream of entropy encoding op-codes and a large value stream. The procedure may further include entropy encoding the large value stream with a multi-byte encoding scheme that uses less bytes for small values than large values and dividing said graphics data into data tiles, processing said data tiles into tile components, and performing said entropy encoding on said tile components. The encoding scheme may be configured to operate only on fixed-byte boundaries.

[0084] Figure 11 depicts an exemplary system for processing graphics data for transmission to a client computer as described above. Referring to Figure 11, system 1100 comprises a processor 1110 and memory 1120. Memory 1120 further comprises computer instructions configured to process graphics data for transmission to a remote computing device. Block 1122 illustrates receiving graphics data representing a client screen associated with a virtual machine session. Block 1124 illustrates dividing said graphics data into data tiles. Block 1126 illustrates entropy encoding coefficients of transformed data tiles using a stream of bit tokens aligned to byte boundaries.

[0085] Any of the above mentioned aspects can be implemented in methods, systems, computer readable media, or any type of manufacture. For example, a computer readable medium can store thereon computer executable instructions for processing graphics data for transmission to a client computer. Such media can comprise a first subset of instructions for receiving graphics data representative of a client screen associated with a virtual machine session and a second subset of instructions for entropy encoding coefficients of transformed graphics data using a compact stream of bit tokens aligned to

byte boundaries such that the encoded data can be decoded using a byte-based decoding process. It will be appreciated by those skilled in the art that additional sets of instructions can be used to capture the various other aspects disclosed herein, and that the two presently disclosed subsets of instructions can vary in detail per the present disclosure.

[0086] The foregoing detailed description has set forth various embodiments of the systems and/or processes via examples and/or operational diagrams. Insofar as such block diagrams, and/or examples contain one or more functions and/or operations, it will be understood by those within the art that each function and/or operation within such block diagrams, or examples can be implemented, individually and/or collectively, by a wide range of hardware, software, firmware, or virtually any combination thereof.

[0087] It should be understood that the various techniques described herein may be implemented in connection with hardware or software or, where appropriate, with a combination of both. Thus, the methods and apparatus of the disclosure, or certain aspects or portions thereof, may take the form of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other machine-readable storage medium wherein, when the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the disclosure. In the case of program code execution on programmable computers, the computing device generally includes a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. One or more programs that may implement or utilize the processes described in connection with the disclosure, e.g., through the use of an application programming interface (API), reusable controls, or the like. Such programs are preferably implemented in a high level procedural or object oriented programming language to communicate with a computer system. However, the program(s) can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language, and combined with hardware implementations.

[0088] While the invention has been particularly shown and described with reference to a preferred embodiment thereof, it will be understood by those skilled in the art that various changes in form and detail may be made without departing from the scope of the present invention as set forth in the following claims. Furthermore, although elements of the invention may be described or claimed in the singular, the plural is contemplated unless limitation to the singular is explicitly stated.

2011314228 11 Jun 2014

[0089] Throughout this specification and the claims which follow, unless the context requires otherwise, the word "comprise", and variations such as "comprises" and "comprising", will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

[0090] The reference in this specification to any prior publication (or information derived from it), or to any matter which is known, is not, and should not be taken as an acknowledgment or admission or any form of suggestion that that prior publication (or information derived from it) or known matter forms part of the common general knowledge in the field of endeavour to which this specification relates.

2011314228 11 Jun 2014

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A method of processing graphics data for transmission to a remote computing device, the method comprising:

receiving graphics data representative of a client screen associated with a virtual machine session;

receiving information indicative of available bandwidth for said transmission and, based on the information, determining that the available bandwidth meets a predetermined threshold, and

entropy encoding coefficients of transformed graphics data using a compact stream of bit tokens that form groups that align to byte boundaries, wherein:

runs of zeroes are encoded in a variable number of multiples of a quantum size;

literal values are encoded using an entry in a cache of recently used literal values; and

other values are encoded using a minimum number of multiples of the quantum size.

2 The method of claim 1, wherein said quantum size is a nibble.

3. The method of claim 1 or 2, further comprising generating a stream of entropy encoding op-codes and a large value stream.

4. The method of claim 3, further comprising entropy encoding the large value stream with a multi-byte encoding scheme that uses less bytes for small values than large values.

5. The method of claim 4, wherein the encoding scheme is configured to operate only on fixed-byte boundaries.

6. The method of claim 4 or 5, further comprising dividing said graphics data into data tiles, processing said data tiles into tile components, and performing said entropy encoding on said tile components.

2011314228 11 Jun 2014

7. The method of any one of claims 1-6, further comprising transmitting the encoded coefficients to a computing device configured to process the encoded coefficients based on the quantum size.

8. The method of claim 7, further comprising entropy decoding the encoded coefficients.

9. A system for processing graphics data for transmission to a remote computing device, comprising:

a computing device comprising at least one processor;

a memory communicatively coupled to said processor when said system is operational; said memory having stored therein computer instructions that upon execution by the at least one processor cause:

receiving graphics data representing a client screen associated with a virtual machine session;

dividing said graphics data into data tiles;

entropy encoding coefficients of transformed data tiles using a stream of bit tokens that form groups that align to byte boundaries, wherein:

runs of zeroes are encoded in a variable number of multiples of a quantum size;

literal values are encoded using an entry in a cache of recently used literal values; and

other values are encoded using a minimum number of units of the quantum size.

10. The system of claim 9, further comprising transmitting the encoded coefficients to a computing device configured to process the encoded coefficients based on the quantum size.

11. The system of claim 9 or 10, wherein the encoded data is operable for efficient decoding by a entropy decoding process configured to operate on the encoded data on a per-byte basis.

12. The system of any one of claims 9-11, wherein said quantum size is a nibble.

2011314228 11 Jun 2014

13. The system of any one of claims 9-12, further comprising generating a stream of entropy encoding op-codes and a large value stream.

