Apple Inc. v. Realtime Data, LLC d/b/a/ IXO
Cases IPR2016-01365, -01366

Table of Contents

“preloading the boot data into a cache memory”

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

(10) **Patent No.: US 7,181,608 B2**
(45) **Date of Patent: Feb. 20, 2007**

(54) **SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS**

(75) Inventors: **James J. Fallon**, Armonk, NY (US); **John Buck**, Oceanside, NY (US); **Paul F. Pickel**, Bethpage, NY (US); **Stephen J. McErlain**, New York, NY (US)

(73) Assignee: **Realtime Data LLC**, New York, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 223 days.

(21) Appl. No.: **09/776,267**

(22) Filed: **Feb. 2, 2001**

(65) **Prior Publication Data**

US 2002/0069354 A1 Jun. 6, 2002

Related U.S. Application Data

(60) Provisional application No. 60/180,114, filed on Feb. 3, 2000.

(51) **Int. Cl.**
G06F 9/24 (2006.01)
G06F 9/00 (2006.01)
G06F 13/00 (2006.01)

(52) **U.S. Cl.** **713/2**; 713/1; 711/113

(58) **Field of Classification Search** 713/2, 713/1, 100; 711/170, 118, 113
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,127,518 A 11/1978 Coy et al.
4,302,775 A 11/1981 Widergren et al.
4,394,774 A 7/1983 Widergren et al.
4,574,351 A 3/1986 Dang et al.

(Continued)

FOREIGN PATENT DOCUMENTS

DE 4127518 A1 2/1992

(Continued)

OTHER PUBLICATIONS

IBM, Fast Dos Soft Boot, Feb. 1, 1994, vol. 37, Issue 2B, pp 185-186.*

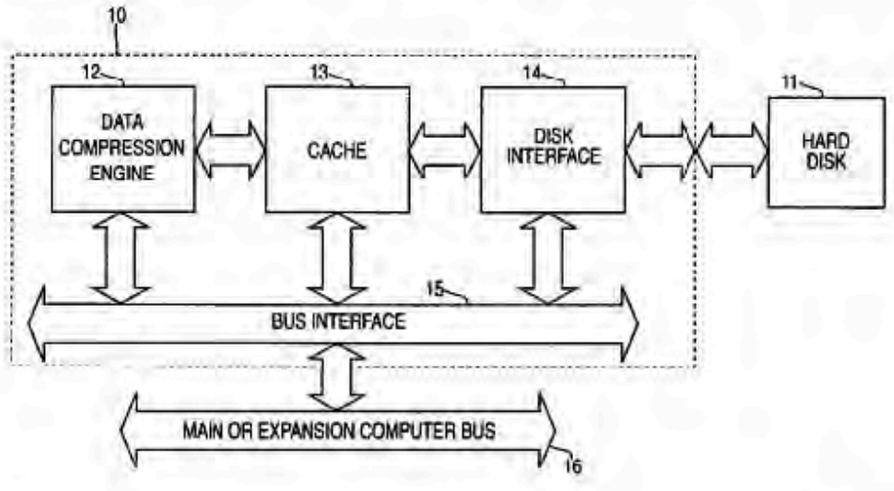
(Continued)

Primary Examiner—Thomas Lee
Assistant Examiner—Suresh K Suryawanshi
(74) *Attorney, Agent, or Firm*—Fish & Neave IP Group of Ropes & Gray LLP

(57) **ABSTRACT**

Systems and methods are provided for accelerated loading of operating system and application programs upon system boot or application launch. In one aspect, a method for providing accelerated loading of an operating system includes maintaining a list of boot data used for booting a computer system, preloading the boot data upon initialization of the computer system, and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. The boot data is retrieved from a boot device and stored in a cache memory device. The boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system.

31 Claims, 13 Drawing Sheets



APPLE 1001

set of decoders, or a sequential set of decoders corresponding to the extracted compression type descriptor. The decoders D1 . . . Dn may include those lossless encoding techniques currently well known within the art, including: run length, Huffman, Lempel-Ziv Dictionary Compression, arithmetic coding, data compaction, and data null suppression. Decoding techniques are selected based upon their ability to effectively decode the various different types of encoded input data generated by the data compression systems described above or originating from any other desired source.

As with the data compression systems discussed in U.S. Pat. No. 6,195,024, the decoder module **165** may include multiple decoders of the same type applied in parallel so as to reduce the data decoding time. An output data buffer or cache **170** may be included for buffering the decoded data block output from the decoder module **165**. The output buffer **70** then provides data to the output data stream. It is to be appreciated by those skilled in the art that the data compression system **180** may also include an input data counter and output data counter operatively coupled to the input and output, respectively, of the decoder module **165**. In this manner, the compressed and corresponding decompressed data block may be counted to ensure that sufficient decompression is obtained for the input data block.

Again, it is to be understood that the embodiment of the data decompression system **180** of FIG. **10** is exemplary of a preferred decompression system and method which may be implemented in the present invention, and that other data decompression systems and methods known to those skilled in the art may be employed for providing accelerated data retrieval in accordance with the teachings herein.

Although illustrative embodiments have been described herein with reference to the accompanying drawings, it is to be understood that the present invention is not limited to those precise embodiments, and that various other changes and modifications may be affected therein by one skilled in the art without departing from the scope or spirit of the invention. All such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.

What is claimed is:

1. A method for providing accelerated loading of an operating system, comprising the steps of:

maintaining a list of boot data used for booting a computer system;

initializing a central processing unit of the computer system;

preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and

servicing requests for boot data from the computer system using the preloaded boot data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache.

2. The method of claim **1**, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.

3. The method of claim **1**, wherein the preloading is performed by a data storage controller connected to the boot device.

4. The method of claim **1**, further comprising updating the list of boot data.

5. The method of claim **4**, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.

6. The method of claim **4**, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.

7. A system for providing accelerated loading of an operating system of a host system comprising:

a digital signal processor (DSP) or controller;

a cache memory device; and

a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system, for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system, and for decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system.

8. The system of claim **7**, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.

9. The method of claim **1**, further comprising:

maintaining a list of application data associated with an application program;

preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device; and

servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.

10. The method of claim **1**, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device.

11. The method of claim **1**, wherein the decompressing is provided by a data compression engine.

12. The method of claim **1**, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine.

13. The method of claim **1**, wherein the compressed boot data is accessed via direct memory access.

14. The method of claim **1**, wherein Huffman encoding is utilized to provide the compressed boot data.

15. The method of claim **1**, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data.

[redacted]

by the appended claims.

What is claimed is:

1. A method for providing accelerated loading of an operating system, comprising the steps of:

maintaining a list of boot data used for booting a computer system;

initializing a central processing unit of the computer system;

preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and

servicing requests for boot data from the computer system using the preloaded boot data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises

[redacted]

pression
y other
in U.S.
include
lel so as
uffer or
led data
output
ım. It is
he data
out data
d to the
165. In
decom-
ufficient
t of the

7. A system for providing accelerated loading of an operating system of a host system comprising:

a digital signal processor (DSP) or controller;

a cache memory device; and

a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system, for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system, and for decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system.

8. The system of claim **7**, wherein the logic code in the

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

(10) **Patent No.: US 8,090,936 B2**
(45) **Date of Patent: *Jan. 3, 2012**

(54) **SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS**

(75) Inventors: **James J. Fallon**, Armonk, NY (US); **John Buck**, Oceanside, NY (US); **Paul F. Pickel**, Bethpage, NY (US); **Stephen J. McErlain**, New York, NY (US)

(73) Assignee: **Realtime Data, LLC**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 286 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/551,204**

(22) Filed: **Oct. 19, 2006**

(65) **Prior Publication Data**

US 2007/0083746 A1 Apr. 12, 2007

Related U.S. Application Data

(63) Continuation of application No. 09/776,267, filed on Feb. 2, 2001, now Pat. No. 7,181,608.

(60) Provisional application No. 60/180,114, filed on Feb. 3, 2000.

(51) **Int. Cl.**

G06F 9/00	(2006.01)
G06F 9/24	(2006.01)
G06F 13/28	(2006.01)

(52) **U.S. Cl.** **713/2**; 713/1; 711/113

(58) **Field of Classification Search** 713/2

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,394,352	A	7/1968	Wernikoff et al.
3,490,690	A	1/1970	Apple et al.
4,021,782	A	5/1977	Hoerning
4,032,893	A	6/1977	Moran
4,054,951	A	10/1977	Jackson et al.
4,127,518	A	11/1978	Coy et al.
4,302,775	A	11/1981	Widergren et al.

(Continued)

FOREIGN PATENT DOCUMENTS

DE	4127518	2/1992

(Continued)

OTHER PUBLICATIONS

Millman, Howard, "Image and video compression", Computerworld, vol. 33, Issue No. 3, Jan. 18, 1999, pp. 78.

(Continued)

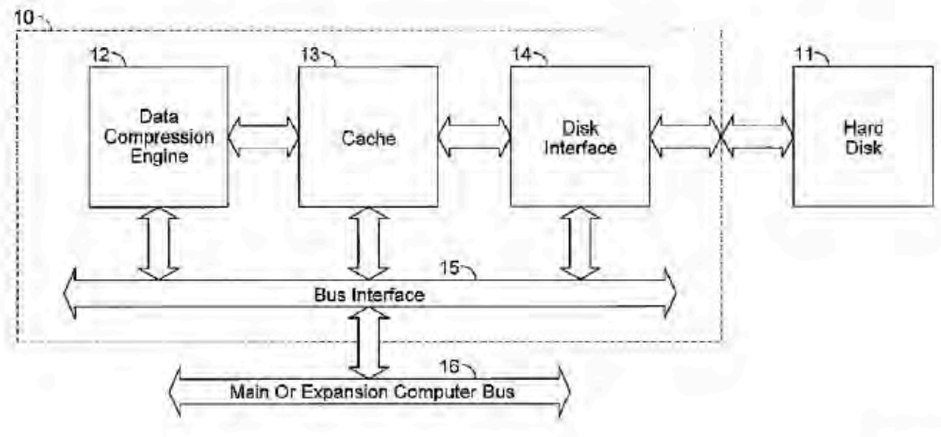
Primary Examiner — Suresh K Suryawanshi

(74) *Attorney, Agent, or Firm* — Sterne, Kessler, Goldstein & Fox P.L.L.C.

(57) **ABSTRACT**

Systems and methods are disclosed for providing accelerated loading of operating system and application programs. In one aspect, a method for providing accelerated loading of an operating system comprises the steps of: maintaining a list of boot data; preloading the boot data upon initialization of the computer system; and servicing requests for boot data from the computer system using the preloaded boot data. In a preferred embodiment, the boot data is retrieved from a boot device and stored in a cache memory device. In another aspect, a method for accelerated loading of an operating system comprises updating the list of boot data during the boot process, wherein updating comprises adding to the list any boot data requested by the computer system not previously stored in the list and/or removing from the list any boot data previously stored in the list and not requested by the computer system.

24 Claims, 13 Drawing Sheets



APPLE 1001

such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.

What is claimed is:

1. A method comprising:

maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;

initializing a central processing unit of said computer system;

preloading said at least a portion of said boot data in compressed form from said boot device to a memory;

accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and

utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.

2. The method of claim **1**, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.

3. The method of claim **1**, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.

4. The method of claim **1**, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.

5. The method of claim **1**, wherein said preloading is performed by a data storage controller connected to said boot device.

6. The method of claim **1**, further comprising updating the list of boot data.

7. The method of claim **1**, wherein Huffman encoding is utilized to provide said at least a portion of said boot data in said compressed form.

8. The method of claim **1**, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.

9. The method of claim **1**, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.

10. The method of claim **1**, wherein a plurality of encoders in a parallel configuration are utilized to provide said at least a portion of said data in compressed form.

11. A system comprising:

a processor;

a memory; and

a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and

a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.

12. The system of claim **11**, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.

13. The system of claim **11**, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.

14. The system of claim **11**, wherein Huffman encoding is utilized to provide said at least a portion of said boot data in compressed form.

15. The system of claim **11**, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.

16. The system of claim **11**, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.

17. The system of claim **11**, wherein a plurality of encoders in a parallel configuration are utilized to provide said at least a portion of said boot data in compressed form.

18. A method of preloading an operating system for booting a computer system comprising:

storing substantially all of the operating system in compressed form on a boot device;

preloading a first portion of the substantially all of the operating system from said boot device to a memory;

accessing and decompressing the first portion from the memory using a data compression engine;

utilizing the decompressed first portion to partially boot said computer system;

responsive to a request, locating a second portion of the substantially all of the operating system using a boot data list and preloading the second portion from the boot device to the memory;

accessing and decompressing the second portion from the memory using the data compression engine; and

utilizing the decompressed second portion to further partially boot said computer system.

19. The method of claim **18**, wherein the preloading is performed by a data storage controller connected to the boot device.

20. The method of claim **18**, further comprising updating the boot data list.

21. The method of claim **18**, wherein Huffman encoding is utilized to obtain the substantially all of the operating system in compressed form.

22. The method of claim **18**, wherein Lempel-Ziv encoding is utilized to obtain the substantially all of the operating system in compressed form.

23. The method of claim **18**, wherein a plurality of encoders are utilized to obtain the substantially all of the operating system in compressed form.

24. The method of claim **18**, wherein a plurality of encoders in a parallel configuration are utilized to obtain the substantially all of the operating system in compressed form.

* * * * *

rogram code
aid computer

ompressed at
rogram code
perating sys-

oading is per-
to said boot

updating the

n encoding is
l boot data in

v encoding is
l boot data in

y of encoders
f compressed

in a parallel configuration are utilized to provide said at least a portion of said boot data in compressed form.

18. A method of preloading an operating system for booting a computer system comprising:

storing substantially all of the operating system in compressed form on a boot device;

preloading a first portion of the substantially all of the operating system from said boot device to a memory;

accessing and decompressing the first portion from the memory using a data compression engine;

utilizing the decompressed first portion to partially boot said computer system;

responsive to a request, locating a second portion of the substantially all of the operating system using a boot data list and preloading the second portion from the boot device to the memory;

accessing and decompressing the second portion from the memory using the data compression engine; and

utilizing the decompressed second portion to further partially boot said computer system.

19. The method of claim **18**, wherein the preloading is performed by a data storage controller connected to the boot

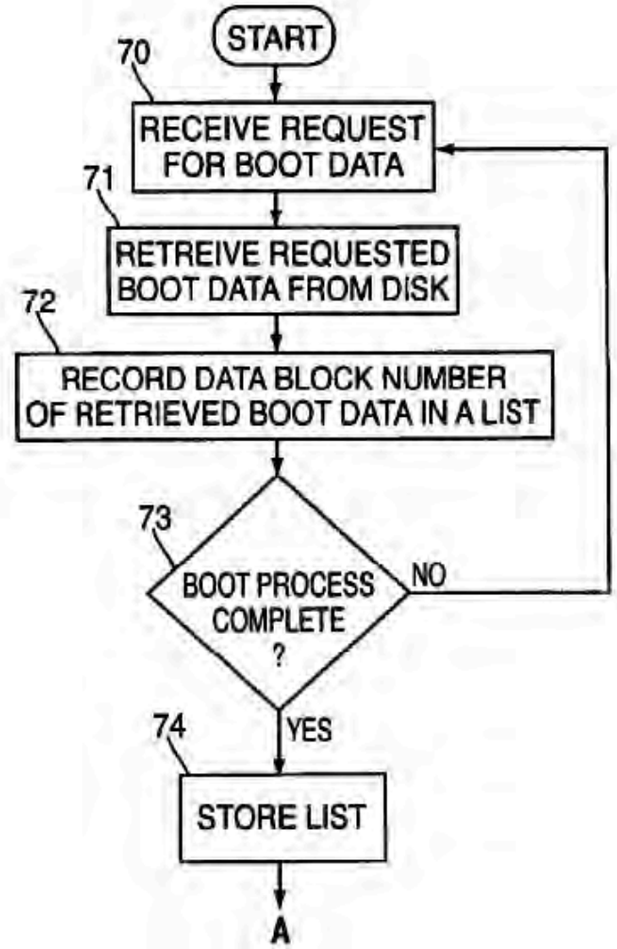

START
70
RECEIVE REQUEST FOR BOOT DATA
71
RETREIVE REQUESTED BOOT DATA FROM DISK
72
RECORD DATA BLOCK NUMBER OF RETRIEVED BOOT DATA IN A LIST
73
BOOT PROCESS COMPLETE ?
NO
YES
74
STORE LIST
A

FIG. 7a

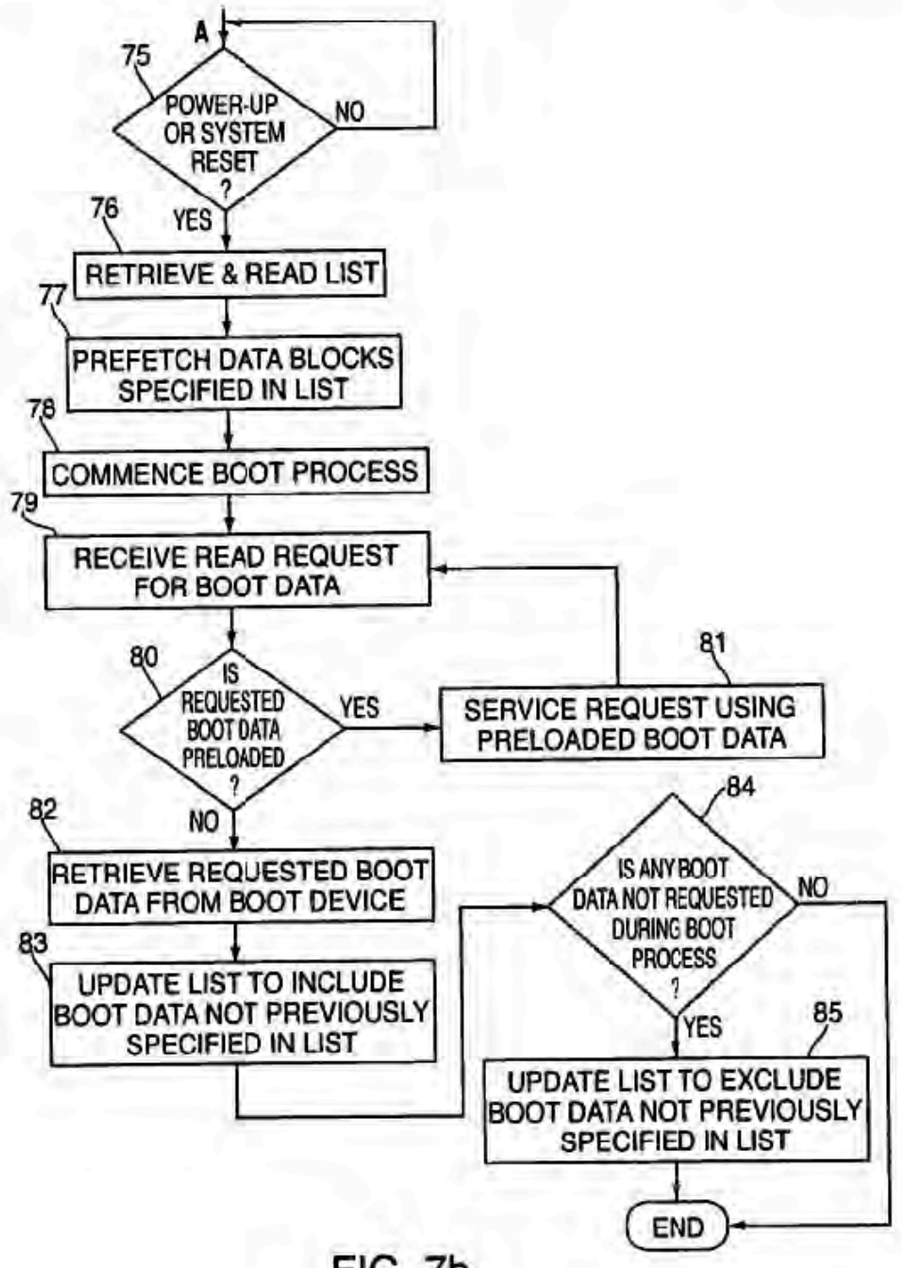
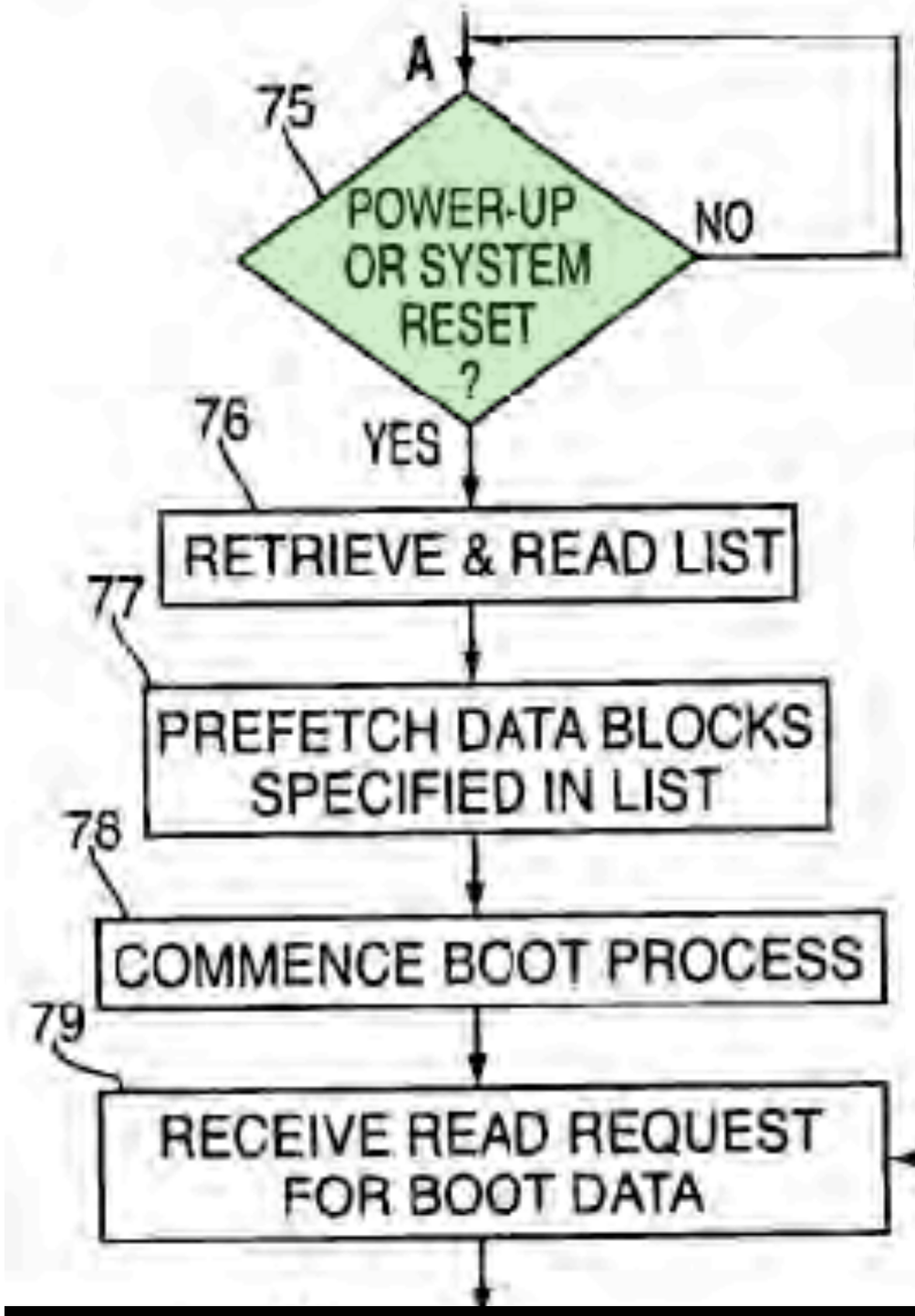
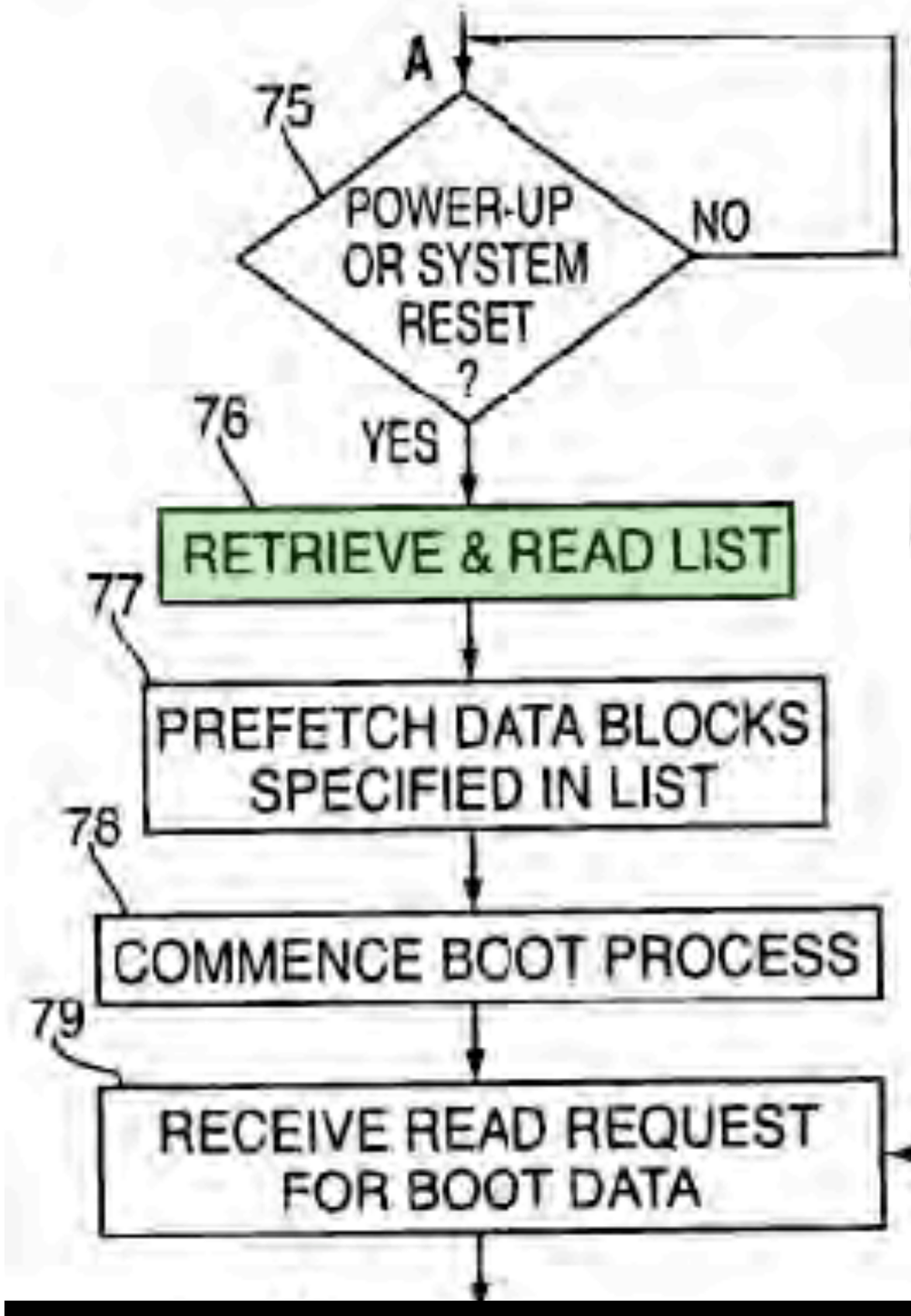

A
75
POWER-UP OR SYSTEM RESET ?
NO
YES
76
RETRIEVE & READ LIST
77
PREFETCH DATA BLOCKS SPECIFIED IN LIST
78
COMMENCE BOOT PROCESS
79
RECEIVE READ REQUEST FOR BOOT DATA
80
IS REQUESTED BOOT DATA PRELOADED ?
YES
81
SERVICE REQUEST USING PRELOADED BOOT DATA
NO
82
RETRIEVE REQUESTED BOOT DATA FROM BOOT DEVICE
83
UPDATE LIST TO INCLUDE BOOT DATA NOT PREVIOUSLY SPECIFIED IN LIST
84
IS ANY BOOT DATA NOT REQUESTED DURING BOOT PROCESS ?
NO
YES
85
UPDATE LIST TO EXCLUDE BOOT DATA NOT PREVIOUSLY SPECIFIED IN LIST
END

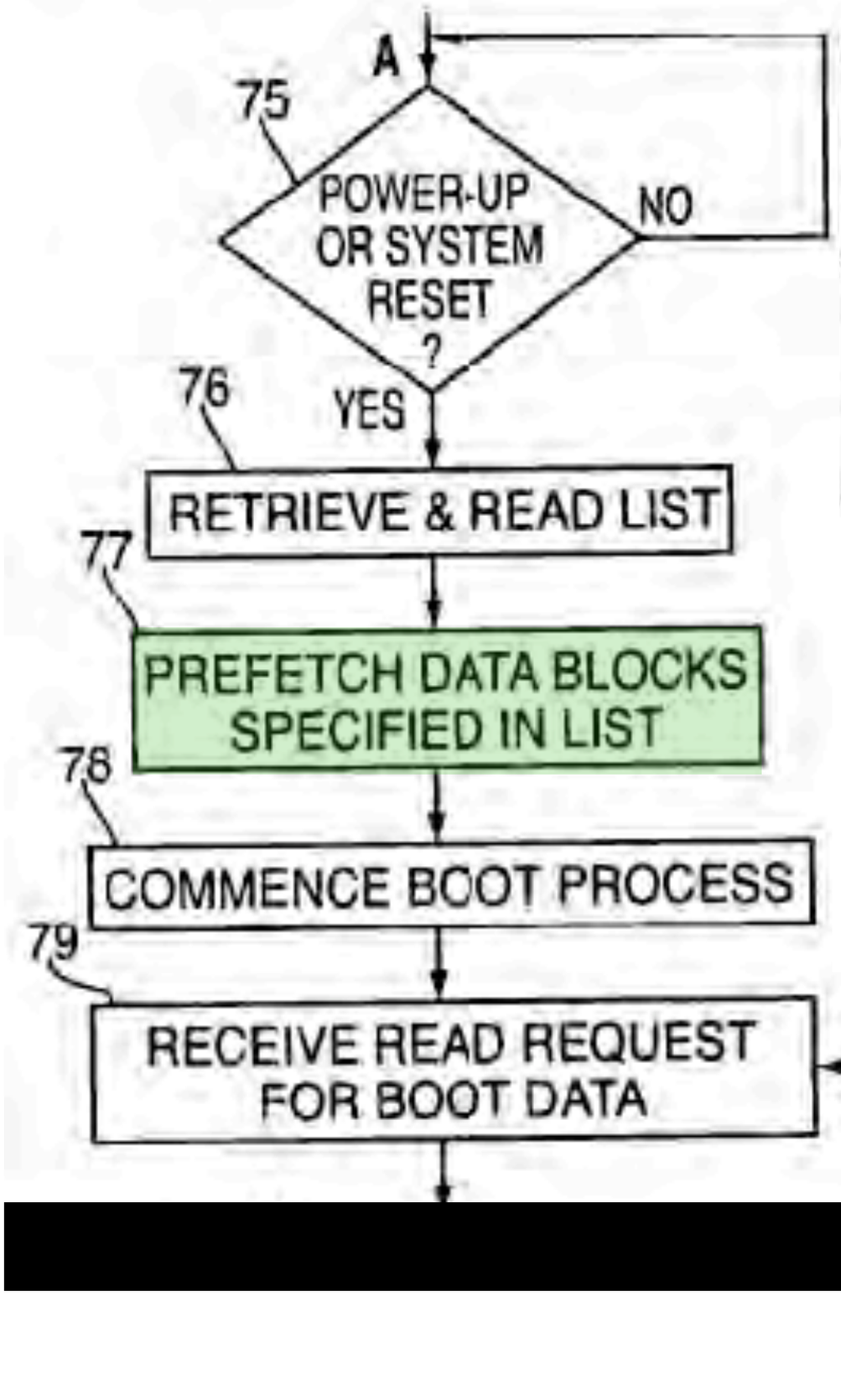
FIG. 7b



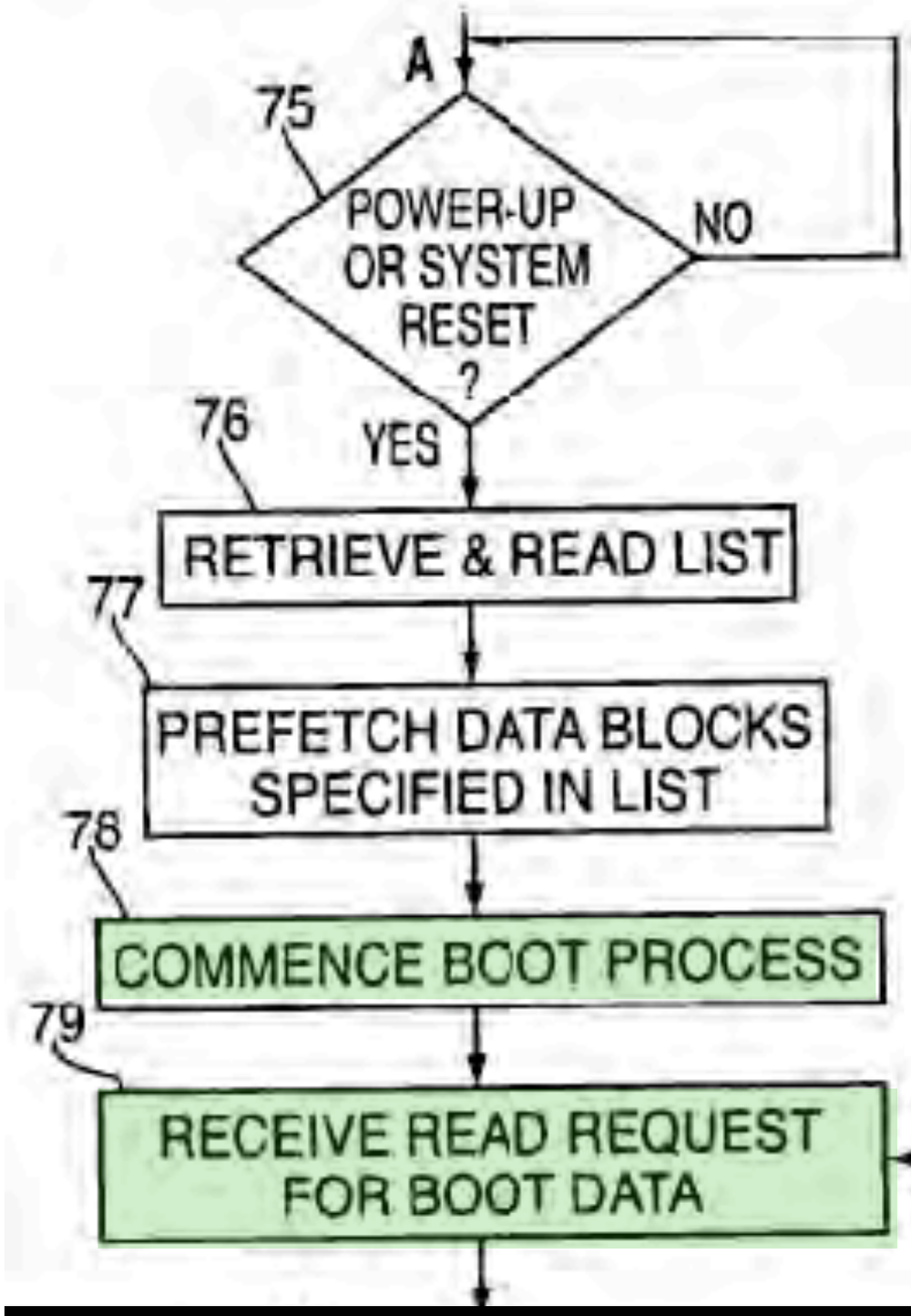
Then, upon each subsequent power-on/reset (affirmative result in step **75**), the data storage controller would retrieve and read the stored list (step **76**) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step **77**). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).



