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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 April 2006 (13.04.2006)

PCT

(10) International Publication Number
WO 2006/039595 A2

(51) International Patent Classification:
G06F 11/00 (2006.01)

(21) International Application Number:
PCT/US2005/035375

(22) International Filing Date:
29 September 2005 (29.09.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/953,887 29 September 2004 (29.09.2004) US

(71) Applicant *(for all designated States except US)*: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **MUKHERJEE, Shubhendu** [IN/US]; 43 GATES STREET, FRAMINGHAM, MA 01702 (US). **EMER, Joel** [US/US]; 20 BLACKHORSE DRIVE, ACTON, MA 01720 (US). **REINHARDT, Steven** [US/US]; 3158 KILBURN PARK CIR., ANN ARBOR, MI 48105 (US). **WEAVER, Christopher** [US/US]; 1009 APPLEBRIAR LANE, MARLBOROUGH, MA 01752 (US).

(74) Agents: **VINCENT, Lester, J.** et al.; BLAKELY SOKOLOFF TAYLOR & ZAFMAN, 12400 Wilshire Boulevard||7th Floor, Los Angeles, CA 950025 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 2006/039595 A2

(54) Title: EXECUTING CHECKER INSTRUCTIONS IN REDUNDANT MULTITHREADING ENVIRONMENTS

(57) Abstract: A method and apparatus for a checker instruction in a redundant multithreading environment is described. In one embodiment, when RMT requires, a processor may issue a checker instruction in both a leading thread and a trailing thread. The checker instruction may travel down individual pipelines for each thread independently until it reaches a buffer at the end of each pipeline. Then, prior to committing the checker instruction, the checker instruction looks for its counterpart and does a comparison of the instructions. If the checker instructions match, the checker instructions commit and retires otherwise an error is declared.

EXECUTING CHECKER INSTRUCTIONS IN REDUNDANT MULTITHREADING ENVIRONMENTS

BACKGROUND INFORMATION

[0001] Current redundant-execution systems commonly employ a checker circuit that is self-checking and is implemented in hardware. Similar to the checker circuit is the compare instruction that would compare the results from two threads (e.g., store address and data). It may be possible to duplicate the compare instruction in both threads to get the effect of self-checking via duplication.

[0002] Unfortunately, by duplicating the compare instruction the architecture would lose the performance advantage of redundant multithreading (RMT). RMT's performance advantage comes from having the leading and trailing threads sufficiently apart such that the leading thread can prefetch cache misses and branch mispredictions for the trailing thread. If the compare instruction is duplicated, not only are additional queues needed, incurring higher overhead, but also the architecture would be unable to keep the two threads sufficiently apart because of the synchronization required in both directions. Thus what is needed is an instruction that can achieve lower failure rate without sacrificing the performance advantage of RMT.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] Various features of the invention will be apparent from the following description of preferred embodiments as illustrated in the accompanying drawings, in which like reference numerals generally refer to the same parts

throughout the drawings. The drawings are not necessarily to scale, the emphasis instead being placed upon illustrating the principles of the inventions.

[0004] Figure 1 is a block diagram of one embodiment of a multithreaded architecture.

[0005] Figure 2 is a flowchart illustrating one method of generating a checker instruction.

[0006] Figure 3 is a flowchart illustrating one implementation of a checker instruction in either thread.

[0007] Figure 4 is a block diagram of one embodiment of a checker instruction.

[0008] Figure 5 is a block diagram of a system that may provide an environment for multithreaded processors.

[0009] Figure 6 is a block diagram of an alternative system that may provide an environment for multithreaded processors.

DETAILED DESCRIPTION

[0010] In the following description, for purposes of explanation and not limitation, specific details are set forth such as particular structures, architectures, interfaces, techniques, etc. in order to provide a thorough understanding of the various aspects of the invention. However, it will be apparent to those skilled in the art having the benefit of the present disclosure that the various aspects of the invention may be practiced in other examples that depart from these specific details. In certain instances, descriptions of well-known devices, circuits, and methods are omitted so as not to obscure the description of the present invention with unnecessary detail.

[0011] Method and apparatuses for a checker instruction in a redundant multithreading environment are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a through understanding of the invention. It will be apparent, however, to one skilled in the art that the invention can be practiced without these specific details.

[0012] Figure 1 is a block diagram of one embodiment of a redundantly multithreaded architecture. In a redundantly multithreaded architecture faults can be detected by executing two copies of a program as separate threads.

[0013] Each thread is provided with identical inputs and the outputs are compared to determine whether an error has occurred. Redundant multithreading can be described with respect to a concept referred to herein as the "sphere of replication." The sphere of replication is the boundary of logically or physically redundant operation.