14. The system of claim 13, further comprising entropy encoding the large value stream with a multi-byte encoding scheme that uses less bytes for small values than large values.

15. The system of any one of claims 9-14, wherein the encoding scheme is configured to operate only on fixed-byte boundaries.

16. A computer readable storage device storing thereon computer executable instructions for processing graphics data for transmission to a client computer, said instructions for:

receiving graphics data representative of a client screen associated with a virtual machine session; and

entropy encoding coefficients of transformed graphics data using a compact stream of bit tokens that form groups that align to byte boundaries such that the encoded data can be decoded using a byte-based decoding process, wherein:

runs of zeroes are encoded in a variable number of multiples of a nibble;

literal values are encoded using an entry in a cache of recently used literal values; and

other values are encoded using a minimum number of multiples of a nibble.

17. The computer readable storage device of claim 16, further comprising instructions for transmitting the encoded coeffficients to a computing device configured to process the encoded coefficients based on a per-byte basis.

18. The computer readable storage device of claim 16 or 17, further comprising generating a stream of entropy encoding op-codes and a large value stream.

19. The computer readable storage device of claim 18, further comprising entropy encoding the large value stream with a multi-byte encoding scheme that uses less bytes for

small values than large values.

20. The computer readable storage device of claim 19, wherein the encoding scheme is configured to operate only on fixed-byte boundaries.

21. The method of any one of claims 1-8, wherein the graphics data comprises an indication of a first portion of the client screen that has changed from a previous frame and an indication of a second portion of the client screen that has not changed from the previous frame, and wherein the entropy encoding includes entropy encoding coefficients of the graphics data corresponding to the first portion of the client screen using the compact stream of bit tokens that form groups that align to the byte boundaries.

22. The system of any one of claims 9-15, further comprising determining whether at least a first data tile has changed from a previous frame and whether available bandwidth meets a predetermined threshold; and

when at least a first data tile has changed from a previous frame and when the available bandwidth meets the predetermined threshold: entropy encoding the coefficients of the first data tile using the stream of bit tokens that form groups that align to the byte boundaries.

23. The device of any one of claims 16-20, wherein the graphics data comprises an indication of a first portion of the client screen that has changed from a previous frame and an indication of a second portion of the client screen that has not changed from the previous frame, and wherein the entropy encoding includes entropy encoding coefficients of the graphics data corresponding to the first portion of the client screen using the compact stream of bit tokens that form groups that align to the byte boundaries such that the encoded data can be decoded using a byte-based decoding process.

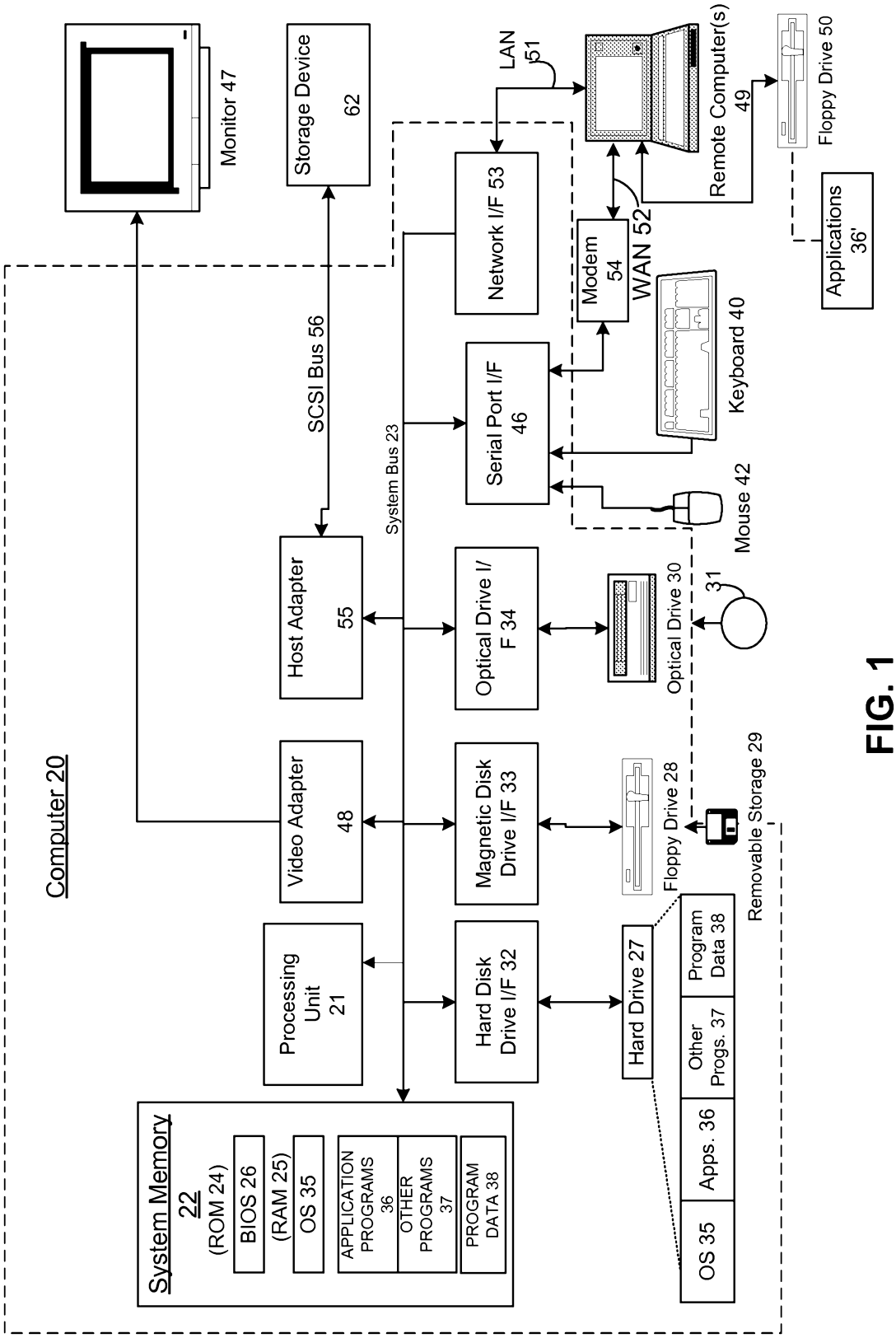
Computer 20
System Memory
22
(ROM 24)
BIOS 26
(RAM 25)
OS 35
APPLICATION PROGRAMS 36
OTHER PROGRAMS 37
PROGRAM DATA 38
Processing Unit 21
Video Adapter 48
Host Adapter 55
Hard Disk Drive I/F 32
Magnetic Disk Drive I/F 33
Optical Drive I/F 34
Serial Port I/F 46
Network I/F 53
Modem 54
WAN 52
LAN 51
Remote Computer(s) 49
Floppy Drive 50
Applications 36'
Monitor 47
Storage Device 62
SCSI Bus 56
System Bus 23
Keyboard 40
Mouse 42
Optical Drive 30
31
Hard Drive 27
Floppy Drive 28
Removable Storage 29
OS 35
Apps. 36
Other Progs. 37
Program Data 38