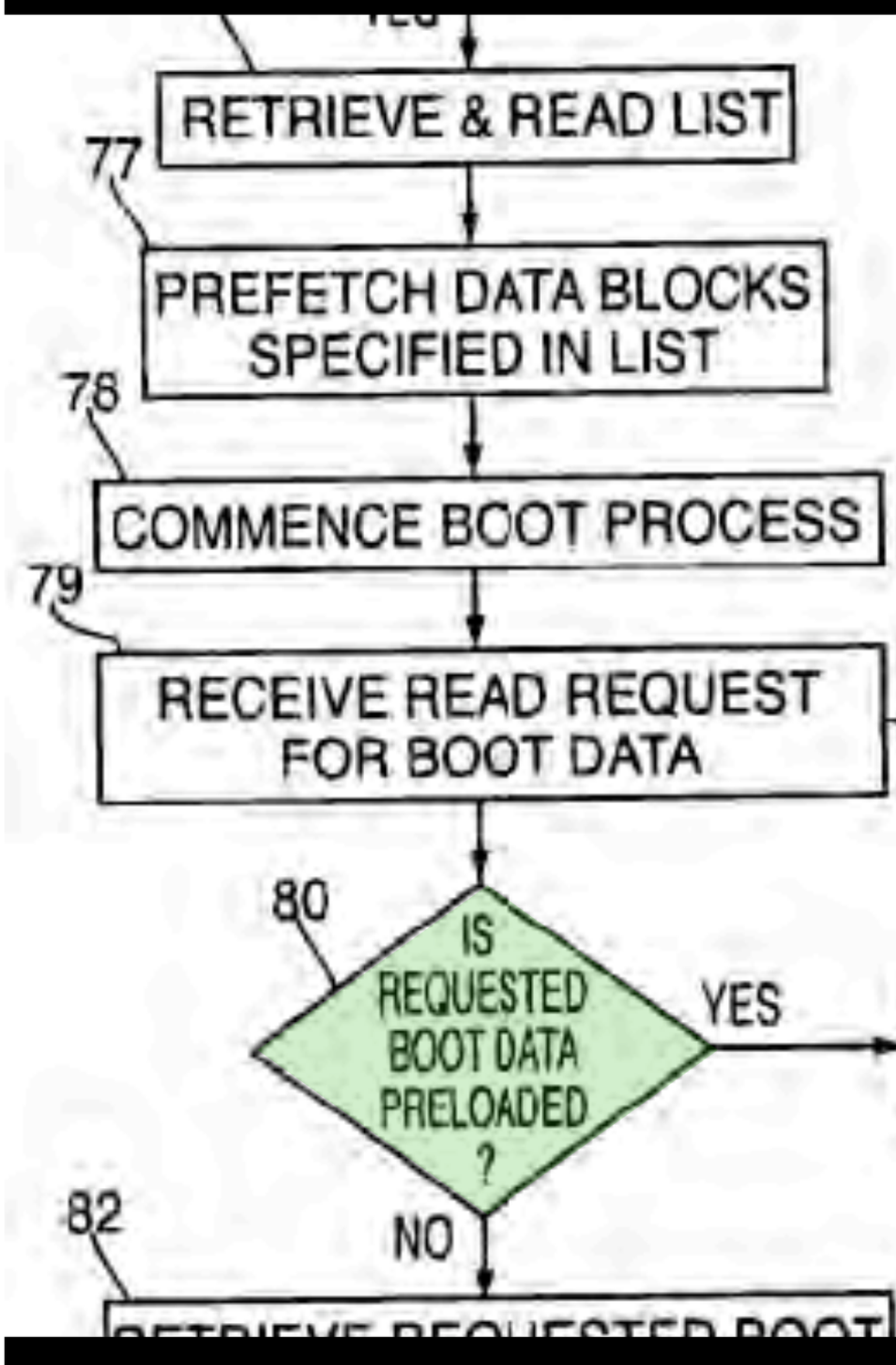
Then, upon each subsequent power-on/reset (affirmative result in step **75**), the data storage controller would retrieve and read the stored list (step **76**) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step **77**). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).



Then, upon each subsequent power-on/reset (affirmative result in step **75**), the data storage controller would retrieve and read the stored list (step **76**) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step **77**). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).



When the boot process begins (step **78**) (i.e., the storage controller is initialized and the system bus reset is deasserted), the data storage controller will receive requests for boot data (step **79**). If the host computer issues a request for boot data that is pre-loaded in the local memory of the data storage controller (affirmative result in step **80**), the request is immediately serviced using the preloaded boot data (step **81**). If the host computer issues a request for boot data that is not preloaded in the local memory of the data storage controller (negative determination in step **80**), the controller



When the boot process begins (step **78**) (i.e., the storage controller is initialized and the system bus reset is deasserted), the data storage controller will receive requests for boot data (step **79**). If the host computer issues a request for boot data that is pre-loaded in the local memory of the data storage controller (affirmative result in step **80**), the request is immediately serviced using the preloaded boot data (step **81**). If the host computer issues a request for boot data that is not preloaded in the local memory of the data storage controller (negative determination in step **80**), the controller

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE INC.,
Petitioner,

v.

REALTIME DATA LLC,
Patent Owner.

Case IPR2016-01365
Patent 7,181,608

PETITIONER'S REPLY TO PATENT OWNER'S RESPONSE

anticipation" of anything, let alone "immediate or near-in-time use." Because the '608 Patent does not even use these terms to describe preloading, it would be improper to import them into the term preloading under BRI.

2) **Realtime's Preloading Construction is Inconsistent with Examples of Preloading in the '608 Patent**

Contrary to BRI, Realtime's construction improperly excludes examples of preloading in the '608 Patent. POR, 3-7, 12-16, 16-26. Specifically, the claims require preloading boot data prior to completion of initialization of the central processing unit. The '608 Patent describes several examples where preloading occurs prior to completion of initialization because the preloading occurs prior to system reset or boot. In these examples, the '608 Patent simply states that preloading occurs prior to system reset or boot; it does not qualify how far in advance of reset or boot the preloading occurs. Thus, these examples align with the claim language, but are not limited to preloading "in anticipation of immediate or near-in-time use," as advocated by Realtime.

For example, the '608 Patent explains that "***prior to host system reset***, the data storage controller can proceed to ***pre-load*** the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory." '608, 21:45-65. In this example, "[s]ince the same portions of the operating system must be loaded ***upon each boot process***, it is advantageous for the boot device

occurs prior to completion of initialization because the preloading occurs prior to system reset or boot. In these examples, the '608 Patent simply states that preloading occurs prior to system reset or boot; it does not qualify how far in advance of reset or boot the preloading occurs. Thus, these examples align with the claim language, but are not limited to preloading "in anticipation of immediate or near-in-time use," as advocated by Realtime.

For example, the '608 Patent explains that "***prior to host system reset***, the data storage controller can proceed to ***pre-load*** the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory." '608, 21:45-65. In this example, "[s]ince the same portions of the operating system must be loaded ***upon each boot process***, it is advantageous for the boot device

If only the PCI Bus and DSP require SDRAM:

PCI Bus Interface	(A+B)/K
DSP Accesses	(A+B)/K

If only the DSP and Disk require SDRAM:

DSP Accesses	2A/K
UltraDMA Disk Interface	2B/K

If only the PCI Bus and Disk require SDRAM:

PCI Bus Interface	2A/K
UltraDMA Disk Interface	2B/K

It should be noted that the resultant ratios may all be scaled by a constant in order to most effectively utilize the bandwidths of the internal busses and external interfaces. In addition each ratio can be scale by an adjustment factor based upon the time required to complete individual cycles. For example if PCI Bus interface takes 20% longer than all other cycles, the PCI time slice should be adjusted longer accordingly.

V. Instant Boot Device for Operating System, Application Program and Loading

Typically, with conventional boot device controllers, after reset, the boot device controller will wait for a command over the computer bus (such as PCI). Since the boot device controller will typically be reset prior to bus reset and before the computer bus starts sending commands, this wait period is unproductive time. The initial bus commands inevitably instruct the boot device controller to retrieve data from the boot device (such as a disk) for the operating system. Since most boot devices are relatively slow compared to the speed of most computer busses, a long delay is seen by the computer user. This is evident in the time it takes for a typical computer to boot.

It is to be appreciated that a data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the computer system boot time. Upon host system power-up or reset, the data storage controller will perform a self-diagnostic and program the programmable logic device (as discussed above) prior to completion of the host system reset (e.g., PCI bus reset) so that the logic device can accept PCI Bus commands after system reset. Further, prior to host system reset, the data storage controller can proceed to pre-load the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory. The data storage controller preloads the needed sectors of data in the order in which they will be needed. Since the same portions of the operating system must be loaded upon each boot process, it is advantageous for the boot device controller to preload such portions and not wait until it is commanded to load the operating system. Preferably, the data storage controller employs a dedicated IO channel of the DSP (with or without data compression) to pre-load computer operating systems and applications.

Once the data is preloaded, when the computer system bus issues its first read commands to the data storage controller seeking operating system data, the data will already be available in the cache memory of the data storage controller. The data storage controller will then be able to instantly start transmitting the data to the system bus. Before transmission to the bus, if the data was stored in compressed format on the boot device, the data will be decompressed. The process of preloading required (compressed) portions of the operating system significantly reduces the computer boot process time.

In addition to preloading operating system data, the data storage controller could also preload other data that the user would likely want to use at startup. An example of this would be a frequently used application such as a word processor and any number of document files.

There are several techniques that may be employed in accordance with the present invention that would allow the data storage controller to know what data to preload from the boot device. One technique utilizes a custom utility program that would allow the user to specify what applications/data should be preloaded.

Another technique (illustrated by the flow diagram of FIGS. **7***a* and **7***b*) that may be employed comprises an automatic process that requires no input from the user. With this technique, the data storage controller maintain a list comprising the data associated with the first series of data requests received by the data storage controller by the host system after a power-on/reset. In particular, referring to FIG. **7***a*, during the computer boot process, the data storage controller will receive requests for the boot data (step **70**). In response, the data storage controller will retrieve the requested boot data from the boot device (e.g., hard disk) in the local cache memory (step **71**). For each requested data block, the data storage controller will record the requested data block number in a list (step **72**). The data storage controller will record the data block number of each data block requested by the host computer during the boot process (repeat steps **70**–**72**). When the boot process is complete (affirmative determination in step **73**), the data storage controller will store the data list on the boot device (or other storage device) (step **74**).

Then, upon each subsequent power-on/reset (affirmative result in step **75**), the data storage controller would retrieve and read the stored list (step **76**) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step **77**). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).

When the boot process begins (step **78**) (i.e., the storage controller is initialized and the system bus reset is deasserted), the data storage controller will receive requests for boot data (step **79**). If the host computer issues a request for boot data that is pre-loaded in the local memory of the data storage controller (affirmative result in step **80**), the request is immediately serviced using the preloaded boot data (step **81**). If the host computer issues a request for boot data that is not preloaded in the local memory of the data storage controller (negative determination in step **80**), the controller will retrieve the requested data from the boot device, store the data in the local memory, and then deliver the requested boot data to the computer bus (step **82**). In addition, the data storage controller would update the boot data list by recording any changes in the actual data requests as compared to the expected data requests already stored in the list (step **83**). Then, upon the next boot sequence, the boot device con-

It is to be appreciated that a data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the computer system boot time. Upon host system power-up or reset, the data storage controller will perform a self-diagnostic and program the programmable logic device (as discussed above) prior to completion of the host system reset (e.g., PCI bus reset) so that the logic device can accept PCI Bus commands after system reset. Further, prior to host system reset, the data storage controller can proceed to pre-load the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory. The data storage controller preloads the needed sectors of data in the order in which they will be needed. Since the same portions of the operating system must be loaded upon each boot process, it is advantageous for the boot device controller to preload such portions and not wait until it is com-

[redacted] storage controller employs a dedicated IO channel of the DSP (with or without data compression) to pre-load computer operating systems and applications.

Once the data is preloaded, when the computer system bus issues its first read commands to the data storage controller

seeking operating system data, the data will already be available in the cache memory of the data storage controller. The data storage controller will then be able to instantly start transmitting the data to the system bus. Before transmission to the bus, if the data was stored in compressed format on the boot device, the data will be decompressed. The process of preloading required (compressed) portions of the operating system significantly reduces the computer boot process time.

In addition to preloading operating system data, the data

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE INC.,
Petitioner,

v.

REALTIME DATA LLC,
Patent Owner.

Case IPR2016-01365
Patent 7,181,608

PETITIONER'S REPLY TO PATENT OWNER'S RESPONSE

controller to ***preload*** such portions and not wait until it is commanded to load the operating system." *Id.*

This example makes explicit that, contrary to Realtime's arguments, preloading occurs "prior to host system reset" (i.e., prior to boot[1]). Indeed, the '608 Patent deems such an approach "advantageous." '608, 21:45-65.

The '608 Patent provides further confirmation that preloading may occur prior to a subsequent boot process through discussion of two alternative examples. In a first example, preloading is completed prior to boot and, in a second, alternative example, booting and preloading are performed simultaneously. '608, 22:20-50. Specifically, the '608 Patent explains that:

> It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the ***preloading*** process may be completed ***prior to commencement of the boot process***, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).

'608, 22:40-50.

This passage demonstrates that, in a system equipped with appropriate resources (e.g., non-volatile cache memory), preloading in the '608 Patent is

[1] The '608 Patent explains, at 11:26-30, that "the boot process begins when the CPU of the host system is released from external reset …."

to a subsequent boot process through discussion of two alternative examples. In a first example, preloading is completed prior to boot and, in a second, alternative example, booting and preloading are performed simultaneously. '608, 22:20-50. Specifically, the '608 Patent explains that:

> It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the ***preloading*** process may be completed ***prior to commencement of the boot process***, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).

'608, 22:40-50.

This passage demonstrates that, in a system equipped with appropriate resources (e.g., non-volatile cache memory), preloading in the '608 Patent is

If only the PCI Bus and DSP require SDRAM:

PCI Bus Interface	(A+B)/K
DSP Accesses	(A+B)/K

If only the DSP and Disk require SDRAM:

DSP Accesses	2A/K
UltraDMA Disk Interface	2B/K

If only the PCI Bus and Disk require SDRAM:

PCI Bus Interface	2A/K
UltraDMA Disk Interface	2B/K

It should be noted that the resultant ratios may all be scaled by a constant in order to most effectively utilize the bandwidths of the internal busses and external interfaces. In addition each ratio can be scale by an adjustment factor based upon the time required to complete individual cycles. For example if PCI Bus interface takes 20% longer than all other cycles, the PCI time slice should be adjusted longer accordingly.

V. Instant Boot Device for Operating System, Application Program and Loading

Typically, with conventional boot device controllers, after reset, the boot device controller will wait for a command over the computer bus (such as PCI). Since the boot device controller will typically be reset prior to bus reset and before the computer bus starts sending commands, this wait period is unproductive time. The initial bus commands inevitably instruct the boot device controller to retrieve data from the boot device (such as a disk) for the operating system. Since most boot devices are relatively slow compared to the speed of most computer busses, a long delay is seen by the computer user. This is evident in the time it takes for a typical computer to boot.

It is to be appreciated that a data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the computer system boot time. Upon host system power-up or reset, the data storage controller will perform a self-diagnostic and program the programmable logic device (as discussed above) prior to completion of the host system reset (e.g., PCI bus reset) so that the logic device can accept PCI Bus commands after system reset. Further, prior to host system reset, the data storage controller can proceed to pre-load the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory. The data storage controller preloads the needed sectors of data in the order in which they will be needed. Since the same portions of the operating system must be loaded upon each boot process, it is advantageous for the boot device controller to preload such portions and not wait until it is commanded to load the operating system. Preferably, the data storage controller employs a dedicated IO channel of the DSP (with or without data compression) to pre-load computer operating systems and applications.

Once the data is preloaded, when the computer system bus issues its first read commands to the data storage controller seeking operating system data, the data will already be available in the cache memory of the data storage controller. The data storage controller will then be able to instantly start transmitting the data to the system bus. Before transmission to the bus, if the data was stored in compressed format on the boot device, the data will be decompressed. The process of preloading required (compressed) portions of the operating system significantly reduces the computer boot process time.

In addition to preloading operating system data, the data storage controller could also preload other data that the user would likely want to use at startup. An example of this would be a frequently used application such as a word processor and any number of document files.

There are several techniques that may be employed in accordance with the present invention that would allow the data storage controller to know what data to preload from the boot device. One technique utilizes a custom utility program that would allow the user to specify what applications/data should be preloaded.

Another technique (illustrated by the flow diagram of FIGS. **7***a* and **7***b*) that may be employed comprises an automatic process that requires no input from the user. With this technique, the data storage controller maintain a list comprising the data associated with the first series of data requests received by the data storage controller by the host system after a power-on/reset. In particular, referring to FIG. **7***a*, during the computer boot process, the data storage controller will receive requests for the boot data (step **70**). In response, the data storage controller will retrieve the requested boot data from the boot device (e.g., hard disk) in the local cache memory (step **71**). For each requested data block, the data storage controller will record the requested data block number in a list (step **72**). The data storage controller will record the data block number of each data block requested by the host computer during the boot process (repeat steps **70**–**72**). When the boot process is complete (affirmative determination in step **73**), the data storage controller will store the data list on the boot device (or other storage device) (step **74**).

Then, upon each subsequent power-on/reset (affirmative result in step **75**), the data storage controller would retrieve and read the stored list (step **76**) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step **77**). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).

When the boot process begins (step **78**) (i.e., the storage controller is initialized and the system bus reset is deasserted), the data storage controller will receive requests for boot data (step **79**). If the host computer issues a request for boot data that is pre-loaded in the local memory of the data storage controller (affirmative result in step **80**), the request is immediately serviced using the preloaded boot data (step **81**). If the host computer issues a request for boot data that is not preloaded in the local memory of the data storage controller (negative determination in step **80**), the controller will retrieve the requested data from the boot device, store the data in the local memory, and then deliver the requested boot data to the computer bus (step **82**). In addition, the data storage controller would update the boot data list by recording any changes in the actual data requests as compared to the expected data requests already stored in the list (step **83**). Then, upon the next boot sequence, the boot device con-

(or other storage device) (step **74**).

Then, upon each subsequent power-on/reset (affirmative result in step **75**), the data storage controller would retrieve and read the stored list (step **76**) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step **77**). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).

When the boot process begins (step **78**) (i.e., the storage controller is initialized and the system bus reset is deasserted), the data storage controller will receive requests for boot data (step **79**). If the host computer issues a request for

Office Action Summary	Application No.	Applicant(s)	
	09/776,267	FALLON ET AL.	
	Examiner	Art Unit	
	Suresh K. Suryawanshi	2115	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *02 May 2005*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,2,4-7,9,10,12,13,15 and 17* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,2,4-7,9,10,12,13,15 and 17* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *02 February 2001* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

U.S. Patent and Trademark Office
PTOL-326 (Rev. 1-04) **Office Action Summary** Part of Paper No./Mail Date 3

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1-2, 4-7, 9-10, 12-13 and 15 are rejected under 35 U.S.C. 103(a) as being unpatentable over Krocker et al (US Patent no 6,073,232) in view of Esfahani et al (US Patent no 6,434,695 B1).

6. As per claim 1, Krocker et al teach

maintaining a list of boot data used for booting a computer system [col. 2, lines 30-47; col. 5, lines 1-7; a prefetch table containing a listing of the disk locations and length of data records that were requested by the host computer in the immediately previous power-on/reset];

preloading the boot data upon initialization of the computer system [col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the RAM cache according to the prefetch table]; and

servicing requests for boot data from the computer system using the preloaded boot data [col. 2, lines 41-47; col. 3, lines 30-39; data is communicated from the cache to the host computer].

[redacted]

5. Claims 1-2, 4-7, 9-10, 12-13 and 15 are rejected under 35 U.S.C. 103(a) as being unpatentable over Krocker et al (US Patent no 6,073,232) in view of Esfahani et al (US Patent no 6,434,695 B1).

6. As per claim 1, Krocker et al teach

maintaining a list of boot data used for booting a computer system [col. 2, lines 30-47; col. 5, lines 1-7; a prefetch table containing a listing of the disk locations and length of data records that were requested by the host computer in the immediately previous power-on/reset];

preloading the boot data upon initialization of the computer system [col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the RAM cache according to the prefetch table]; and

[redacted]

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE, INC.,
Petitioner

v.

REALTIME DATA LLC,
Patent Owner

Case IPR2016-01365
Patent 7,181,608

EXPERT DECLARATION OF DR. GODMAR BACK IN SUPPORT OF THE PATENT OWNER'S RESPONSE

boot device.[24] And specification also explains that the data is preloaded from a boot device into the onboard cache memory of an exemplary data storage controller.[25]

51. Based on the disclosures of the '608 Patent, a POSITA would have understood that this movement of data from storage into memory is performed in anticipation of immediate or near-in-time use. Again, this is the manner in which the '608 Patent uses "preloading." For example, the '608 Patent discloses that the invention is designed to "provide[ ] accelerated loading of operating system and application programs upon system boot or application launch."[26] To this end, certain claims recite that boot or application data is preloaded to service requests for that data.[27] And in certain embodiments of the '608 Patent, the specification describes "preloading" as loading data because it will be needed, or is likely to be needed, for use by the system.[28]

52. In addition, the '608 Patent indicates that "preloading" is similar to "prefetching," which refers to the process of retrieving data before it is needed.

[24] Ex. 1001, '608 Patent at 3:60-61.

[25] Ex. 1001, '608 Patent at 21:50-54, 22:41-45.

[26] Ex. 1001, '608 Patent at 1:15-21; *see also* '608 Patent at Abs., 3:34-40.

[27] Ex. 1001, '608 Patent at claims 1, 7-9, 22, 27.

[28] Ex. 1001, '608 Patent at Abs., 21:56-62, 22:9-19, 23:13-38; Figs. 7b, 8b.

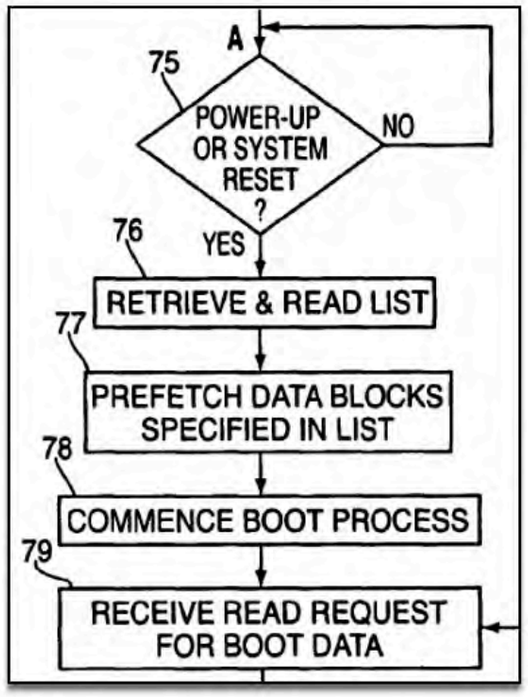
[redacted]

51. Based on the disclosures of the '608 Patent, a POSITA would have understood that this movement of data from storage into memory is performed in anticipation of immediate or near-in-time use. Again, this is the manner in which the '608 Patent uses "preloading." For example, the '608 Patent discloses that the invention is designed to "provide[ ] accelerated loading of operating system and application programs upon system boot or application launch."[26] To this end, certain claims recite that boot or application data is preloaded to service requests for that data.[27] And in certain embodiments of the '608 Patent, the specification describes "preloading" as loading data because it will be needed, or is likely to be needed, for use by the system.[28]

52. In addition, the '608 Patent indicates that "preloading" is similar to "prefetching," which refers to the process of retrieving data before it is needed.

[redacted]

Specifically, in one embodiment, boot data is preloaded into the onboard cache memory of an exemplary data storage controller in step 77 of Figure 7b.[29] As shown below, Figure 7b describes step 77 as "Prefetch Data Blocks Specified in List":[30]



53. Figure 7b therefore indicates that "preloading" has a meaning similar to "prefetching." And "prefetching" refers to the process of retrieving data before it

[29] Ex. 1001, '608 Patent at 22:40-45; *see also* '608 Patent at 22:20-39.

[30] Ex. 1001, '608 Patent at Fig. 7b.

53. Figure 7b therefore indicates that "preloading" has a meaning similar to "prefetching." And "prefetching" refers to the process of retrieving data before it is needed,[31] which further supports my interpretation of how a POSITA would have understood "preloading" at the time of invention.

54. This interpretation is also consistent with the file history of the '608 Patent. For example, during prosecution, the Patent Examiner initially rejected the

---

[31] Ex. 2011, The Design and Implementation of the 4.4BSD Operating System at 535 (defining "prefetching" as "[t]he retrieval of data before they are needed. Many machines prefetch machine instructions so they can overlap the time spent fetching instructions from memory with the time spent decoding instructions.");

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE INC.,
Petitioner,

v.

REALTIME DATA LLC,
Patent Owner.

Case IPR2016-01365
Patent 7,181,608

PETITIONER'S REPLY TO PATENT OWNER'S RESPONSE

completed before a boot process commences. Indeed, and as the passage makes clear by its alternative example, preloading during the boot process is only necessary in systems lacking appropriate resources.

Along these lines, the '608 Patent explains, and Dr. Back confirms[2], that the cache memory device in which preloaded boot data is stored "may comprise…***non-volatile memory***" in which data is persistently stored across boot cycles. '608, 6:60-63. Indeed, Dr. Back explained that, "if boot data were stored in the cache memory and if a form of non-volatile memory were used, then this data stored in the cache ***would be available after subsequent power on***." APPLE-1017 42, 76. Dr. Back added, "[t]hat's what non-volatile means." *Id.,* 76.

By describing examples of preloading prior to boot, including examples of preloading into non-volatile memory, the intrinsic record of the '608 Patent runs contrary to Realtime's attempt to narrow the term preloading. Indeed, Realtime's "preloading" construction is improper because it excludes examples within the '608 Patent specification and is inconsistent with the testimony of Realtime's own expert, Dr. Back.

---

[2] In his deposition on June 20, 2017, Dr. Back was asked whether the claimed cache memory "could be either volatile or non-volatile," and he concluded that yes, "it could be either way." APPLE-1017, 29-30.

[redacted]

in systems lacking appropriate resources.

Along these lines, the '608 Patent explains, and Dr. Back confirms[2], that the cache memory device in which preloaded boot data is stored "may comprise…***non-volatile memory***" in which data is persistently stored across boot cycles. '608, 6:60-63. Indeed, Dr. Back explained that, "if boot data were stored in the cache memory and if a form of non-volatile memory were used, then this data stored in the cache ***would be available after subsequent power on***." APPLE-1017 42, 76. Dr. Back added, "[t]hat's what non-volatile means." *Id.*, 76.

By describing examples of preloading prior to boot, including examples of preloading into non-volatile memory, the intrinsic record of the '608 Patent runs

[redacted]

[redacted]

contrary to Realtime's attempt to narrow the term preloading." Indeed, Realtime's "preloading" construction is improper because it excludes examples within the '608 Patent specification and is inconsistent with the testimony of Realtime's own expert, Dr. Back.

---

[2] In his deposition on June 20, 2017, Dr. Back was asked whether the claimed cache memory "could be either volatile or non-volatile," and he concluded that yes, "it could be either way." APPLE-1017, 29-30.

[redacted]

UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE, INC.,
Petitioners,
-vs- Case IPR2016-01365
Patent 7,181,608B2
REALTIME DATA LLC,
Patent Owner.

APPLE, INC.,
Petitioners,
-vs- Case IPR2016-01366
Patent 8,090,936B2
REALTIME DATA LLC,
Patent Owner.

VIDEOTAPED DEPOSITION OF DR. GODMAR BACK
11:18 a.m. to 2:28 p.m.
June 20, 2017
Blacksburg, Virginia

Job No. 2634898
REPORTED BY: Rhonda D. Tuck, RPR, CRR

device in the context of the specification.

Q. In your opinion, can the Data Controller 10 which is shown in Figure 1 and described elsewhere in the patent, can that function as described in the '936 Patent if Cache 13 is a nonvolatile memory?

MR. EDELL: Objection. Form.

THE WITNESS: I would say no. No. The -- well, let me qualify this.

The -- most of the, if not all, of the specification of both the '608 and the '936 Patents focus on how this particular controller would operate if the cache memory is a volatile memory. But if it weren't a volatile memory, then the cache controller could -- would operate in a way that actually does not benefit from the nonvolatility of the memory.

BY MS. VIDAL:

Q. So nonvolatility of the memory would just be an added feature?

MR. EDELL: Objection to form.

THE WITNESS: In this hypothetical setup that you are proposing, the nonvolatility of the memory would be superfluous, and it would not be needed.

Q. In your opinion, can the Data Controller 10 which is shown in Figure 1 and described elsewhere in the patent, can that function as described in the '936 Patent if Cache 13 is a nonvolatile memory?

MR. EDELL: Objection. Form.

THE WITNESS: I would say no. No. The -- well, let me qualify this.

The -- most of the, if not all, of the specification of both the '608 and the '936 Patents focus on how this particular controller would operate if the cache memory is a volatile memory. But if it weren't a volatile memory, then the cache controller could -- would operate in a way that actually does not benefit from the nonvolatility of the memory.

memory. But if it weren't a volatile memory, then the cache controller could -- would operate in a way that actually does not benefit from the nonvolatility of the memory.

BY MS. VIDAL:

Q. So nonvolatility of the memory would just be an added feature?

MR. EDELL: Objection to form.

THE WITNESS: In this hypothetical setup that you are proposing, the nonvolatility of the memory would be superfluous, and it would not be needed.

THE WITNESS: The earlier question, my understanding is that you asked whether Disk 11 is an example of the boot device that is referenced in Claim 1 of the '936 Patent.

BY MS. VIDAL:

Q. Correct. Would it be the same for the '608?

A. Let us double check if the '608 refers to the boot device.

Q. Thank you.

A. It does and, it would be the same.

Q. Thank you. In your opinion, can the claimed cache memory of both the '608 and the '936 Patents be nonvolatile memory?

MR. EDELL: Objection to form.

THE WITNESS: The Cache Memory 13 in both the '608 and the '936 Patent could be either volatile or nonvolatile memory. As I explained, if it were nonvolatile memory, there would not be any benefit derived from the fact that it is nonvolatile.

BY MS. VIDAL:

Q. And the same is true for the claimed cache memory? That could be either volatile or nonvolatile?

A. It does and, it would be the same.

Q. Thank you. In your opinion, can the claimed cache memory of both the '608 and the '936 Patents be nonvolatile memory?

MR. EDELL: Objection to form.

THE WITNESS: The Cache Memory 13 in both the '608 and the '936 Patent could be either volatile or nonvolatile memory. As I explained, if it were nonvolatile memory, there would not be any benefit derived from the fact that it is nonvolatile.

BY MS. VIDAL:

Q. And the same is true for the claimed

MR. EDELL: Objection to form.
THE WITNESS: By "the claimed," you mean the claimed cache memory in the claim of the patent?
BY MS. VIDAL:
Q. That's correct.
A. For claims, we apply a broader interpretation than for the specification.
Q. So then your answer would be yes to that?
A. The answer would be yes to whether it could be either volatile or nonvolatile memory? I think it could be either way.
Q. I'd like to turn to Paragraph 46 of the '936 declaration. That would be Exhibit 3.
A. Yes. Page 17?
Q. That's correct. In this paragraph, you state that "the term 'preloading' as used in Claims 1, 18 and 19 of the '936 Patent, means transferring data from storage to memory in anticipation of immediate or near-in-time use." Do you see that?
A. I do not see that. I don't state that. No. The paragraph reads that it is my opinion that a person of ordinary skill would have understood the term "preloading" as used in Claims 1, 18 and 19 to mean transferring data from storage to memory in

Q. That's correct.

A. For claims, we apply a broader interpretation than for the specification.

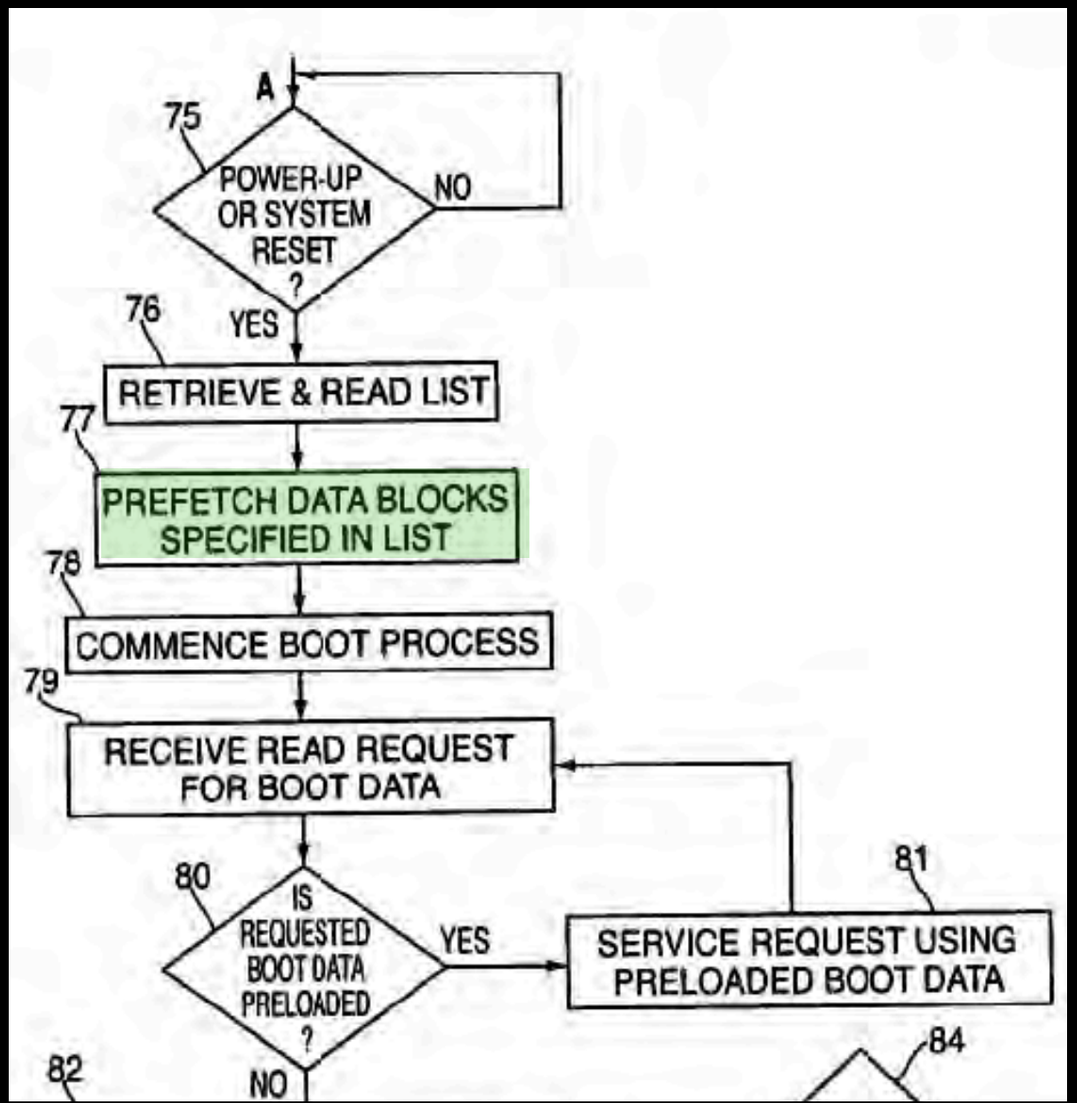
Q. So then your answer would be yes to that?

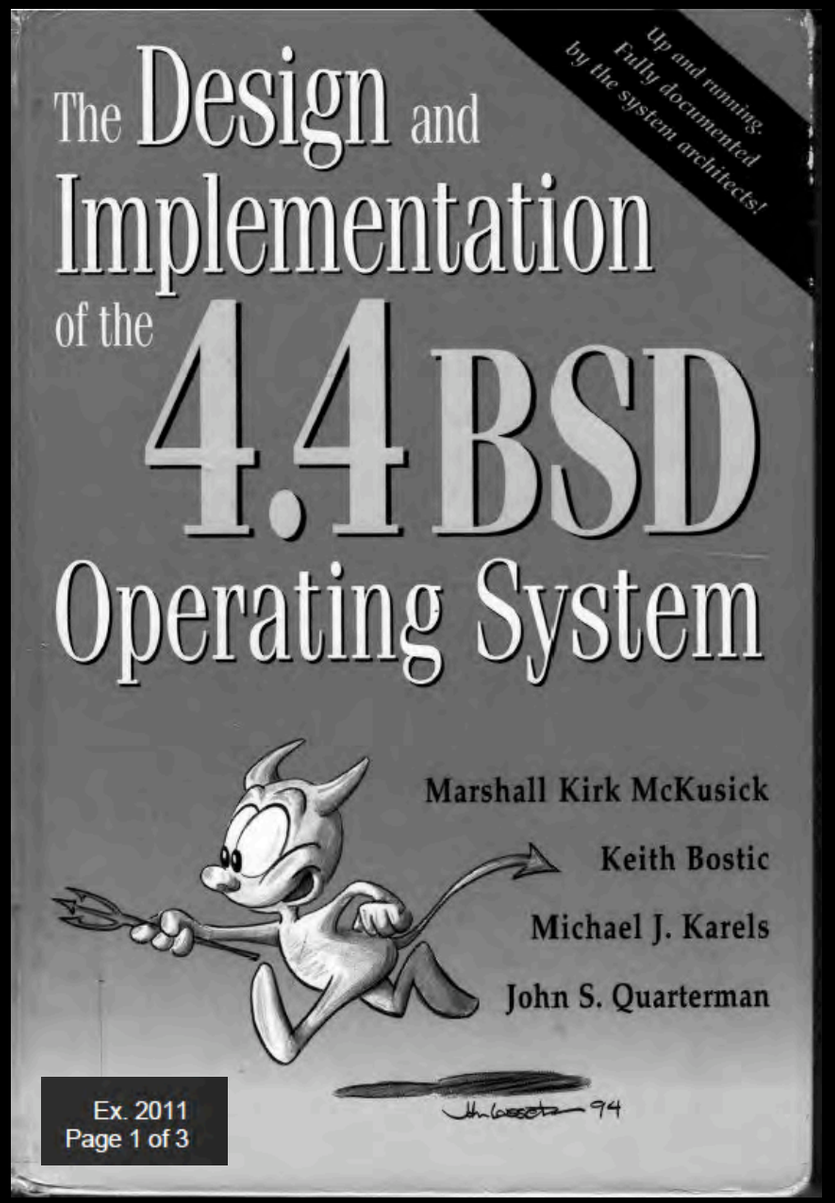
A. The answer would be yes to whether it could be either volatile or nonvolatile memory? I think it could be either way.