[0014] Components within the sphere of replication 100 (e.g., a processor executing leading thread 105 and a processor executing trailing thread 110) are subject to redundant execution. In contrast, components outside sphere of replication 100 (e.g., memory 115) are not subject to redundant execution. Fault protection is provided by other techniques, for example, error correcting code for memory 115. Other devices may be outside of sphere of replication 100 and/or other techniques can be used to provide fault protection for devices outside of sphere of replication 100.

[0015] Data entering sphere of replication 100 enter through input replication agent 120 that replicates the data and sends a copy of the data to leading thread 105 and to trailing thread 110. Similarly, data exiting sphere of

replication 100 exit through output comparison agent 125 that compares the data and determines whether an error has occurred. Varying the boundary of sphere of replication 100 results in a performance versus amount of hardware tradeoff. For example, replicating memory 115 would allow faster access to memory by avoiding output comparison of store instructions, but would increase system cost by doubling the amount of memory in the system.

[0016] One embodiment of the present invention proposes a mechanism to check a checker circuit in a software implementation of RMT. Because RMT compares outputs of committed instructions (requiring instruction-by-instruction comparison), it may also be implemented in software. If the software implementation of RMT compared every instruction, it would incur significant overhead. Instead, however, RMT allows the comparison of only store instructions and replication of only load instructions, which may significantly reduce the software overhead of an RMT implementation.

[0017] Figure 2 illustrates one method of generating a checker instruction. Initially, as in most computers, a complier generates instructions. From the complier, the computer now has a binary program which may be, but not limited to, a sequence of store instructions 200. Next, a binary translator may insert a checker instruction prior to each store instruction in the binary program 205. The binary translator may be any binary translator well known in the art. Upon translating the binary program, the system creates a binary program for both the leading thread and the trailing thread. The binary program for the leading thread adds the checker instruction to the store instruction 210. The binary program for the trailing thread replaces the store instruction with the peer checker instruction of the leading thread 215.

[0018] Figure 3 illustrates one implementation of the checker instruction. When RMT requires a compare, a processor may issue a checker instruction in both the leading thread and the trailing thread 300. Each checker instruction may carry a 64-bit quantity from each thread. The checker instruction may travel down individual pipelines for each thread independently 305 until it reaches a buffer at the end of each pipeline. The checker instruction waits for its peer checker instruction in the buffer 310. These two checker instructions may then do a comparison of the 64-bit quantities they are carrying 315. On a mismatch, both may report errors. On a match, they may let the processors commit the checker instruction 320. The pipelines can be from different processors in a CMP or from the same multithreaded processor as in a SMT processor.

[0019] In this implementation, the checker instruction does not hold up the leading instruction from processing instructions from the instruction queue. Rather, it only holds up the retire pointer till the corresponding checker instruction from the trailing thread shows up. Also, if the environment is not a RMT environment, then the checker instruction may be treated as a NOP.

[0020] Figure 4 is a block diagram of one example of a checker instruction flowing through two pipelines. Assuming a store instruction: R1 → [R2], stores the value in register R1 to the memory location pointed by the address in register R2. This store instruction may be replicated in both the leading thread and the trailing thread with the checker instruction. The store instruction in the leading thread may include both the checker instruction and the store instruction as shown below:

Checkerinst R1

Checkerinst R2

Store: R1 → [R2]. Thus, the leading thread may contain the checker instruction along with the store instruction when traveling through its pipeline 400.

[0021] The store instruction in the trailing thread may include only the checker instruction as shown below:

Checkerinst R1

Checkerinst R2. Thus, the trailing thread does not have the store instruction traveling through its pipeline 405.

[0022] The checkerinst R1 from the leading thread in pipeline 400 waits for peer checker instruction in buffer 410. The checkerinst R1 from the trailing thread in pipeline 405 waits for it peer checker instruction in buffer 415. The checker instruction always looks or waits for its counterpart or peer. If there is a mirror thread, the checker instruction will look or wait for the thread in the buffers 410, 405 to make sure the mirror thread is there and then compares the checker instructions.

[0023] The checkerinst R1 from the leading thread and checkerinst R1 from the trailing thread may pair up, due to commit order, and compare the register specifier and value of R1 to ensure that the registers did not have any errors in them. If no errors are found, the checker instructions commit 420. Once the checker instructions commit, the value of R1 is stored. The value of R1 is carried through to the commit point and then stored. Thus, the system is able to check all the stores simultaneously instead store by store as done previously.