FIG. 1

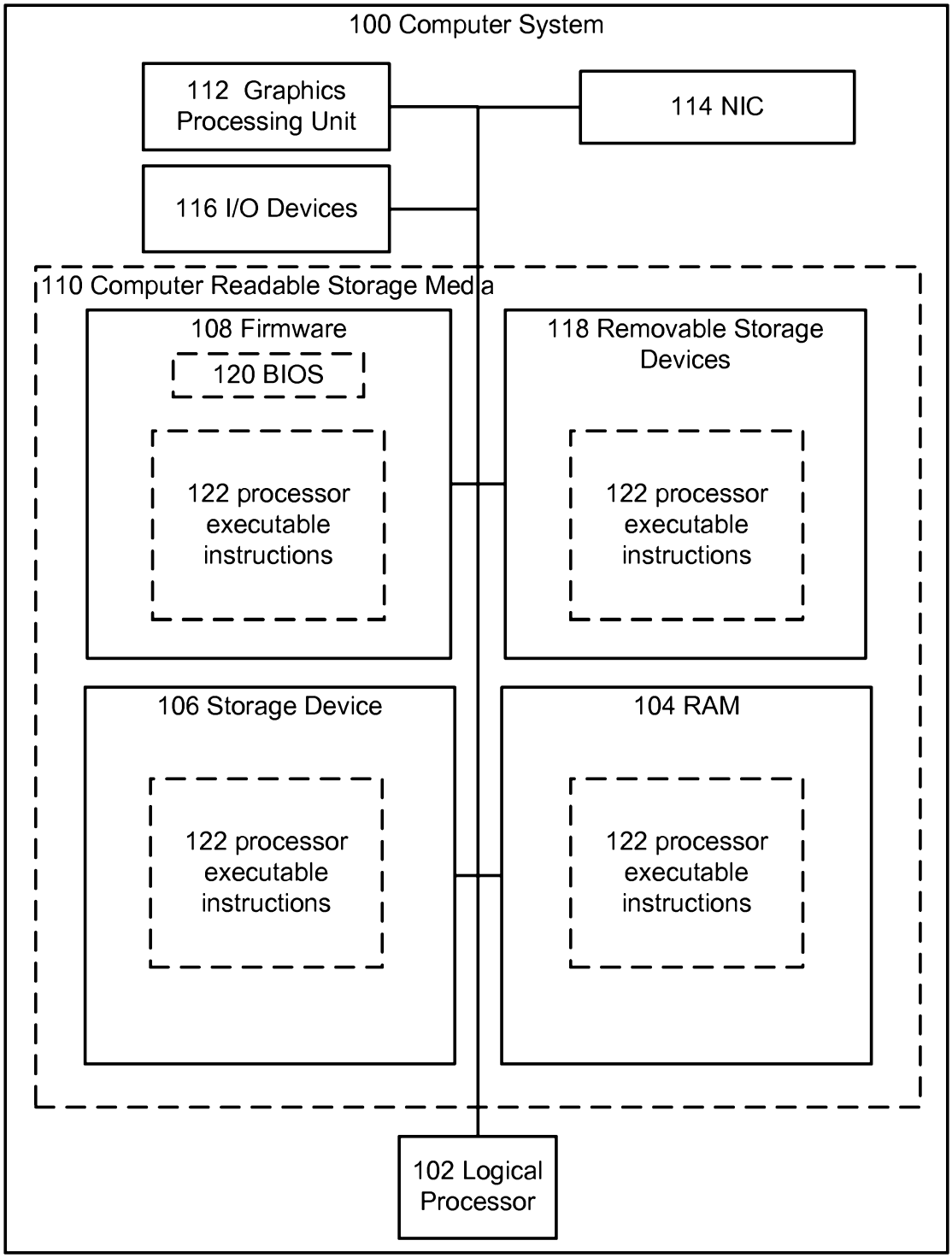
100 Computer System
112 Graphics Processing Unit
114 NIC
116 I/O Devices
110 Computer Readable Storage Media
108 Firmware
120 BIOS
122 processor executable instructions
118 Removable Storage Devices
122 processor executable instructions
106 Storage Device
122 processor executable instructions
104 RAM
122 processor executable instructions
102 Logical Processor