Q. I'd like to turn to Paragraph 46 of the '936 declaration. That would be Exhibit 3.

A. Yes. Page 17?

Q. That's correct. In this paragraph, you state that "the term 'preloading' as used in Claims 1, 18 and 19 of the '936 Patent, means transferring


A
75
POWER-UP OR SYSTEM RESET ?
NO
YES
76
RETRIEVE & READ LIST
77
PREFETCH DATA BLOCKS SPECIFIED IN LIST
78
COMMENCE BOOT PROCESS
79
RECEIVE READ REQUEST FOR BOOT DATA
80
IS REQUESTED BOOT DATA PRELOADED ?
YES
81
SERVICE REQUEST USING PRELOADED BOOT DATA
82
NO
84


Up and running.
Fully documented
by the system architects!
The Design and Implementation of the 4.4 BSD Operating System
Marshall Kirk McKusick
Keith Bostic
Michael J. Karels
John S. Quarterman
Ex. 2011
Page 1 of 3

s into main memory
mory when they are
l to reside in virtual
eside in main mem-
needed.
the kernel. 4.4BSD
chine, repairing any
on. See also *crash*

process as a result of

h an optional slash
ated by slashes, and
ns with a slash, it is
ins at the root direc-
hname, and the path
process. A slash by
the current working

low on a connection.
but the send window
tstanding. If no win-
v probe is sent.
:h the system maps a

to as the *pmap* struc-
on and access tables
y-management hard-
ccess rights, in addi-

ts the unidirectional
stream-oriented, reli-
with the "|" symbol.
m a to the standard
| b".
tput of one process is

ystem to place pages

polling I/O The normal mode for a descriptor whereby the system will block if a read request has no data available or a write request has no buffering available. A process can determine whether an I/O operation will block by polling the kernel using the *select* system call. The *select* system call can be requested to return immediately with the information or to block until at least one of the requested I/O operations can be completed. See also *nonblocking I/O; signal-driven I/O*.

POSIX The standards group for P1003, the portable operating-system interfaces established by the IEEE. Its first established standard was the kernel interface, 1003.1, which was ratified in 1988.

prefetching The retrieval of data before they are needed. Many machines prefetch machine instructions so that they can overlap the time spent fetching instructions from memory with the time spent decoding instructions.

prepaging The prefetching of pages of memory. Prepaging is a technique used by virtual-memory systems to reduce the number of page faults.

probing The operation of checking to see whether a hardware device is present on a machine. Each different type of hardware device usually requires its own technique for probing.

process In operating systems, a task or thread of execution. In UNIX, user processes are created with the *fork* system call.

process control block (PCB) A data structure used to hold process context. The hardware-defined PCB contains the hardware portion of this context. The software PCB contains the software portion, and is located in memory immediately after the hardware PCB.

process group A collection of processes on a single machine that all have the same process-group identifier. The kernel uses this grouping to arbitrate among multiple jobs contending for the same terminal.

process-group identifier A positive integer used to identify uniquely each active process group in the system. Process-group identifiers are typically defined to be the PID of the process-group leader. Process-group identifiers are used by command interpreters in implementing job control, when the command interpreter is broadcasting signals with the *killpg* system call, and when the command interpreter is altering the scheduling priority of all processes in a process group with the *setpriority* system call.

process-group leader The process in a process group whose PID is used as the process-group identifier. This process is typically the first process in a pipeline.

process identifier (PID) A nonnegative integer used to identify uniquely each active process in the system.

process open-file table See *descriptor table*.

processor priority level A priority that the kernel uses to control the delivery of interrupts to the CPU. Most machines support multiple priority levels at which the processor may execute. Similarly, interrupts also occur at multiple

the kernel using the *select* system call. The *select* system call can be requested to return immediately with the information or to block until at least one of the requested I/O operations can be completed. See also *nonblocking I/O; signal-driven I/O*.

POSIX The standards group for P1003, the portable operating-system interfaces established by the IEEE. Its first established standard was the kernel interface, 1003.1, which was ratified in 1988.

prefetching The retrieval of data before they are needed. Many machines prefetch machine instructions so that they can overlap the time spent fetching instructions from memory with the time spent decoding instructions.

prepaging The prefetching of pages of memory. Prepaging is a technique used by virtual-memory systems to reduce the number of page faults.

probing The operation of checking to see whether a hardware device is present on a machine. Each different type of hardware device usually requires its own technique for probing.

process In operating systems, a task or thread of execution. In UNIX, user processes are created with the *fork* system call.

process control block (PCB) A data structure used to hold process context. The hardware-defined PCB contains the hardware portion of this context. The

(54) **SYSTEM AND METHOD FOR REDUCING THE FOOTPRINT OF PRELOADED CLASSES**

(75) Inventors: **Hideya Kawahara**, Mountain View, CA (US); **Nedim Fresko**, San Francisco, CA (US)

(73) Assignee: **Sun Microsystems, Inc.**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/045,508**

(22) Filed: **Mar. 20, 1998**

(51) **Int. Cl.**[7] **G06F 9/00**

(52) **U.S. Cl.** **709/332**; 717/166; 717/159

(58) **Field of Search** 709/332, 331; 717/151–167

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,734,822	A *	3/1998	Houha et al.	709/216
5,815,718	A *	9/1998	Tock	709/331
5,966,542	A *	10/1999	Tock	713/1
5,966,702	A *	10/1999	Fresko et al.	707/1
6,052,778	A *	4/2000	Hagy et al.	709/331
6,061,520	A *	5/2000	Yellin et al.	703/22
6,223,346	B1 *	4/2001	Tock	713/1
6,324,637	B1 *	11/2001	Hamilton	711/216
6,339,841	B1 *	1/2002	Merrick et al.	707/103 R
6,349,344	B1 *	2/2002	Sauntry et al.	709/1
6,363,436	B1 *	3/2002	Hagy et al.	709/331
6,366,898	B2 *	4/2002	Taivalsaari et al.	707/1
6,526,565	B1 *	2/2003	Nally	717/108
6,530,080	B2 *	3/2003	Fresko et al.	717/166

FOREIGN PATENT DOCUMENTS

EP	943989 A2 *	9/1999	.....	G06F/9/44

OTHER PUBLICATIONS

Sun Microsystems. "PersonalJava 1.1 Memory Usage Technical Note." 1998.*

"The Java™ Virtual Machine Specification", Tim Lindholm, Frank Yellin, 475 pages, ©1997, ISBN 0201–63452–X.

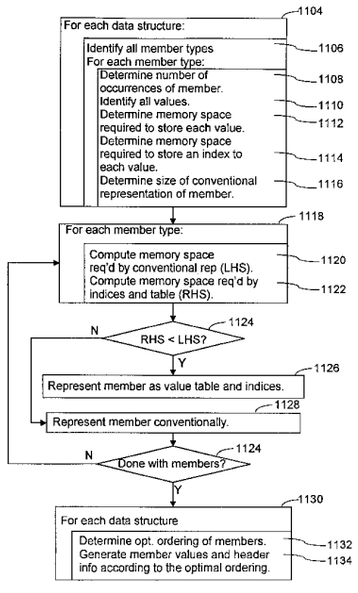
* cited by examiner

Primary Examiner—John Follansbee
Assistant Examiner—Lewis A. Bullock, Jr.
(74) *Attorney, Agent, or Firm*—Pennie & Edmonds LLP

(57) **ABSTRACT**

A method and system that reduces the space allocated for internal data structures by a runtime engine. The internal data structures store member information for preloaded classes used by applications executed by the runtime engine. The system determines the different types of internal data structures represented in the classes and identifies thee possible values of each type's members. The system next determines the amount of space required to store the values for each type in a respective value table and the number of bits needed to index each entry of that table. The system determines based on the stored information whether occurrences of a member are optimally represented as a set of value table indices and a value table or, in the conventional manner, as a general variable that stores the member's value for each occurrence. The system then emits appropriate information for the member and its parent data structure.

44 Claims, 12 Drawing Sheets



patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/045,508**

(22) Filed: **Mar. 20, 1998**

(51) **Int. Cl.**[7] **G06F 9/00**
(52) **U.S. Cl.** **709/332**; 717/166; 717/159
(58) **Field of Search** 709/332, 331; 717/151–167

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,734,822	A	*	3/1998	Houha et al.	709/216
5,815,718	A	*	9/1998	Tock	709/331
5,966,542	A	*	10/1999	Tock	713/1
5,966,702	A	*	10/1999	Fresko et al.	707/1
6,052,778	A	*	4/2000	Hagy et al.	709/331
6,061,520	A	*	5/2000	Yellin et al.	703/22
6,223,346	B1	*	4/2001	Tock	713/1
6,324,637	B1	*	11/2001	Hamilton	711/216
6,339,841	B1	*	1/2002	Merrick et al.	707/103 R
6,349,344	B1	*	2/2002	Sauntry et al.	709/1
6,363,436	B1	*	3/2002	Hagy et al.	709/331
6,366,898	B2	*	4/2002	Taivalsaari et al.	707/1

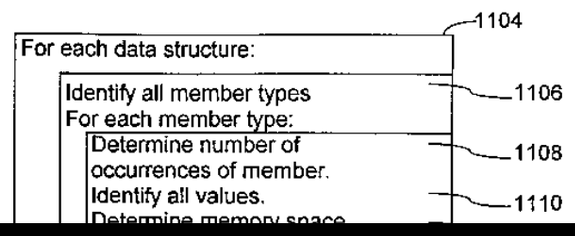
* cited by examiner

Primary Examiner—John Follansbee
Assistant Examiner—Lewis A. Bullock, Jr.
(74) *Attorney, Agent, or Firm*—Pennie & Edmonds LLP

(57) **ABSTRACT**

A method and system that reduces the space allocated for internal data structures by a runtime engine. The internal data structures store member information for preloaded classes used by applications executed by the runtime engine. The system determines the different types of internal data structures represented in the classes and identifies thee possible values of each type's members. The system next determines the amount of space required to store the values for each type in a respective value table and the number of bits needed to index each entry of that table. The system determines based on the stored information whether occurrences of a member are optimally represented as a set of value table indices and a value table or, in the conventional manner, as a general variable that stores the member's value for each occurrence. The system then emits appropriate information for the member and its parent data structure.

44 Claims, 12 Drawing Sheets



SYSTEM AND METHOD FOR REDUCING THE FOOTPRINT OF PRELOADED CLASSES

The present invention relates generally to a class preloader and, particularly, to a system and method for reducing the size in read only memory of preloaded Java classes.

BACKGROUND OF THE INVENTION

A Java program comprises a number of small software components called classes. Each class contains code and data and is defined by information in a respective class file. Each class file is organized according to the same platform-independent "class file format". Referring to FIG. **1**, there is shown a block diagram of the class file format, according to which each class file **400** includes header information **402**, a constant pool **404**, a methods table **406** and a fields table **408**. The header information **402** identifies the class file format, the size of the constant pool, the number of methods in the methods table **406** and the number of fields in the fields table **408**. The constant pool **404** is a table of structures representing various string constants, class names, field names and other constants that are referred to within the class file structure and its sub-structures. The methods table **406** includes one or more method structures, each of which gives a complete description of and Java code for a method explicitly declared by the class. The fields table **408** includes one or more field structures, each of which gives a complete description of a field declared by the class. An example of the fields table **408** is now described in reference to FIG. **1**B.

A Java program is executed on a computer containing a program called a virtual machine (VM), which is responsible for executing the code in Java classes. It is customary for the classes of a Java program to be loaded as late in the program's execution as possible: they are loaded on demand from a network server or from a local file system when first referenced during the program's execution. The VM locates and loads each class, parses the class file format, allocates internal data structures for its various components, and links it in with other already loaded classes. This process makes the method code in the class readily executable by the VM.

For small and embedded systems for which facilities, required for class loading, such as a network connection, a local file system or other permanent storage, are unavailable, it is desirable to preload the classes into read only memory (ROM). One preloading scheme is described in U.S. patent application Ser. No. 08/655,474 ("A Method and System for Loading Classes in Read-Only Memory"), which is entirely incorporated herein by reference. In this method and system, the VM data structures representing classes, fields and methods in memory are generated offline by a class preloader. The preloader output is then linked in a system that includes a VM and placed in read-only memory. This eliminates the need for storing class files and doing dynamic class loading.

Referring to FIG. **2**A, there is shown a more detailed block diagram of the VM data structures **1200**.generated by the class preloader. The data structures **1200** include a class block **1202**, a plurality of method blocks **1204**, a plurality of field blocks **1214** and a constant pool **1224**.

The class block **1202** is a fixed-size data structure that can include the following information:

the class name **1230**;

a pointer **1232** to the class block of the current class's immediate superclass;

a pointer **1234** to the method blocks **1204**;

a pointer **1236** to the field blocks **1214**; and

a pointer **1238** to the class' constant pool;

The elements of a class block data structure are referred to herein as class block members.

A method block **1204** is a fixed-sized data structure that represents one of the class's methods. The elements of a method block data structure are referred to herein as method block members. A field block **1214** is a fixed-size data structure that represents one of the class's instance variables. The elements of a field block data structure are referred to herein as field block members.

Each type of VM data structure, including the class block **1202**, method blocks **1204**, field blocks **1214** and constant pool **1224**, has a format defined by a corresponding data structure declaration. For example, a single method block declaration defines the format of all method blocks **1204**. The data structure declarations also define accessor functions (or macros) that are used by the VM to access data structure members. These data structure declarations are internal to the VM and are not used by class components. The prior art data structure declarations are now described in reference to FIG. **2**B.

Referring to FIG. **2**B, there is shown a depiction of data structure declarations **1230** that define the format of all data structure types employed by a particular VM. Each declaration **1230** includes a set of member declarations **1232** and accessor functions **1234** for accessing respective members. The member declarations **1232** and accessor functions **1234** are defined conventionally, according to the syntax of the language used in the implementation of the VM. For example, assuming the C language is used in the data structure declarations **1230**, a generic field data structure **1230**.N (shown in FIG. **2**B) could be defined as a structure T with five members of the following types with respective accessor functions:

member name	member type	accessor functions
member1	mtype1	mem1 of (T) T->member1
member2	mtype2	mem2 of (T) T->member2
member3	mtype3	mem3 of (T) T->member3
member4	mtype4	mem4 of (T) T->member4
member5	mtype5	mem5 of (T) T->member5

In this example, the member types can be any type defined by the relevant computer language, including user defined types or language types, such as integer, float, char or double. The accessor functions are macros used by the VM to access the fields without needing to access directly the structure containing the field. For example, instead of employing the expression "T→member**1**" to access field**1** in structure type T, the VM need only employ the expression "mem**1** of (T)". Accessor functions are well known in programming languages, such as C, that provide sophisticated data structure capabilities.

The internal data structures used to store "class meta data" (i.e., the class, method and field blocks **1202**, **1204**, **1214**) require large, fixed amounts of space in read-only memory. In fact, measurements indicate that this sort of class meta data often takes up much more space than the bytecodes for the class methods themselves. These internal data structures are therefore often unsuitable for use in small, resource-constrained devices in which class preloading is desirable and/or necessary.

Moreover, if the internal data structures were individually modified to save memory space, the VM code would need to

programming languages, such as C, that provide sophisticated data structure capabilities.

The internal data structures used to store "class meta data" (i.e., the class, method and field blocks **1202**, **1204**, **1214**) require large, fixed amounts of space in read-only memory. In fact, measurements indicate that this sort of class meta data often takes up much more space than the bytecodes for the class methods themselves. These internal data structures are therefore often unsuitable for use in small, resource-constrained devices in which class preloading is desirable and/or necessary.

Moreover, if the internal data structures were individually modified to save memory space, the VM code would need to

US005815718A

United States Patent [19]

Tock

[11] Patent Number: 5,815,718

[45] Date of Patent: Sep. 29, 1998

[54] **METHOD AND SYSTEM FOR LOADING CLASSES IN READ-ONLY MEMORY**

[75] Inventor: **Theron D. Tock**, Sunnyvale, Calif.

[73] Assignee: **Sun Microsystems, Inc.**, Mountain View, Calif.

[21] Appl. No.: **655,474**

[22] Filed: **May 30, 1996**

[51] **Int. Cl.**[6] **G06F 9/45**

[52] **U.S. Cl.** **395/705**; 395/710; 395/685

[58] **Field of Search** 395/701, 702, 395/710, 651, 652, 685, 705, 708

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,051,893	9/1991	Tenny et al.	364/200
5,303,380	4/1994	Tenny et al.	395/700
5,369,766	11/1994	Nakano et al.	395/700
5,594,903	1/1997	Bunnell et al.	395/712
5,613,120	3/1997	Palay et al.	395/710
5,664,128	9/1997	Bauer	345/334
5,671,413	9/1997	Shipman et al.	395/700

FOREIGN PATENT DOCUMENTS

2 242 293	9/1991	United Kingdom .

OTHER PUBLICATIONS

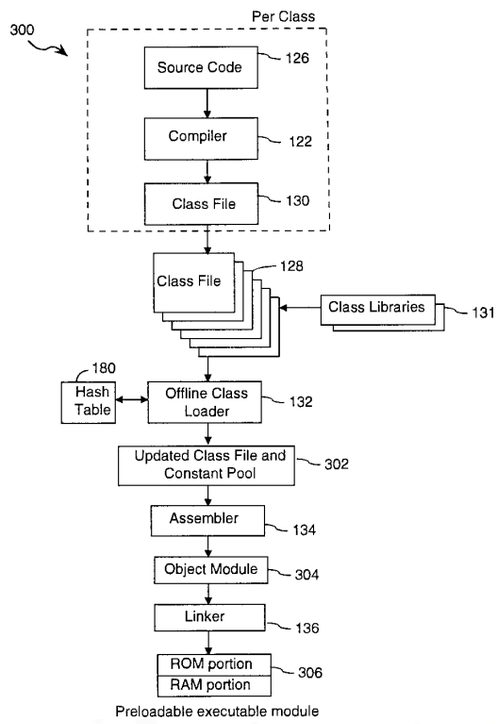
"Java Intermediate Bytecodes"; J. Gosling; 1995 ACM SIGPLAN Workshop on Intermediate Representations; pp. 111–118.

Primary Examiner—Emanuel Todd Voeltz
Assistant Examiner—Kakali Chaki
Attorney, Agent, or Firm—Gary S. Williams; Flehr Hohbach Test Albritton & Herbert LLP

[57] **ABSTRACT**

A method and system for providing an executable module having an address space for storing program data that is to reside in a read-only storage medium and an address space for storing program data that is to reside in a random access memory is herein described. The executable module represents Java classes that are structured for dynamic class loading. A static class loader is used to modify the class structure to accommodate static loading. The static class loader also identifies methods that contain unresolved symbolic references and data that varies during the execution of the module. These methods and data are identified in order to place them in the address space that resides in the random access memory. The static loader is beneficial in a distributed computing environment having a client computer that has little or no secondary storage thereby requiring applications to run entirely in random access memory. By utilizing a read-only memory to store statically loadable classes, the random access memory is left available for other uses.

20 Claims, 12 Drawing Sheets



[73] Assignee: **Sun Microsystems, Inc.**, Mountain View, Calif.

[21] Appl. No.: **655,474**

[22] Filed: **May 30, 1996**

[51] **Int. Cl.**[6] **G06F 9/45**

[52] **U.S. Cl.** **395/705**; 395/710; 395/685

[58] **Field of Search** 395/701, 702, 395/710, 651, 652, 685, 705, 708

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,051,893	9/1991	Tenny et al.	364/200
5,303,380	4/1994	Tenny et al.	395/700
5,369,766	11/1994	Nakano et al.	395/700
5,594,903	1/1997	Bunnell et al.	395/712
5,613,120	3/1997	Palay et al.	395/710
5,664,128	9/1997	Bauer	345/334
5,671,413	9/1997	Shipman et al.	395/700

FOREIGN PATENT DOCUMENTS

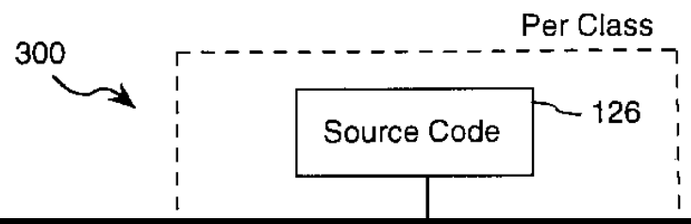
2 242 293	9/1991	United Kingdom .

Primary Examiner—Emanuel Todd Voeltz
Assistant Examiner—Kakali Chaki
Attorney, Agent, or Firm—Gary S. Williams; Flehr Hohbach Test Albritton & Herbert LLP

[57] **ABSTRACT**

A method and system for providing an executable module having an address space for storing program data that is to reside in a read-only storage medium and an address space for storing program data that is to reside in a random access memory is herein described. The executable module represents Java classes that are structured for dynamic class loading. A static class loader is used to modify the class structure to accommodate static loading. The static class loader also identifies methods that contain unresolved symbolic references and data that varies during the execution of the module. These methods and data are identified in order to place them in the address space that resides in the random access memory. The static loader is beneficial in a distributed computing environment having a client computer that has little or no secondary storage thereby requiring applications to run entirely in random access memory. By utilizing a read-only memory to store statically loadable classes, the random access memory is left available for other uses.

20 Claims, 12 Drawing Sheets



METHOD AND SYSTEM FOR LOADING CLASSES IN READ-ONLY MEMORY

The present invention relates generally to object-oriented computer systems having classes that are dynamically loaded at runtime, and particularly to a system and method for preloading a subset of the classes in a read-only memory.

BACKGROUND OF THE INVENTION

A current trend in object-oriented programming languages is to extend the functionality of the language to accommodate the distribution of dynamic content in a distributed computing environment. In one such language, this is accomplished by dynamically loading classes at runtime. A class is a collection of variables and methods that model the behavior of an object. By dynamically loading classes at runtime, existing applications can add functionality by linking in new classes that reside on any computer system within the distributed computing environment.

In such languages, symbolic references are used to refer to the class members (i.e., the class' methods and variables). When a class is invoked, the dynamic loader determines the storage schema for the class and resolves the symbolic reference. Such a loading scheme is beneficial when accessing classes that are updated often. However, a limitation of such a loading scheme is its dependency on a read/write memory device such as a random access memory (RAM). In a computing environment that has little or no secondary storage (e.g., non-volatile magnetic disk storage), dynamic loading of the classes in this manner can quickly use up the storage capacity of the RAM. As the capacity of the RAM is limited, it is desirable to minimize the amount of RAM that is used by an application. Accordingly, there exists a need to limit the amount of RAM that is utilized to execute object-oriented program code having dynamically loadable classes.

It would be beneficial to provide a method and system which overcomes the deficiencies of the prior art.

SUMMARY OF THE INVENTION

In summary, this disclosure pertains to an offline class loader that is used to produce an executable module whose classes are preloaded into memory without requiring runtime dynamic loading. The executable module, nevertheless, contains a class structure that is tailored for runtime dynamic loading. Thus, the offline class loader modifies the existing class structures to accommodate static loading. However, the class structure allows for varying data and methods that contain unresolved references. The offline class loader tags these methods and data specifying that they are to be stored in a random access memory. All other data is stored in a read-only memory. At the completion of the static loading process, a preloadable executable module is generated that contains two addresses spaces. A first address space that contains methods having unresolved references and data that varies during the execution of the module is loaded in a random access memory. The second address space contains methods having static loaded classes and constant data which is loaded into a read-only memory.

A preloadable executable module of this fashion is advantageous in a distributed computer system having client computers with little or no secondary storage. Such client computers require applications to run entirely in random access memory which quickly turns into a limited resource. By utilizing the offline class loader to partition an application into two address spaces, the amount of RAM utilized by the preloadable module is minimized.

In an embodiment, a client computer having minimal secondary storage utilizes an offline class loader to preload a browser in the client's read-only memory. The browser is partitioned into the aforementioned two address spaces. At system initialization or power up, the random access memory portion of the browser is loaded from read-only memory into the random access memory. By executing a large portion of the browser from read-only memory, the browser has additional RAM storage to store information-content and executable modules that it can obtain from other server computers that the client is in communication with.

BRIEF DESCRIPTION OF THE DRAWINGS

Additional objects and features of the invention will be more readily apparent from the following detailed description and appended claims when taken in conjunction with the drawings, in which:

FIG. **1** is a block diagram of a distributed computer system.

FIG. **2** is a block diagram of a client computer in the distributed computer system of FIG. **1**.

FIG. **3** is a flow diagram illustrating the processing components used to produce the preloadable executable module.

FIG. **4** illustrates the file layout for a class file.

FIG. **5** illustrates the file layout for a constant pool.

FIG. **6** illustrates the class block data structures.

FIG. **7** illustrates an instruction bytecode stream.

FIG. **8**, which is a combination of FIGS. **8**A and **8**B, represents a flow chart of the method used by the offline class loader.

FIG. **9** is a flow chart of the method for building the class block data structures.

FIG. **10** is a flow chart of the method for eliminating duplicate constants.

FIG. **11** is a flow chart of the method for converting a non-quick instruction format into a quick instruction format.

FIG. **12** is a block diagram showing the mapping of a preloaded application into read-only memory and random-access memory and indicating the loading of the portion of the methods and data mapped into random-access memory by a static class initializer.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The method and system described herein utilizes a distributed computing environment having a communication link that connects at least one server computer and a number of client computers. Some of the client computers have little or no secondary storage (e.g., non-volatile magnetic disk storage) thereby requiring applications to be run entirely from random access memory. An application developed in the Java programming language is executed on such a client computer. Preferably, the application is a browser that is used to import Java content, such as Java applets, from one or more server computers. Typically, the browser is an interpreted program module that retrieves Web documents utilizing a HyperText Transfer Protocol (HTTP) to access one or more Web pages formatted as HyperText Markup Language (HTML) documents from a server acting as a Web site. The HTML documents are interpreted and presented to the user associated with the client computer. Often, the HTML documents embed applets. An applet is a executable module represented as a Java class. The browser loads in the applet and its associated classes in order to execute the applet.

METHOD AND SYSTEM FOR LOADING CLASSES IN READ-ONLY MEMORY

The present invention relates generally to object-oriented computer systems having classes that are dynamically loaded at runtime, and particularly to a system and method for preloading a subset of the classes in a read-only memory.

BACKGROUND OF THE INVENTION

A current trend in object-oriented programming languages is to extend the functionality of the language to accommodate the distribution of dynamic content in a distributed computing environment. In one such language, this is accomplished by dynamically loading classes at runtime. A class is a collection of variables and methods that

US006463509B1

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

(10) **Patent No.: US 6,463,509 B1**
(45) **Date of Patent: Oct. 8, 2002**

(54) **PRELOADING DATA IN A CACHE MEMORY ACCORDING TO USER-SPECIFIED PRELOAD CRITERIA**

(75) Inventors: **Deniz Teoman**, San Mateo; **John M. Neil**, San Francisco, both of CA (US)

(73) Assignee: **Motive Power, Inc.**, San Mateo, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/238,656**

(22) Filed: **Jan. 26, 1999**

(51) **Int. Cl.**[7] **G06F 12/00**
(52) **U.S. Cl.** **711/137**; 711/118; 711/141; 711/152; 711/113
(58) **Field of Search** 711/118, 137, 711/138, 129, 113, 141, 152, 3, 125; 710/52; 713/400; 714/5

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,609,665	A		9/1971	Kronies
3,806,888	A		4/1974	Brickman et al.
4,020,466	A		4/1977	Cordi et al.
4,215,400	A		7/1980	Denko
4,295,205	A		10/1981	Kunstadt
4,342,079	A		7/1982	Stewart et al.
4,435,775	A		3/1984	Brantingham et al.
4,500,954	A	*	2/1985	Duke et al. 711/138
4,637,024	A	*	1/1987	Dixon et al. 714/819
5,128,810	A		7/1992	Halford
5,131,089	A		7/1992	Cole
5,146,576	A		9/1992	Beardsley
5,218,689	A		6/1993	Hotle
5,226,168	A		7/1993	Kobayashi et al.
5,263,142	A		11/1993	Watkins et al.
5,287,457	A		2/1994	Arimilli et al.
5,291,584	A		3/1994	Challa et al.
5,293,622	A		3/1994	Nicholson et al.
5,359,713	A	*	10/1994	Moran et al. 710/52
5,396,596	A		3/1995	Hashemi et al.
5,420,998	A		5/1995	Horning
5,437,018	A		7/1995	Kobayashi et al.
5,448,719	A	*	9/1995	Schultz et al. 714/5

(List continued on next page.)

OTHER PUBLICATIONS

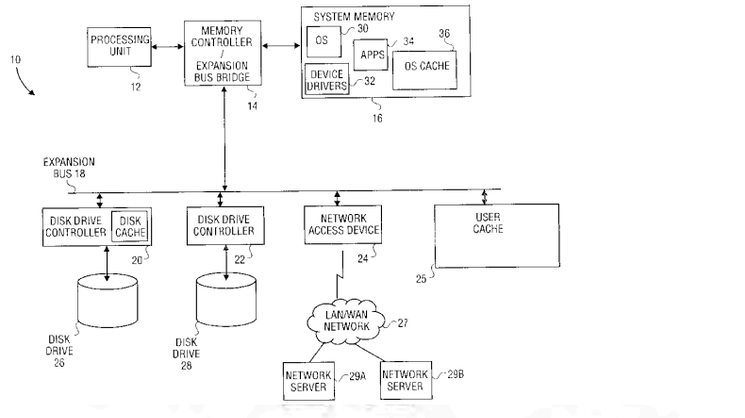
International Search Report in connection with International Application No. PCT/US00/02156 (6 pages).
"The I/O System", Inside Windows NT Second Edition, Microsoft Press, David A. Solomon, pp. v–xiv, 325–393, 1998.
"Filter Drivers", Windows NT File System Internals A Developer's Guide, Rajeev Nagar, O'Rielly & Associates, Inc., pp. vii–x, 615–667, 1997.

Primary Examiner—David Hudspeth
Assistant Examiner—Fred F. Tzeng
(74) *Attorney, Agent, or Firm*—Blakely, Sokoloff, Taylor & Zafman LLP

(57) **ABSTRACT**

An apparatus and method for caching data in a storage device of a computer system. A relatively high-speed, intermediate-volume storage device is operated as a user-configurable cache. Requests to access a mass storage device such as a disk or tape are intercepted by a device driver that compares the access request against a directory of the contents of the user-configurable cache. If the user-configurable cache contains the data sought to be accessed, the access request is carried out in the user-configurable cache instead of being forwarded to the device driver for the target mass storage device. Because the user-cache is implemented using memory having a dramatically shorter access time than most mechanical mass storage devices, the access request is fulfilled much more quickly than if the originally intended mass storage device was accessed. Data is preloaded and responsively cached in the user-configurable cache memory based on user preferences.

26 Claims, 12 Drawing Sheets



1

Apple v. Realtime
Proceeding No. IPR2016-01365
APPLE 1020

the user cache. One user-configurable memory management policy is whether to reserve storage space in the user cache for storing a system page file. The system page file is the portion of allocated virtual memory not mapped to system RAM. Ordinarily the system page file is mapped to a disk drive or other mass storage device. By reserving capacity for the system page file in the user-cache, the system page file can be swapped between the user cache and system RAM much more quickly than if the disk drive or other mass storage device was used to hold the page file. Another user configurable memory management policy is the maximum preloadable file size. As shown in FIG. **10**, for example, the user may specify that files larger than a user entered threshold are not to be preloaded.

In the context of reserving capacity in the user cache, it should be noted that a significant benefit of the user cache is that storage space is provided for both preloading and responsive caching operations without having to specifically dedicate respective regions of the user cache storage space for those operations. In an alternate embodiment, however, the user cache may be partitioned into respective dedicated storage regions for the preloading and responsive caching operations.

Examples of preloading policies include, but are not limited to, preloading complete files in response to file segment access, preloading all files within the directory or folder of a launched application, preloading all files in a directory or folder if a threshold number of files from the directory or folder have already been accessed, preloading files in the system directory or folder, preloading files having a particular file type identifier if a threshold number of files having the file type identifier have been accessed, and so forth. The file type identifier may be a filename extension such as ".doc" or ".psd", as used in many operating systems, or the file type identifier may be a file attribute that does not appear in the file name. Also, as indicated in FIG. **10**, the threshold number of files that have a particular file type identifier and the threshold number of files that are from a directory or folder are specified by the user. In many cases, the preloading policies translate directly into criteria for triggered preloading. For example, a policy to preload all files in a directory or folder if a threshold number of the files have been accessed sets up a preload trigger. The user cache manager periodically inspects the access table maintained by the observer to determine if the trigger criteria is met (e.g., whether the threshold number of files from the indicated directory have been accessed). Other preload policies give rise to commanded preload operations. For example, a policy to preload the system directory causes the user cache manager to begin commanded preloads of the directory contents.

FIG. **11** is an exemplary user interface **127** generated by the user cache manager to allow the user to specify commanded preloads. According to one embodiment, a user enters a commanded preload in the interface of FIG. **11** by clicking the add button on the user interface **127**. The user cache manager responds by generating a view of the file storage within the computer system. The user may then double click selected logical drives, directories or filenames to indicate that files meeting the specified criteria are to be preloaded. For example, the user may select the file "C:\Program Files\Netscape\netscape.exe" to indicate that the netscape.exe file is to be preloaded into the user cache from the specified logical drive and directory (likewise for pshop.exe, FirstBirthday.mov and Business Plan.wrd). Similarly, the user may select the directory "C:\Program Files\AutoCAD\" to indicate that all the files in the drive C: subdirectory "\Program Files\AutoCAD\" are to be preloaded. The user may also select the drive E:\ to indicate that all the files in drive E are to be preloaded. The user may also enter wildcards within filenames to indicate that files having filenames that match the wildcard are to be preloaded. For example, to load all files having the extension ".doc" from the logical drive and directory, "C:\Program Files\Winword\", the user would select the indicated logical drive and directory and enter "*.doc" (i.e., "C:\Program Files\Winword*.doc"). After entering file parameters in the interface of FIG. **11**, the user may click the apply button to initiate commanded preloading.

The user may also indicate, via the user interface **127** of FIG. **11**, to lock down selected files in the user cache. In one embodiment, for example, selected files are locked down in the user cache in response to the user clicking in the left most column of the interface adjacent a commanded preload entry. A lock symbol is displayed to indicate that the files covered by the commanded preload entry will be locked down in the user cache. The lock down indication can be removed by clicking on the lock symbol.

The column to the right of the lock down column is an exclude column and can be used to exclude certain files from the user cache. For example, by clicking in the exclude column adjacent the preload entry "D:\Video Benchmark\Disk Test\speed.exe", the executable file speed.exe is prevented from being loaded into the user cache, either by preloading or by responsive caching.

FIG. **12** is an exemplary user interface **129** generated by the user cache manager to permit the user to generate a report of the user cache contents, test the memory in the user cache, flush the user cache or backup the contents of the user cache to a mass storage such as a tape backup or a disk drive. Other operations may be prompted in the user interface **129** of FIG. **12** including, but not limited to, a battery test, an age report indicating the relative order in which files have been loaded into the user cache and so forth.

In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made to the specific exemplary embodiments without departing from the broader spirit and scope of the invention as set forth in the appended claims. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method of pre-loading data into a storage array operating as a user cache for a computer system, the method comprising:

causing data to be retrieved from a mass storage device in accordance with pre-load criteria, the pre-load criteria identifying data that is to be pre-loaded into the storage array prior to receiving a command on the data;

storing the data in the storage army for subsequent access by commands on the data;

inspecting a request for data from a process; causing the requested data to be retrieved from a mass storage device if the requested data is not in the storage array; and p**1** storing the requested data in the storage array in accordance with exclusion criteria that specifies data excluded from storage in the storage array using a file identifier.

2. The method of claim **1** further comprising:

obtaining the pre-load criteria from a user of the computer system.

, a
che
ory
by
m-
ser
by
ser
file
nen
nes
be
file
hat
che
for
d).

1. A method of pre-loading data into a storage array operating as a user cache for a computer system, the method comprising:

causing data to be retrieved from a mass storage device in accordance with pre-load criteria, the pre-load criteria identifying data that is to be pre-loaded into the storage array prior to receiving a command on the data;

storing the data in the storage army for subsequent access by commands on the data;

inspecting a request for data from a process; causing the requested data to be retrieved from a mass storage device if the requested data is not in the storage array; and p**1** storing the requested data in the storage array in accordance with exclusion criteria that specifies data excluded from storage in the storage array using a file identifier.