[0024] Figure 5 is a block diagram of a system that can provide an

environment for multithreaded processors. The system illustrated in Fig.5 is intended to represent a range of systems. Alternative systems may include more, fewer and/or different components.

[0025] System 500 includes bus 510 or other communication device to communicate information, and processor(s) 520 coupled to bus 510 to process information. System 500 further includes random access memory (RAM) or other dynamic memory as well as static memory, for example, a hard disk or other storage device 535 (referred to as memory), couple to bus 510 via memory controller 530 to store information and instructions to be executed by processor(s) 520. Memory 535 also can be used to store temporary variables or other intermediate information during execution of instructions by processor(s) 520. Memory controller 530 can include one or more components to control one or more types of memory and/or associated memory devices. System 500 also includes read only memory (ROM) and/or other static storage device 540 coupled to bus 510 to store static information and instructions for processor(s) 520.

[0026] System 500 can also be coupled via a bus 510 to input/output (I/O) interface 550. I/O interface 550 provides an interface to I/O devices 555, which can include, for example, a cathode ray tube (CRT) or liquid crystal display (LCD), to display information to a computer user, an alphanumeric input device including alphanumeric and other keys and/or a cursor control device, such as a mouse, a trackball, or cursor direction keys. System 500 further includes network interface 560 to provide access to a network, such as a local area network, whether wired or wireless.

[0027] Instructions are provided to memory 535 from a storage device,

such as magnetic disk, a read-only memory (ROM) integrated circuit, CD_ROM, DVD, via a remote connection (e.g., over a network via network interface 860) that is either wired or wireless, etc.

[0028] Referring now to Fig, 6, the system 600 generally shows a system where processors, memory, and input/output devices are interconnected by a number of point-to-point interfaces. The system 600 may also include several processors, of which only two, processors 605, 610 are shown for clarity. Processors 605, 610 may each include a local memory controller hub (MCH) 615, 620 to connect with memory 625, 630. Processors 605, 610 may exchange data via a point-to-point interface 635 using point-to-point interface circuits 640, 645. Processors 605, 610 may each exchange data with a chipset 650 via individual point-to-point interfaces 655, 660 using point to point interface circuits 665, 670, 675, 680. Chipset 650 may also exchange data with a high-performance graphics circuit 685 via a high-performance graphics interface 690.

[0029] The chipset 650 may exchange data with a bus 616 via a bus interface 695. In either system, there may be various input/output I/O devices 614 on the bus 616, including in some embodiments low performance graphics controllers, video controllers, and networking controllers. Another bus bridge 618 may in some embodiments be used to permit data exchanges between bus 616 and bus 620. Bus 620 may in some embodiments be a small computer system interface (SCSI) bus, an integrated drive electronics (IDE) bus, or a universal serial bus (USB) bus. Additional I/O devices may be connected with bus 620. These may include keyboard and cursor control devices 622, including mouse, audio I/O 624, communications devices 626, including

modems and network interfaces, and data storage devices 628. Software code 630 may be stored on data storage device 628. In some embodiments, data storage device 628 may be a fixed magnetic disk, a floppy disk drive, an optical disk drive, a magneto-optical disk drive, a magnetic tape, or non-volatile memory including flash memory.

[0030] Throughout the specification, the term, "instruction" is used generally to refer to instructions, macro-instructions, instruction bundles or any of a number of other mechanisms used to encode processor operations.

[0031] In the following description, for purposes of explanation and not limitation, specific details are set forth such as particular structures, architectures, interfaces, techniques, etc. in order to provide a thorough understanding of the various aspects of the invention. However, it will be apparent to those skilled in the art having the benefit of the present disclosure that the various aspects of the invention may be practiced in other examples that depart from these specific details. In certain instances, descriptions of well-known devices, circuits, and methods are omitted so as not to obscure the description of the present invention with unnecessary detail.

WHAT IS CLAIMED IS:

1. A method comprising:

generating checker instructions in a leading thread and a trailing thread;

waiting for a peer checker instruciton from the leading thread and the trailing thread; and

comparing the peer checker instructions from the leading thread and the trailing thread.

2. The method of claim 1 wherein the leading thread includes the checker instruction and a selected instruction and the trailing thread includes the checker instruction.

3. The method of claim 2 wherein the generating checker instruction in leading thread further includes inserting the checker instruction prior to the selected instruction.

4. The method of claim 1 wherein the checker instructions traveling through corresponding pipelines for the leading thread and the trailing thread.

5. The method of claim 1 further comprising generating a checker instruction.

6. The method of claim 2 further comprising committing the checker instructions.

7. The method of claim 6 further comprising storing the selected instruction for the leading thread if the comparing the corresponding checker instructions from the leading thread and the trailing thread match.