FIG. 2

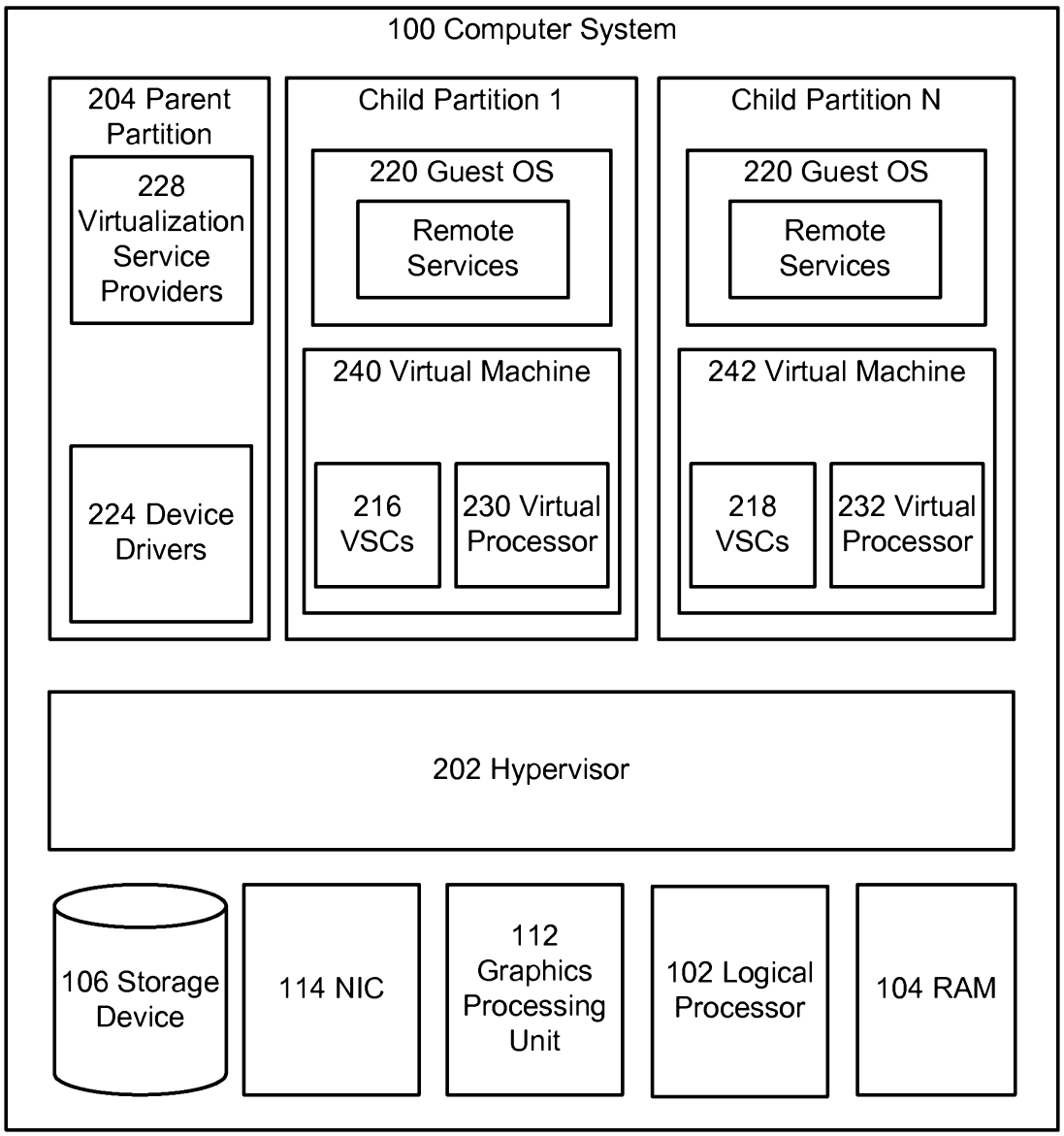
100 Computer System
204 Parent Partition
228 Virtualization Service Providers
224 Device Drivers
Child Partition 1
220 Guest OS
Remote Services
240 Virtual Machine
216 VSCs
230 Virtual Processor
Child Partition N
220 Guest OS
Remote Services
242 Virtual Machine
218 VSCs
232 Virtual Processor
202 Hypervisor
106 Storage Device
114 NIC
112 Graphics Processing Unit
102 Logical Processor
104 RAM