2. The method of claim **1** further comprising:

United States Patent [19]

Sukegawa

[11] **Patent Number: 5,860,083**

[45] **Date of Patent: Jan. 12, 1999**

[54] **DATA STORAGE SYSTEM HAVING FLASH MEMORY AND DISK DRIVE**

[75] Inventor: **Hiroshi Sukegawa**, Tokyo, Japan

[73] Assignee: **Kabushiki Kaisha Toshiba**, Kawasaki, Japan

[21] Appl. No.: **818,983**

[22] Filed: **Mar. 14, 1997**

[30] **Foreign Application Priority Data**

Nov. 26, 1996 [JP] Japan 8-314850

[51] **Int. Cl.**[6] **G06F 12/08**

[52] **U.S. Cl.** **711/103**; 711/117; 711/171; 711/173

[58] **Field of Search** 711/103, 113, 711/117, 170, 171, 173

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,175,842	12/1992	Totani	711/161
5,371,876	12/1994	Ewertz et al.	711/159
5,437,018	7/1995	Kobayashi et al.	395/652
5,535,357	7/1996	Moran et al.	711/103
5,644,539	7/1997	Yamagami et al.	365/200
5,701,492	12/1997	Wadsworth et al.	395/712
5,745,418	4/1998	Ma et al.	365/185.33
5,778,418	7/1998	Auclair et al.	711/101

FOREIGN PATENT DOCUMENTS

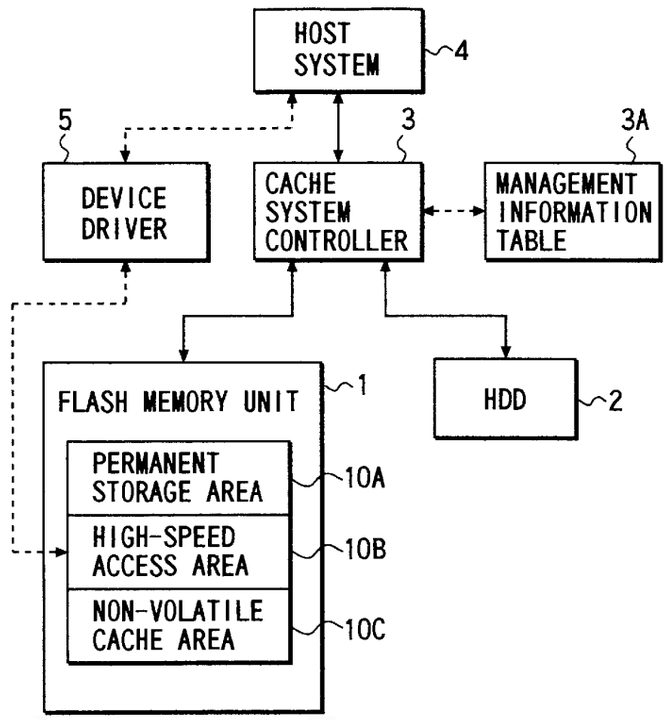
0 706 135	4/1996	European Pat. Off. .
5-11933	1/1993	Japan .
8-63395	3/1996	Japan .
8-115241	5/1996	Japan .

Primary Examiner—Tod R. Swann
Assistant Examiner—Conley B. King, Jr.
Attorney, Agent, or Firm—Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P.

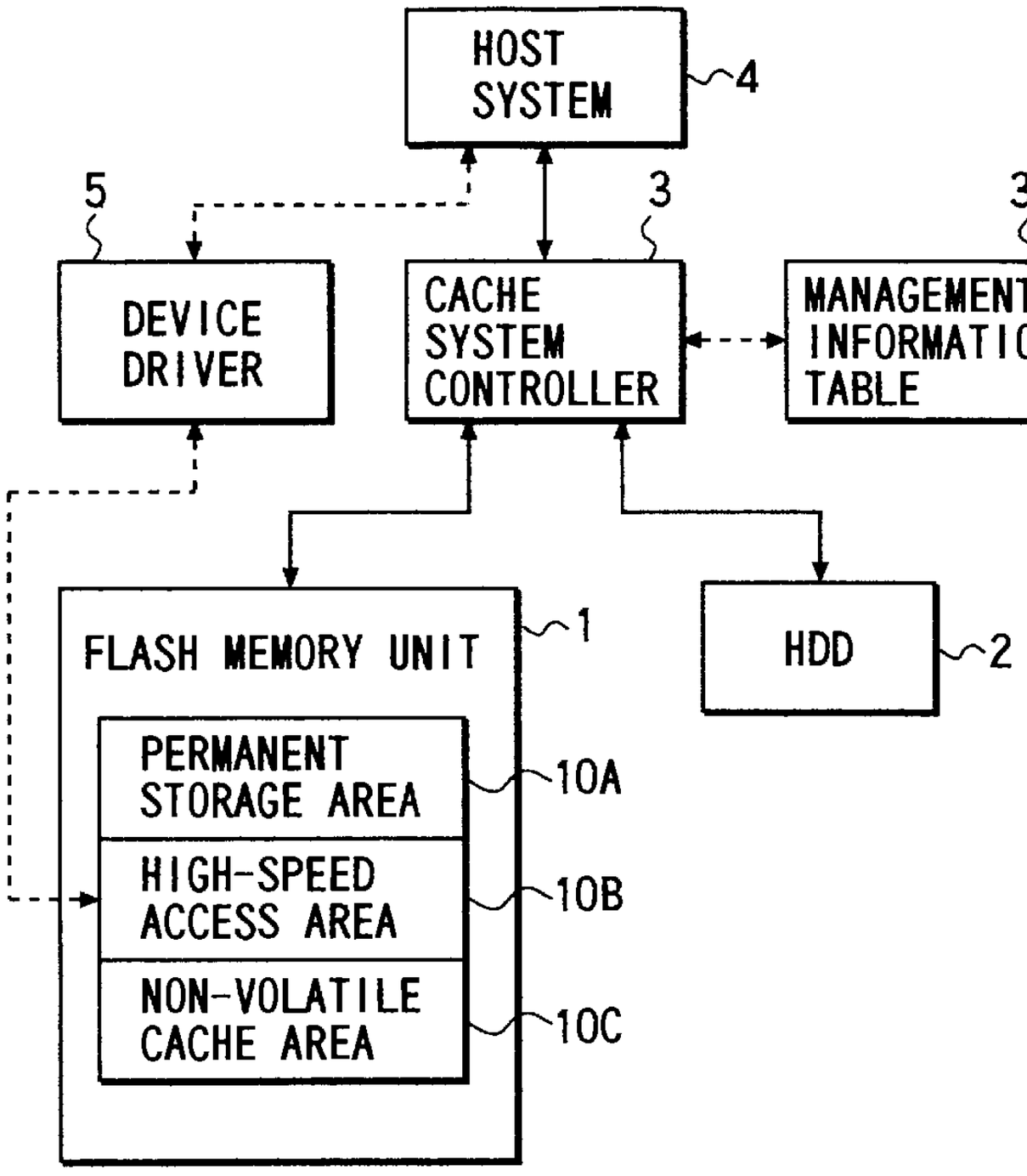
[57] **ABSTRACT**

In a data storage system using a flash memory unit and an HDD, the storage area of the flash memory unit is logically divided into a permanent storage area, a non-volatile cache area, which are used as cache memory areas of the HDD, and a high-speed access area. These divided areas are individually managed. The permanent storage area stores data which is used frequently for a relatively long time period. The non-volatile cache area is used as an ordinary cache memory area in which data, which is updated relatively frequently, is stored. The high-speed access area is a storage area to be used by, e.g. an operating system (OS) of a host system. For example, a swap file, which needs to be accessed at high speed, is shifted into the high-speed access area.

28 Claims, 10 Drawing Sheets



1


HOST SYSTEM
4
5
3
3A
DEVICE DRIVER
CACHE SYSTEM CONTROLLER
MANAGEMENT INFORMATION TABLE
FLASH MEMORY UNIT
1
HDD
2
PERMANENT STORAGE AREA
10A
HIGH-SPEED ACCESS AREA
10B
NON-VOLATILE CACHE AREA
10C

DATA STORAGE SYSTEM HAVING FLASH MEMORY AND DISK DRIVE

BACKGROUND OF THE INVENTION

The present invention relates to a data storage system which is applied to a computer system and has a flash memory unit (also known as "semiconductor disk unit") and a disk drive.

In a conventional computer system, a hard disk drive (HDD) is used as an external memory device wherein a disk is used as a storage medium. The HDD can be used as a large-capacity file apparatus. However, as compared to a main memory comprising a semiconductor memory (e.g. a DRAM), the access speed of the HDD is lower. A cache system for the HDD has been known as means for increasing the access speed of the HDD.

In the cache system, in particular, frequently used data is-stored in a storage medium having a higher access speed than the HDD, thereby compensating the low access speed of the HDD. In a specific system, a part of the storage area of the main memory (volatile IC memory) comprising a DRAM is used as a cache area of the HDD (this system being called "smartdrive"). In this system, however, the main memory is cleared when the power to the system is switched off. Thus, the cache system does not function when the power is switched on. Accordingly, after the power is switched on, the HDD is accessed to enable the cache system to function effectively, thereby achieving a learning effect. The learning effect will now be described. When a request for access to the HDD occurs, this request first fails since no data is stored in the cache memory (a part of the main memory in this case). Then, the data associated with the access request is read out from the HDD and stored in the cache memory. Thus, if the next access request occurs, the data stored in the cache memory is quickly read out from the cache memory in place of the cache memory. This effect of achieving the cache function is called "leaning effect."

The cache system using the above-described main memory does not effectively function when the first access request for the HDD occurs at the time of turning-on of power. Consequently, when the computer system is started up, the cache system cannot be utilized to run the operating system (OS) or frequently used application programs (AP). The OA and AP are thus started up by using the low access-speed HDD. With an increase in the scale of the OS and AP, the low access speed of the HDD elongates the time needed to start up the OS and AP. This is considered a serious problem.

To solve this problem, there has been proposed a cache system for an HDD, which uses a flash memory unit comprising a flash EEPROM (electrically erasable programmable read-only memory). The flash memory, unlike the main memory, is a non-volatile storage medium and has a higher access speed than the HDD. Accordingly, in the cache system using the flash memory, the data stored in the flash memory functioning as cache memory can be retained even if the power is switched off, and the cache function is effectively performed at the time of turning on power. Moreover, the flash memory having a higher access speed than the HDD can perform a high-speed buffer function.

As described above, the cache system using the cache memory as flash memory can effectively perform the cache function for the HDD even when the power is turned on. Therefore, the cache system as combined with the HDD can constitute a high-speed, large-capacity external storage system.

In other words, the data storage system comprising the combination of the large-capacity HDD and high-speed, non-volatile flash memory can achieve not only the above-described function but also a function which will influence the performance of the computer system by the effective use of the respective memory units. Specifically, the storage area of the flash. memory is set in accordance with the contents of data to be stored, or the cooperative function of the flash memory and the HDD is set, thereby to effectively use the data storage system, as will be described below.

For example, when information (or control information in the present invention) necessary for starting up the OS or a frequently used AP is to be stored in the storage area in the flash memory, it is desirable that the OS and AP are permanently stored in the flash memory since the frequency of use of both programs is high. On the other hand, for example, when a word search utility program is run for a number of files and the HDD is accessed, it is not important, in general, to permanently store the file accessed from the HDD in the cache memory area of the flash memory. Since such an accessed file is frequently updated, there is no need to permanently store it.

Besides, when the computer system performs a swapping operation, it is possible to store a swap file in the flash memory in place of the HDD. Since the file size of the swap file produced in the swapping operation is variable, the size of the storage area set in the flash memory needs to be variable accordingly. However, since the storage area of the flash memory is limited, it is desirable to perform a cooperative function with the HDD, for example, to use the storage area in the HDD in accordance with the increase in file size of the swap file.

BRIEF SUMMARY OF THE INVENTION

The object of the present invention is to provide a data storage system having a disk drive and a flash memory unit, wherein the storage area of the flash memory unit as well as a cache system is efficiently used, and cooperative functions of the flash memory and an HDD are achieved, whereby the data storage system can be efficiently used.

A data storage system according to the present invention comprises a disk drive, flash memory means, and control means. The flash memory means uses a non-volatile flash memory as a data storage medium, and has an entire storage area logically sorted into a plurality of storage areas assigned to predetermined functions. The control means controls data input/output of the disk drive and the flash memory means and stores all data or specified data stored in the disk of the disk drive into that one of the logically sorted storage areas in the flash memory means, which has the associated function.

The flash memory means has the entire storage area logically sorted into a first storage area for permanently storing data, a second storage area which can be associated with the host system and is used for high-speed access, and a third storage area for use as a non-volatile cache memory area. When the control means accesses the flash memory means according to an instruction from the host system, the control means individually manages the first storage area, second storage area and third storage area. The data stored in the disk drive is read out and stored in the first and third storage areas, and the data transferred from the host system is stored in the second storage area.

According to this system, for example, control information necessary for starting an application program (AP) and an OS, which are frequently used, is stored in the first

[redacted]

data storage system, as will be described below.

For example, when information (or control information in the present invention) necessary for starting up the OS or a frequently used AP is to be stored in the storage area in the flash memory, it is desirable that the OS and AP are permanently stored in the flash memory since the frequency of use of both programs is high. On the other hand, for example, when a word search utility program is run for a number of files and the HDD is accessed, it is not important, in general, to permanently store the file accessed from the HDD in the cache memory area of the flash memory. Since such an accessed file is frequently updated, there is no need to permanently store it.

Besides, when the computer system performs a swapping operation, it is possible to store a swap file in the flash memory in place of the HDD. Since the file size of the swap

[redacted]

In the present invention, it is assumed that the entire storage area of the flash memory unit **1** is logically divided into permanent storage area **10**A, high-speed access area **10**B and non-volatile cache area **10**C and the divided areas are managed. The controller **3** manages the storage areas **10**A to **10**C of the flash memory unit **1** by using a management information table **3**A. The management information table **3**A is stored, for example, in the non-volatile cache area **10**C of flash memory unit **1**.

The first embodiment relates to a system wherein the permanent storage area **10**A of flash memory unit **1** is used as a cache memory area. In this embodiment, it is supposed that the user desires to start a frequently used application program (AP) at high speed at all times.

The user starts a data storage utility program of the cache system controller **3** via a user interface provided in the host system **4** (step S**1**). The data storage utility program reads specified data from the HDD **2** and stores the read data in a specified storage area in the flash memory unit **1**. In this case, it is assumed that the user sets the permanent storage area **10**A in the flash memory unit **1** as the data storage area, at the time of instructing the start of the data storage utility program (step S**2**).

Then, the user instructs the host system **4** to start the AP (step S**3**). The host system **4**, upon receiving the AP start instruction, issues a read command to the controller **3** in order to read control information necessary for the start of the AP from the HDD **2**.

The controller **3** controls the HDD **2**, reads out the control information necessary for the start of the AP and transfers the read-out control information to the host system **4** (step S**4**). At this time, according to the started-up data storage utility program, the controller **3** stores the AP control information read out from the HDD **2** in the permanent storage area **10**A of flash memory unit **1** (step S**5**). When the AP has been prepared to start, the data storage utility program is stopped by the instruction from the user ("YES" in step S**6**; step S**7**). Through these operations, the control information necessary for starting the AP is stored in the permanent storage area **10**A in the flash memory unit **1**.

In this case, the control information is stored in the permanent storage area **10**A in flash memory unit **1** under the file name designated by the user. Information for correlating the file name and the AP and information of other comment is recorded on the management information table **3**A by the data storage utility program. The user inputs the file name to the controller **3** via the user interface, thereby referring to the file (the control information of the AP in this case) stored in the permanent storage area **10**A. The user can delete the file, if unnecessary. In other words, the control information necessary for starting the AP is permanently stored in the permanent storage area **10**A in the flash memory unit **1** as one file, until the user instructs the deletion of the file.

If the user instructs the start of the same AP via the user interface, the host system **4** issues the read command, as described above, to read from the HDD **2** the control information necessary for starting the AP ("YES" in step S**8**). Upon receiving the read command, the controller **3** determines whether the control information to be accessed is stored in the flash memory unit **1** by using the management information table **3**A. Since the AP control information is stored in the permanent storage area **10**A, the cache memory area is successfully accessed. Accordingly, the controller **3** reads out the AP control information from the permanent storage area **10**A of flash memory unit **1**, without accessing the HDD **2**, and transfers the read-out control information to the host system **4** (step S**9**). The host system **4** starts the AP designated by the user on the basis of the transferred AP control information (step S**10**).

By the above-described cache system, the control information of the frequently used AP designated by the user is read out from the HDD **2** and stored in the permanent storage area **10**A in the flash memory unit **1** used as the cache memory area. Accordingly, when the AP is to be started next time, the AP control information can be read out quickly from the permanent storage area **10**A used as cache memory area, and not from the HDD **2**. Thereby, the host system **4** can quickly acquire the control information at the time of starting the AP. As a result, the AP can be quickly started. Since the AP control information is permanently stored in the permanent storage area **10**A until the user instructs the deletion of the control information, as described above, the frequently used AP can be started quickly at all times.

(First Modification of the First Embodiment)

FIG. **4** is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area **10**A of flash memory unit **1**, for example, when the OS of the host system **4** is started in a series of operations from the turn-on of power to the completion of the starting operation.

When the system is switched on and the user sets the data storage mode via the user interface, the controller **3** stores the information representing the data storage mode in the permanent storage area **10**A in flash memory unit **1** (steps S**11** to S**13**).

After the power is turned off and then turned on again, the controller **3** starts the above-mentioned data storage utility program on the basis of the information of the set data storage mode ("YES" in step S**14**; step S**15**). According to the data storage utility program, the control information, which is pre-stored in the HDD **2** and necessary for starting the OS, is read out and stored in the permanent storage area **10**A (steps S**16** and S**17**). The controller **3** transfers to the host system **4** the control information necessary for starting the OS read out from the HDD **2**. Based on the control information, the host system **4** starts the OS. After the preparation for starting the OS is completed, the data storage utility program is stopped ("YES" in step S**18**; step S**19**).

According to this system, when the OS is automatically started by the control information read out from the HDD **2** at the time of turning-on of power, the control information is stored in the permanent storage area **10**A used as the cache memory area for the HDD **2**. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD **2** but from the permanent storage area **10**A or cache memory area, and the read-out control information is transferred to the host system **4**. Thus, the control information can be accessed from the permanent storage area **10**A in the flash memory unit **1** having a higher access speed than the HDD **2**. As a result, the OS can be started at higher speed.

Like the AP control information, the OS control information may be stored as one file in the permanent storage area **10**A. Thereby, the user can refer to, or delete, the OS control information on an as-needed basis. For example, at the time of shipment of the personal computer, if the OS is pre-installed in the flash memory unit **1** functioning as cache memory area, the user can delete the OS control information based on the user's judgment. Specifically, the OS control information may be deleted by a user who is used to starting

preparation for starting the OS is completed, the data storage utility program is stopped ("YES" in step **S18**; step **S19**).

According to this system, when the OS is automatically started by the control information read out from the HDD **2** at the time of turning-on of power, the control information is stored in the permanent storage area **10**A used as the cache memory area for the HDD **2**. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD **2** but from the permanent storage area **10**A or cache memory area, and the read-out control information is transferred to the host system **4**. Thus, the control information can be accessed from the permanent storage area **10**A in the flash memory unit **1** having a higher access speed than the HDD **2**. As a result, the OS can be started at higher speed.

Like the AP control information, the OS control informa-

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE, INC.,
Petitioner

v.

REALTIME DATA LLC,
Patent Owner

Case IPR2016-01365
Patent 7,181,608

EXPERT DECLARATION OF DR. GODMAR BACK IN SUPPORT OF THE PATENT OWNER'S RESPONSE

user input.[42] Regardless, like the first technique, data is saved to non-volatile cache area 10C.[43]

62. As such, Dr. Neuhauser alleges that the one-time act of "storing" or "permanently storing" data in a particular location corresponds to the "preloading" claimed by the '608 Patent. This interpretation of "preloading," however, is unreasonably broad and contrary to the '608 Patent and its file history. As explained in detail below, the '608 Patent discloses that "preloading" may be performed in volatile memory devices, which is expressly foreclosed by Sukegawa, which requires storage in non-volatile flash memory to retain the data even when the system is turned off.

63. As described above in Section VI.B., "preloading" means "transferring data from storage to memory in anticipation of immediate or near-in-time use." Thus, a POSITA would have understood that "storing" data in a particular location, as Sukegawa does, cannot comprise "preloading." The '608 Patent explains that "storing" is different from "loading," and "storing" is different from "preloading" (which is one form of "loading"). As one example, the specification explains that "[m]ass storage devices (such as a 'hard disk') typically

[42] *See* Ex. 1003, Neuhauser Dec. at ¶¶ 110, 114-115; Petition at 31-32. *See also* Ex. 1005, Sukegawa, 7:28-55.

[43] *See* Petition at 31-32; Ex. 1005, Sukegawa at 7:17-65.

area 10C.[43]

62. As such, Dr. Neuhauser alleges that the one-time act of "storing" or "permanently storing" data in a particular location corresponds to the "preloading" claimed by the '608 Patent. This interpretation of "preloading," however, is unreasonably broad and contrary to the '608 Patent and its file history. As explained in detail below, the '608 Patent discloses that "preloading" may be performed in volatile memory devices, which is expressly foreclosed by Sukegawa, which requires storage in non-volatile flash memory to retain the data even when the system is turned off.

63. As described above in Section VI.B., "preloading" means "transferring data from storage to memory in anticipation of immediate or near-in-

“preloading…prior to completion of initialization of the central processing unit”

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

(10) **Patent No.: US 7,181,608 B2**
(45) **Date of Patent: Feb. 20, 2007**

(54) **SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS**

(75) Inventors: **James J. Fallon**, Armonk, NY (US); **John Buck**, Oceanside, NY (US); **Paul F. Pickel**, Bethpage, NY (US); **Stephen J. McErlain**, New York, NY (US)

(73) Assignee: **Realtime Data LLC**, New York, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 223 days.

(21) Appl. No.: **09/776,267**

(22) Filed: **Feb. 2, 2001**

(65) **Prior Publication Data**

US 2002/0069354 A1 Jun. 6, 2002

Related U.S. Application Data

(60) Provisional application No. 60/180,114, filed on Feb. 3, 2000.

(51) **Int. Cl.**
G06F 9/24 (2006.01)
G06F 9/00 (2006.01)
G06F 13/00 (2006.01)

(52) **U.S. Cl.** **713/2**; 713/1; 711/113

(58) **Field of Classification Search** 713/2, 713/1, 100; 711/170, 118, 113
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,127,518 A 11/1978 Coy et al.
4,302,775 A 11/1981 Widergren et al.
4,394,774 A 7/1983 Widergren et al.
4,574,351 A 3/1986 Dang et al.

(Continued)

FOREIGN PATENT DOCUMENTS

DE 4127518 A1 2/1992

(Continued)

OTHER PUBLICATIONS

IBM, Fast Dos Soft Boot, Feb. 1, 1994, vol. 37, Issue 2B, pp 185-186.*

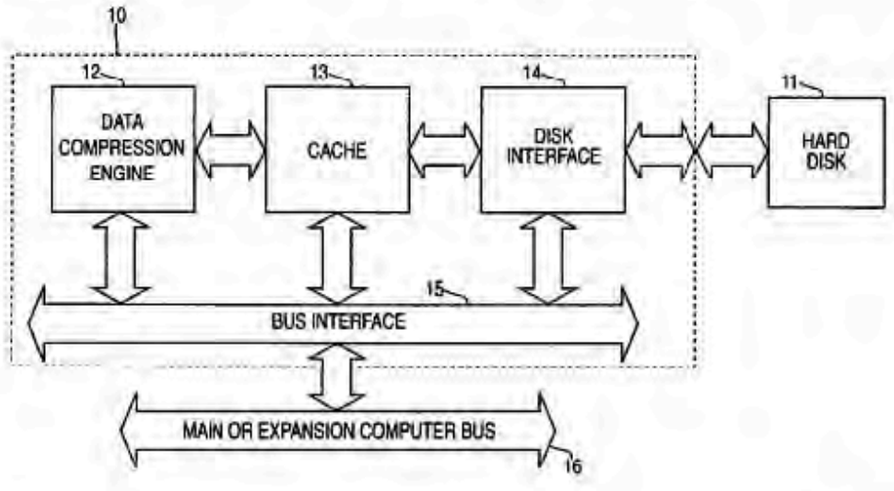
(Continued)

Primary Examiner—Thomas Lee
Assistant Examiner—Suresh K Suryawanshi
(74) *Attorney, Agent, or Firm*—Fish & Neave IP Group of Ropes & Gray LLP

(57) **ABSTRACT**

Systems and methods are provided for accelerated loading of operating system and application programs upon system boot or application launch. In one aspect, a method for providing accelerated loading of an operating system includes maintaining a list of boot data used for booting a computer system, preloading the boot data upon initialization of the computer system, and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. The boot data is retrieved from a boot device and stored in a cache memory device. The boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system.

31 Claims, 13 Drawing Sheets



APPLE 1001

set of decoders, or a sequential set of decoders corresponding to the extracted compression type descriptor. The decoders D1 . . . Dn may include those lossless encoding techniques currently well known within the art, including: run length, Huffman, Lempel-Ziv Dictionary Compression, arithmetic coding, data compaction, and data null suppression. Decoding techniques are selected based upon their ability to effectively decode the various different types of encoded input data generated by the data compression systems described above or originating from any other desired source.

As with the data compression systems discussed in U.S. Pat. No. 6,195,024, the decoder module **165** may include multiple decoders of the same type applied in parallel so as to reduce the data decoding time. An output data buffer or cache **170** may be included for buffering the decoded data block output from the decoder module **165**. The output buffer **70** then provides data to the output data stream. It is to be appreciated by those skilled in the art that the data compression system **180** may also include an input data counter and output data counter operatively coupled to the input and output, respectively, of the decoder module **165**. In this manner, the compressed and corresponding decompressed data block may be counted to ensure that sufficient decompression is obtained for the input data block.

Again, it is to be understood that the embodiment of the data decompression system **180** of FIG. **10** is exemplary of a preferred decompression system and method which may be implemented in the present invention, and that other data decompression systems and methods known to those skilled in the art may be employed for providing accelerated data retrieval in accordance with the teachings herein.

Although illustrative embodiments have been described herein with reference to the accompanying drawings, it is to be understood that the present invention is not limited to those precise embodiments, and that various other changes and modifications may be affected therein by one skilled in the art without departing from the scope or spirit of the invention. All such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.

What is claimed is:

1. A method for providing accelerated loading of an operating system, comprising the steps of:

maintaining a list of boot data used for booting a computer system;

initializing a central processing unit of the computer system;

preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and

servicing requests for boot data from the computer system using the preloaded boot data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache.

2. The method of claim **1**, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.

3. The method of claim **1**, wherein the preloading is performed by a data storage controller connected to the boot device.

4. The method of claim **1**, further comprising updating the list of boot data.

5. The method of claim **4**, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.

6. The method of claim **4**, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.

7. A system for providing accelerated loading of an operating system of a host system comprising:

a digital signal processor (DSP) or controller;

a cache memory device; and

a non-volatile memory device, for storing logic code associated with the DSP or controller, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system, for preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system, and for decompressing the preloaded compressed boot data, at a rate that increases the effective access rate of the cache, to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system.

8. The system of claim **7**, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.

9. The method of claim **1**, further comprising:

maintaining a list of application data associated with an application program;

preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device; and

servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.

10. The method of claim **1**, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data and the data compression engine provides the compressed boot data to the boot device.

11. The method of claim **1**, wherein the decompressing is provided by a data compression engine.

12. The method of claim **1**, further comprising a data compression engine for compressing, wherein the compressing provides the compressed boot data, the data compression engine provides the compressed boot data to the boot device, and the decompressing is provided by the data compression engine.

13. The method of claim **1**, wherein the compressed boot data is accessed via direct memory access.

14. The method of claim **1**, wherein Huffman encoding is utilized to provide the compressed boot data.

15. The method of claim **1**, wherein Lempel-Ziv encoding is utilized to provide the compressed boot data.

What is claimed is:

1. A method for providing accelerated loading of an operating system, comprising the steps of:

maintaining a list of boot data used for booting a computer system;

initializing a central processing unit of the computer system;

preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and

servicing requests for boot data from the computer system using the preloaded boot data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data at a rate that increases the effective access rate of the cache.

2. The method of claim **1**, wherein the boot data com-

servicin
syste
comp
unit c
comp
the c
tion c

10. The
compressi
ing provic
pression e
boot devic

11. The
provided l

12. The
compressi
ing provid
engine pro
and the de
engine.

quently compared. This tends to decrease data bandwidth from even that of a single comparable disk drive. In systems that offer hot swap capability, the failed drive is removed and a replacement drive is inserted. The data on the failed drive is then copied in the background while the entire system continues to operate in a performance degraded but fully operational mode. Once the data rebuild is complete, normal operation resumes. Hence, another problem with RAID systems is the high cost of increased reliability and associated decrease in performance.

RAID Level 5 employs disk data striping and parity error detection to increase both data bandwidth and reliability simultaneously. A minimum of three disk drives is required for this technique. In the event of a single disk drive failure, that drive may be rebuilt from parity and other data encoded on disk remaining disk drives. In systems that offer hot swap capability, the failed drive is removed and a replacement drive is inserted. The data on the failed drive is then rebuilt in the background while the entire system continues to operate in a performance degraded but fully operational mode. Once the data rebuild is complete, normal operation resumes.

Thus another problem with redundant modem mass storage devices is the degradation of data bandwidth when a storage device fails. Additional problems with bandwidth limitations and reliability similarly occur within the art by all other forms of sequential, pseudo-random, and random access mass storage devices. These and other limitations within the current art are addressed by the present invention.

SUMMARY OF THE INVENTION

The present invention is directed to systems and methods for providing accelerated loading of operating system and application programs upon system boot or application launch and, more particularly, to data storage controllers employing lossless and/or lossy data compression and decompression to provide accelerated loading of operating systems and application programs.

In one aspect of the present invention, a method for providing accelerated loading of an operating system comprises the steps of: maintaining a list of boot data used for booting a computer system; preloading the boot data upon initialization of the computer system; and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. In a preferred embodiment, the boot data is retrieved from a boot device and stored in a cache memory device.

In another aspect, the method for accelerated loading of an operating system comprises updating the list of boot data during the boot process. The step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list and/or removing from the list any boot data previously stored in the list and not requested by the computer system.

In yet another aspect, the boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system.

In another aspect, a method for providing accelerated launching of an application program comprises the steps of: maintaining a list of application data associated with an application program; preloading the application data upon launching the application program; and servicing requests for application data from a computer system using the preloaded application data.

In yet another aspect, a boot device controller for providing accelerated loading of an operating system of a host system comprises: a digital signal processor (DSP); a programmable logic device, wherein the programmable logic device is programmed by the digital signal processor to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system; and a non-volatile memory device, for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system, preloading the boot data upon initialization of the host system, and servicing requests for boot data from the host system using the preloaded boot data. The boot device controller further includes a cache memory device for storing the preloaded boot data.

The present invention is realized due to recent improvements in processing speed, inclusive of dedicated analog and digital hardware circuits, central processing units, (and any hybrid combinations thereof), that, coupled with advanced data compression and decompression algorithms are enabling of ultra high bandwidth data compression and decompression methods that enable improved data storage and retrieval bandwidth

These and other aspects, features and advantages, of the present invention will become apparent from the following detailed description of preferred embodiments that is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a data storage controller according to one embodiment of the present invention;

FIG. **2** is a block diagram of a data storage controller according to another embodiment of the present invention;

FIG. **3** is a block diagram of a data storage controller according to another embodiment of the present invention;

FIG. **4** is a block diagram of a data storage controller according to another embodiment of the present invention;

FIG. **5** is a block diagram of a data storage controller according to another embodiment of the present invention;

FIGS. **6***a* and **6***b* comprise a flow diagram of a method for initializing a data storage controller according to one aspect of the present invention;

FIGS. **7***a* and **7***b* comprise a flow diagram of a method for providing accelerated loading of an operating system and/or application programs upon system boot, according to one aspect of the present invention;

FIGS. **8***a* and **8***b* comprise a flow diagram of a method for providing accelerated loading of application programs according to one aspect of the present invention;

FIG. **9** is a diagram of an exemplary data compression system that may be employed in a data storage controller according to the present invention; and

FIG. **10** is a diagram of an exemplary data decompression system that may be employed in a data storage controller according to the present invention.

decompression to provide accelerated loading of operating systems and application programs.

In one aspect of the present invention, a method for providing accelerated loading of an operating system comprises the steps of: maintaining a list of boot data used for booting a computer system; preloading the boot data upon initialization of the computer system; and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. In a preferred embodiment, the boot data is retrieved from a boot device and stored in a cache memory device.

In another aspect, the method for accelerated loading of an operating system comprises updating the list of boot data

device is programmed by the digital signal processor to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system; and a non-volatile memory device, for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system, preloading the boot data upon initialization of the host system, and servicing requests for boot data from the host system using the preloaded boot data. The boot device controller further includes a cache memory device for storing the preloaded boot data.

The present invention is realized due to recent improve-

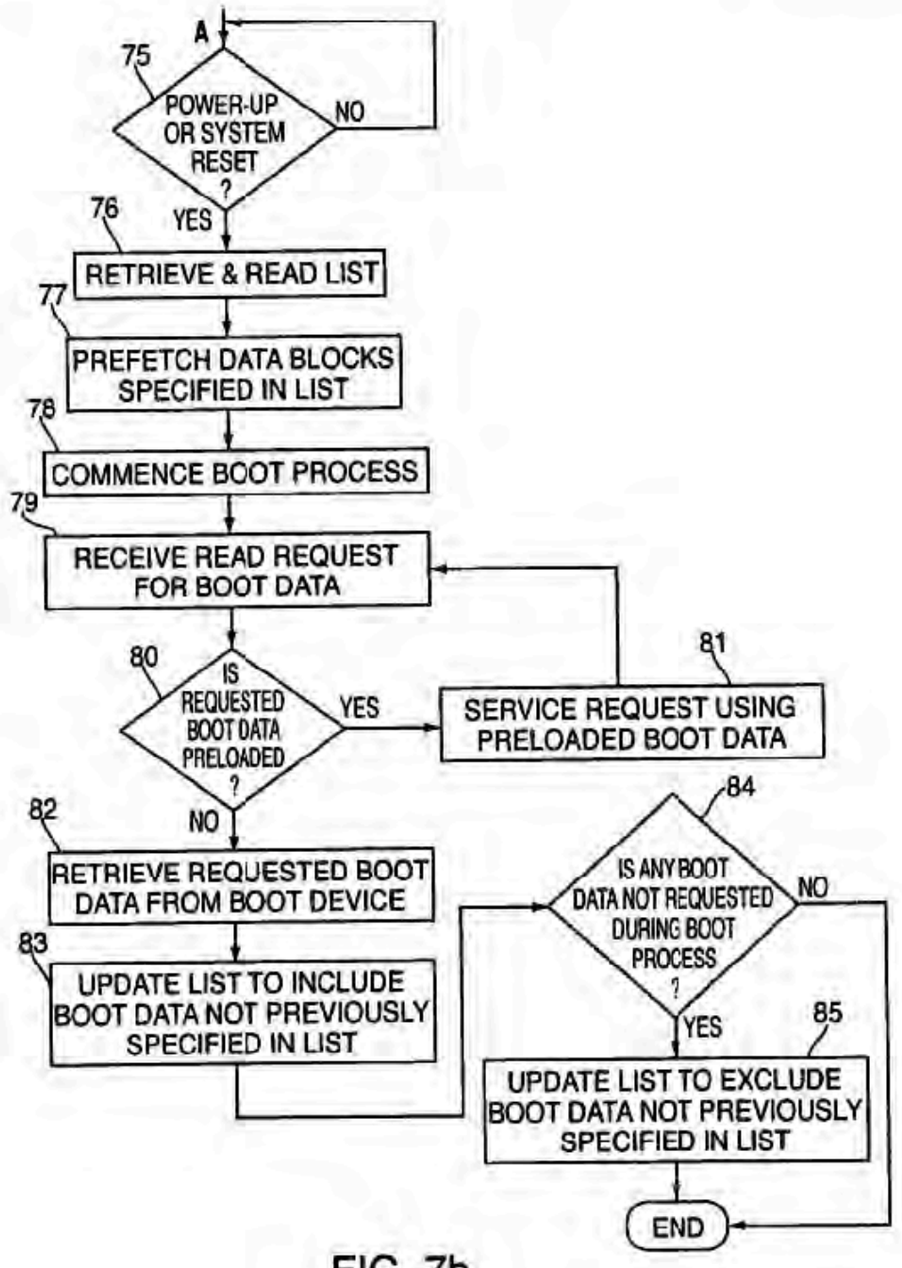
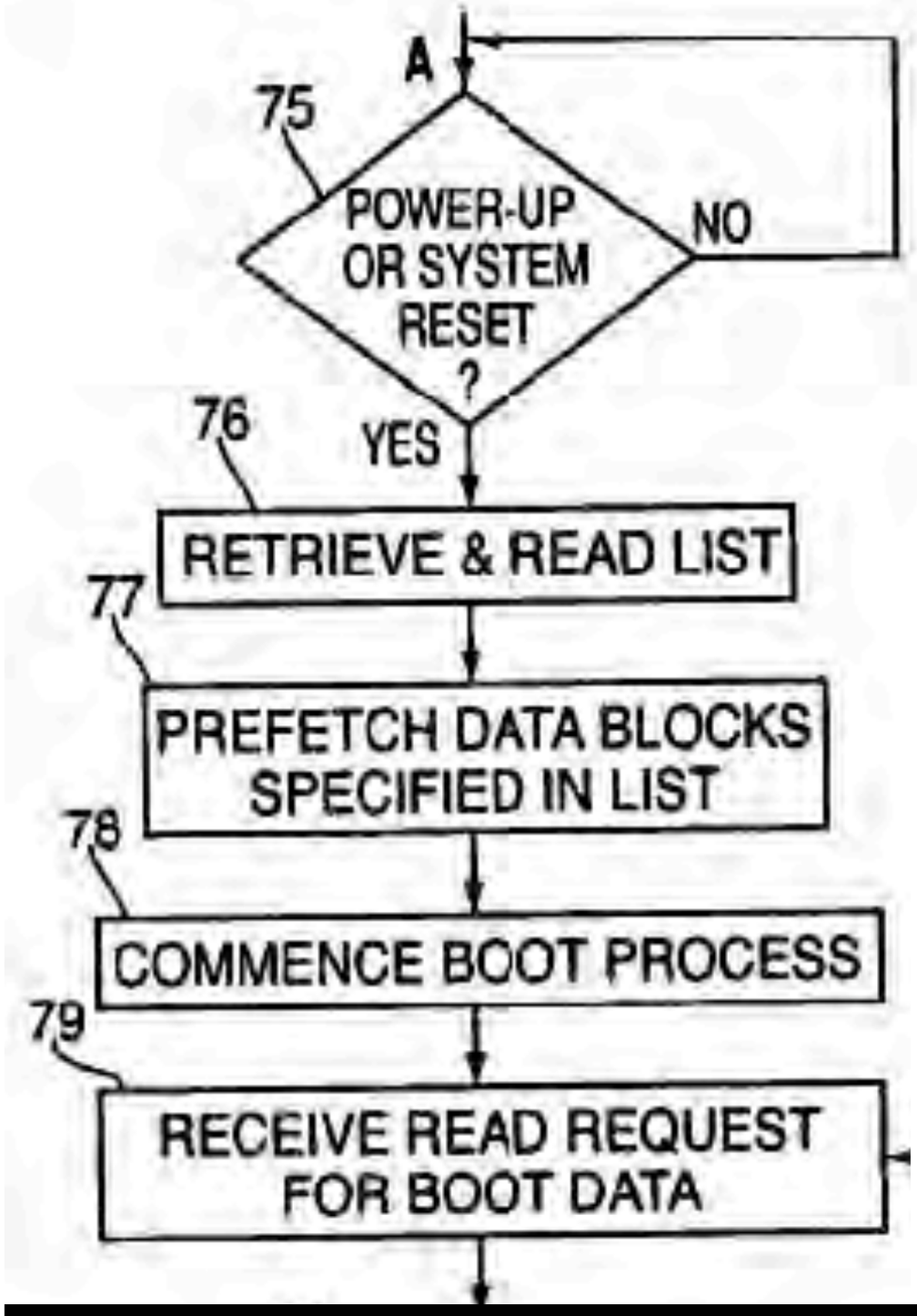

A
75
POWER-UP OR SYSTEM RESET ?
NO
YES
76
RETRIEVE & READ LIST
77
PREFETCH DATA BLOCKS SPECIFIED IN LIST
78
COMMENCE BOOT PROCESS
79
RECEIVE READ REQUEST FOR BOOT DATA
80
IS REQUESTED BOOT DATA PRELOADED ?
YES
81
SERVICE REQUEST USING PRELOADED BOOT DATA
NO
82
RETRIEVE REQUESTED BOOT DATA FROM BOOT DEVICE
83
UPDATE LIST TO INCLUDE BOOT DATA NOT PREVIOUSLY SPECIFIED IN LIST
84
IS ANY BOOT DATA NOT REQUESTED DURING BOOT PROCESS ?
NO
YES
85
UPDATE LIST TO EXCLUDE BOOT DATA NOT PREVIOUSLY SPECIFIED IN LIST
END

FIG. 7b



Then, upon each subsequent power-on/reset (affirmative result in step **75**), the data storage controller would retrieve and read the stored list (step **76**) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step **77**). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE, INC.,
Petitioner

v.