8. The method of claim 1 wherein the leading thread and the trailing thread are executed by a single processor.

9. The method of claim 1 wherein the leading thread and the trailing thread are executed by multiple processors.

10. An apparatus comprising:

leading thread circuitry to execute a leading thread of instructions;

trailing thread circuitry to execute a trailing thread of instructions; and

a commit unit to commit corresponding checker instructions from the leading thread and the trailing thread.

11. The apparatus of claim 10 wherein the leading thread of instructions comprises checker instructions and selected instructions.

12. The apparatus of claim 11 wherein the trailing thread of instructions comprises checker instructions.

13. The apparatus of claim 10 wherein the leading thread circuitry and trailing thread circuitry include a pipeline.

14. The apparatus of claim 13 wherein the leading thread and trailing thread are executed by a single processor.

15. The apparatus of claim 13 wherein the leading thread and the trailing thread are executed by multiple processors.

16. The apparatus of claim 12 further comprising buffers coupled to the leading thread circuitry and the trailing thread circuitry.

17. The apparatus of claim 16 wherein the checker instruction of the leading thread and the checker instruction of the trailing thread wait for corresponding checker instructions in the buffer.

18. The apparatus of claim 2 wherein the selected instruction is stored if the corresponding checker instructions match.

19. The apparatus of claim 2 wherein the commit unit generates an error if the corresponding checker instructions do not match.

20. The apparatus of claim 11 wherein the checker instruction is placed prior to the selected instruction by a binary translator.

21. The apparatus of claim 11 wherein the selected instruction is a store instruction.

22. A system comprising:

a first processor comprising:

leading thread circuitry to execute a leading thread of checker instructions;

trailing thread circuitry to execute a trailing thread of the checker instructions; and

a retire unit to retire corresponding checker instructions from the leading thread and the trailing thread,

a first interface to a second processor;

a second interface to input/output devices; and

an audio input-output device coupled to the second interface.

23. The system of claim 22 wherein the leading thread of instructions comprises checker instructions and selected instructions.

24. The system of claim 23 wherein the trailing thread of instructions comprises checker instructions.

25. The system of claim 22 wherein the leading thread circuitry and trailing thread circuitry include a pipeline.

26. The system of claim 24 wherein the selected instructions are stored if the corresponding checker instructions from the leading thread and the trailing thread match.

27. The system of claim 26 further comprising buffers coupled with the leading thread circuitry and the trailing thread circuitry.

28. The system of claim 26 wherein the retire unit generates an error if the corresponding checker instructions do not match.

29. The system of claim 23 wherein the checker instruction is place prior to the selected instruction by a binary translator.

