

1032 U.S. PTO  
02/13/01

Please type a plus sign (+) inside this box ☒

Approved for use through 10/31/2002. OMB 0651-0032  
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE  
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

# **PROVISIONAL APPLICATION FOR PATENT COVER SHEET**

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

JC929 U.S. PTO  
60/268394  
02/13/01

INVENTOR(S)					
Given Name (first and middle [if any])		Family Name or Surname		Residence (City and either State or Foreign Country)	
James J. Stephen J.		Fallon McErlain		11 Wamous Close, Armonk, NY 10504 325 E. 17th St., New York, NY 10003	
<input type="checkbox"/> Additional inventors are being named on the _____ separately numbered sheets attached hereto					
TITLE OF THE INVENTION (280 characters max)					
BANDWIDTH SENSITIVE DATA COMPRESSION AND DECOMPRESSION					
Direct all correspondence to: CORRESPONDENCE ADDRESS					
<input type="checkbox"/> Customer Number				Place Customer Number Bar Code Label here	
OR Type Customer Number here					
<input checked="" type="checkbox"/> Firm or Individual Name		F. CHAU & ASSOCIATES, LLP			
Address		1900 Hempstead Turnpike			
Address		Suite 501			
City		State	New York	ZIP	11554
Country		USA	Telephone	516-357-0091	Fax 516-357-0092
ENCLOSED APPLICATION PARTS (check all that apply)					
<input checked="" type="checkbox"/> Specification		Number of Pages		12	
<input type="checkbox"/> Drawing(s)		Number of Sheets		2	
<input type="checkbox"/> Application Data Sheet. See 37 CFR 1.76		<input type="checkbox"/> CD(s), Number			
		<input checked="" type="checkbox"/> Other (specify)		Check for \$75.00	
METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT					
<input checked="" type="checkbox"/> Applicant claims small entity status. See 37 CFR 1.27.				FILING FEE AMOUNT (\$)	
<input checked="" type="checkbox"/> A check or money order is enclosed to cover the filing fees				75.00	
<input type="checkbox"/> The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number:		50-0679			
<input type="checkbox"/> Payment by credit card. Form PTO-2038 is attached.					
The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.					
<input checked="" type="checkbox"/> No.					
<input type="checkbox"/> Yes, the name of the U.S. Government agency and the Government contract number are: _____					

Respectfully submitted,

SIGNATURE

Frank V. DeRosa

Date

2/13/01

TYPED or PRINTED NAME

Frank V. DeRosa

REGISTRATION NO.

(if appropriate)

Docket Number:

43,584

8011-5

TELEPHONE

516-357-0091

## **USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT**

This collection of information is required by 37 CFR 1.51. The information is used by the public to file (and by the PTO to process) a provisional application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing, and submitting the complete provisional application to the PTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Box Provisional Application, Assistant Commissioner for Patents, Washington, D.C. 20231.



02-14-01

A/PROV

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Fallon et al.

Docket: 8011-5

Serial No.: To Be Assigned

Filed: Concurrently herewith

For: BANDWIDTH SENSITIVE DATA COMPRESSION AND  
DECOMPRESSION


Assistant Commissioner for Patents  
Washington, D.C. 20231



CERTIFICATION UNDER 37 C.F.R. § 1.10

I hereby certify that this Provisional Application enclosed herein is being deposited with the United States Postal Service on this date February 13, 2001 in an envelope as "Express Mail Post Office to Addressee" Mail Label Number EL679454069US addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

Dated: 2/13/01

  
Frank V. DeRosa  
Reg. No. 43,584

F. CHAU & ASSOCIATES, LLP  
1900 Hempstead Turnpike  
Suite 501  
East Meadow, New York 11554  
(516) 357-0091  
(516) 357-0092 (FAX)

# **BANDWIDTH SENSITIVE DATA COMPRESSION AND DECOMPRESSION**

## **BACKGROUND**

### **1. Technical Field:**

The present invention relates generally to data compression and decompression and, in particular, to a system and method for compressing and decompressing based on the actual or expected throughput (bandwidth) of a system employing data compression. In addition, the present invention relates generally to a virtual mass-storage data management system.

### **2. Description of Related Art:**

There are a variety of data compression algorithms, both well defined and novel. Many of them have multiple parameters that may take on multiple values thus changing the performance characteristics of the algorithm. For example, a typical dictionary based compression algorithm such as Lempel-Ziv may employ a large dictionary that would yield very good compression ratios but would take a long time to execute. If speed were more important than compression ratio, then one could limit the algorithm to using a small dictionary. This would produce a much faster compression time, but would not compress the data as well.