REALTIME DATA LLC,
Patent Owner

Case IPR2016-01365
Patent 7,181,608

EXPERT DECLARATION OF DR. GODMAR BACK IN SUPPORT OF THE PATENT OWNER'S RESPONSE

volt levels, converging platform clocks, and other tests. Specifically, claims 1 and 22 show that the "initializing" and "preloading prior to completion of initialization of the central processing unit" elements must be performed during the same initialization. A POSITA would have understood that the use of the definite article "the" in the term "the central processing unit" indicates that the term has an antecedent basis. The antecedent basis for "the central processing unit" can be found in "initializing *a central processing unit* of the computer system." As such, a POSITA would have understood that "preloading prior to completion of initialization of the central processing unit" finds its antecedent basis in the step "initializing a central processing unit." Therefore, "preloading prior to completion of *initialization* of the central processing unit" and "*initializing* a central processing unit" must occur during the *same* initialization process—not two or more separate processes.

77. Moreover, the "initialization" process referred to in "prior to completion of initialization of the central processing unit of the computer system" must be part of the same "initializing" process referred to in "initializing a central processing unit of the computer system" when read in the context of the claims. The only reasonable interpretation from the perspective of a POSITA is that the plain language of the "initializing" and "preloading…prior to completion of initialization" steps is that these steps must occur during the same initialization.

77. Moreover, the "initialization" process referred to in "prior to completion of initialization of the central processing unit of the computer system" must be part of the same "initializing" process referred to in "initializing a central processing unit of the computer system" when read in the context of the claims. The only reasonable interpretation from the perspective of a POSITA is that the plain language of the "initializing" and "preloading…prior to completion of initialization" steps is that these steps must occur during the same initialization. This is confirmed by the claims, which discuss only a single initialization identified in the claims.

78. This requirement is also confirmed by the specification of the '608

in one "turning-on of power of the host system 4." Then, when addressing Sukegawa's disclosure of "prior to completion of initialization of the central processing unit of the computer system," Dr. Neuhauser points to an initialization of the CPU in the "*next* turn-on of power and initialization of the CPU."[75]

83. A POSITA would have understood that each power-on cycle is associated with a different initialization. For example, Sukegawa would have an initialization that occurs every time the system is powered on. Dr. Neuhauser confirms this fact when explaining that a "POST" or "Power On Self Test" is performed as part of the initialization process.[76] Therefore, the initialization that occurs in the first "turning-on of power" when "preloading," (which is simply storage as explained in Section VII.A.) cannot be the same initialization that occurs in a subsequent "turning-on of power" for the claimed "completion of initialization."

84. Further, Dr. Neuhauser's interpretation of the "preloading…prior to completion of initialization of the central processing unit" element effectively strips all meaning out of the phrase "completion of." As explained above, Dr. Neuhauser alleges that Sukegawa teaches preloading "prior to completion of initialization" because in Sukegawa, storing of control information "has been

[75] Ex. 1003, Neuhauser Dec. at ¶ 115; *see also* Neuhauser Dec. at ¶¶ 225, 238.

[76] Ex. 1003, Neuhauser Dec. at ¶ 106.

of the CPU in the "*next* turn-on of power and initialization of the CPU."[75]

83. A POSITA would have understood that each power-on cycle is associated with a different initialization. For example, Sukegawa would have an initialization that occurs every time the system is powered on. Dr. Neuhauser confirms this fact when explaining that a "POST" or "Power On Self Test" is performed as part of the initialization process.[76] Therefore, the initialization that occurs in the first "turning-on of power" when "preloading," (which is simply storage as explained in Section VII.A.) cannot be the same initialization that occurs in a subsequent "turning-on of power" for the claimed "completion of initialization."

United States Patent [19]

Sukegawa

[11] **Patent Number: 5,860,083**

[45] **Date of Patent: Jan. 12, 1999**

[54] **DATA STORAGE SYSTEM HAVING FLASH MEMORY AND DISK DRIVE**

[75] Inventor: **Hiroshi Sukegawa**, Tokyo, Japan

[73] Assignee: **Kabushiki Kaisha Toshiba**, Kawasaki, Japan

[21] Appl. No.: **818,983**

[22] Filed: **Mar. 14, 1997**

[30] **Foreign Application Priority Data**

Nov. 26, 1996 [JP] Japan 8-314850

[51] **Int. Cl.**[6] **G06F 12/08**

[52] **U.S. Cl.** **711/103**; 711/117; 711/171; 711/173

[58] **Field of Search** 711/103, 113, 711/117, 170, 171, 173

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,175,842	12/1992	Totani	711/161
5,371,876	12/1994	Ewertz et al.	711/159
5,437,018	7/1995	Kobayashi et al.	395/652
5,535,357	7/1996	Moran et al.	711/103
5,644,539	7/1997	Yamagami et al.	365/200
5,701,492	12/1997	Wadsworth et al.	395/712
5,745,418	4/1998	Ma et al.	365/185.33
5,778,418	7/1998	Auclair et al.	711/101

FOREIGN PATENT DOCUMENTS

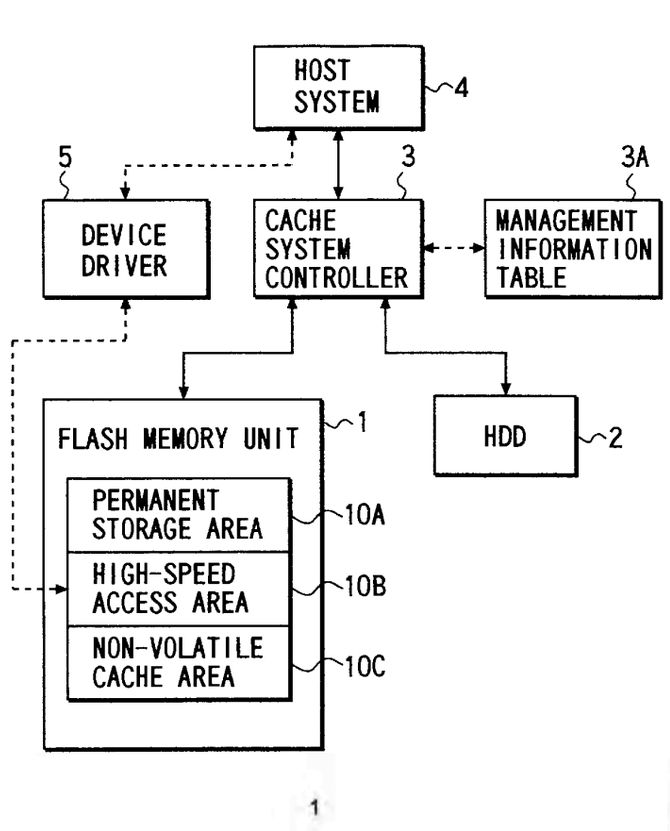
0 706 135	4/1996	European Pat. Off. .
5-11933	1/1993	Japan .
8-63395	3/1996	Japan .
8-115241	5/1996	Japan .

Primary Examiner—Tod R. Swann
Assistant Examiner—Conley B. King, Jr.
Attorney, Agent, or Firm—Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P.

[57] **ABSTRACT**

In a data storage system using a flash memory unit and an HDD, the storage area of the flash memory unit is logically divided into a permanent storage area, a non-volatile cache area, which are used as cache memory areas of the HDD, and a high-speed access area. These divided areas are individually managed. The permanent storage area stores data which is used frequently for a relatively long time period. The non-volatile cache area is used as an ordinary cache memory area in which data, which is updated relatively frequently, is stored. The high-speed access area is a storage area to be used by, e.g. an operating system (OS) of a host system. For example, a swap file, which needs to be accessed at high speed, is shifted into the high-speed access area.

28 Claims, 10 Drawing Sheets



1

APPLE 1005

In the present invention, it is assumed that the entire storage area of the flash memory unit **1** is logically divided into permanent storage area **10**A, high-speed access area **10**B and non-volatile cache area **10**C and the divided areas are managed. The controller **3** manages the storage areas **10**A to **10**C of the flash memory unit **1** by using a management information table **3**A. The management information table **3**A is stored, for example, in the non-volatile cache area **10**C of flash memory unit **1**.

The first embodiment relates to a system wherein the permanent storage area **10**A of flash memory unit **1** is used as a cache memory area. In this embodiment, it is supposed that the user desires to start a frequently used application program (AP) at high speed at all times.

The user starts a data storage utility program of the cache system controller **3** via a user interface provided in the host system **4** (step S**1**). The data storage utility program reads specified data from the HDD **2** and stores the read data in a specified storage area in the flash memory unit **1**. In this case, it is assumed that the user sets the permanent storage area **10**A in the flash memory unit **1** as the data storage area, at the time of instructing the start of the data storage utility program (step S**2**).

Then, the user instructs the host system **4** to start the AP (step S**3**). The host system **4**, upon receiving the AP start instruction, issues a read command to the controller **3** in order to read control information necessary for the start of the AP from the HDD **2**.

The controller **3** controls the HDD **2**, reads out the control information necessary for the start of the AP and transfers the read-out control information to the host system **4** (step S**4**). At this time, according to the started-up data storage utility program, the controller **3** stores the AP control information read out from the HDD **2** in the permanent storage area **10**A of flash memory unit **1** (step S**5**). When the AP has been prepared to start, the data storage utility program is stopped by the instruction from the user ("YES" in step S**6**; step S**7**). Through these operations, the control information necessary for starting the AP is stored in the permanent storage area **10**A in the flash memory unit **1**.

In this case, the control information is stored in the permanent storage area **10**A in flash memory unit **1** under the file name designated by the user. Information for correlating the file name and the AP and information of other comment is recorded on the management information table **3**A by the data storage utility program. The user inputs the file name to the controller **3** via the user interface, thereby referring to the file (the control information of the AP in this case) stored in the permanent storage area **10**A. The user can delete the file, if unnecessary. In other words, the control information necessary for starting the AP is permanently stored in the permanent storage area **10**A in the flash memory unit **1** as one file, until the user instructs the deletion of the file.

If the user instructs the start of the same AP via the user interface, the host system **4** issues the read command, as described above, to read from the HDD **2** the control information necessary for starting the AP ("YES" in step S**8**). Upon receiving the read command, the controller **3** determines whether the control information to be accessed is stored in the flash memory unit **1** by using the management information table **3**A. Since the AP control information is stored in the permanent storage area **10**A, the cache memory area is successfully accessed. Accordingly, the controller **3** reads out the AP control information from the permanent storage area **10**A of flash memory unit **1**, without accessing the HDD **2**, and transfers the read-out control information to the host system **4** (step S**9**). The host system **4** starts the AP designated by the user on the basis of the transferred AP control information (step S**10**).

By the above-described cache system, the control information of the frequently used AP designated by the user is read out from the HDD **2** and stored in the permanent storage area **10**A in the flash memory unit **1** used as the cache memory area. Accordingly, when the AP is to be started next time, the AP control information can be read out quickly from the permanent storage area **10**A used as cache memory area, and not from the HDD **2**. Thereby, the host system **4** can quickly acquire the control information at the time of starting the AP. As a result, the AP can be quickly started. Since the AP control information is permanently stored in the permanent storage area **10**A until the user instructs the deletion of the control information, as described above, the frequently used AP can be started quickly at all times.

(First Modification of the First Embodiment)

FIG. **4** is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area **10**A of flash memory unit **1**, for example, when the OS of the host system **4** is started in a series of operations from the turn-on of power to the completion of the starting operation.

When the system is switched on and the user sets the data storage mode via the user interface, the controller **3** stores the information representing the data storage mode in the permanent storage area **10**A in flash memory unit **1** (steps S**11** to S**13**).

After the power is turned off and then turned on again, the controller **3** starts the above-mentioned data storage utility program on the basis of the information of the set data storage mode ("YES" in step S**14**; step S**15**). According to the data storage utility program, the control information, which is pre-stored in the HDD **2** and necessary for starting the OS, is read out and stored in the permanent storage area **10**A (steps S**16** and S**17**). The controller **3** transfers to the host system **4** the control information necessary for starting the OS read out from the HDD **2**. Based on the control information, the host system **4** starts the OS. After the preparation for starting the OS is completed, the data storage utility program is stopped ("YES" in step S**18**; step S**19**).

According to this system, when the OS is automatically started by the control information read out from the HDD **2** at the time of turning-on of power, the control information is stored in the permanent storage area **10**A used as the cache memory area for the HDD **2**. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD **2** but from the permanent storage area **10**A or cache memory area, and the read-out control information is transferred to the host system **4**. Thus, the control information can be accessed from the permanent storage area **10**A in the flash memory unit **1** having a higher access speed than the HDD **2**. As a result, the OS can be started at higher speed.

Like the AP control information, the OS control information may be stored as one file in the permanent storage area **10**A. Thereby, the user can refer to, or delete, the OS control information on an as-needed basis. For example, at the time of shipment of the personal computer, if the OS is pre-installed in the flash memory unit **1** functioning as cache memory area, the user can delete the OS control information based on the user's judgment. Specifically, the OS control information may be deleted by a user who is used to starting

preparation for starting the OS is completed, the data storage utility program is stopped ("YES" in step **S18**; step **S19**).

According to this system, when the OS is automatically started by the control information read out from the HDD **2** at the time of turning-on of power, the control information is stored in the permanent storage area **10**A used as the cache memory area for the HDD **2**. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD **2** but from the permanent storage area **10**A or cache memory area, and the read-out control information is transferred to the host system **4**. Thus, the control information can be accessed from the permanent storage area **10**A in the flash memory unit **1** having a higher access speed than the HDD **2**. As a result, the OS can be started at higher speed.

Like the AP control information, the OS control informa-

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of:	James J. Fallon, et al.	
U.S. Patent No.:	7,181,608	Attorney Docket No.: 39521-0023IP1
Issue Date:	February 20, 2007	Control No. IPR2016-01365
Appl. Serial No.:	09/776,267	
Filing Date:	February 2, 2001	
Title:	SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS	

Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

PETITION FOR *INTER PARTES* REVIEW OF UNITED STATES PATENT NO. 7,181,608 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42

OS based on control information that is transferred to it "from the permanent storage area 10A." Id., 6:19-58. A POSITA would have understood that, for the host computer system to start the OS using this transferred control information, its CPU would first have to be initialized. Dec., ¶¶100, 106-107 (citing Collins, 6:24-35, 6:47-52, 6:61-7:19, 7:56-65, 8:60-65). Thus, Sukegawa renders obvious initialization of the host computer system's CPU. Dec., ¶107.

1.3: preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system

As explained in Section IV.B., Sukegawa's system features a flash memory unit 1, a hard disk drive ("HDD") 2, and a controller 3 that "performs data input/output control (including cache operation control) for the flash memory unit 1 and HDD." Sukegawa, 4:1-21, 5:1-6:17, 6:18-7:2. The Sukegawa controller 3 performs "a cache function" by "using the flash memory unit 1 as a cache memory." Id., 4:7-10; Dec., ¶109.

Also, as explained in Section IV.B., Sukegawa describes two techniques for preloading boot data into this cache memory: (1) user selection of data to preload, and (2) automatic selection of data to preload. See Sukegawa, 5:14-6:17, 6:19-58, 7:28-55.

For the first preloading technique, Sukegawa explains that a "data storage utility program of the cache system controller 3" reads user-specified control information (boot data as described at 1.1) out of HDD2 and preloads it "in the permanent storage area 10A" of flash memory unit 1, which has "a higher access speed than the HDD 2." Id., 5:10-40, 6:19-58.

For the second preloading technique, Sukegawa describes automated preloading of control information in flash memory unit 1's non-volatile cache area 10C, which is "used independently by controller 3," rather than being "used by the user's intent." Id., 7:28-55. If controller 3 determines, in response to a read command issued by the host system 4 to HDD2, that the data to be accessed is not already present in areas 10A or 10C, and is not data designated by the user, "the controller 3 stores the data in the non-volatile cache area 10C ...." Id., 7:40-55. By preloading control information into areas 10A and 10C of flash memory unit 1, Sukegawa discloses preloading boot data into a cache memory. Dec., ¶¶111-114.

This preloading of boot data into flash memory unit 1's areas 10A and 10C occurs prior to the next turning-on of power of the host system 4 and, therefore, prior to an initialization of the host computer system's CPU. Sukegawa, 5:10-12, 6:19-26, 6:45-54; Dec., ¶¶111-114.

For the preloading of control information into area 10A, "the [host computer] system is switched on and the user sets the data storage mode via the user

preloading control information into areas 10A and 10C of flash memory unit 1, Sukegawa discloses preloading boot data into a cache memory. Dec., ¶¶111-114.

This preloading of boot data into flash memory unit 1's areas 10A and 10C occurs prior to the next turning-on of power of the host system 4 and, therefore, prior to an initialization of the host computer system's CPU. Sukegawa, 5:10-12, 6:19-26, 6:45-54; Dec., ¶¶111-114.

For the preloading of control information into area 10A, "the [host computer] system is switched on and the user sets the data storage mode via the user

‘936 Patent’s Claim 18

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

(10) **Patent No.: US 8,090,936 B2**
(45) **Date of Patent: *Jan. 3, 2012**

(54) **SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS**

(75) Inventors: **James J. Fallon**, Armonk, NY (US); **John Buck**, Oceanside, NY (US); **Paul F. Pickel**, Bethpage, NY (US); **Stephen J. McErlain**, New York, NY (US)

(73) Assignee: **Realtime Data, LLC**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 286 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/551,204**

(22) Filed: **Oct. 19, 2006**

(65) **Prior Publication Data**

US 2007/0083746 A1 Apr. 12, 2007

Related U.S. Application Data

(63) Continuation of application No. 09/776,267, filed on Feb. 2, 2001, now Pat. No. 7,181,608.

(60) Provisional application No. 60/180,114, filed on Feb. 3, 2000.

(51) **Int. Cl.**

G06F 9/00	(2006.01)
G06F 9/24	(2006.01)
G06F 13/28	(2006.01)

(52) **U.S. Cl.** **713/2**; 713/1; 711/113

(58) **Field of Classification Search** 713/2

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,394,352	A	7/1968	Wernikoff et al.
3,490,690	A	1/1970	Apple et al.
4,021,782	A	5/1977	Hoerning
4,032,893	A	6/1977	Moran
4,054,951	A	10/1977	Jackson et al.
4,127,518	A	11/1978	Coy et al.
4,302,775	A	11/1981	Widergren et al.

(Continued)

FOREIGN PATENT DOCUMENTS

DE	4127518	2/1992

(Continued)

OTHER PUBLICATIONS

Millman, Howard, "Image and video compression", Computerworld, vol. 33, Issue No. 3, Jan. 18, 1999, pp. 78.

(Continued)

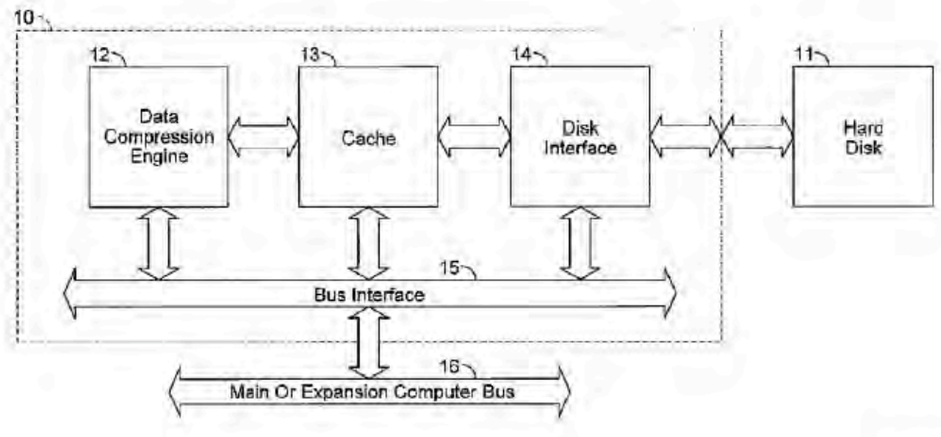
Primary Examiner — Suresh K Suryawanshi

(74) *Attorney, Agent, or Firm* — Sterne, Kessler, Goldstein & Fox P.L.L.C.

(57) **ABSTRACT**

Systems and methods are disclosed for providing accelerated loading of operating system and application programs. In one aspect, a method for providing accelerated loading of an operating system comprises the steps of: maintaining a list of boot data; preloading the boot data upon initialization of the computer system; and servicing requests for boot data from the computer system using the preloaded boot data. In a preferred embodiment, the boot data is retrieved from a boot device and stored in a cache memory device. In another aspect, a method for accelerated loading of an operating system comprises updating the list of boot data during the boot process, wherein updating comprises adding to the list any boot data requested by the computer system not previously stored in the list and/or removing from the list any boot data previously stored in the list and not requested by the computer system.

24 Claims, 13 Drawing Sheets



APPLE 1001

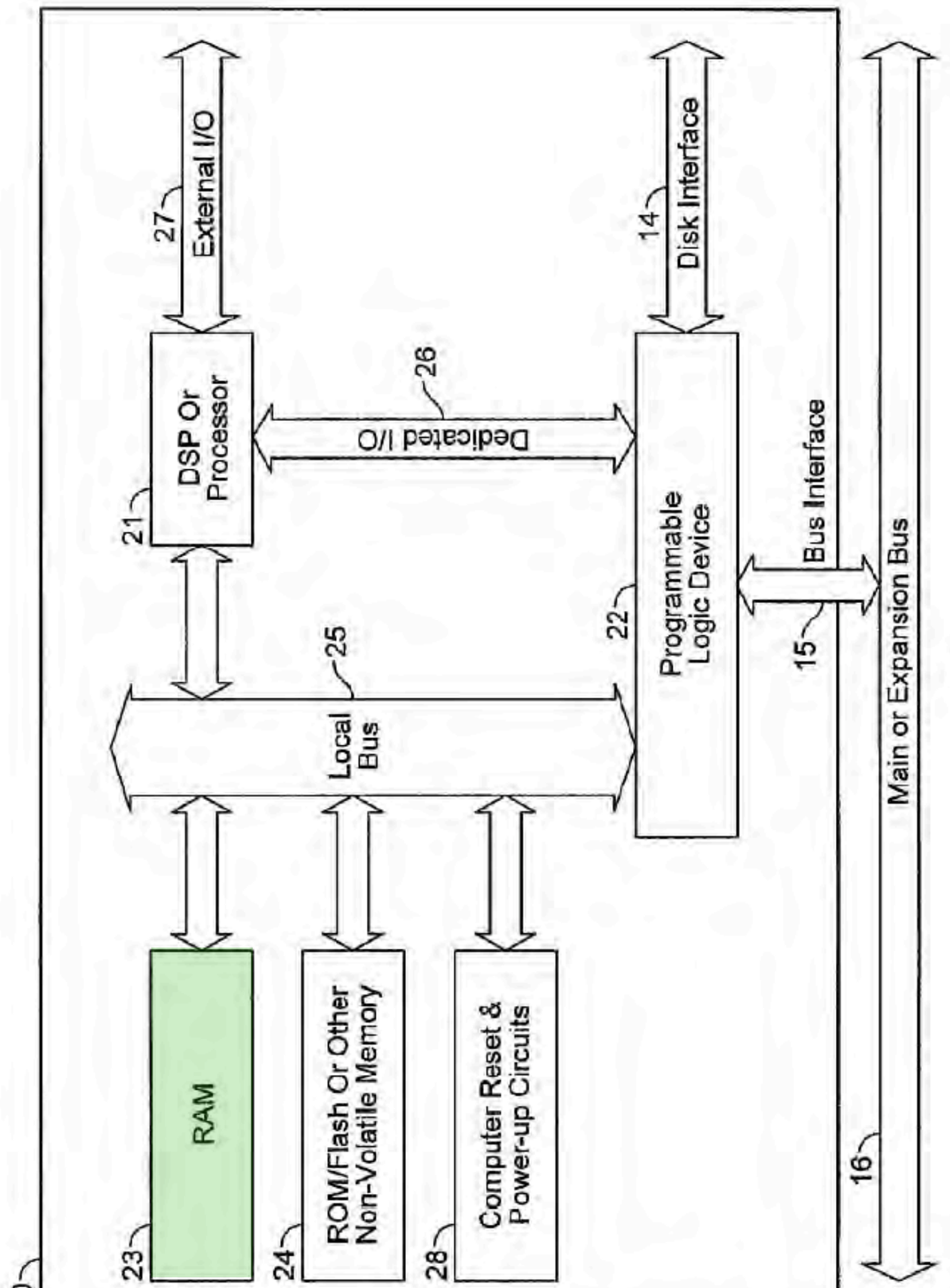


FIG. 2

V. Instant Boot Device for Operation System, Application Program and Loading

Typically, with conventional boot device controllers, after reset, the boot device controller will wait for a command over the computer bus (such as PCI). Since the boot device controller will typically be reset prior to bus reset and before the computer bus starts sending commands, this wait period is unproductive time. The initial bus commands inevitably instruct the boot device controller to retrieve data from the boot device (such as a disk) for the operating system. Since most boot devices are relatively slow compared to the speed of most computer busses, a long delay is seen by the computer user. This is evident in the time it takes for a typical computer to boot.

It is to be appreciated that a data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the computer system boot time. Upon host system power-up or reset, the data storage controller will perform a self-diagnostic and program the programmable logic device (as discussed above) prior to completion of the host system reset (e.g., PCI bus reset) so that the logic device can accept PCI Bus commands after system reset. Further, prior to host system reset, the data storage controller can proceed to pre-load the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory. The data storage controller preloads the needed sectors of data in the order in which they will be needed. Since the same portions of the operating system must be loaded upon each boot process, it is advantageous for the boot device controller to preload such portions and not wait until it is commanded to load the operating system. Preferably, the data storage controller employs a dedicated IO channel of the DSP (with or without data compression) to pre-load computer operating systems and applications.

Once the data is preloaded, when the computer system bus issues its first read commands to the data storage controller seeking operating system data, the data will already be available in the cache memory of the data storage controller. The data storage controller will then be able to instantly start transmitting the data to the system bus. Before transmission to the bus, if the was stored in compressed format on the boot device, the data will be decompressed. The process of preloading required (compressed) portions of the operating system significantly reduces the computer boot process time.

In addition to preloading operating system data, the data storage controller could also preload other data that the user would likely want to use at startup. An example of this would be a frequently used application such as a word processor and any number of document files.

There are several techniques that may be employed in accordance with the present invention that would allow the data storage controller to know what data to preload from the boot device. One technique utilizes a custom utility program that would allow the user to specify what applications/data should be preloaded.

Another technique (illustrated by the flow diagram of FIGS. **7***a* and **7***b*) that may be employed comprises an automatic process that requires no input from the user. With this technique, the data storage controller maintain a list comprising the data associated with the first series of data requests received by the data storage controller by the host system after a power-on/reset. In particular, referring to FIG. **7***a*, during the computer boot process, the data storage controller will receive requests for the boot data (step **70**). In response, the data storage controller will retrieve the requested boot data from the boot device (e.g., hard disk) in the local cache memory (step **71**). For each requested data block, the data storage controller will record the requested data block number in a list (step **72**). The data storage controller will record the data block number of each data block requested by the host computer during the boot process (repeat steps **70**-**72**). When the boot process is complete (affirmative determination in step **73**), the data storage controller will store the data list on the boot device (or other storage device) (step **74**).

Then, upon each subsequent power-on/reset (affirmative result in step **75**), the data storage controller would retrieve and read the stored list (step **76**) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step **77**). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).

When the boot process begins (step **78**) (i.e., the storage controller is initialized and the system bus reset is deasserted), the data storage controller will receive requests for boot data (step **79**). If the host computer issues a request for boot data that is pre-loaded in the local memory of the data storage controller (affirmative result in step **80**), the request is immediately serviced using the preloaded boot data (step **81**). If the host computer issues a request for boot data that is not preloaded in the local memory of the data storage controller (negative determination in step **80**), the controller will retrieve the requested data from the boot device, store the data in the local memory, and then deliver the requested boot data to the computer bus (step **82**). In addition, the data storage controller would update the boot data list by recording any changes in the actual data requests as compared to the expected data requests already stored in the list (step **83**). Then, upon the next boot sequence, the boot device controller would pre-load that data into the local cache memory along with the other boot data previously on the list.

Further, during the boot process, if no request is made by the host computer for a data block that was pre-loaded into the local memory of the data storage controller (affirmative result in step **84**), then the boot data list will be updated by removing the non-requested data block from the list (step **85**). Thereafter, upon the next boot sequence, the data storage controller will not pre-load that data into local memory.

VI. Quick Launch for Operating System, Application Program, and Loading

It is to be appreciated that the data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the time to load application programs (referred to as "quick launch"). Conventionally, when a user launches an application, the file system reads the first few blocks of the file off the disk, and then the portion of the loaded software will request via the file system what additional data it needs from the disk. For example, a user may open a spreadsheet program, and the program may be configured to always load a company spreadsheet each time the program is started. In addition, the company spreadsheet may require data from other spreadsheet files.

In accordance with the present invention, the data storage controller may be configured to "remember" what data is typically loaded following the launch of the spreadsheet program, for example. The data storage controller may then proceed to preload the company spreadsheet and all the necessary data in the order is which such data is needed. Once this is accomplished, the data storage controller can service read commands using the preloaded data. Before transmission to the bus, if the preloaded data was stored in compressed for-

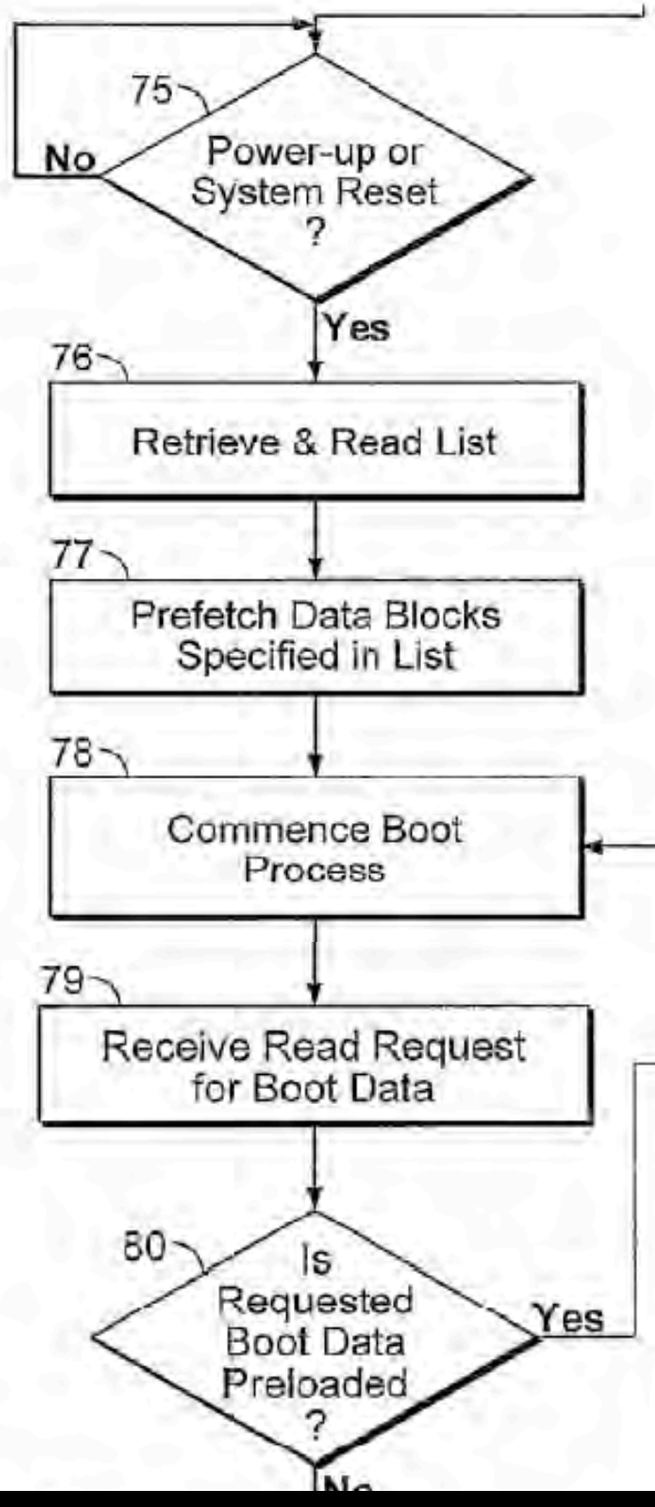
to boot.

It is to be appreciated that a data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the computer system boot time. Upon host system power-up or reset, the data storage controller will perform a self-diagnostic and program the programmable logic device (as discussed above) prior to completion of the host system reset (e.g., PCI bus reset) so that the logic device can accept PCI Bus commands after system reset. Further, prior to host system reset, the data storage controller can proceed to pre-load the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory. The data storage controller preloads the needed sectors of data in the order in which they will be needed. Since the same portions of the operating system must be loaded upon each boot process, it is advantageous for the boot device controller to preload such portions and not wait until it is commanded to load the operating system. Preferably, the data storage controller employs a dedicated IO channel of the DSP (with or without data compression) to pre-load computer operating systems and applications.

Once the data is preloaded, when the computer system bus issues its first read commands to the data storage controller

of the given syst
cess may be con
process, or conti
case booting and

When the boo
controller is init
serted), the data
boot data (step 7
boot data that is
storage controlle
immediately serv
If the host compu
preloaded in the
(negative determ
retrieve the reque
in the local mem
to the computer
controller would
changes in the
expected data re
Then, upon the n
would pre-load t
with the other bo



Then, upon each subsequent power-on/reset (affirmative result in step 75), the data storage controller would retrieve and read the stored list (step 76) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step 77). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed simultaneously).

such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.

What is claimed is:

1. A method comprising:

maintaining a list of boot data used for booting a computer system, wherein at least a portion of said boot data is compressed by a data compression engine to provide said at least a portion of said boot data in compressed form, and stored in compressed form on a boot device;

initializing a central processing unit of said computer system;

preloading said at least a portion of said boot data in compressed form from said boot device to a memory;

accessing and decompressing said at least a portion of said boot data in said compressed form from said memory; and

utilizing said decompressed at least a portion of said boot data to boot said computer system, wherein said at least a portion of said boot data is decompressed by said data compression engine.