30. The system of claim 23 wherein the selected instruction is a store instruction.

31. The system of claim 22 wherein the first and second interfaces are point to point interfaces.

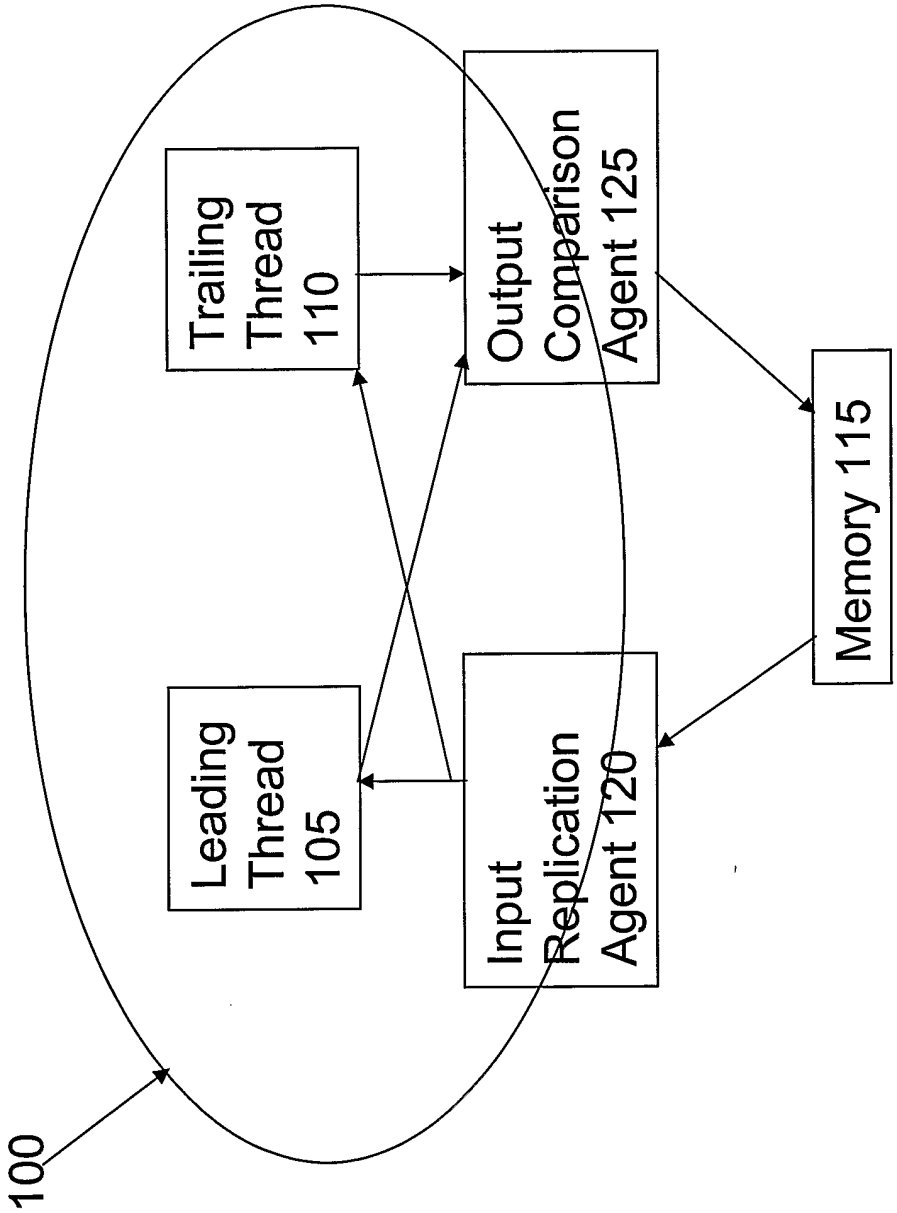
Trailing Thread 110
Output Comparison Agent 125
Memory 115
Leading Thread 105
Input Replication Agent 120
100

Fig.1

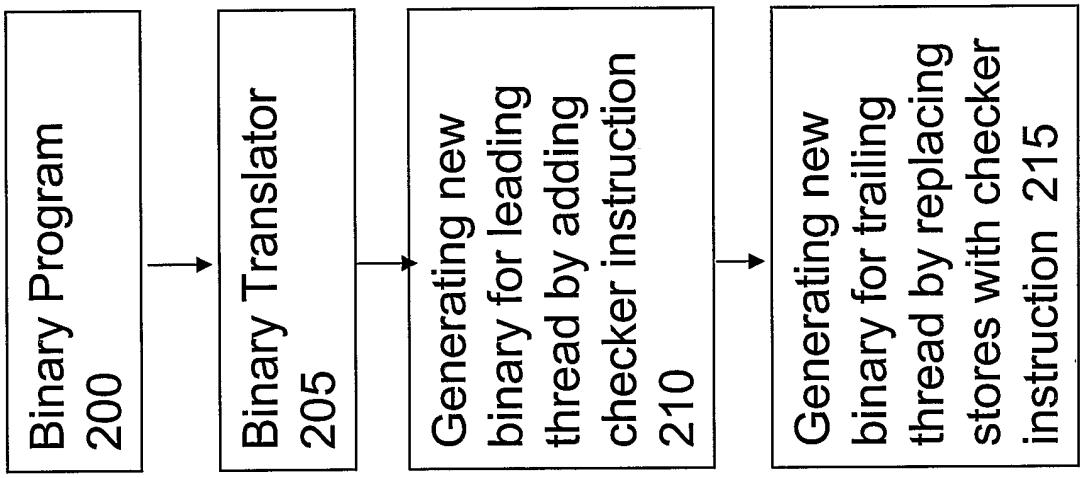
Binary Program 200
Binary Translator 205
Generating new binary for leading thread by adding checker instruction 210
Generating new binary for trailing thread by replacing stores with checker instruction 215