FIG. 3

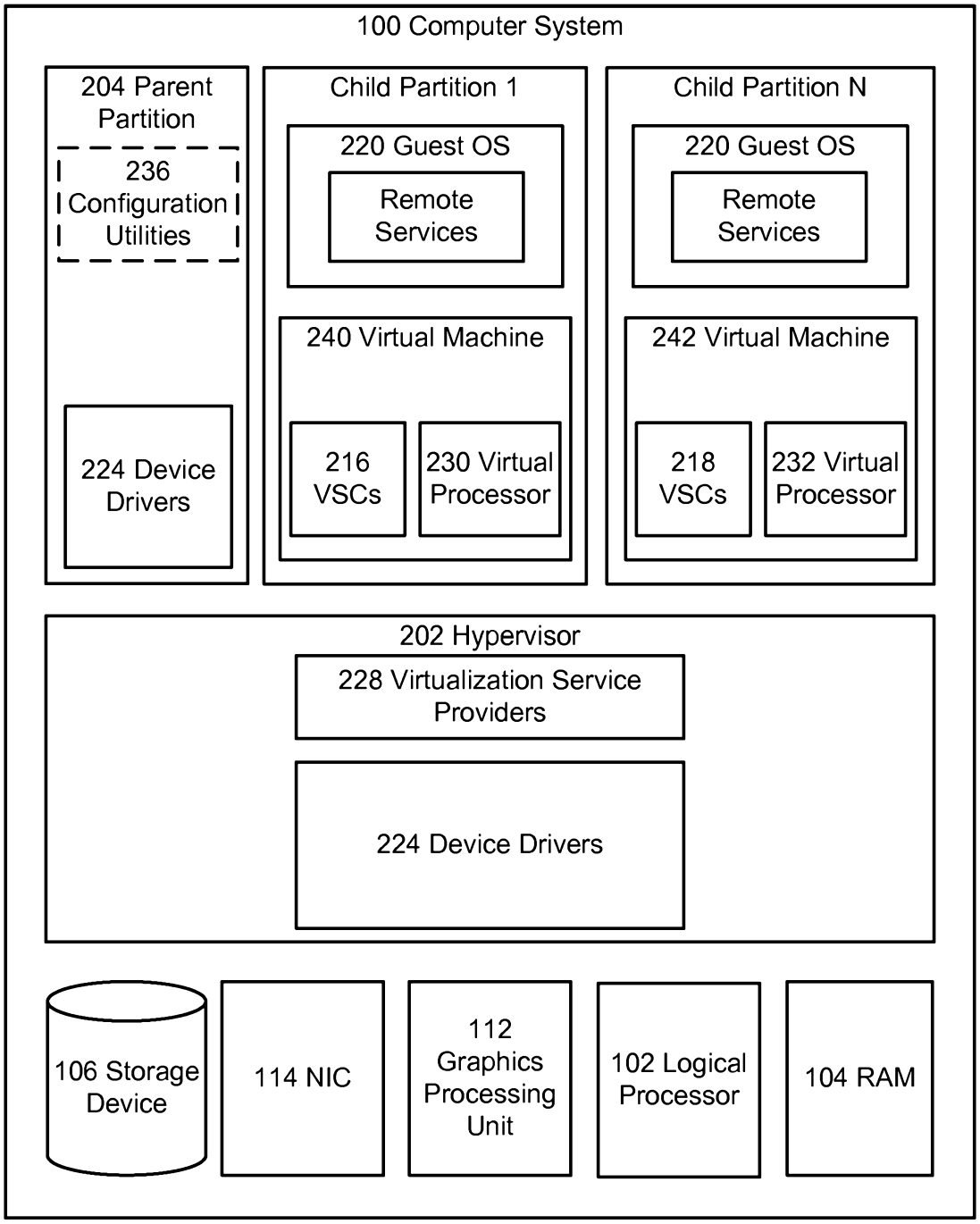
100 Computer System
204 Parent Partition
236 Configuration Utilities
224 Device Drivers
Child Partition 1
220 Guest OS
Remote Services
240 Virtual Machine
216 VSCs
230 Virtual Processor
Child Partition N
220 Guest OS
Remote Services
242 Virtual Machine
218 VSCs
232 Virtual Processor
202 Hypervisor
228 Virtualization Service Providers
224 Device Drivers
106 Storage Device
114 NIC
112 Graphics Processing Unit
102 Logical Processor
104 RAM

FIG. 4

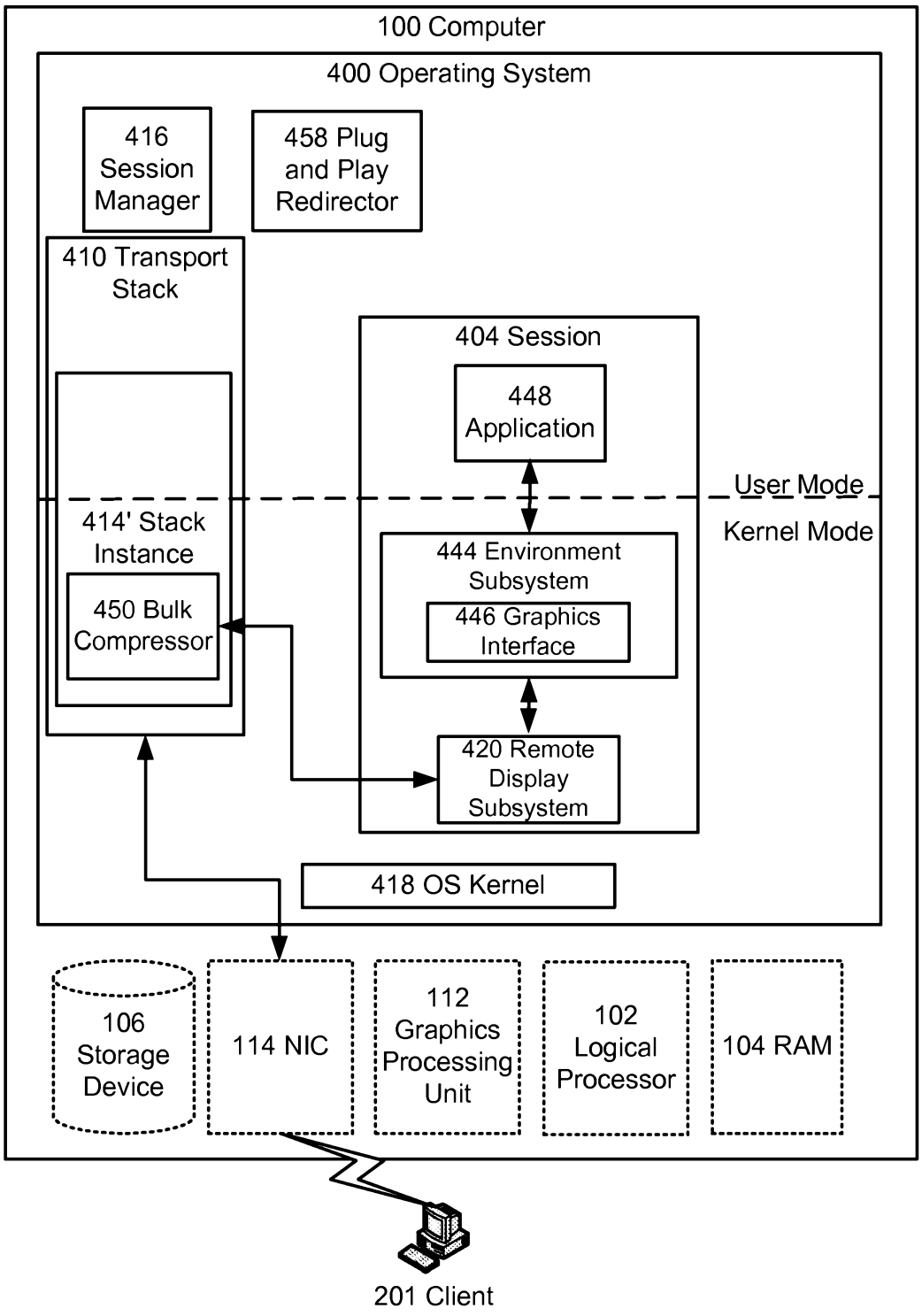
100 Computer
400 Operating System
416 Session Manager
458 Plug and Play Redirector
410 Transport Stack
404 Session
448 Application
User Mode
Kernel Mode
414' Stack Instance
450 Bulk Compressor
444 Environment Subsystem
446 Graphics Interface
420 Remote Display Subsystem
418 OS Kernel
106 Storage Device
114 NIC
112 Graphics Processing Unit
102 Logical Processor
104 RAM
201 Client