2. The method of claim **1**, wherein said decompressed at least a portion of said boot data comprises program code associated with an operating system of said computer system.

3. The method of claim **1**, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program of said computer system.

4. The method of claim **1**, wherein said decompressed at least a portion of said boot data comprises program code associated with an application program and an operating system of said computer system.

5. The method of claim **1**, wherein said preloading is performed by a data storage controller connected to said boot device.

6. The method of claim **1**, further comprising updating the list of boot data.

7. The method of claim **1**, wherein Huffman encoding is utilized to provide said at least a portion of said boot data in said compressed form.

8. The method of claim **1**, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in said compressed form.

9. The method of claim **1**, wherein a plurality of encoders are utilized to provide said at least a portion of compressed data in compressed form.

10. The method of claim **1**, wherein a plurality of encoders in a parallel configuration are utilized to provide said at least a portion of said data in compressed form.

11. A system comprising:

a processor;

a memory; and

a non-volatile memory device for storing logic code associated with the processor, wherein said logic code comprises instructions executable by the processor for maintaining a list of boot data used for booting the host system, at least a portion of said boot data is stored in compressed form in said non-volatile memory device, said at least a portion of said boot data in compressed form is preloaded into said memory, and said preloaded at least a portion of boot data in compressed form is decompressed and utilized to boot said computer system; and

a data compression engine for providing said at least a portion of said boot data in compressed form by compressing said at least a portion of said boot data and decompressing said at least a portion of said boot data in compressed form to provide said decompressed at least a portion of boot data.

12. The system of claim **11**, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program.

13. The system of claim **11**, wherein said logic code further comprises program instructions executable by said processor for maintaining a list of application data associated with an application program, and wherein said application data is preloaded upon launching the application program and utilized by said computer system.

14. The system of claim **11**, wherein Huffman encoding is utilized to provide said at least a portion of said boot data in compressed form.

15. The system of claim **11**, wherein Lempel-Ziv encoding is utilized to provide said at least a portion of said boot data in compressed form.

16. The system of claim **11**, wherein a plurality of encoders are utilized to provide said at least a portion of said boot data in compressed form.

17. The system of claim **11**, wherein a plurality of encoders in a parallel configuration are utilized to provide said at least a portion of said boot data in compressed form.

18. A method of preloading an operating system for booting a computer system comprising:

storing substantially all of the operating system in compressed form on a boot device;

preloading a first portion of the substantially all of the operating system from said boot device to a memory;

accessing and decompressing the first portion from the memory using a data compression engine;

utilizing the decompressed first portion to partially boot said computer system;

responsive to a request, locating a second portion of the substantially all of the operating system using a boot data list and preloading the second portion from the boot device to the memory;

accessing and decompressing the second portion from the memory using the data compression engine; and

utilizing the decompressed second portion to further partially boot said computer system.

19. The method of claim **18**, wherein the preloading is performed by a data storage controller connected to the boot device.

20. The method of claim **18**, further comprising updating the boot data list.

21. The method of claim **18**, wherein Huffman encoding is utilized to obtain the substantially all of the operating system in compressed form.

22. The method of claim **18**, wherein Lempel-Ziv encoding is utilized to obtain the substantially all of the operating system in compressed form.

23. The method of claim **18**, wherein a plurality of encoders are utilized to obtain the substantially all of the operating system in compressed form.

24. The method of claim **18**, wherein a plurality of encoders in a parallel configuration are utilized to obtain the substantially all of the operating system in compressed form.

* * * * *

rogram code
ıid computer

ɔmpressed at
rogram code
perating sys-

ɔading is per-
to said boot

updating the

ı encoding is
l boot data in

v encoding is
l boot data in

/ of encoders
: compressed

in a parallel configuration are utilized to provide said at least a portion of said boot data in compressed form.

18. A method of preloading an operating system for booting a computer system comprising:

storing substantially all of the operating system in compressed form on a boot device;

preloading a first portion of the substantially all of the operating system from said boot device to a memory;

accessing and decompressing the first portion from the memory using a data compression engine;

utilizing the decompressed first portion to partially boot said computer system;

responsive to a request, locating a second portion of the substantially all of the operating system using a boot data list and preloading the second portion from the boot device to the memory;

accessing and decompressing the second portion from the memory using the data compression engine; and

utilizing the decompressed second portion to further partially boot said computer system.

19. The method of claim **18**, wherein the preloading is

United States Patent [19]

Sukegawa

[11] **Patent Number: 5,860,083**

[45] **Date of Patent: Jan. 12, 1999**

[54] **DATA STORAGE SYSTEM HAVING FLASH MEMORY AND DISK DRIVE**

[75] Inventor: **Hiroshi Sukegawa**, Tokyo, Japan

[73] Assignee: **Kabushiki Kaisha Toshiba**, Kawasaki, Japan

[21] Appl. No.: **818,983**

[22] Filed: **Mar. 14, 1997**

[30] **Foreign Application Priority Data**

Nov. 26, 1996 [JP] Japan 8-314850

[51] **Int. Cl.**[6] **G06F 12/08**

[52] **U.S. Cl.** **711/103**; 711/117; 711/171; 711/173

[58] **Field of Search** 711/103, 113, 711/117, 170, 171, 173

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,175,842	12/1992	Totani	711/161
5,371,876	12/1994	Ewertz et al.	711/159
5,437,018	7/1995	Kobayashi et al.	395/652
5,535,357	7/1996	Moran et al.	711/103
5,644,539	7/1997	Yamagami et al.	365/200
5,701,492	12/1997	Wadsworth et al.	395/712
5,745,418	4/1998	Ma et al.	365/185.33
5,778,418	7/1998	Auclair et al.	711/101

FOREIGN PATENT DOCUMENTS

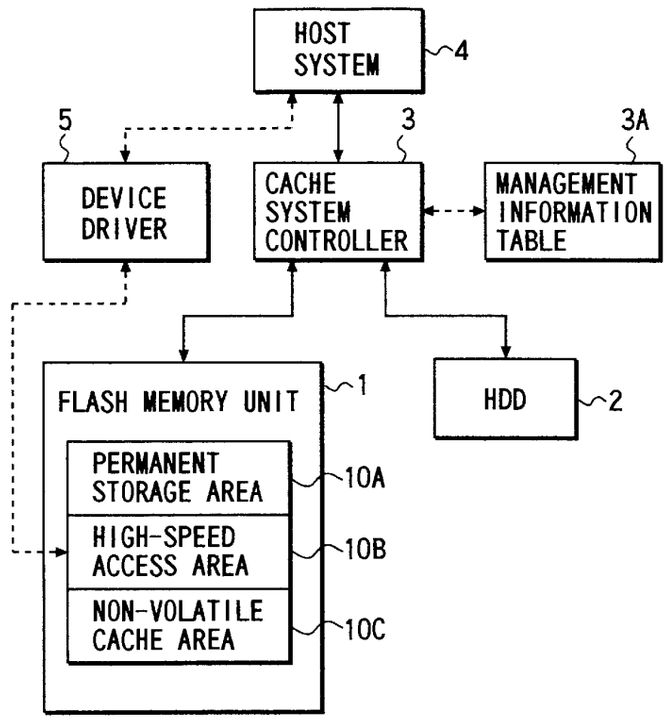
0 706 135	4/1996	European Pat. Off. .
5-11933	1/1993	Japan .
8-63395	3/1996	Japan .
8-115241	5/1996	Japan .

Primary Examiner—Tod R. Swann
Assistant Examiner—Conley B. King, Jr.
Attorney, Agent, or Firm—Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P.

[57] **ABSTRACT**

In a data storage system using a flash memory unit and an HDD, the storage area of the flash memory unit is logically divided into a permanent storage area, a non-volatile cache area, which are used as cache memory areas of the HDD, and a high-speed access area. These divided areas are individually managed. The permanent storage area stores data which is used frequently for a relatively long time period. The non-volatile cache area is used as an ordinary cache memory area in which data, which is updated relatively frequently, is stored. The high-speed access area is a storage area to be used by, e.g. an operating system (OS) of a host system. For example, a swap file, which needs to be accessed at high speed, is shifted into the high-speed access area.

28 Claims, 10 Drawing Sheets



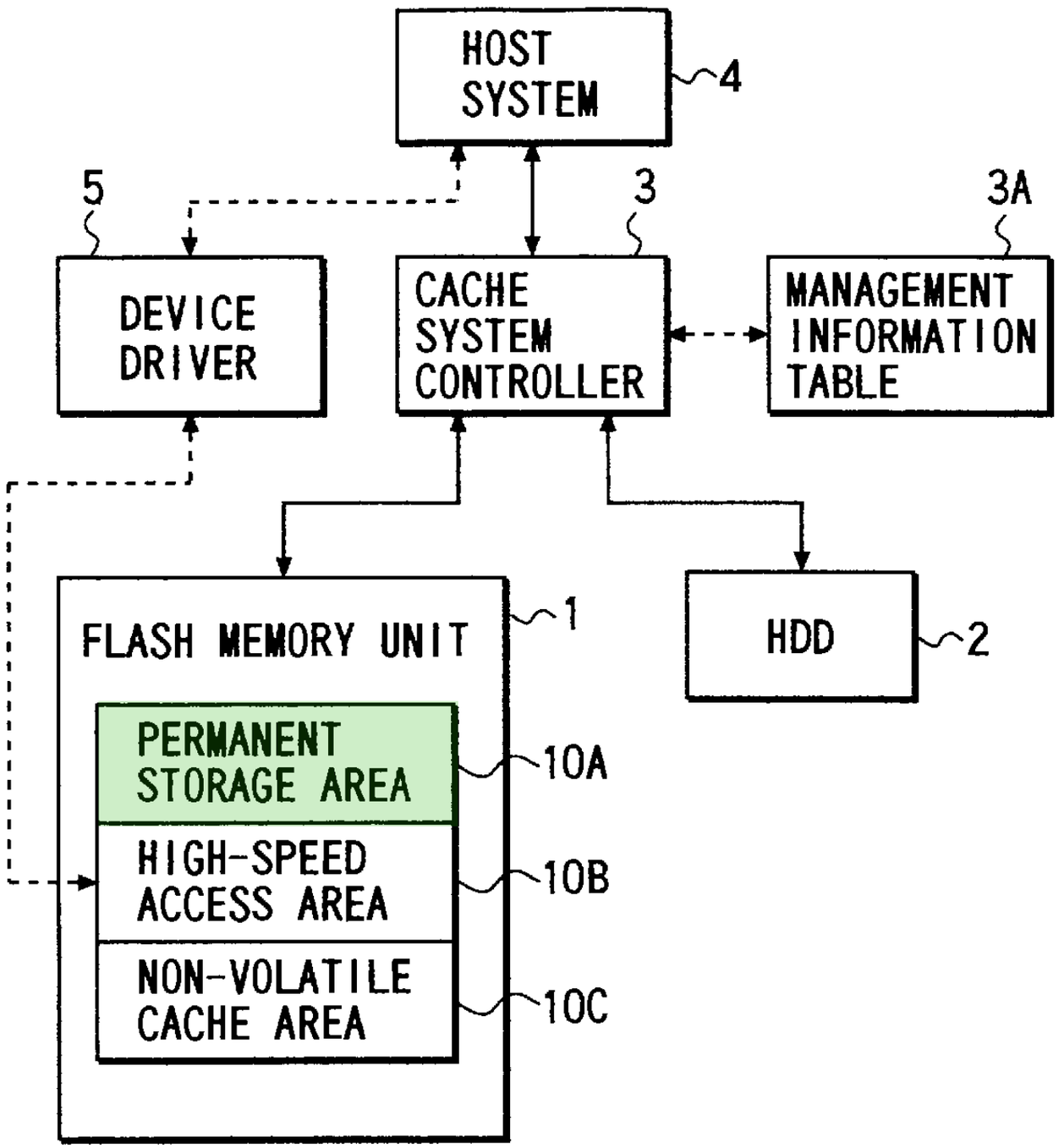
1

APPLE 1005

5,535,357	7/1996	Moran et al.	711/103
5,644,539	7/1997	Yamagami et al.	365/200
5,701,492	12/1997	Wadsworth et al.	395/712
5,745,418	4/1998	Ma et al.	365/185.33

area.

28 Claims, 10 Drawing Sheets



In the present invention, it is assumed that the entire storage area of the flash memory unit **1** is logically divided into permanent storage area **10**A, high-speed access area **10**B and non-volatile cache area **10**C and the divided areas are managed. The controller **3** manages the storage areas **10**A to **10**C of the flash memory unit **1** by using a management information table **3**A. The management information table **3**A is stored, for example, in the non-volatile cache area **10**C of flash memory unit **1**.

The first embodiment relates to a system wherein the permanent storage area **10**A of flash memory unit **1** is used as a cache memory area. In this embodiment, it is supposed that the user desires to start a frequently used application program (AP) at high speed at all times.

The user starts a data storage utility program of the cache system controller **3** via a user interface provided in the host system **4** (step S**1**). The data storage utility program reads specified data from the HDD **2** and stores the read data in a specified storage area in the flash memory unit **1**. In this case, it is assumed that the user sets the permanent storage area **10**A in the flash memory unit **1** as the data storage area, at the time of instructing the start of the data storage utility program (step S**2**).

Then, the user instructs the host system **4** to start the AP (step S**3**). The host system **4**, upon receiving the AP start instruction, issues a read command to the controller **3** in order to read control information necessary for the start of the AP from the HDD **2**.

The controller **3** controls the HDD **2**, reads out the control information necessary for the start of the AP and transfers the read-out control information to the host system **4** (step S**4**). At this time, according to the started-up data storage utility program, the controller **3** stores the AP control information read out from the HDD **2** in the permanent storage area **10**A of flash memory unit **1** (step S**5**). When the AP has been prepared to start, the data storage utility program is stopped by the instruction from the user ("YES" in step S**6**; step S**7**). Through these operations, the control information necessary for starting the AP is stored in the permanent storage area **10**A in the flash memory unit **1**.

In this case, the control information is stored in the permanent storage area **10**A in flash memory unit **1** under the file name designated by the user. Information for correlating the file name and the AP and information of other comment is recorded on the management information table **3**A by the data storage utility program. The user inputs the file name to the controller **3** via the user interface, thereby referring to the file (the control information of the AP in this case) stored in the permanent storage area **10**A. The user can delete the file, if unnecessary. In other words, the control information necessary for starting the AP is permanently stored in the permanent storage area **10**A in the flash memory unit **1** as one file, until the user instructs the deletion of the file.

If the user instructs the start of the same AP via the user interface, the host system **4** issues the read command, as described above, to read from the HDD **2** the control information necessary for starting the AP ("YES" in step S**8**). Upon receiving the read command, the controller **3** determines whether the control information to be accessed is stored in the flash memory unit **1** by using the management information table **3**A. Since the AP control information is stored in the permanent storage area **10**A, the cache memory area is successfully accessed. Accordingly, the controller **3** reads out the AP control information from the permanent storage area **10**A of flash memory unit **1**, without accessing the HDD **2**, and transfers the read-out control information to the host system **4** (step S**9**). The host system **4** starts the AP designated by the user on the basis of the transferred AP control information (step S**10**).

By the above-described cache system, the control information of the frequently used AP designated by the user is read out from the HDD **2** and stored in the permanent storage area **10**A in the flash memory unit **1** used as the cache memory area. Accordingly, when the AP is to be started next time, the AP control information can be read out quickly from the permanent storage area **10**A used as cache memory area, and not from the HDD **2**. Thereby, the host system **4** can quickly acquire the control information at the time of starting the AP. As a result, the AP can be quickly started. Since the AP control information is permanently stored in the permanent storage area **10**A until the user instructs the deletion of the control information, as described above, the frequently used AP can be started quickly at all times.

(First Modification of the First Embodiment)

FIG. **4** is a flow chart illustrating a first modification of the first embodiment. This modification relates to a system having a mode (data storage mode) for storing the control information necessary for starting the OS in the permanent storage area **10**A of flash memory unit **1**, for example, when the OS of the host system **4** is started in a series of operations from the turn-on of power to the completion of the starting operation.

When the system is switched on and the user sets the data storage mode via the user interface, the controller **3** stores the information representing the data storage mode in the permanent storage area **10**A in flash memory unit **1** (steps S**11** to S**13**).

After the power is turned off and then turned on again, the controller **3** starts the above-mentioned data storage utility program on the basis of the information of the set data storage mode ("YES" in step S**14**; step S**15**). According to the data storage utility program, the control information, which is pre-stored in the HDD **2** and necessary for starting the OS, is read out and stored in the permanent storage area **10**A (steps S**16** and S**17**). The controller **3** transfers to the host system **4** the control information necessary for starting the OS read out from the HDD **2**. Based on the control information, the host system **4** starts the OS. After the preparation for starting the OS is completed, the data storage utility program is stopped ("YES" in step S**18**; step S**19**).

According to this system, when the OS is automatically started by the control information read out from the HDD **2** at the time of turning-on of power, the control information is stored in the permanent storage area **10**A used as the cache memory area for the HDD **2**. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD **2** but from the permanent storage area **10**A or cache memory area, and the read-out control information is transferred to the host system **4**. Thus, the control information can be accessed from the permanent storage area **10**A in the flash memory unit **1** having a higher access speed than the HDD **2**. As a result, the OS can be started at higher speed.

Like the AP control information, the OS control information may be stored as one file in the permanent storage area **10**A. Thereby, the user can refer to, or delete, the OS control information on an as-needed basis. For example, at the time of shipment of the personal computer, if the OS is pre-installed in the flash memory unit **1** functioning as cache memory area, the user can delete the OS control information based on the user's judgment. Specifically, the OS control information may be deleted by a user who is used to starting

the controller **3** via the user interface, thereby referring to the file (the control information of the AP in this case) stored in the permanent storage area **10**A. The user can delete the file, if unnecessary. In other words, the control information necessary for starting the AP is permanently stored in the permanent storage area **10**A in the flash memory unit **1** as one file, until the user instructs the deletion of the file.

If the user instructs the start of the same AP via the user interface, the host system **4** issues the read command, as described above, to read from the HDD **2** the control information necessary for starting the AP ("YES" in step **S8**). Upon receiving the read command, the controller **3** determines whether the control information to be accessed is stored in the flash memory unit **1** by using the management information table **3**A. Since the AP control information is stored in the permanent storage area **10**A, the cache memory area is successfully accessed. Accordingly, the controller **3**

at th
is sto
mem
starte
contr
not f
10A
matic
infor
area
speed
highe
Li
tion
10A.
infor
of st

rol
tep
r **3**
l is
ent
ı is
ory
r **3**
ent
ing
ı to
AP

area **10**A in the flash memory unit **1** having a higher access speed than the HDD **2**. As a result, the OS can be started at higher speed.

Like the AP control information, the OS control information may be stored as one file in the permanent storage area **10**A. Thereby, the user can refer to, or delete, the OS control information on an as-needed basis. For example, at the time of shipment of the personal computer, if the OS is pre-installed in the flash memory unit **1** functioning as cache memory area, the user can delete the OS control information based on the user's judgment. Specifically, the OS control information may be deleted by a user who is used to starting

ing
ent
the
e to
the
l in
ile,
ion
the
as
ıser
as
trol
tep
r **3**
d is

preparation for starting the OS is completed, the data storage utility program is stopped ("YES" in step **S18**; step **S19**).

According to this system, when the OS is automatically started by the control information read out from the HDD **2** at the time of turning-on of power, the control information is stored in the permanent storage area **10**A used as the cache memory area for the HDD **2**. Accordingly, when the OS is started at the time of the next turning-on of power, the control information necessary for starting the OS is read out not from the HDD **2** but from the permanent storage area **10**A or cache memory area, and the read-out control information is transferred to the host system **4**. Thus, the control information can be accessed from the permanent storage area **10**A in the flash memory unit **1** having a higher access speed than the HDD **2**. As a result, the OS can be started at higher speed.

Like the AP control information, the OS control informa-

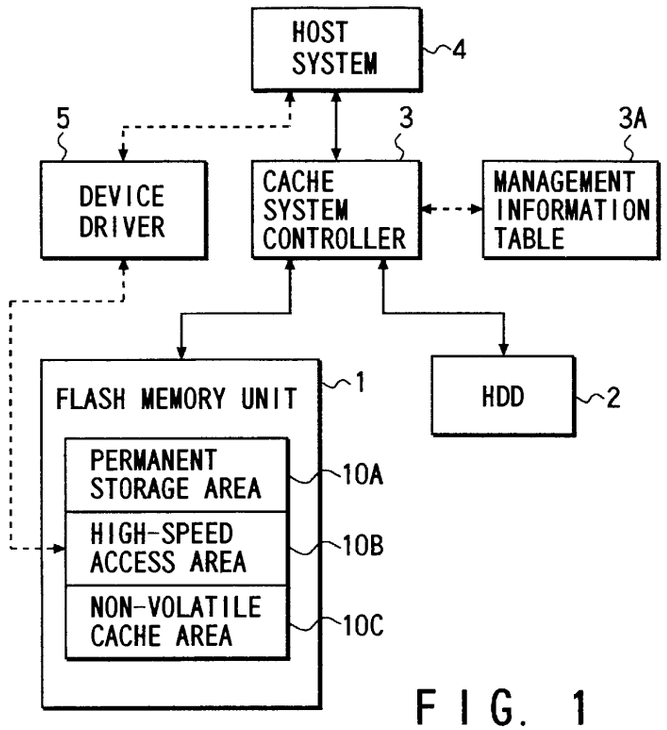

HOST SYSTEM
4
5
3
3A
DEVICE DRIVER
CACHE SYSTEM CONTROLLER
MANAGEMENT INFORMATION TABLE
FLASH MEMORY UNIT
1
HDD
2
PERMANENT STORAGE AREA
10A
HIGH-SPEED ACCESS AREA
10B
NON-VOLATILE CACHE AREA
10C

FIG. 1

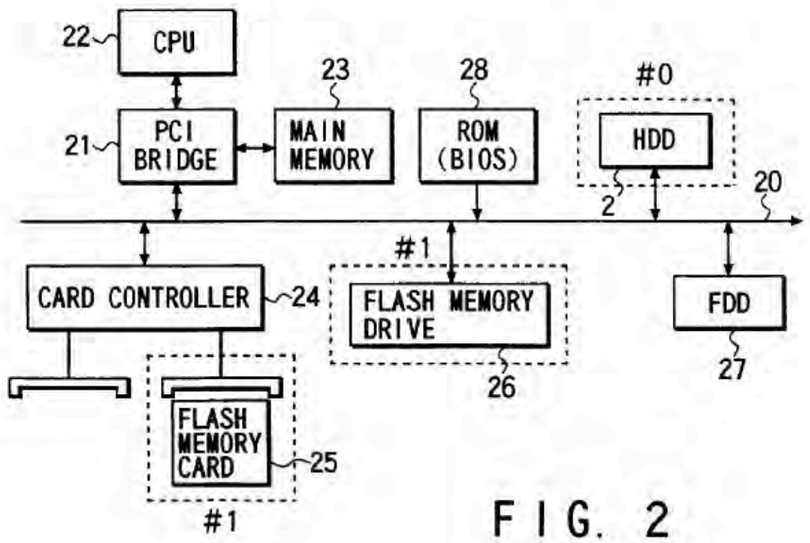

22
CPU
23
28
#0
21
PCI BRIDGE
MAIN MEMORY
ROM (BIOS)
HDD
2
20
#1
CARD CONTROLLER
24
FLASH MEMORY DRIVE
FDD
26
27
FLASH MEMORY CARD
25
#1

FIG. 2

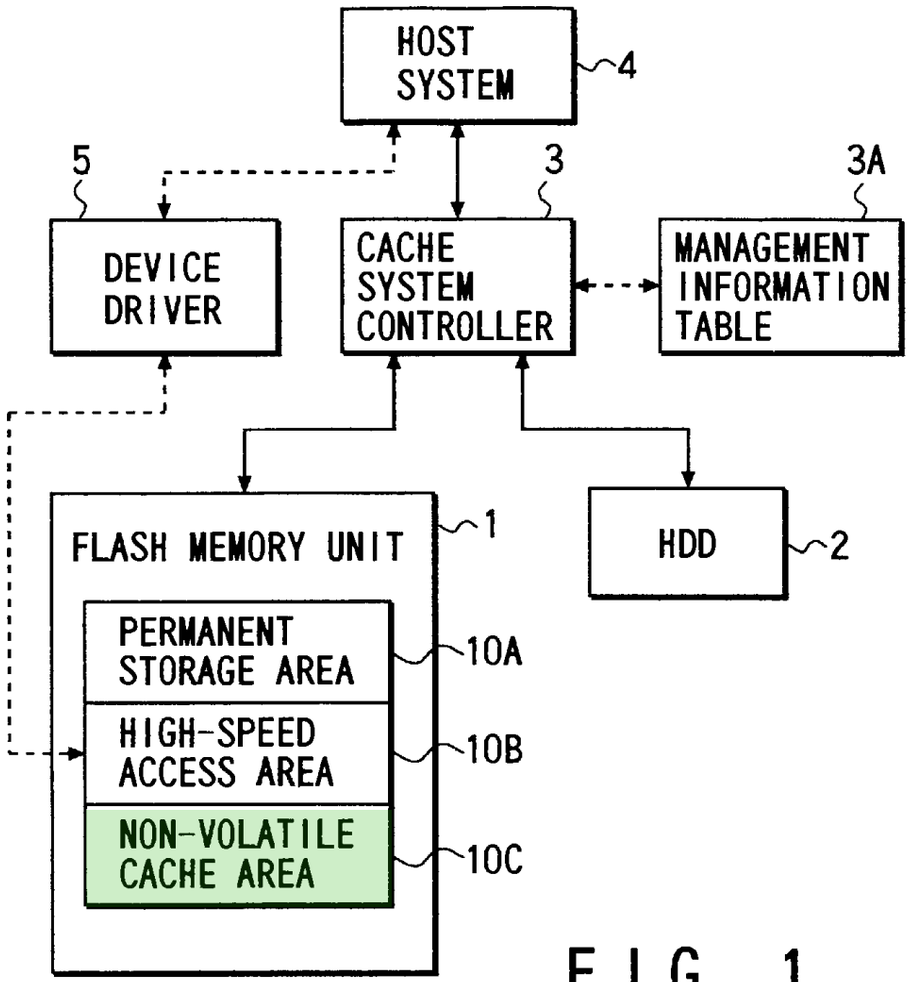


FIG. 1

storage area **10**A or non-volatile cache area **10**C, which is the cache memory area (or whether the cache memory area is "hit") (steps S**20** and S**21**), as shown in FIGS. **5**. If the data to be accessed is "hit", the controller **3** reads the data from the permanent storage area **10**A or non-volatile cache area **10**C and transfers the read-out data to the host system **4** ("YES" in step S**21**; step S**25**).

On the other hand, if the cache memory area is not "hit", the controller **3** accesses the HDD **2**, reads out the data to be accessed and transfers the read-out data to the host system **4**. In this case, as described above, if the data to be accessed is the permanent data designated by the user, the controller **3** stores it in the permanent storage area **10**A ("NO" in step

US006374353B1

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

(10) **Patent No.: US 6,374,353 B1**
(45) **Date of Patent: Apr. 16, 2002**

(54) **INFORMATION PROCESSING APPARATUS METHOD OF BOOTING INFORMATION PROCESSING APPARATUS AT A HIGH SPEED**

(75) Inventors: **Atsushi Settsu**; **Noriyuki Baba**; **Naoto Sugai**, all of Tokyo (JP)

(73) Assignee: **Mitsubishi Denki Kabushiki Kaisha**, Tokyo (JP)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/261,255**

(22) Filed: **Mar. 3, 1999**

(30) **Foreign Application Priority Data**

Mar. 16, 1998 (JP) 10-065957

(51) **Int. Cl.**[7] **G06F 9/445**
(52) **U.S. Cl.** **713/2**
(58) **Field of Search** 713/2

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,307,497 A		4/1994	Feigenbaum et al.	
5,355,498 A		10/1994	Provino et al.	
5,918,048 A	*	6/1999	Mealey et al.	713/2
5,933,631 A	*	8/1999	Mealey et al.	713/2
6,052,778 A	*	4/2000	Hagy et al.	713/2

OTHER PUBLICATIONS

M. M. Mckusick et al. Maruzen Co., Ltd., Jun. 30, 1991 pp. 413–433.

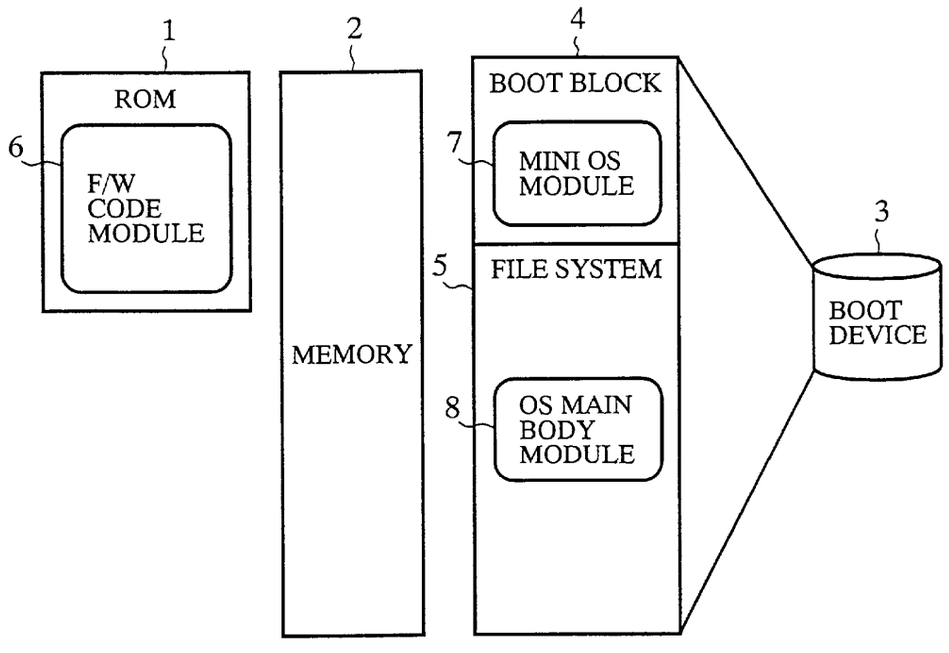
* cited by examiner

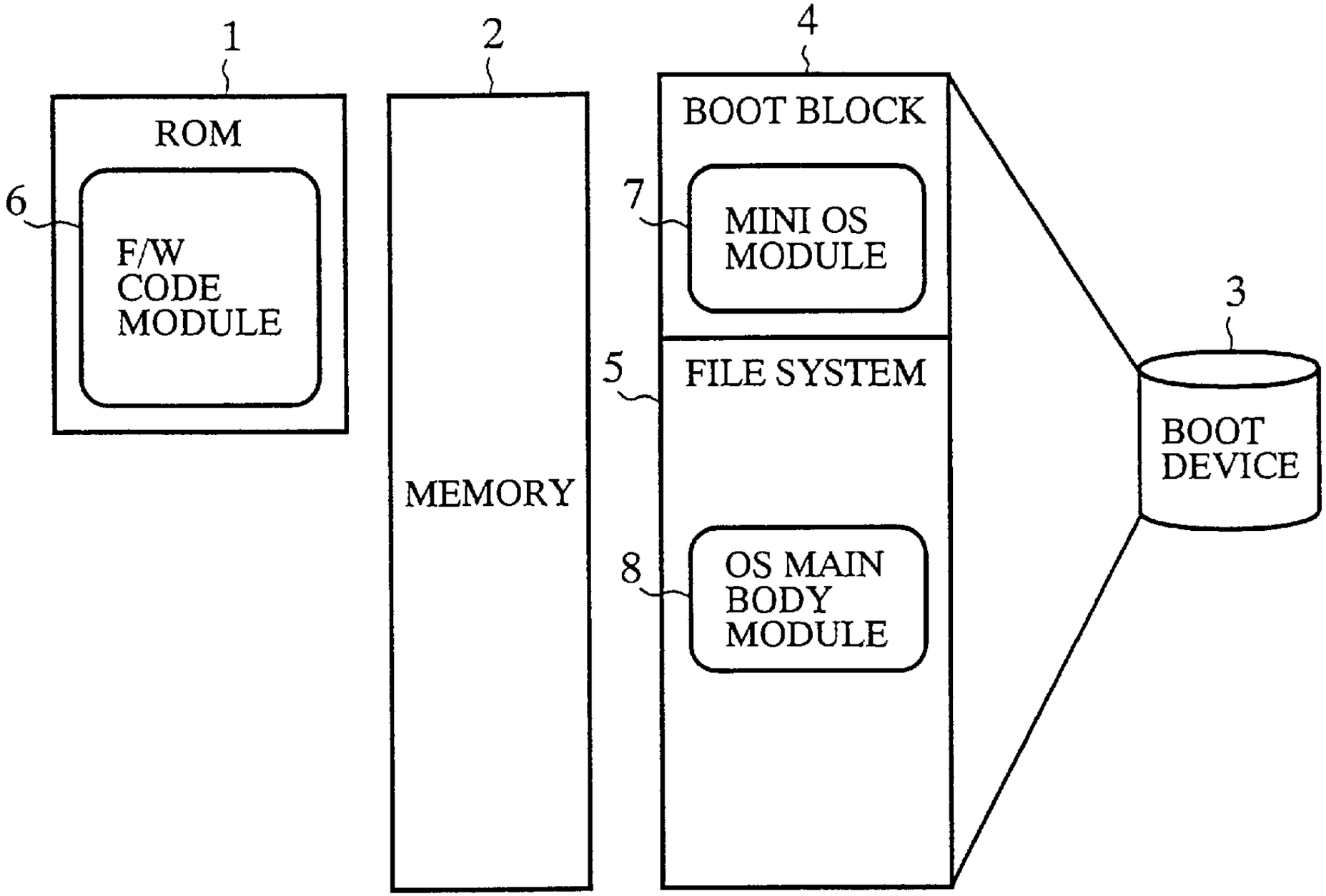
Primary Examiner—Thomas M. Heckler

(57) **ABSTRACT**

A method of booting up an information processing apparatus is provided. An operating system is divided into a mini operating system (OS) module having a function of bootstrap and an OS main body module having functions other than the function of bootstrap. The mini OS module can be located in a boot block of a boot device, whereas the OS main body module can be located in a file system of the boot device. A firmware or F/W code module stored in a ROM loads the mini OS module into memory when booting up the information processing apparatus. The mini OS module then loads the OS main body module into memory and then initializes the OS main body module.

12 Claims, 26 Drawing Sheets




1
ROM
6
F/W
CODE
MODULE
2
MEMORY
4
BOOT BLOCK
7
MINI OS
MODULE
5
FILE SYSTEM
8
OS MAIN
BODY
MODULE
3
BOOT
DEVICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of:	James J. Fallon, et al.	
U.S. Patent No.:	8,090,936	Attorney Docket No.: 39521-0024IP1
Issue Date:	January 3, 2012	Control No. IPR2016-01366
Appl. Serial No.:	11/551,204	
Filing Date:	October 19, 2006	
Title:	SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS	

Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

PETITION FOR *INTER PARTES* REVIEW OF UNITED STATES PATENT NO. 8,090,936 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42

POSITA additional motivation for incorporating compression/decompression (such as Dye's compression/decompression engine) into Sukegawa. Thus, Petitioner proposes, as Grounds 2-4, a combination of Sukegawa with Dye as well as Settsu and/or Burrows.

In the combinations, Sukegawa and Dye are applied in a similar manner to that discussed above in Section VI.A. Indeed, all of the analysis and citations to Sukegawa and Dye presented in Section VI.A. apply equally to Grounds 2-4. Grounds 2-4 simply rely on Settsu and/or Burrows for additional motivation and guidance to arrive at the combination of Sukegawa and Dye presented in Ground 1. Examples of additional motivation and guidance, including discussion of how Settsu and Burrows reinforce the use of compression/decompression in Sukegawa, are provided above in Section IV.D. and below for claim elements 1.2, 1.5, 7.0, 8.0, and 18.1-18.7. As shown by the mappings in the table that follows the analysis of Settsu and Burrows, these examples of additional disclosure apply to all claim elements in Grounds 2-4 where Section VI.A. references claim elements 1.2, 1.5, 7.0, 8.0, and 18.1-18.7. The remaining claim elements are addressed fully by the analysis and citations to Sukegawa and Dye presented in Section VI.A.

Thus, as discussed in Sections IV.D., VI.A., and below, claims 1-24 are obvious over Sukegawa and Dye in view of Settsu and/or Burrows.

[redacted]

In the combinations, Sukegawa and Dye are applied in a similar manner to that discussed above in Section VI.A. Indeed, all of the analysis and citations to Sukegawa and Dye presented in Section VI.A. apply equally to Grounds 2-4.

Grounds 2-4 simply rely on Settsu and/or Burrows for additional motivation and guidance to arrive at the combination of Sukegawa and Dye presented in Ground 1. Examples of additional motivation and guidance, including discussion of how Settsu and Burrows reinforce the use of compression/decompression in Sukegawa, are provided above in Section IV.D. and below for claim elements 1.2, 1.5, 7.0, 8.0, and 18.1-18.7. As shown by the mappings in the table that follows the analysis of Settsu and Burrows, these examples of additional disclosure apply to all claim elements in Grounds 2-4 where Section VI.A. references claim elements 1.2, 1.5, 7.0, 8.0, and 18.1-18.7. The remaining claim elements are addressed fully by

[redacted]

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE, INC.,
Petitioner

v.