Fig. 2

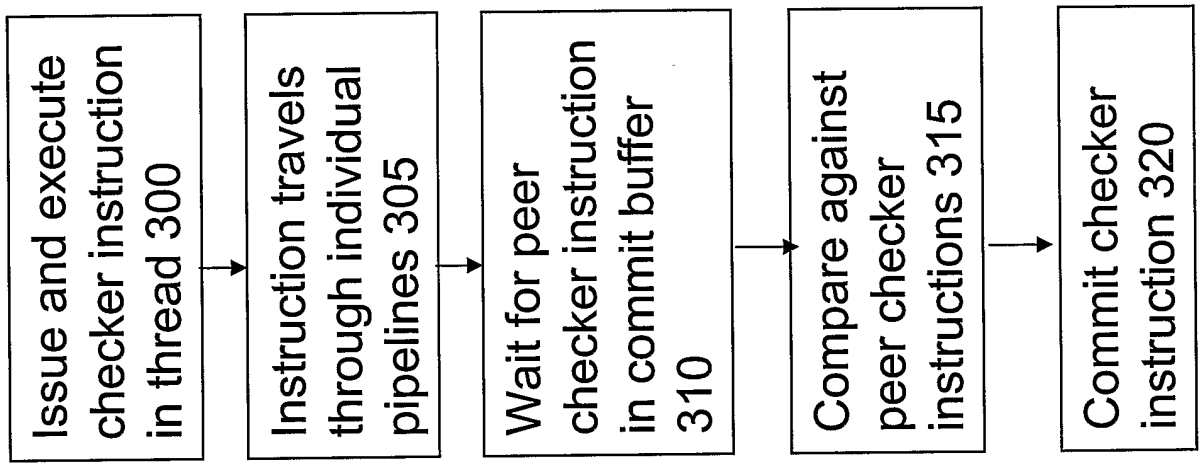
Issue and execute checker instruction in thread 300
Instruction travels through individual pipelines 305
Wait for peer checker instruction in commit buffer 310
Compare against peer checker instructions 315
Commit checker instruction 320

Fig. 3

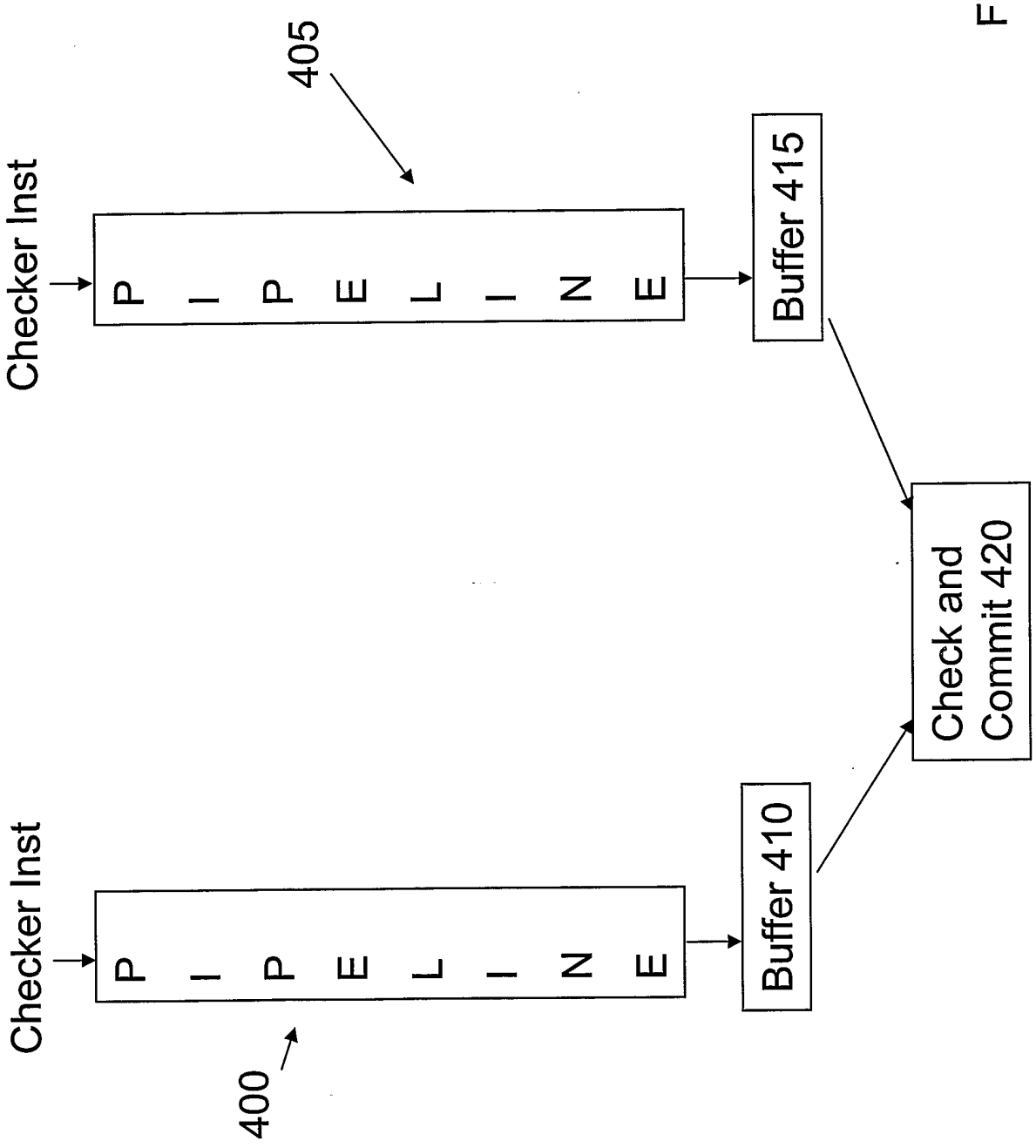
Checker Inst
P I P E L I N E
405
Buffer 415
Checker Inst
P I P E L I N E
400
Buffer 410
Check and Commit 420