FIG. 5

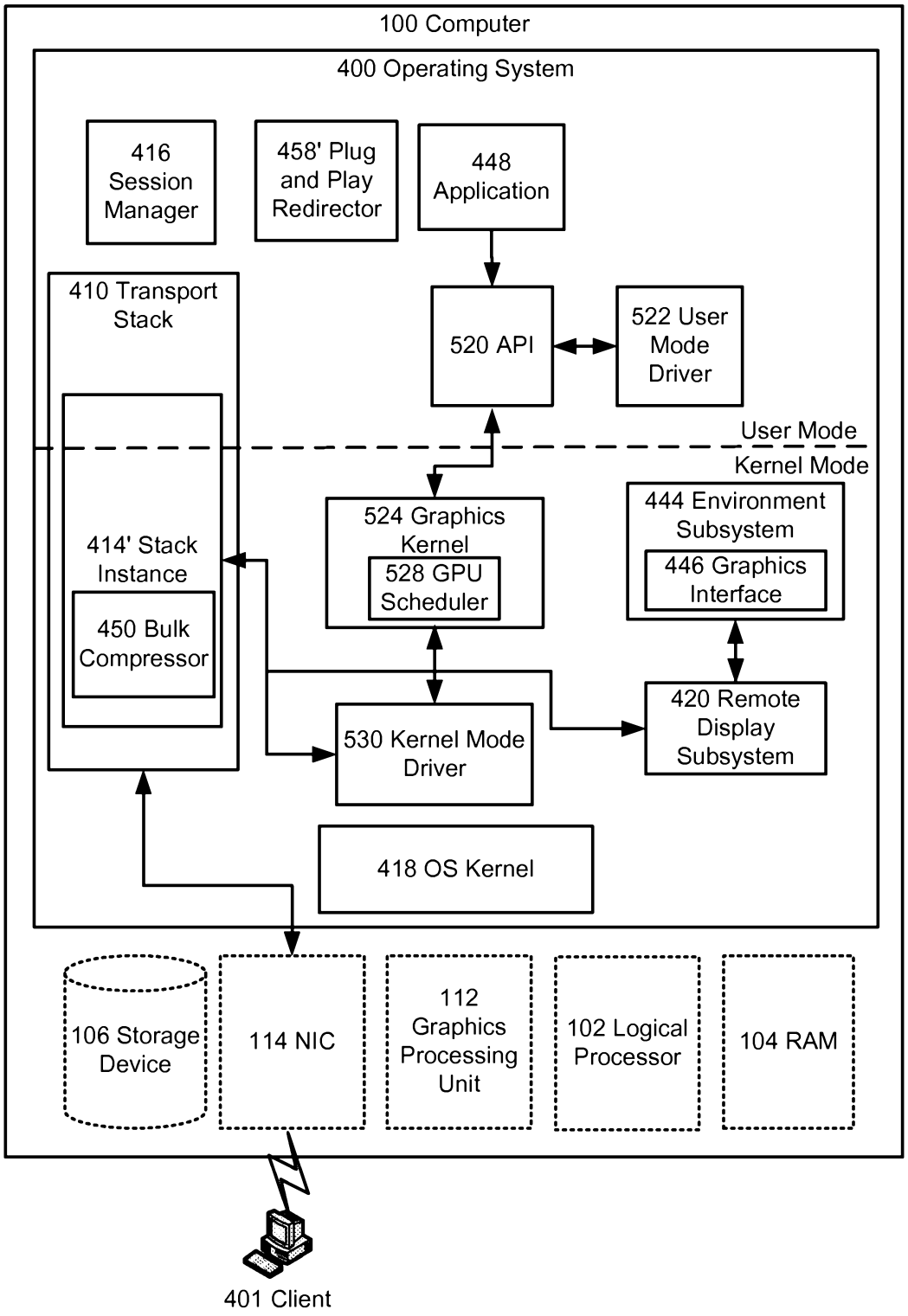
100 Computer
400 Operating System
416 Session Manager
458' Plug and Play Redirector
448 Application
410 Transport Stack
520 API
522 User Mode Driver
User Mode
Kernel Mode
414' Stack Instance
450 Bulk Compressor
524 Graphics Kernel
528 GPU Scheduler
444 Environment Subsystem
446 Graphics Interface
420 Remote Display Subsystem
530 Kernel Mode Driver
418 OS Kernel
106 Storage Device
114 NIC
112 Graphics Processing Unit
102 Logical Processor
104 RAM
401 Client