REALTIME DATA LLC,
Patent Owner

Case IPR2016-01366
Patent 8,090,936

EXPERT DECLARATION OF DR. GODMAR BACK IN SUPPORT OF THE PATENT OWNER'S RESPONSE

understood as being storing "substantially all" of the OS in compressed form. Dr. Neuhauser equates Settsu's disclosure of "storing the mini OS module 7 and the OS main body module 8 as a plurality of compressed files in boot device 3"[95] to disclosing storing substantially all of the OS in compressed form but does not point to any support in Settsu. Settsu only discloses compressing the OS kernel, which a POSITA would have understood to be only a small portion of the "operating system." A POSITA therefore would not have interpreted Settsu's teachings as "storing substantially all of the [OS] in compressed form," as required by claim 18.

85. Based on these differences between the '936 Patent and Sukegawa, Sukegawa, Dye, and Settsu fail to disclose or render obvious "preloading" a "first portion" "to partially boot" the system" and "preloading" a "second portion" "to further partially boot" the system, as recited in claim 18. Moreover, Sukegawa, Dye, and Settsu do not teach "storing substantially all of the operating system in compressed form on a boot device."

C. Dye Does Not Disclose Storing Compressed Boot Data in a Hard Disk Drive.

86. Independent claims 1, 11, and 18 of the '936 Patent each recite storing "compressed" boot data on either a "boot device" or a "non-volatile memory

[95] Ex. 1003, Neuhauser Dec. at ¶ 284.

[redacted]

Neuhauser equates Settsu's disclosure of "storing the main OS module 7 and the OS main body module 8 as a plurality of compressed files in boot device 3"[95] to disclosing storing substantially all of the OS in compressed form but does not point to any support in Settsu. Settsu only discloses compressing the OS kernel, which a POSITA would have understood to be only a small portion of the "operating system." A POSITA therefore would not have interpreted Settsu's teachings as "storing substantially all of the [OS] in compressed form," as required by claim 18.

85. Based on these differences between the '936 Patent and Sukegawa, Sukegawa, Dye, and Settsu fail to disclose or render obvious "preloading" a "first portion" "to partially boot" the system" and "preloading" a "second portion" "to further partially boot" the system, as recited in claim 18. Moreover, Sukegawa,

[redacted]

“preloading boot data, in compressed form…from a boot device into a cache memory”

United States Patent [19]

Sukegawa

[11] **Patent Number: 5,860,083**

[45] **Date of Patent: Jan. 12, 1999**

[54] **DATA STORAGE SYSTEM HAVING FLASH MEMORY AND DISK DRIVE**

[75] Inventor: **Hiroshi Sukegawa**, Tokyo, Japan

[73] Assignee: **Kabushiki Kaisha Toshiba**, Kawasaki, Japan

[21] Appl. No.: **818,983**

[22] Filed: **Mar. 14, 1997**

[30] **Foreign Application Priority Data**

Nov. 26, 1996 [JP] Japan 8-314850

[51] **Int. Cl.**[6] **G06F 12/08**

[52] **U.S. Cl.** **711/103**; 711/117; 711/171; 711/173

[58] **Field of Search** 711/103, 113, 711/117, 170, 171, 173

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,175,842	12/1992	Totani	711/161
5,371,876	12/1994	Ewertz et al.	711/159
5,437,018	7/1995	Kobayashi et al.	395/652
5,535,357	7/1996	Moran et al.	711/103
5,644,539	7/1997	Yamagami et al.	365/200
5,701,492	12/1997	Wadsworth et al.	395/712
5,745,418	4/1998	Ma et al.	365/185.33
5,778,418	7/1998	Auclair et al.	711/101

FOREIGN PATENT DOCUMENTS

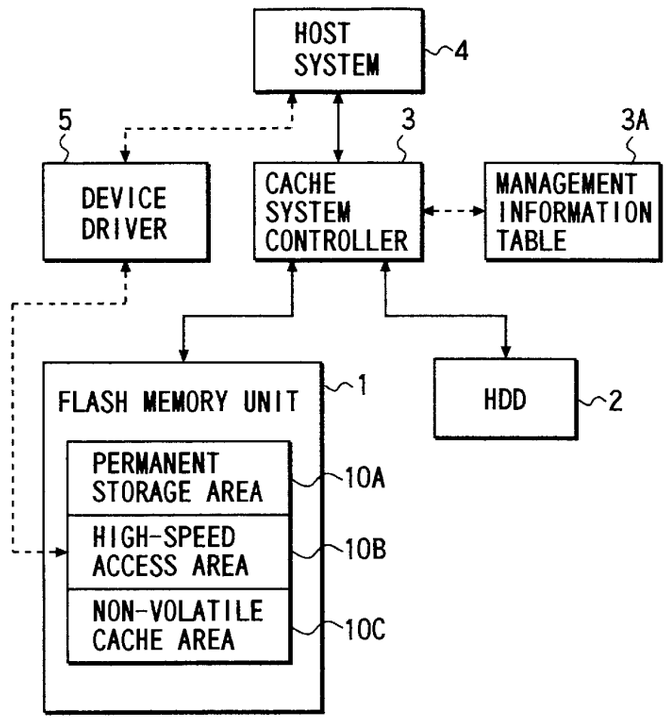
0 706 135	4/1996	European Pat. Off. .
5-11933	1/1993	Japan .
8-63395	3/1996	Japan .
8-115241	5/1996	Japan .

Primary Examiner—Tod R. Swann
Assistant Examiner—Conley B. King, Jr.
Attorney, Agent, or Firm—Finnegan, Henderson, Farabow, Garrett & Dunner, L.L.P.

[57] **ABSTRACT**

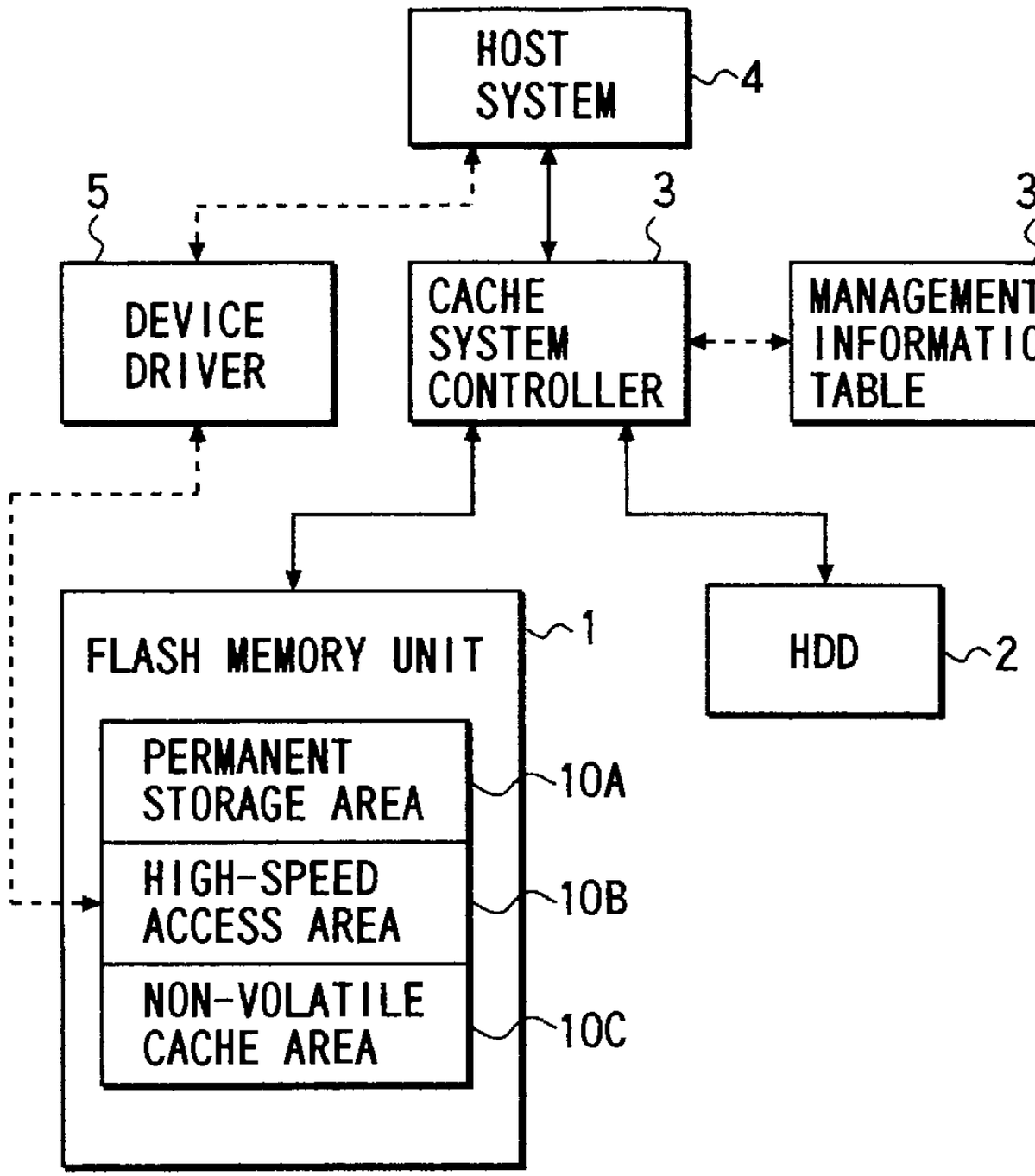
In a data storage system using a flash memory unit and an HDD, the storage area of the flash memory unit is logically divided into a permanent storage area, a non-volatile cache area, which are used as cache memory areas of the HDD, and a high-speed access area. These divided areas are individually managed. The permanent storage area stores data which is used frequently for a relatively long time period. The non-volatile cache area is used as an ordinary cache memory area in which data, which is updated relatively frequently, is stored. The high-speed access area is a storage area to be used by, e.g. an operating system (OS) of a host system. For example, a swap file, which needs to be accessed at high speed, is shifted into the high-speed access area.

28 Claims, 10 Drawing Sheets



1

APPLE 1005


HOST SYSTEM
4
5
3
3A
DEVICE DRIVER
CACHE SYSTEM CONTROLLER
MANAGEMENT INFORMATION TABLE
FLASH MEMORY UNIT
1
HDD
2
PERMANENT STORAGE AREA
10A
HIGH-SPEED ACCESS AREA
10B
NON-VOLATILE CACHE AREA
10C

US006145069A

United States Patent [19]

Dye

[11] Patent Number: 6,145,069

[45] Date of Patent: Nov. 7, 2000

[54] **PARALLEL DECOMPRESSION AND COMPRESSION SYSTEM AND METHOD FOR IMPROVING STORAGE DENSITY AND ACCESS SPEED FOR NON-VOLATILE MEMORY AND EMBEDDED MEMORY DEVICES**

[75] Inventor: **Thomas A. Dye**, Austin, Tex.

[73] Assignee: **Interactive Silicon, Inc.**, Austin, Tex.

[21] Appl. No.: **09/299,966**

[22] Filed: **Apr. 26, 1999**

Related U.S. Application Data

[63] Continuation-in-part of application No. 09/239,659, Jan. 29, 1999.

[51] **Int. Cl.**[7] **G06F 12/00**

[52] **U.S. Cl.** **711/170**; 711/103; 710/68; 382/233; 345/521; 345/501

[58] **Field of Search** 711/103, 170; 710/68; 714/763, 764; 709/247; 382/232, 233; 345/521, 501, 507, 509

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,008,460	2/1977	Bryant et al.	395/463
4,688,108	8/1987	Cotton et al.	358/261.1
4,876,541	10/1989	Storer .	
4,881,075	11/1989	Weng	341/87
5,003,307	3/1991	Whiting et al. .	
5,016,009	5/1991	Whiting et al. .	
5,126,739	6/1992	Whiting et al. .	
5,146,221	9/1992	Whiting et al. .	
5,155,484	10/1992	Chambers, IV .	
5,237,460	8/1993	Miller et al.	395/888
5,237,675	8/1993	Hannon, Jr.	710/68
5,247,638	9/1993	O'Brien et al.	710/68
5,247,646	9/1993	Osterlund et al.	395/888
5,337,275	8/1994	Garner	365/189.01
5,341,339	8/1994	Wells	365/185.11
5,353,024	10/1994	Graybill .	
5,353,425	10/1994	Malamy et al.	711/144
5,357,614	10/1994	Pattisam et al.	710/68
5,371,499	12/1994	Graybill et al. .	
5,379,036	1/1995	Storer .	
5,396,343	3/1995	Hanselman	358/426
5,406,278	4/1995	Graybill et al. .	
5,406,279	4/1995	Anderson et al. .	
5,412,429	5/1995	Glover .	
5,414,425	5/1995	Whiting et al. .	

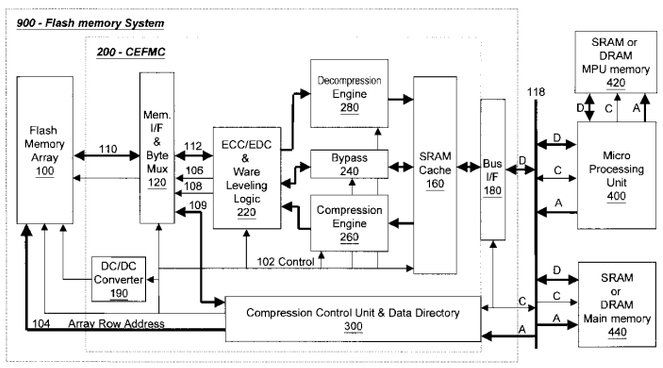
(List continued on next page.)

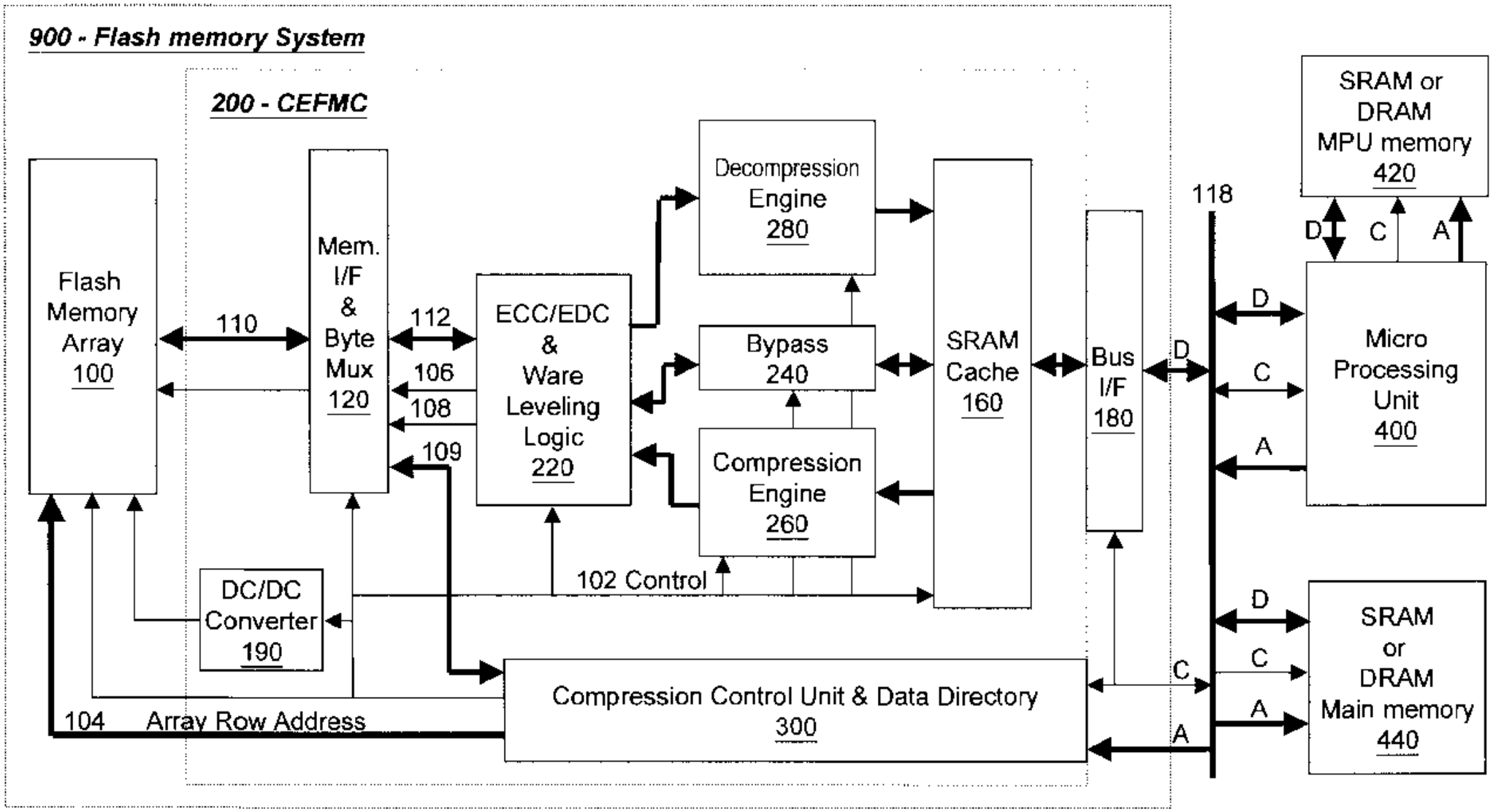
Primary Examiner—Eddie P. Chan
Assistant Examiner—Hong Kim
Attorney, Agent, or Firm—Conley, Rose & Tayon PC; Jeffrey C. Hood

[57] **ABSTRACT**

A flash memory controller and/or embedded memory controller including MemoryF/X Technology that uses data compression and decompression for improved system cost and performance. The Compression Enhanced Flash Memory Controller (CEFMC) of the present invention preferably uses parallel lossless compression and decompression engines embedded into the flash memory controller unit for improved memory density and data bandwidth. In addition, the invention includes a Compression Enhanced Memory Controller (CEMC) where the parallel compression and decompression engines are introduced into the memory controller of the microprocessor unit. The Compression Enhanced Memory Controller (CEMC) invention improves system wide memory density and data bandwidth. The disclosure also indicates preferred methods for specific applications such as usage of the invention for solid-state disks, embedded memory and Systems on Chip (SOC) environments. The disclosure also indicates a novel memory control method for the execute in place (XIP) architectural model. The integrated parallel data compression and decompression capabilities of the CEFMC and CEMC inventions remove system bottle-necks and increase performance matching the data access speeds of the memory subsystem to that of the microprocessor. Thus, the invention allows lower cost systems due to smaller data storage, reduced bandwidth requirements, reduced power and noise.

39 Claims, 24 Drawing Sheets




900 - Flash memory System
200 - CEFMC
Flash Memory Array 100
110
Mem. I/F & Byte Mux 120
112
106
108
109
ECC/EDC & Ware Leveling Logic 220
Decompression Engine 280
Bypass 240
Compression Engine 260
SRAM Cache 160
Bus I/F 180
118
SRAM or DRAM MPU memory 420
D
C
A
Micro Processing Unit 400
102 Control
DC/DC Converter 190
Compression Control Unit & Data Directory 300
104 Array Row Address
SRAM or DRAM Main memory 440

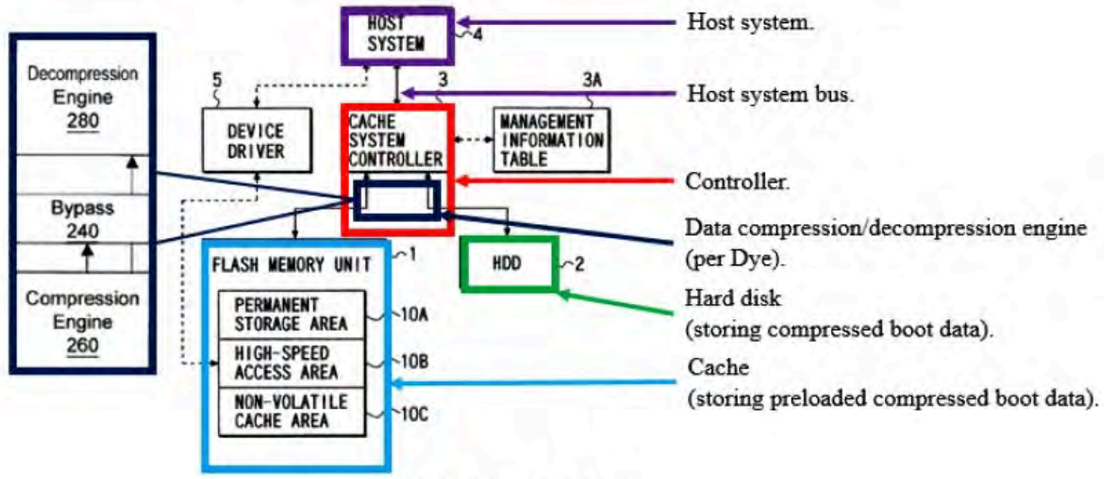
IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of:	James J. Fallon, et al.	
U.S. Patent No.:	7,181,608	Attorney Docket No.: 39521-0023IP1
Issue Date:	February 20, 2007	Control No. IPR2016-01365
Appl. Serial No.:	09/776,267	
Filing Date:	February 2, 2001	
Title:	SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS	

Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

PETITION FOR *INTER PARTES* REVIEW OF UNITED STATES PATENT NO. 7,181,608 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42

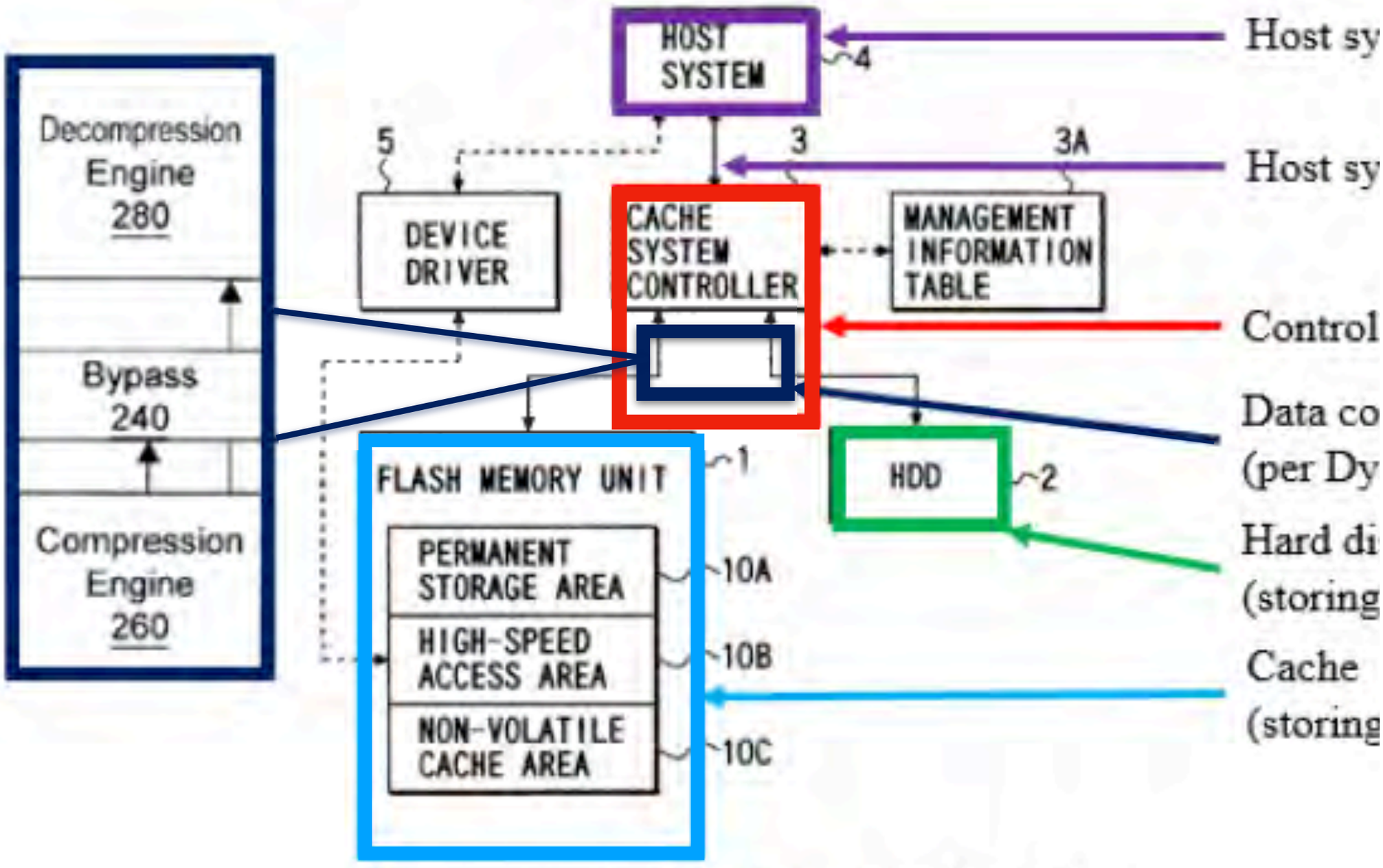
and decompressing the compressed data at a rate that increases flash memory unit 1's effective access rate. Id., ¶¶67, 120.



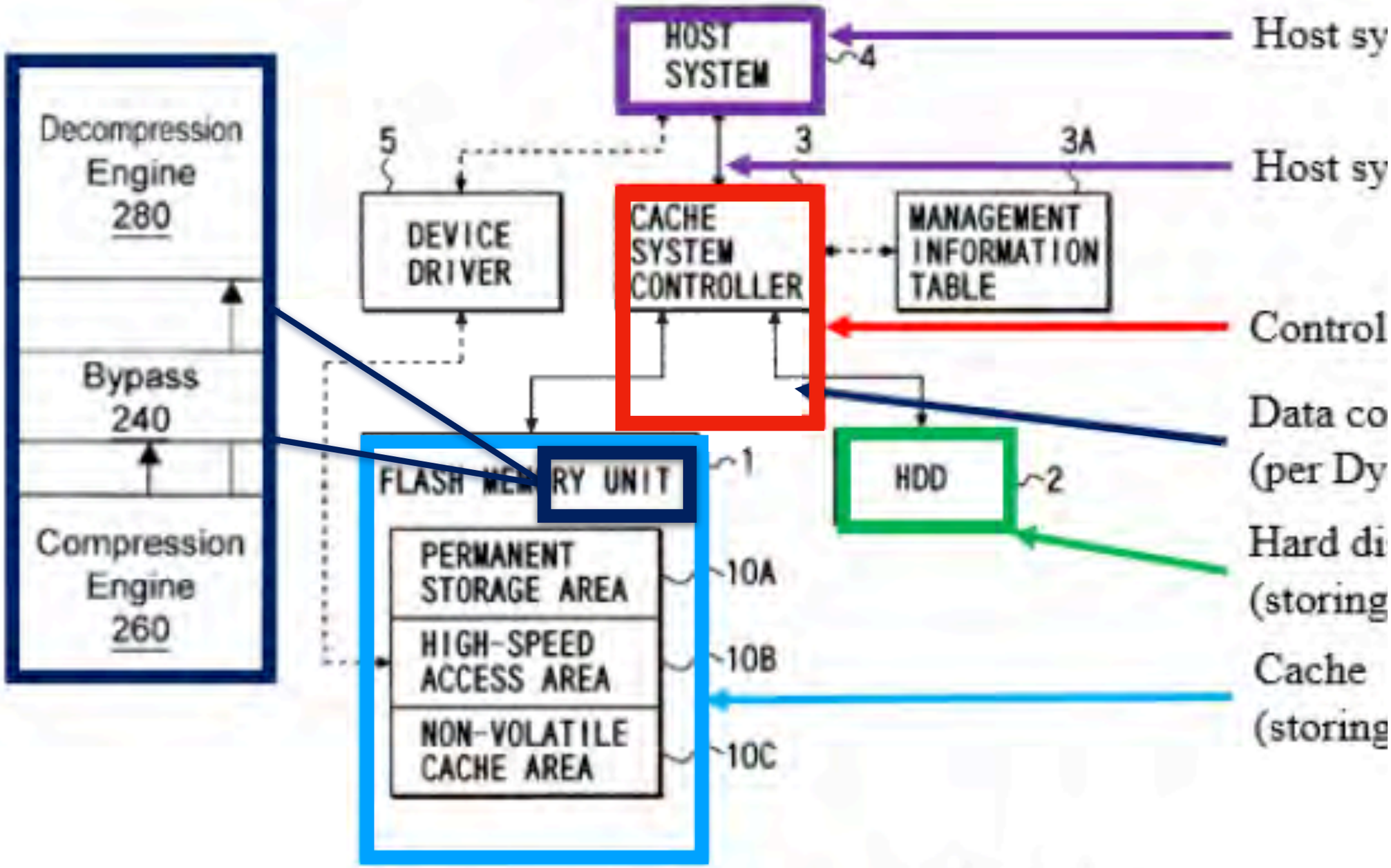
Sukegawa FIG. 1 and Dye FIG. 3
(combined excerpts, annotated).

With the increase in data access rates and data storage capacity, a POSITA would have understood that preloading compressed control information would further speed Sukegawa's boot process and, thus, further improve on Sukegawa's stated desire to increase boot speed. Id., ¶¶61-71. With these modifications, Sukegawa's modified system operates similarly to the '608 Patent. And, as explained below in Ground 1, Sukegawa and Dye render obvious all of the Challenged Claims.

D. Settsu and Burrows Further Confirm that Compression Was Well-Known and Would Have Been Obvious to Add to Sukegawa



Sukegawa FIG. 1 and Dye FIG. 3 (combined excerpts, annotated).



Sukegawa FIG. 1 and Dye FIG. 3 (combined excerpts, annotated).

31.0: ... a plurality of encoders in a parallel configuration are utilized by the data compression engine to compress the additional boot data.

See Sections IV.C., 10.0, 17.0; Dec., ¶389.

B. GROUNDS 2-4 – Claims 1-31 are obvious over Sukegawa and Dye in view of Settsu and/or Burrows

As discussed in Section VI.A., claims 1-31 are obvious over Sukegawa and Dye. As explained in Section IV.D., Settsu and/or Burrows would have provided a POSITA additional motivation for incorporating compression/decompression (such as that disclosed by Dye) into Sukegawa. Thus, Petitioner proposes, as Grounds 2-4, combinations of Sukegawa with Dye as well as Settsu and/or Burrows.

In the combinations, Sukegawa and Dye are applied in a similar manner to that discussed above in Section VI.A. Indeed, all of the analysis and citations to Sukegawa and Dye presented in Section VI.A. apply equally to Grounds 2-4. Grounds 2-4 simply rely on Settsu and/or Burrows for additional motivation and guidance to arrive at the combination of Sukegawa and Dye presented in Ground 1. Examples of additional motivation and guidance, including discussion of how Settsu and Burrows reinforce the use of compression/decompression in Sukegawa, are provided above in Section IV.D. and below for claim elements 1.4, 10.0, 14.0, and 15.0. As shown by the mappings in the table that follows the analysis of Settsu and Burrows, these examples of additional disclosure apply to all claim elements in Grounds 2-4 where Section VI.A. references claim elements 1.4, 10.0,

[redacted]

B. GROUNDS 2-4 – Claims 1-31 are obvious over Sukegawa and Dye in view of Settsu and/or Burrows

As discussed in Section VI.A., claims 1-31 are obvious over Sukegawa and Dye. As explained in Section IV.D., Settsu and/or Burrows would have provided a POSITA additional motivation for incorporating compression/decompression (such as that disclosed by Dye) into Sukegawa. Thus, Petitioner proposes, as Grounds 2-4, combinations of Sukegawa with Dye as well as Settsu and/or Burrows.

In the combinations, Sukegawa and Dye are applied in a similar manner to that discussed above in Section VI.A. Indeed, all of the analysis and citations to Sukegawa and Dye presented in Section VI.A. apply equally to Grounds 2-4. Grounds 2-4 simply rely on Settsu and/or Burrows for additional motivation and guidance to arrive at the combination of Sukegawa and Dye presented in Ground 1.

[redacted]

“a plurality of encoders are utilized to provide the compressed boot data”

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

(10) **Patent No.: US 7,181,608 B2**
(45) **Date of Patent: Feb. 20, 2007**

(54) **SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS**

(75) Inventors: **James J. Fallon**, Armonk, NY (US); **John Buck**, Oceanside, NY (US); **Paul F. Pickel**, Bethpage, NY (US); **Stephen J. McEerlain**, New York, NY (US)

(73) Assignee: **Realtime Data LLC**, New York, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 223 days.

(21) Appl. No.: **09/776,267**

(22) Filed: **Feb. 2, 2001**

(65) **Prior Publication Data**

US 2002/0069354 A1 Jun. 6, 2002

Related U.S. Application Data

(60) Provisional application No. 60/180,114, filed on Feb. 3, 2000.

(51) **Int. Cl.**
G06F 9/24 (2006.01)
G06F 9/00 (2006.01)
G06F 13/00 (2006.01)

(52) **U.S. Cl.** **713/2**; 713/1; 711/113

(58) **Field of Classification Search** 713/2, 713/1, 100; 711/170, 118, 113
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,127,518 A 11/1978 Coy et al.
4,302,775 A 11/1981 Widergren et al.
4,394,774 A 7/1983 Widergren et al.
4,574,351 A 3/1986 Dang et al.

(Continued)

FOREIGN PATENT DOCUMENTS

DE 4127518 A1 2/1992

(Continued)

OTHER PUBLICATIONS

IBM, Fast Dos Soft Boot, Feb. 1, 1994, vol. 37, Issue 2B, pp. 185-186.*

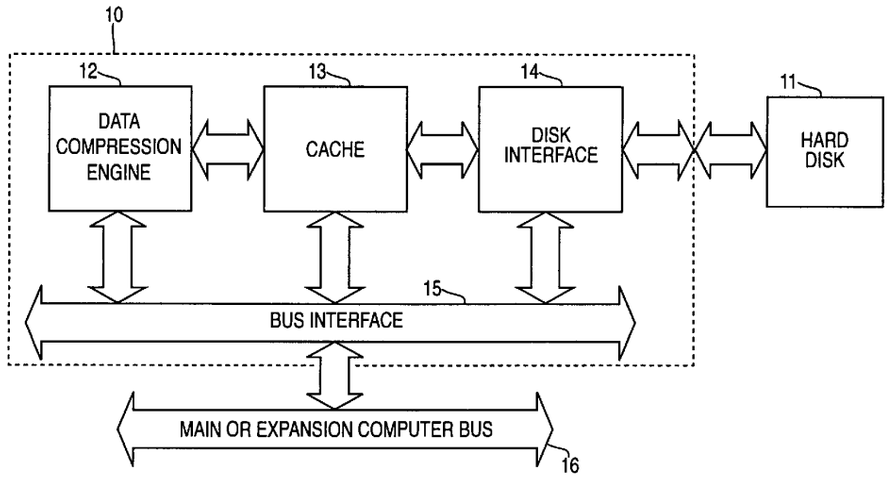
(Continued)

Primary Examiner—Thomas Lee
Assistant Examiner—Suresh K Suryawanshi
(74) *Attorney, Agent, or Firm*—Fish & Neave IP Group of Ropes & Gray LLP

(57) **ABSTRACT**

Systems and methods are provided for accelerated loading of operating system and application programs upon system boot or application launch. In one aspect, a method for providing accelerated loading of an operating system includes maintaining a list of boot data used for booting a computer system, preloading the boot data upon initialization of the computer system, and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. The boot data is retrieved from a boot device and stored in a cache memory device. The boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system.