Fig. 4

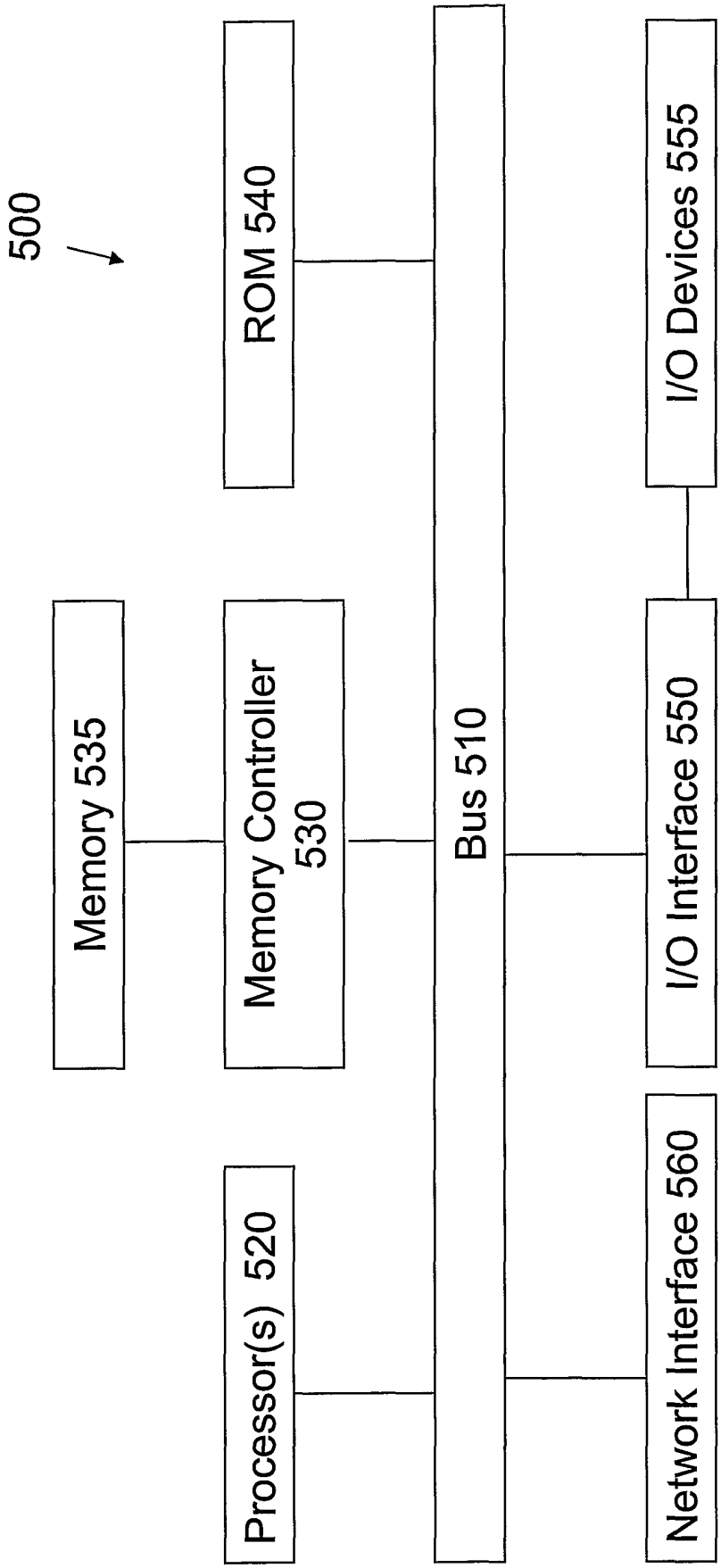
500
ROM 540
I/O Devices 555
Memory 535
Memory Controller 530
Bus 510
I/O Interface 550
Processor(s) 520
Network Interface 560