One challenge in data compression is the selection of one or more optimal compression algorithms to utilize from the variety of available algorithms. A trade-off between speed and efficiency should be considered in determining which algorithm to employ for a given set of data. Algorithms that compress particularly well usually take longer to execute whereas algorithms that execute quickly usually do not compress particularly well.

## **DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS**

Two categories of compression algorithms are defined - asymmetrical data compression algorithms and symmetrical data compression algorithms. An asymmetrical data compression algorithm is referred to herein as one in which the execution time for the compression and decompression routines differ significantly. Either the compression routine may be slow and the decompression routine fast or the compression routine would be fast and the decompression routine would be slow. A

symmetrical data compression algorithm is referred to herein as one in which the execution time for the compression and the decompression routines are similar. When summing the time to perform one compress and one decompress of a data set, the asymmetrical algorithms usually take longer to execute than the symmetrical algorithms, however they usually achieve higher compression ratios.

The decision regarding which routines will be used at compression time (write) and at decompression time (read) is preferably made before or at the time of compression. This is because once data is compressed using a certain algorithm; only the matching decompression routine can be used to decompress the data, regardless of how much processing time is available at the time of decompression.

### **Data Access Profiles**

In a preferred embodiment, the overall throughput (bandwidth) is a determining factor in deciding whether to use an asymmetrical compression algorithm or a symmetrical compression algorithm.

Consider the following three access profiles for a data set.

1. Data is written once (or very few times) but is read many times. This is the case with most programs and websites. This is a very common scenario.
2. Data is written often but read few times. This is the least common scenario.
3. Data is accessed as a read or a write equally often (approximately). This is the case with most user-generated data such as documents and spreadsheets. This is a common scenario.

With access profile number 1, where data is read significantly more times than it is written, the decompression routine would be executed significantly more times than the compression routine. Therefore, if an asymmetrical algorithm is utilized that provides a slow compression routine and a fast decompression routine, the overall system performance would be faster as compared to using a symmetrical compression algorithm. Further, the compression ratio achieved would likely be higher.

With access profile number 2, where data is written significantly more than it is read, the compression routine would be executed significantly more times than the decompression routine. Therefore, if an asymmetrical algorithm is utilized that provides a fast compression routine and a slow decompression routine, the overall system

performance would be faster as compared to using a symmetrical algorithm. In addition, the compression ratio achieved would likely be higher.

With access profile number 3, where data is accessed with a similar number of reads and writes, the compression routine would be executed approximately the same number of times as the decompression routine. Therefore, if a symmetrical algorithm were utilized that provides a fast compression and decompression routine, the overall system performance would be faster as compared to using an asymmetrical algorithm. Although the compression ratio achieved would likely be lower.

The following table summarizes the three data access profiles and the type of compression algorithm that would produce optimum throughput.

Access Profile	Example of Data Type	Compression	Compressed Data Characteristics	Decompression
1. Write few, Read many	Operating systems, Programs, Web sites	Asymmetrical algorithm (Slow compress)	Very high compression ratio	Asymmetrical algorithm (Fast decompress)
2. Write many, Read few	Automatically updated inventory database	Asymmetrical algorithm (Fast compress)	Very high compression ratio	Asymmetrical algorithm (Slow decompress)
3. Similar number of Reads and Writes	User generated documents	Symmetrical algorithm	Standard compression ratio	Symmetrical algorithm

#### Determination of Access Profile

In accordance with the present invention, the access profile of a given data set is with known a priori or determined prior to compression so that the optimum category of compression algorithm can be selected. The selection process may be performed either by the user or automatically by the file system of a host system employing a data compression/decompression system for, e.g., data storage and retrieval and data transmission over a network.

In one embodiment, prior to installing new software, the user could indicate to a data compression controller that the data being installed comprises an application program that would fall under access profile number 1. The system would then preferably utilize an asymmetric compression algorithm employing a slow compression routine and a fast decompression routine. The result would be a one-time penalty during program installation (slow compression), but with fast access to the data on all subsequent executions (reads) of the program, as well as a high compression ratio. In another embodiment, the file system may use certain criteria to select a compression algorithm type. For example, in an effort to achieve the optimal compression ratios, the file system may instruct the data compression controller to use an asymmetric compression algorithm. If the file system determines that the compression controller in a real-time system can maintain the required/requested data throughput using the slow (highly efficient) asymmetrical compression algorithm, then the file system will leave it in this mode. If the file system determines that the compression controller cannot maintain the required/requested data rates, then the file system will command the data compression controller to utilize a fast symmetric compression algorithm. If the required/requested data rates are subsequently lower and the compression controller is maintaining the data rate, then the file system can command the compression controller to use the slow (highly effective) asymmetric compression algorithm. Fig. 1 is a flow diagram of a method implemented by a file management system for providing bandwidth sensitive data compression according to one aspect of the invention.