FIG. 6

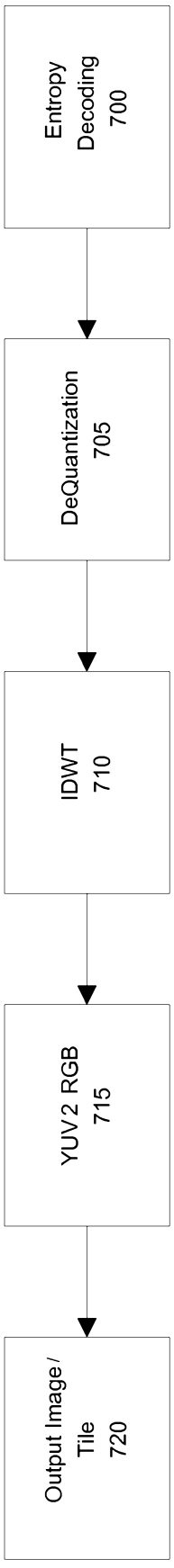
Entropy Decoding 700
DeQuantization 705
IDWT 710
YUV 2 RGB 715
Output Image / Tile 720

FIG. 7

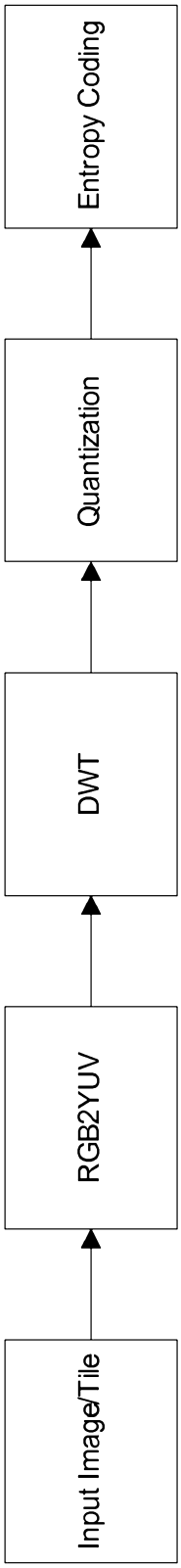

FIG. 8

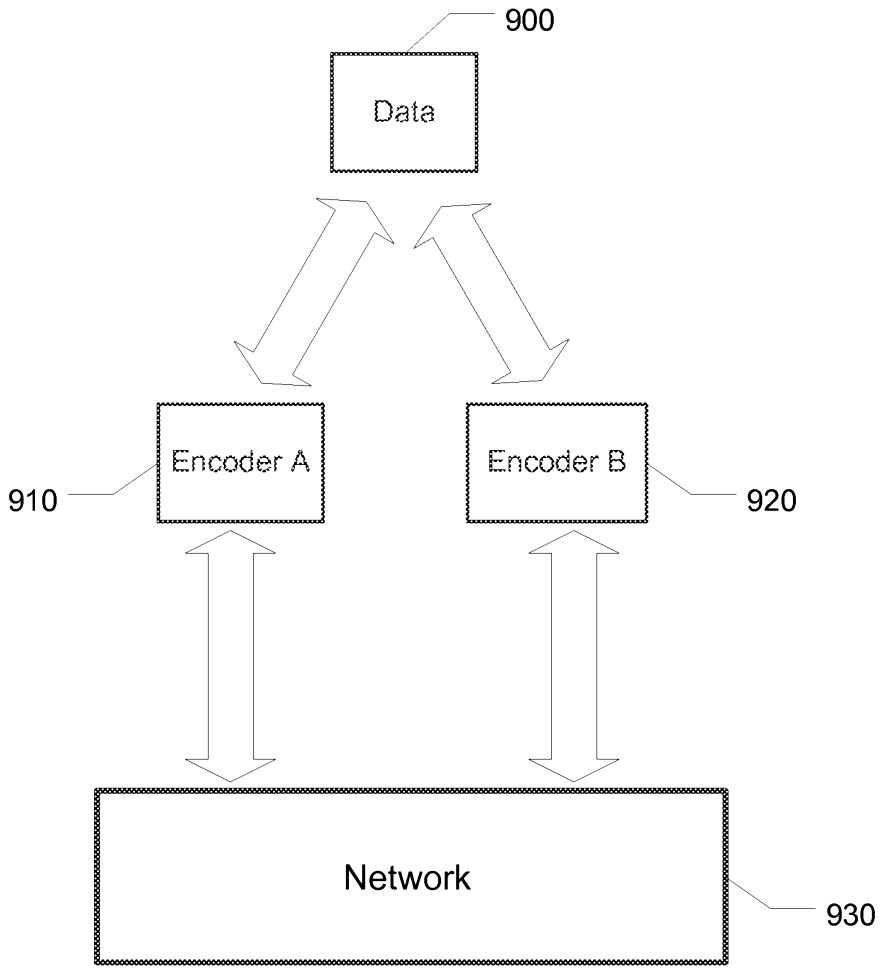

FIG. 9

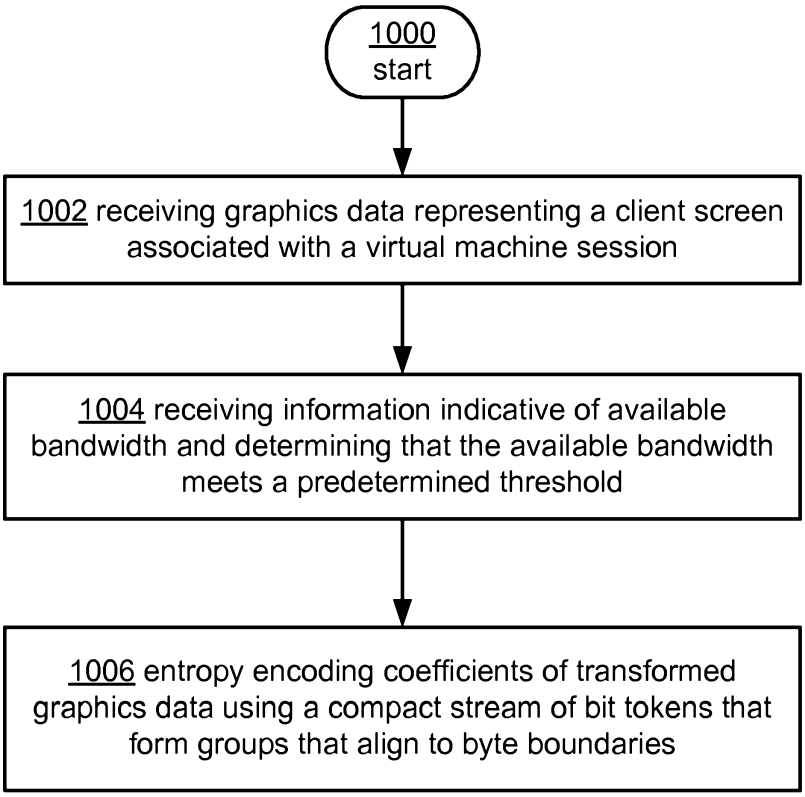
1000
start
1002 receiving graphics data representing a client screen associated with a virtual machine session
1004 receiving information indicative of available bandwidth and determining that the available bandwidth meets a predetermined threshold