Fig. 5

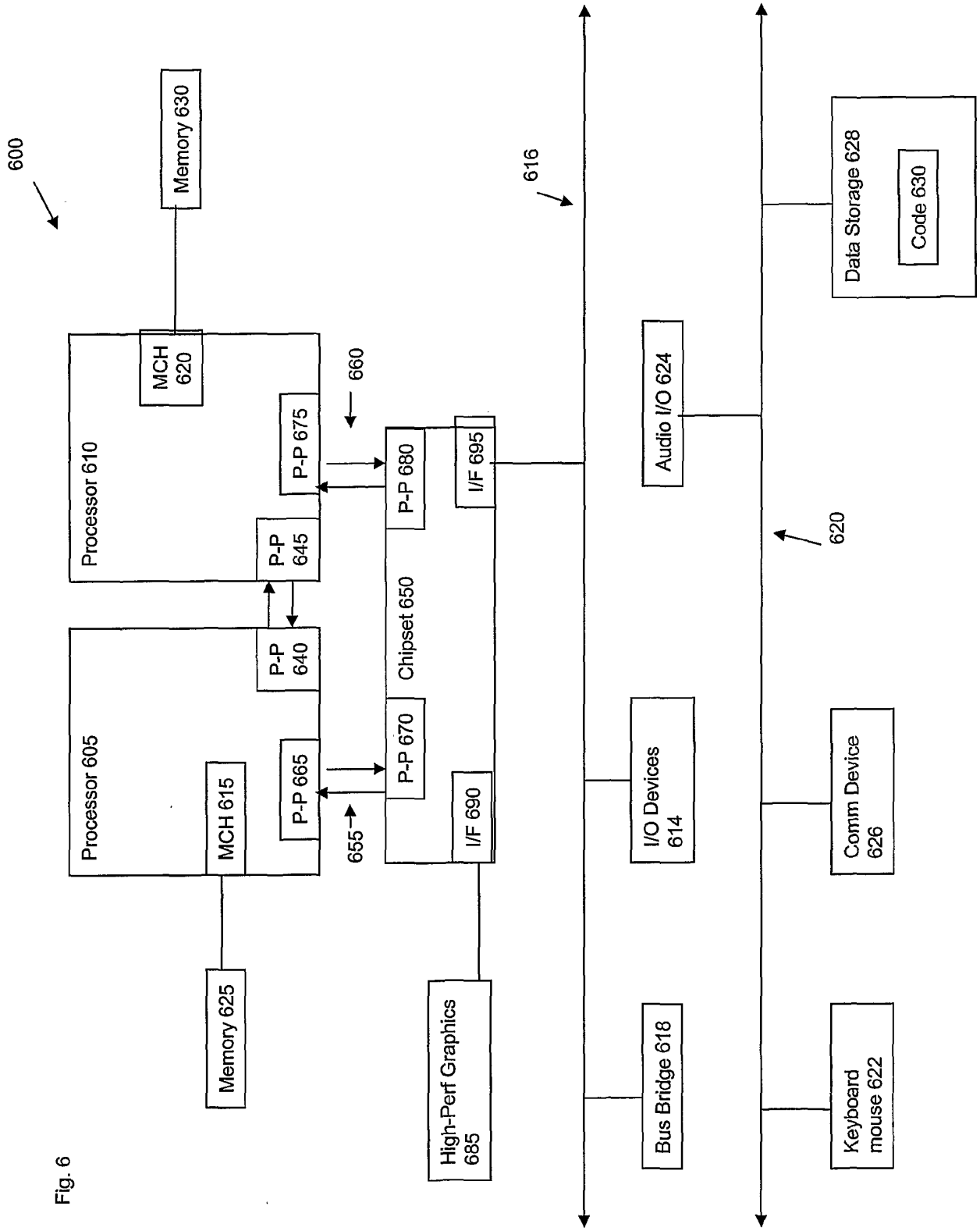
600
Memory 630
MCH 620
Processor 610
P-P 675
660
P-P 680
I/F 695
616
Audio I/O 624
Data Storage 628
Code 630
P-P 645
620
P-P 640
Chipset 650
Processor 605
MCH 615
P-P 665
P-P 670
655
I/F 690
I/O Devices 614
Comm Device 626
Memory 625
High-Perf Graphics 685
Bus Bridge 618
Keyboard mouse 622

Fig. 6