The present invention may be implemented in any data processing system, device, or apparatus using data compression. For instance, the present invention may be employed in a data storage controller utilizing data compression and decompression to provide accelerated data storage and retrieval from a mass storage device. Exemplary embodiments of such a data storage controller are described, for example, in U.S. patent application No. \_\_\_\_\_ (Attorney Docket No. 8011-14), filed on February 2, 2001 (Express Mail Label No. EL679454205US), which is commonly assigned and fully incorporated herein by reference. For example, a data storage controller having compression controller may be utilized in a personal computer for compressing data stored to a hard disk and decompressing data retrieved from the hard disk.

Fig. 2 illustrates a data storage controller 20 that may utilize a bandwidth sensitive data compression technique described herein. The storage controller 20 comprises a DSP (digital signal processor) 21 (or any other micro-processor device) that implements a data compression/decompression routine. The DSP 21 preferably employs the data compression/decompression techniques disclosed in U.S. Serial No. 09/210,491 entitled "Content Independent Data Compression Method and System," filed on December 11, 1998, which is commonly assigned and which is fully incorporated herein by reference. The data storage controller 20 further comprises at least one programmable logic device 22 (or volatile logic device). The programmable logic device 22 preferably implements the logic (program code) for instantiating and driving both a disk interface 14 and a bus interface 15 and for providing full DMA capability for the disk and bus interfaces 14, 15. Further, upon host computer power-up and/or assertion of a system-level "reset" (e.g., PCI Bus reset), the DSP 21 initializes and programs the programmable logic device 22 before of the completion of initialization of the host computer. This advantageously allows the data storage controller 20 to be ready to accept and process commands from the host computer (via the bus 16) and retrieve boot data from the disk (assuming the data storage controller 20 is implemented as the boot device and the hard disk stores the boot data (e.g., operating system, etc.)).

The data storage controller 20 further comprises a plurality of memory devices including a RAM (random access memory) device 23 and a ROM (read only memory) device 24 (or FLASH memory or other types of non-volatile memory). The RAM device 23 is utilized as on-board cache and is preferably implemented as SDRAM. The ROM device 24 is utilized for non-volatile storage of logic code associated with the DSP 21 - and configuration data used by the DSP 21 to program the programmable logic device 22.

The DSP 21 is operatively connected to the memory devices 23, 24 and the programmable logic device 22 via a local bus 25. The DSP 21 is also operatively connected to the programmable logic device 22 via an independent control bus 26. The programmable logic device 22 provides data flow control between the DSP 21 and the host computer system attached to the bus 16, as well as data flow control between the DSP 21 and the storage device. A plurality of external I/O ports 27 are included for data transmission and/or loading of one or more programmable logic devices. Preferably, the

disk interface 14 driven by the programmable logic device 22 supports a plurality of hard drives.

The storage controller 20 further comprises computer reset and power up circuitry 28 (or "boot configuration circuit") for controlling initialization (either cold or warm boots) of the host computer system and storage controller 20. A preferred boot configuration circuit and preferred computer initialization systems and protocols are described in U.S. Patent Application Serial No. \_\_\_\_\_ (Attorney Docket No. 8011-10), filed on February 2, 2001, (Express Mail Label No. EL679454245US), which is commonly assigned and incorporated herein by reference. Preferably, the boot configuration circuit 28 is employed for controlling the initializing and programming the programmable logic device 22 during configuration of the host computer system (i.e., while the CPU of the host is held in reset). The boot configuration circuit 28 ensures that the programmable logic device 22 (and possibly other volatile or partially volatile logic devices) is initialized and programmed before the bus 16 (such as a PCI bus) is fully reset. In particular, when power is first applied to the boot configuration circuit 28, the boot configuration circuit 28 generates a control signal to reset the local system (e.g., storage controller 20) devices such as a DSP, memory, and I/O interfaces. Once the local system is powered-up and reset, the controlling device (such as the DSP 21) will then proceed to automatically determine the system environment and configure the local system to work within that environment. By way of example, the DSP 21 of the disk storage controller 20 would sense that the data storage controller 20 is on a PCI computer bus (expansion bus) and has attached to it a hard disk on an IDE interface. The DSP 21 would then load the appropriate PCI and IDE interfaces into the programmable logic device 22 prior to completion of the host system reset. Once the programmable logic device 22 is configured for its environment, the boot device controller is reset and ready to accept commands over the computer/expansion bus 16.