31 Claims, 13 Drawing Sheets



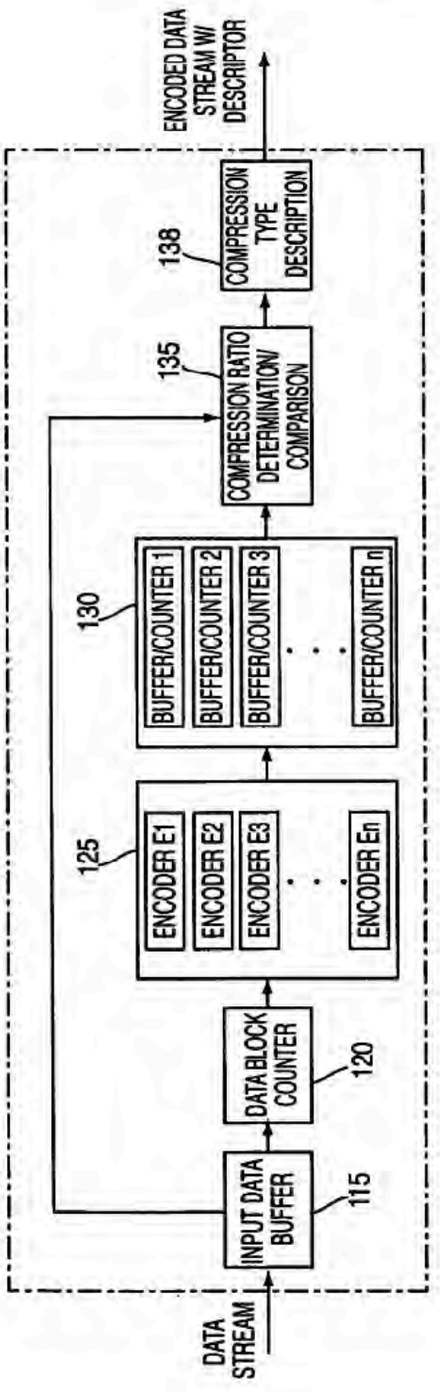


FIG. 9

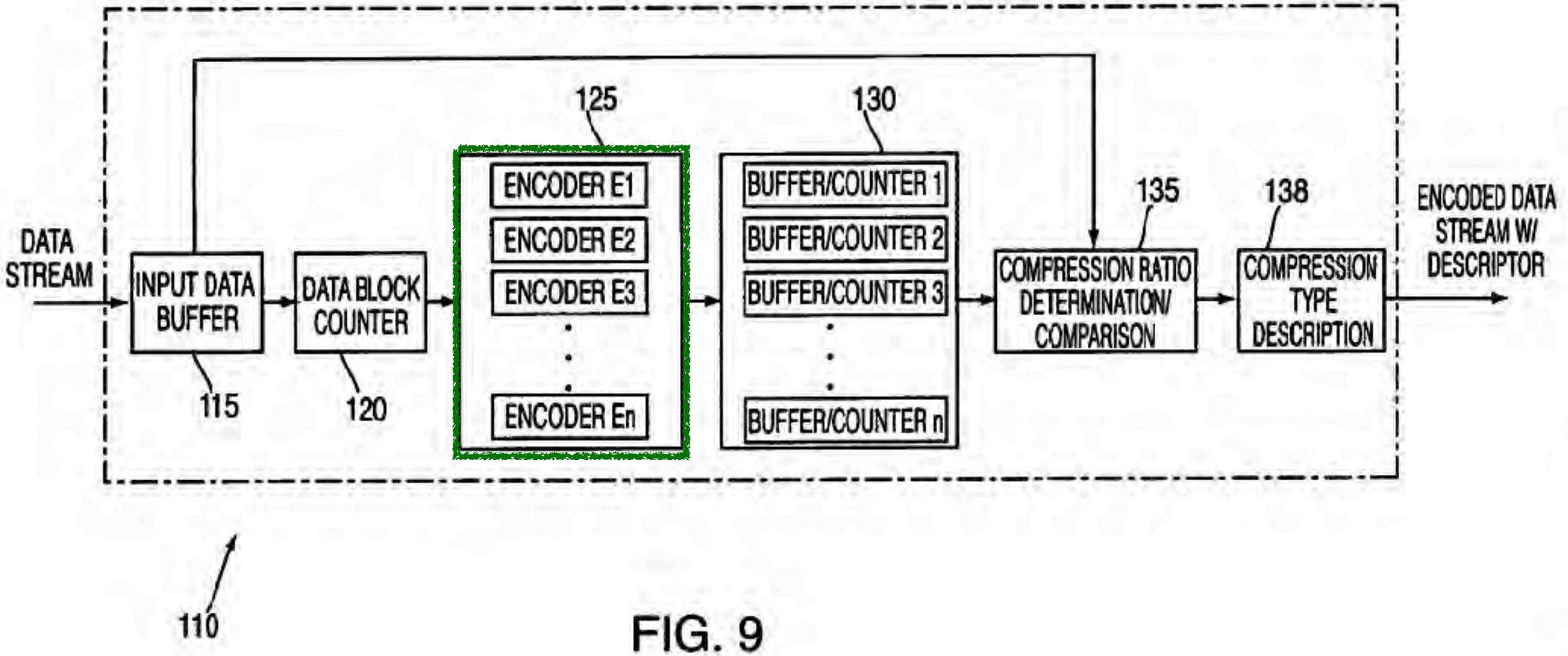

DATA STREAM
INPUT DATA BUFFER
DATA BLOCK COUNTER
125
ENCODER E1
ENCODER E2
ENCODER E3
ENCODER En
130
BUFFER/COUNTER 1
BUFFER/COUNTER 2
BUFFER/COUNTER 3
BUFFER/COUNTER n
135
COMPRESSION RATIO DETERMINATION/ COMPARISON
138
COMPRESSION TYPE DESCRIPTION
ENCODED DATA STREAM W/ DESCRIPTOR
115
120
110

FIG. 9

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of:	James J. Fallon, et al.	
U.S. Patent No.:	7,181,608	Attorney Docket No.: 39521-0023IP1
Issue Date:	February 20, 2007	Control No. IPR2016-01365
Appl. Serial No.:	09/776,267	
Filing Date:	February 2, 2001	
Title:	SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS	

Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

PETITION FOR *INTER PARTES* REVIEW OF UNITED STATES PATENT NO. 7,181,608 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42

16.0: … a plurality of encoders are utilized to provide the compressed boot data.

As explained in Sections IV.C., 1.4-1.7, and 10.0, a POSITA would have modified Sukegawa's controller 3 to include Dye's compression/decompression engine to provide compressed boot data. Dec., ¶245.

Dye's compression/decompression engine uses a "parallel lossless compression/decompression" "designed to process stream data at more than a single byte or symbol (character) at one time." Dye, 4:9-30. As shown in the annotated versions of FIGS. 10A and 10B reproduced below, Dye's compression algorithm analyzes multiple symbols in parallel, and provides "multiple compressed outputs" in parallel. Dye, 18:43-19:30, 22:66-23:24, FIG. 13 (depicting a hardware encoder used in parallel with others of the same type); Dye '284 12:61-13:7, 13:52-56, 14:49-50, 29:13-14. Thus, Dye describes using a plurality of encoders to provide compressed data. Dec., ¶¶274-288 (citing Ziv, 337-343; Storer, 928-951).

16.0: … a plurality of encoders are utilized to provide the compressed boot data.

As explained in Sections IV.C., 1.4-1.7, and 10.0, a POSITA would have modified Sukegawa's controller 3 to include Dye's compression/decompression engine to provide compressed boot data. Dec., ¶245.

Dye's compression/decompression engine uses a "parallel lossless compression/decompression" "designed to process stream data at more than a single byte or symbol (character) at one time." Dye, 4:9-30. As shown in the annotated versions of FIGS. 10A and 10B reproduced below, Dye's compression algorithm analyzes multiple symbols in parallel, and provides "multiple compressed outputs" in parallel. Dye, 18:43-19:30, 22:66-23:24, FIG. 13 (depicting a hardware encoder used in parallel with others of the same type); Dye '284 12:61-13:7, 13:52-56, 14:49-50, 29:13-14. Thus, Dye describes using a plurality of encoders to provide compressed data. Dec., ¶¶274-288 (citing Ziv, 337-343; Storer, 928-951).

Prior Art algorithm, using a single encoder to provide compressed data.

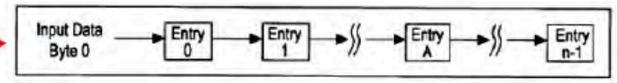


Fig. 10A
(Prior Art)

Dye's algorithm, using a plurality of encoders in parallel configuration to provide compressed data.

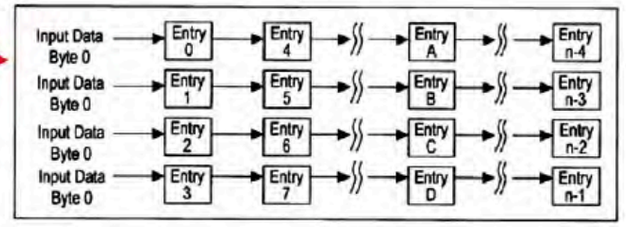


Fig. 10B
(New Art)

Dye FIGS. 10A and 10B (annotated)

17.0: … a plurality of encoders in a parallel configuration are utilized to provide the compressed boot data.

As explained in 16.0, Sukegawa and Dye render obvious using a plurality of encoders to provide the compressed control information. Dec., ¶288. Dye's encoders analyze multiple symbols in parallel, and provide "multiple compressed outputs" in parallel. Dye, 18:43-19:30. Thus, Sukegawa and Dye render obvious the plurality of encoders in a parallel configuration. Dec., ¶¶289-290.

18.0: … Huffman encoding is utilized to provide the compressed boot data.

See Sections IV.C., 14.0; Dec., ¶¶291-295.

19.0: … Lempel-Ziv encoding is utilized to provide the compressed boot data.

See Sections IV.C., 15.0; Dec., ¶¶296-300.

Prior Art algorithm, using a single encoder to provide compressed data.

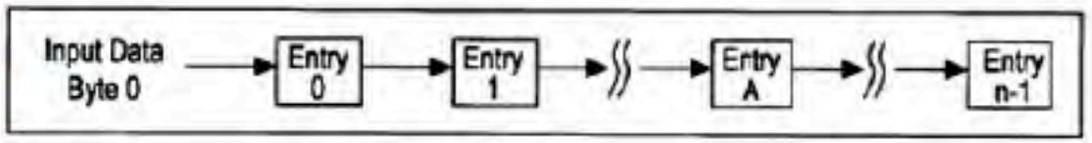


Fig. 10A
(Prior Art)

Dye's algorithm, using a plurality of encoders in parallel configuration to provide compressed data.

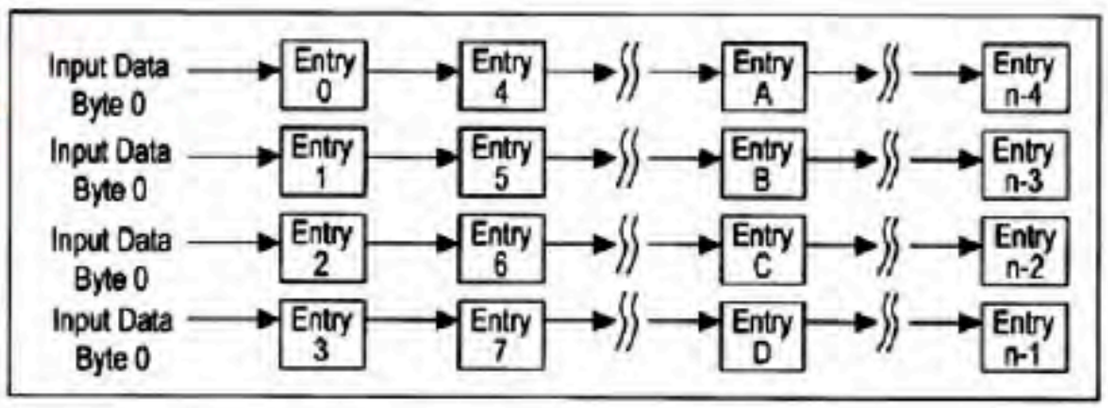


Fig. 10B
(New Art)

Dye FIGS. 10A and 10B (annotated)

17.0: … a plurality of encoders in a parallel configuration are utilized to pro-

vide the compressed boot data.

US006145069A

United States Patent [19]

Dye

[11] Patent Number: 6,145,069

[45] Date of Patent: Nov. 7, 2000

[54] **PARALLEL DECOMPRESSION AND COMPRESSION SYSTEM AND METHOD FOR IMPROVING STORAGE DENSITY AND ACCESS SPEED FOR NON-VOLATILE MEMORY AND EMBEDDED MEMORY DEVICES**

[75] Inventor: **Thomas A. Dye**, Austin, Tex.

[73] Assignee: **Interactive Silicon, Inc.**, Austin, Tex.

[21] Appl. No.: **09/299,966**

[22] Filed: **Apr. 26, 1999**

Related U.S. Application Data

[63] Continuation-in-part of application No. 09/239,659, Jan. 29, 1999.

[51] **Int. Cl.**[7] **G06F 12/00**

[52] **U.S. Cl.** **711/170**; 711/103; 710/68; 382/233; 345/521; 345/501

[58] **Field of Search** 711/103, 170; 710/68; 714/763, 764; 709/247; 382/232, 233; 345/521, 501, 507, 509

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,008,460	2/1977	Bryant et al.	395/463
4,688,108	8/1987	Cotton et al.	358/261.1
4,876,541	10/1989	Storer .	
4,881,075	11/1989	Weng	341/87
5,003,307	3/1991	Whiting et al. .	
5,016,009	5/1991	Whiting et al. .	
5,126,739	6/1992	Whiting et al. .	
5,146,221	9/1992	Whiting et al. .	
5,155,484	10/1992	Chambers, IV .	
5,237,460	8/1993	Miller et al.	395/888
5,237,675	8/1993	Hannon, Jr.	710/68
5,247,638	9/1993	O'Brien et al.	710/68
5,247,646	9/1993	Osterlund et al.	395/888
5,337,275	8/1994	Garner	365/189.01
5,341,339	8/1994	Wells	365/185.11
5,353,024	10/1994	Graybill .	
5,353,425	10/1994	Malamy et al.	711/144
5,357,614	10/1994	Pattisam et al.	710/68
5,371,499	12/1994	Graybill et al. .	
5,379,036	1/1995	Storer .	
5,396,343	3/1995	Hanselman	358/426
5,406,278	4/1995	Graybill et al. .	
5,406,279	4/1995	Anderson et al. .	
5,412,429	5/1995	Glover .	
5,414,425	5/1995	Whiting et al. .	

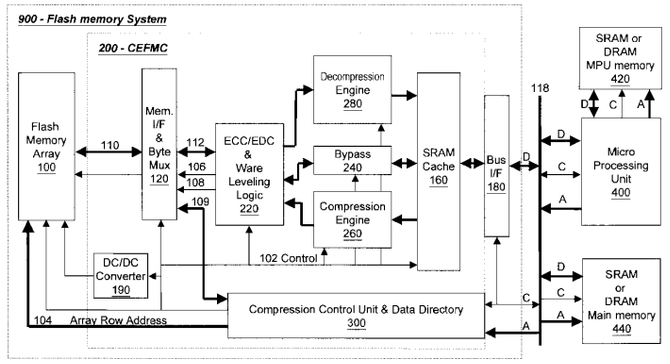
(List continued on next page.)

Primary Examiner—Eddie P. Chan
Assistant Examiner—Hong Kim
Attorney, Agent, or Firm—Conley, Rose & Tayon PC; Jeffrey C. Hood

[57] **ABSTRACT**

A flash memory controller and/or embedded memory controller including MemoryF/X Technology that uses data compression and decompression for improved system cost and performance. The Compression Enhanced Flash Memory Controller (CEFMC) of the present invention preferably uses parallel lossless compression and decompression engines embedded into the flash memory controller unit for improved memory density and data bandwidth. In addition, the invention includes a Compression Enhanced Memory Controller (CEMC) where the parallel compression and decompression engines are introduced into the memory controller of the microprocessor unit. The Compression Enhanced Memory Controller (CEMC) invention improves system wide memory density and data bandwidth. The disclosure also indicates preferred methods for specific applications such as usage of the invention for solid-state disks, embedded memory and Systems on Chip (SOC) environments. The disclosure also indicates a novel memory control method for the execute in place (XIP) architectural model. The integrated parallel data compression and decompression capabilities of the CEFMC and CEMC inventions remove system bottle-necks and increase performance matching the data access speeds of the memory subsystem to that of the microprocessor. Thus, the invention allows lower cost systems due to smaller data storage, reduced bandwidth requirements, reduced power and noise.

39 Claims, 24 Drawing Sheets



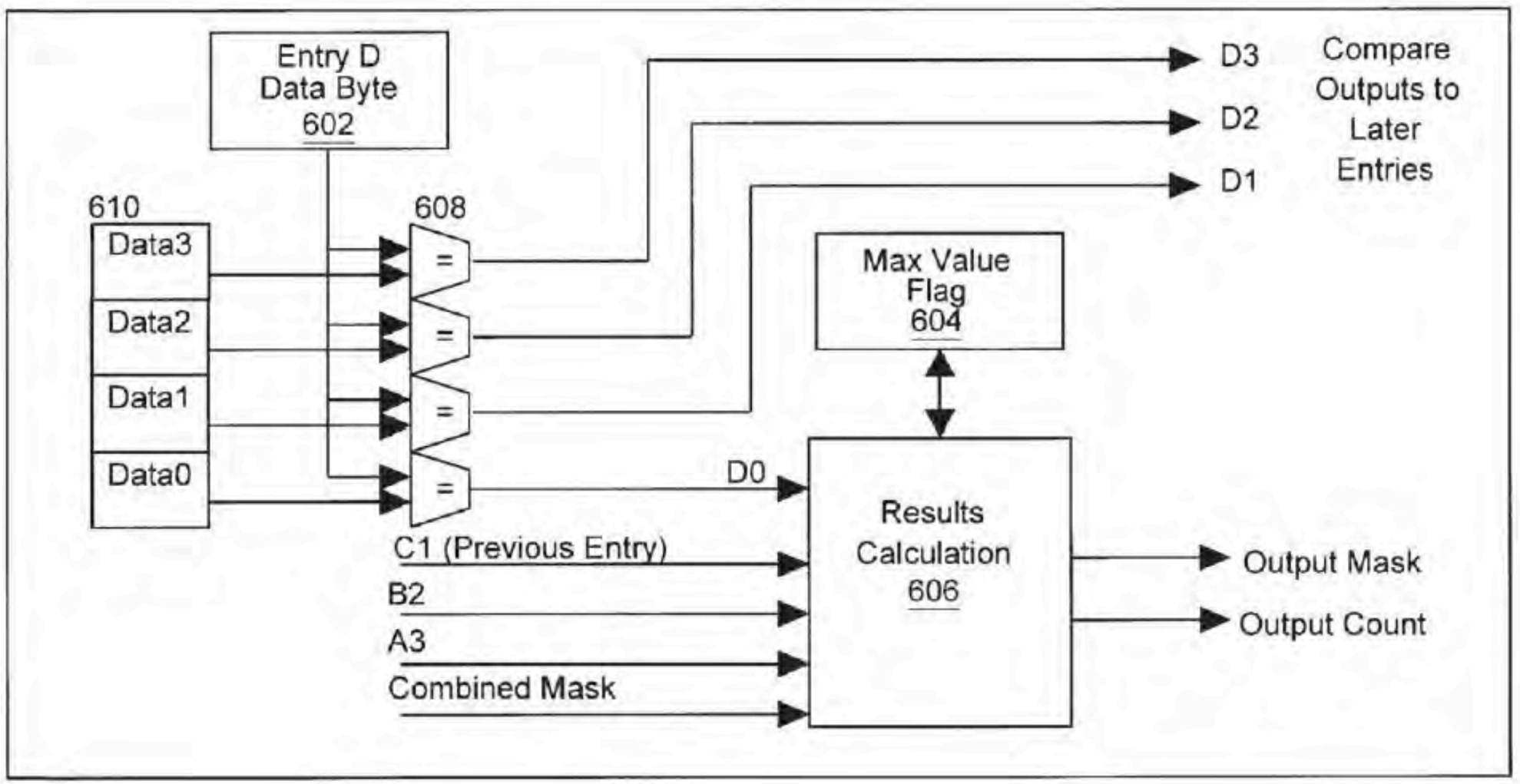

Entry D
Data Byte
602
610
608
Data3
Data2
Data1
Data0
=
=
=
=
D3
D2
D1
D0
Compare
Outputs to
Later
Entries
Max Value
Flag
604
Results
Calculation
606
C1 (Previous Entry)
B2
A3
Combined Mask
Output Mask
Output Count

Fig. 13

Input Matches				New Counter Value	Output Counter	Output Mask	Reset Value
D0	C1	B2	A3				
1	1	1	1	Saved+4	Saved +4	10000	0
1	1	1	0	0	Saved+3	10001	1
1	1	0	1	1	Saved+2	10010	2
1	1	0	0	0	Saved+2	10011	2
1	0	1	1	2	Saved+1	10100	3
1	0	1	0	0	Saved+1	10101	3
1	0	0	1	1	Saved+1	10110	3
1	0	0	0	0	Saved+1	10111	3
0	1	1	1	3	Saved	11000	4
0	1	1	0	0	Saved	01111	1
0	1	0	1	1	Saved	11010	4
0	1	0	0	0	Saved	11011	4
0	0	1	1	2	Saved	11100	4
0	0	1	0	0	Saved	11101	4
0	0	0	1	1	Saved	11110	4
0	0	0	0	0	Saved	11111	4

Fig. 15

"in-line" with data also preferably located within the Flash memory control circuit

The CEFMC is designed for the reduction of data bandwidth and is located between the main memory and/or system memory and the flash memory controller. The CEFMC Technology reduces the bandwidth requirements while increasing the memory efficiency for almost all data types within the computer system. Thus, conventional standard Flash Memory cells can achieve higher bandwidth, more effective density, with less system power and noise than when used in conventional systems without the CEFMC technology.

The CEFMC transfers data between the Flash Memory Array and the system MPU and its optional execution and data memories. Therefore, the CEFMC technology of the present invention typically resides between the MPU, main memory and the Flash Memory Array. In an alternate embodiment, the compression and/or decompression engines may reside in the MPU memory control unit, thus all memory data including flash memory can make use of lower pin-out interconnect buses, more effective memory performance, and increased effective memory density for all types of memory coupled to the MPU device.

The CEFMC technology is designed to embed into prior art flash memory control circuits. Thus, the current invention, using the novel parallel architecture to compress and decompress data streams, substantially improves bandwidth and effective storage density within the computing system. In addition, the CEFMC Technology has a "scalable" architecture designed to function in a plurality of memory configurations or compression modes with a plurality of performance requirements as indicated in U.S. patent application Ser. No. 09/239,659 titled "Bandwidth Reducing Memory Controller Including Scalable Embedded Parallel Data Compression and Decompression Engines" and filed Jan. 29, 1999 (5143-01700). Scalability allows for a non-symmetric compression rate as compared to the decompression rate. Write data can match the effective write speed of the Flash Memory Array, using fewer input symbols in parallel during compression, thus reducing gate count and size. Read data can be decompressed with a different number of input symbols per clock or access, thus allowing the read data to be decompressed at an alternate rate. Thus, the non-symmetric nature of the invention during reads and writes allows tuning of the memory access time vs. gate count to greatly improve performance and cost.

When configured for "execute in place" (XIP model), compressed data is programmed in to the flash memory for execution by the system MPU. The CEFMC invention decompresses the data as it is read by the MPU from the flash memory. In an alternate embodiment a DMA device can also be used to read data in a parallel fashion from the flash memory device. In the preferred embodiment, data presented at the output bus of the Flash Memory system is retrieved when the "ready" output (ready is a control signal associated with the MPU and Flash controller interface) transitions state during a read data request. The "ready" output indicates that the data has been successfully read from the Flash Memory Array and decompressed for consumption by the MPU. Any form of ready output indication can be used, as the "wait" is due to the decompression of a new block of data not previously stored in the SRAM buffer or cache. Alternatively, the timing specifications can include delay time specification indicating a "maximum delay" such that the MPU of system device waits for some period of time in order to process the decompressed requested data.

The CEFMC technology allows data to be stored in multiple compression formats and blocks sizes, as indicated in U.S. patent application Ser. No. 09/239,659 titled "Bandwidth Reducing Memory Controller Including Scalable Embedded Parallel Data Compression and Decompression Engines", referenced above. Thus, data can be saved in either a normal or compressed format, retrieved from the Flash Memory Array for MPU execution in a normal or compressed format, or transmitted and stored on a medium in a normal or compressed format.

To improve latency and reduce performance degradations normally associated with compression and decompression techniques the CEFMC encompasses multiple novel techniques such as: 1) Compiler directives for data types and block sizes for optimal compression and access speeds; 2) parallel lossless compression/decompression; selectable compression modes such as lossless, lossy or no compression; 3) data caching techniques; 4) unique address translation, attribute, and address directory structures, as illustrated in U.S. patent application Ser. No. 09/239,659, referenced above.

The CEFMC Technology preferably includes novel parallel compression and decompression engines designed to process stream data at more than a single byte or symbol (character) at one time. These parallel compression and decompression engines modify the single stream dictionary based (or history table based) data compression method described by Lempel and Ziv to provide a scalable, high bandwidth compression and decompression operation. The parallel compression method examines a plurality of symbols in parallel, thus providing greatly increased compression performance. The CEFMC technology, in an alternate embodiment, reduces latency further by use of multiple compiler hooks to distinguish program data space from table look-up data. Thus, if indicated, a bypass of the decompression engine will send data directly to the output interface bus without delay. A priority scheme can be applied such that compression and decompression operations are suspended as higher priority non-compressed data is transferred. Thus, reduction of latency and improved efficiency can be achieved at the cost of additional parallel buffers and comparison logic. Compiler directives interpreted by the decompression controller, can be embedded within the compiled XIP code for notification of compression/decompression bypass.

In summary, the integrated data compression and decompression capabilities of the present invention removes system bottlenecks allowing a higher frequency MPU clock by de-coupling the Flash Memory access time from MPU clock frequency. In addition, the present invention reduces the data storage size allowing more storage per Flash Memory Array. This lower cost system is due to reduced data storage requirements and improved bandwidth results. This also increases system bandwidth and hence increases system performance. Thus the compression based Flash Memory Controller of the present invention is a significant advance over the operation of current memory controllers.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

FIG. **1** illustrates a typical embodiment for the prior art Flash Memory Controller architecture without Compression Enhancement for the solid-state disk;

FIG. **2** illustrates a typical embodiment for the prior art Flash Memory Controller without Compression Enhancement for the execute in place (XIP) model;

The CEFMC Technology preferably includes novel parallel compression and decompression engines designed to process stream data at more than a single byte or symbol (character) at one time. These parallel compression and decompression engines modify the single stream dictionary based (or history table based) data compression method described by Lempel and Ziv to provide a scalable, high bandwidth compression and decompression operation. The parallel compression method examines a plurality of symbols in parallel, thus providing greatly increased compression performance. The CEFMC technology, in an alternate embodiment, reduces latency further by use of multiple compiler hooks to distinguish program data space from table look-up data. Thus, if indicated, a bypass of the decompression engine will send data directly to the output interface bus without delay. A priority scheme can be applied such that compression and decompression operations are suspended as higher priority non-compressed data is transferred. Thus,

data in the Cache Memory **425**. Once the write data is merged, step **3550** updates the LRU/MRU cache state and proceeds to complete the write cycle. In step **3560** if the latest write block forces a write back of the LRU block, then the process continues with step **3570**. If not, and open cache blocks were available then the process returns into the idle state of step **3440** and waits for the next transaction request. If LRU data is retired to the Flash Memory System **900**, the Flash Memory Controller **200** embedded within the Flash Memory System **900** must return the address of a cleared block of flash memory for write back of the compressed LRU as indicated in step **3570**. The retired LRU data may be compressed as indicated in step **3580** and then written back into the Flash Memory Array **100** shown in step **3590**. Thus, for increased system performance and lower costs the embedded compression and decompression architecture of the present invention is a substantial improvement from prior art Flash memory controllers.

Now referring to the increased performance aspects of the present invention, the advantages of using compression and decompression within the embedded system are shown. For the present embodiment a good example of the performance and cost advantages can be illustrated using the execute in place application of FIG. **5**. Data is clocked out of the Flash Memory Array **100** on a 32-bit bus at 40 ns for every four bytes. This is considered the flash memory "source" rate. The "sink" rate is the maximum System Bus **118** bandwidth running at 66 MHZ, which is equivalent to four bytes every 16 ns. If the average compression ratio for the parallel compression algorithm is 2.5:1 then the output rate of the Flash Memory Array **100** after decompression will be 40/2.5 ns per four bytes read, or 16 ns/Byte thus matching the 66 MHZ maximum bus speed requirements. In addition the effective density of the flash memory is now 2.5 times larger than without the use of the present invention. Thus the use of the present invention can greatly increase the system performance while decreasing the per-bit effective cost of storage.

Parallel Lossless Compression and Decompression

The parallel compression/decompression units or engines **260** and **280**, in combination referred to as codec engine **260/280**, which perform parallel compression and decompression functions, are now discussed. The codec engine **2601280** is preferably a dedicated codec hardware engine, e.g., the engine is comprised of logic circuitry. In one embodiment, the codec engine **260/280** comprises a programmable DSP or CPU core, or programmable compression/decompression processor, with one or more ROMs or RAMs which store different sets of microcode for certain functions, such as compression, decompression, special types of graphical compression and decompression, and bit blit operations, as desired. In another embodiment, the codec engine **260/280** dynamically shifts between the different sets of microcode in the one or more memories, depending on the function being performed. The compression/decompression engine may also be implemented using reconfigurable or programmable logic, e.g., one or more FPGAs.

As shown in FIGS. **3** and **4**, in some embodiments, the engine **260/280** preferably includes a lossless parallel data compression engine **260** and parallel decompression engine **280** designed to compress and decompress data as data is transferred to/from flash memory. Other embodiments, as illustrated in FIG. **5**, may be implemented with only a decompression engine **280**. The compression engine **260** and decompression engine **280** may be constructed using any of the techniques described with reference to the engine **260/280**, including hardware engines comprised of logic circuitry, programmable CPUs, DSPs, a dedicated compression/decompression processor, or reconfigurable or programmable logic, to perform the parallel compression and decompression method of the present invention. Various other implementations may be used to embed a compression/decompression within the flash memory controller according to the present invention. In one embodiment, the compression engine **260** and decompression engine **280** comprise hardware engines in the CEFMC **200** as shown in FIG. **3**. In another embodiment, the compression engine **260** and decompression engine **280** comprise hardware engines in the CEMC **910** as shown in FIG. **4**. In yet another embodiment, the decompression engine **280** comprises a hardware engine in the CEFMC **200** as shown in FIG. **5**, In the following description, the parallel compression and decompression unit is described as having separate compression and decompression engines **260** and **280**.

In the various embodiments, the compression engine **260** and decompression engine **280** comprise one or more hardware engines that perform a novel parallel lossless compression method, preferably a "parallel" dictionary based compression and decompression algorithm. The parallel algorithm may be based on a serial dictionary based algorithm, such as the LZ**77** (preferably LZSS) dictionary based compression and decompression algorithm. The parallel algorithm may be based on any variation of conventional serial LZ compression, including LZ**77**, LZ**78**, LZW and/or LZRW**1**, among others,

The parallel algorithm could also be based on Run length Encoding, Predictive Encoding, Huffman, Arithmetic, or any other lossless compression algorithm. However, the paralleling of these is less preferred due to their lower compression capabilities and/or higher hardware costs.

As a base technology, any of various lossless compression methods may be used as desired. As noted above, a parallel implementation of LZSS compression is preferably used, although other lossless compression methods may allow for fast parallel compression and decompression specifically designed for the purpose of improved memory bandwidth and efficiency.

FIG. **10**A—Prior Art, Serial LZ Compression

Prior art has made use of the LZ compression algorithm for design of computer hardware, but the bandwidth of the data stream has been limited due to the need to serially review the incoming data to properly generate the compressed output stream. FIG. **10**A depicts the prior art normal history table implementation.

The LZ compression algorithm attempts to reduce the number of bits required to store data by scarching that data for repeated symbols or groups of symbols. A hardware implementation of an LZ**77** algorithm would make use of a history table to remember the last n symbols of a data stream so that they could be compared with the incoming data. When a match is found between the incoming stream and the history table, the matching symbols from the stream are replaced by a compressed symbol, which describes how to recover the symbols from the history table.

FIG. **10**B—Parallel Algorithm

The preferred embodiment of the present invention provides a parallel implementation of dictionary based (or history table based) compression/decompression. By designing a parallel history table, and the associated compare logic, the bandwidth of the compression algorithm can be increased many times. This specification describes the implementation of a 4 symbol parallel algorithm which

so that they could be compared with the incoming data. When a match is found between the incoming stream and the history table, the matching symbols from the stream are replaced by a compressed symbol, which describes how to recover the symbols from the history table.

FIG. **10**B—Parallel Algorithm

The preferred embodiment of the present invention provides a parallel implementation of dictionary based (or history table based) compression/decompression. By designing a parallel history table, and the associated compare logic, the bandwidth of the compression algorithm can be increased many times. This specification describes the implementation of a 4 symbol parallel algorithm which

results in a 4 times improvement in the bandwidth of the implementation with no reduction in the compression ratio of the data. In alternate embodiments, the number of symbols and parallel history table can be increased and scaled beyond four for improved parallel operation and bandwidth, or reduced to ease the hardware circuit requirements. In general, the parallel compression algorithm can be a 2 symbol parallel algorithm or greater, and is preferably a multiple of 2, e.g., 2, 4, 8, 16, 32, etc. The parallel algorithm is described below with reference to a 4 symbol parallel algorithm for illustrative purposes.

The parallel algorithm comprises paralleling three parts of the serial algorithm: the history table (or history window), analysis of symbols and compressed stream selection, and the output generation. In the preferred embodiment the data-flow through the history table becomes a 4 symbol parallel flow instead of a single symbol history table. Also, 4 symbols are analyzed in parallel, and multiple compressed outputs may also be provided in parallel. Other alternate embodiments may contain a plurality of compression windows for decompression of multiple streams, allowing a context switch between decompression of individual data blocks. Such alternate embodiments may increase the cost and gate counts with the advantage of suspending current block decompression in favor of other block decompression to reduce latency during fetch operations. For case of discussion, this disclosure will assume a symbol to be a byte of data. Symbols can be any reasonable size as required by the implementation. FIG. **10**B shows the data-flow for the parallel history table.

FIG. **11**—High Level Flowchart of the Parallel Compression Algorithm

FIG. **11** is a high-level flowchart diagram illustrating operation of the parallel compression algorithm in the preferred embodiment. Steps in the flowchart may occur concurrently or in different orders.

In step **402** the method maintains a history table (also called a history window) comprising entries, wherein each entry may comprise one symbol. The history table is preferably a sliding window which stores the last n symbols of the data stream.

In step **404** the method maintains a current count of prior matches which occurred when previous symbols were compared with entries in the history table, A count is maintained for each entry in the history table.

It is noted that maintenance of the history table and the current counts are performed throughout the algorithm based on previously received symbols, preferably starting when the first plurality of symbols are received for compression.

In step **406** the method receives uncompressed data, wherein the uncompressed data comprises a plurality of symbols. Thus the parallel compression algorithm operates on a plurality of symbols at a time. This is different than conventional prior art serial algorithms, which operate in a serial manner on only one symbol at a time. The plurality of symbols comprises 2 or more symbols, preferably a power of 2. In the preferred embodiment, the parallel compression algorithm operates on 4 symbols at a time. However, implementations using 8, 16, 32 or more symbols, as well as other non-power of 2 numbers, may be readily accomplished using the algorithm described herein.

In step **408** the method compares the plurality of symbols with each entry in the history table in a parallel fashion. This comparison produces compare results. Each entry in the history table preferably compares with each of the plurality of symbols concurrently, i.e., in a parallel fashion, for improved speed.

In step **410** the method determines match information for each of the plurality of symbols based on the current count and the compare results. Step **410** of determining match information includes determining zero or more matches of the plurality of symbols with each entry in the history table. More specifically, step **410** may include determining a longest contiguous match based on the current count and the compare results, and then determining if the longest contiguous match has stopped matching. If the longest contiguous match has stopped matching, then the method resets or updates the current counts.

As noted above, stop **410** also includes resetting the counts for all entries if the compare results indicate a contiguous match did not match one of the plurality of symbols. The counts for all entries are preferably reset based on the number of the plurality of symbols that did not match in the contiguous match. In the preferred embodiment, the method generates a reset value for all entries based on the compare results for a contiguous match. The reset value indicates a number of the plurality of symbols that did not match in the contiguous match as indicated in the compare results. The method then updates the current counts according to the compare results and the reset value.

In step **412** the method outputs compressed data information in response to the match information. Step **412** may involve outputting a plurality of sets or compressed data information in parallel, e.g., for different matches and/or for non-matching symbols. Step **412** includes outputting compressed data information corresponding to the longest contiguous match that stopped matching, if any. The contiguous match may involve a match from a prior plurality of symbols. Step **412** may also include outputting compressed data information solely from a prior match. Step **412** also includes, for non-matching symbols that do not match any entry in the history table, outputting the non-matching symbols in an uncompressed format.

For a contiguous match, the compressed data information includes a count value and an entry pointer. The entry pointer points to the entry in the history table that produced the contiguous match, and the count value indicates a number of matching symbols in the contiguous match. In one embodiment, an encoded value is output as the count value, wherein more often occurring counts are encoded with fewer bits than less often occurring counts.

Steps **402**–**412** are repeated one or more times until no more data is available. When no more data is available, then, if any current counts air, non-zero, the method outputs compressed data for the longest remaining match in the history table.

Since the method performs parallel compression, operating on a plurality of symbols at a time, the method preferably accounts for symbol snatches comprised entirely within a given plurality of symbols, referred to as the "special case". Here presume that the plurality of symbols includes a first symbol, a last symbol, and one or more middle symbols. Step **410** of determining match information includes detecting if at least one contiguous match occurs with one or more respective contiguous middle symbols, and the one or more respective contiguous middle symbols are not involved in a match with either the symbol before or after the respective contiguous middle symbols. If this condition is detected, then the method selects the one or more largest non-overlapping contiguous matches involving the middle symbols. In this instance, step **412** includes outputting compressed data for each of the selected matches involving the middle symbols.

[redacted]

is described below with reference to a 4 symbol parallel algorithm for illustrative purposes.

The parallel algorithm comprises paralleling three parts of the serial algorithm: the history table (or history window), analysis of symbols and compressed stream selection, and the output generation. In the preferred embodiment the data-flow through the history table becomes a 4 symbol parallel flow instead of a single symbol history table. Also, 4 symbols are analyzed in parallel, and multiple compressed outputs may also be provided in parallel. Other alternate embodiments may contain a plurality of compression windows for decompression of multiple streams, allowing a context switch between decompression of individual data blocks. Such alternate embodiments may increase the cost and gate counts with the advantage of suspending current block decompression in favor of other block decompression to reduce latency during fetch operations. For case of