1006 entropy encoding coefficients of transformed graphics data using a compact stream of bit tokens that form groups that align to byte boundaries

FIG. 10

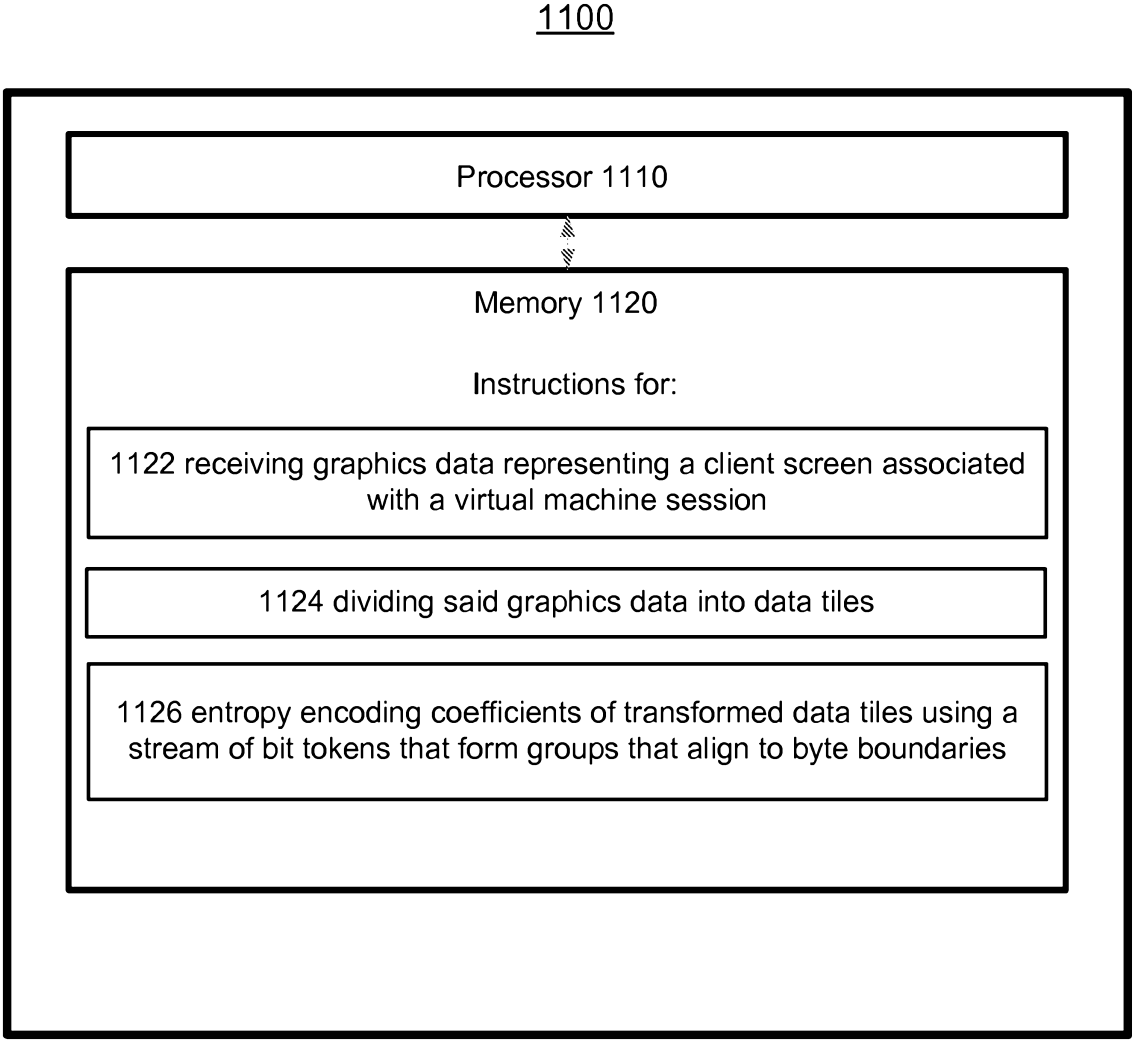
1100
Processor 1110
Memory 1120
Instructions for:
1122 receiving graphics data representing a client screen associated with a virtual machine session
1124 dividing said graphics data into data tiles
1126 entropy encoding coefficients of transformed data tiles using a stream of bit tokens that form groups that align to byte boundaries

FIG. 11