It is to be understood that the data storage controller 20 may be utilized as a controller for transmitting data (compressed or uncompressed) to and from remote locations over the DSP I/O ports 27 or system bus 16, for example. Indeed, the I/O ports 27 of the DSP 21 may be used for transmitting data (compressed or uncompressed) that is either retrieved from the disk or received from the host system via the bus 16, to remote



locations for processing and/or storage. Indeed, the I/O ports may be operatively connected to other data storage controllers or to a network communication channels. Likewise, the data storage controller 20 may receive data (compressed or uncompressed) over the I/O ports 27 of the DSP 21 from remote systems that are connected to the I/O ports 27 of the DSP, for local processing by the data storage controller 20. For instance, a remote system may remotely access the data storage controller (via the I/O ports of the DSP or system bus 16) to utilize the data compression, in which case the data storage controller would transmit the compressed data back to the system that requested compression.

In accordance with the present invention, the system (e.g., data storage controller 20) can boot-up in a mode using asymmetrical data compression. It is to be understood that the boot process would not be affected whether the system boots up defaulting to an asymmetrical mode or to a symmetrical mode. This is because during the computer's boot process, it is reading the operating system from the disk, not writing. However, once data is written to the disk using a compression algorithm, it must retrieve and read the data using the corresponding decompression algorithm.

As the user creates, deletes and edits files, the disk controller will preferably utilize an asymmetrical compression routine that provides slow compression and fast decompression. Since using the asymmetrical compression algorithm will take longer than a symmetrical algorithm, the computer's file system will keep track of whether the disk controller has disk accesses pending. If the disk controller does have disk accesses pending, then it is starting to slow down the system and the file management system will then command the disk controller to use a faster symmetrical compression algorithm. If there are no disk accesses pending, the file management system will leave the disk controller in the mode of using the asymmetrical compression algorithm.

If the disk controller was switched to using a symmetrical algorithm, the file management system will preferably switch it back to a default asymmetrical algorithm when the rate of the disk accessed slow to the point where there are no pending disk accesses.

At some point the user may decide to install a piece of software. Before installing the software, the user could indicate to the disk controller (via a software utility) that is

should go into and remain in the mode of using an asymmetric compression algorithm with a slow compression routine and a very fast decompression routine. The disk controller would continue to use the asymmetrical algorithm until commanded otherwise, regardless of the number of pending disk accesses.

The resulting software installation would likely be slower using this method than using a symmetrical algorithm. However, subsequent reads of the software, which happens upon execution of the loaded software, would be much faster than if a symmetrical algorithm were used during the installation. Therefore, the user can pay a one-time speech penalty during program installation and then benefit from faster program executions indefinitely. In addition, using the asymmetrical algorithm during program installation will most likely produce a higher compression ration than would the symmetrical algorithm.

After completing the software installation, the user would then release the disk controller from this “asymmetrical only” mode of operation (via a software utility).

When the user is not commanding the disk controller to remain in a certain mode, the file management system will determine whether the disk controller should use the asymmetrical compression algorithms or the symmetrical compression algorithms. The file management system will use the amount of backlogged disk activity to determine which type of algorithms to use. If there is a lot of backlogged disk activity, then the file management system will preferably command the disk controller to use a faster compression algorithm, even though compression performance may suffer. Otherwise, the file management system will command the disk controller to use the asymmetrical algorithm that will yield greater compression performance.

The following defines a preferred file system format for a disk controller according to one embodiment of the present invention.

# FILE SYSTEM FORMAT

The following defines the file system format for the RealTime disk controller.

## PHYSICAL DISK

Superblock
VBT
↓
VBT
Data
Data
Data
.
.
.
Data
Superblock
Data
Data
.
.
.
Data

## SUPERBLOCK DEFINITION

The Superblock is a grouping of configuration information necessary for the operation of the disk management system. The Superblock will always reside in the first sector of the disk. Additional copies of the Superblock will kept on the disk for backup purposes. The number of copies will depend on the size of the disk. One sector will be allocated for each copy of the Superblock on the disk. This will allow room to add additional parameters in the future. The Superblock will contain the following information:

- Compress Size
- VBT Address
- VBT Size
- Allocation Size
- Number Of Free Sectors (approximate)
- ID ("Magic") Number
- Checksum

Compress Size: This is the maximum uncompressed size of data which we will group together for compression (call this a chunk of data). Therefore if the compress size was set to 16k and a 40k data block is sent to the disk controller for storage, it would be divided into two 16k chunks and one 8k chunk. Each chunk would be compressed separately and have it's own header.

For many compression algorithms increasing the compression size will increase the compression ratio obtained. However when even a single byte is needed from a compressed chunk of data, the whole chunk must be decompressed. This constrains us from using a very large compression size.

VBT Address: The physical address of the virtual block table.

VBT Size: The size of the virtual block table.

Allocation Size: This is the minimum number of contiguous sectors on the disk to reserve for each new data entry. For example, if we were to allow 4 sectors for each allocation and a compressed data entry only took up 1 sector then the remaining 3 sectors would be left unused. Then if that piece of data were to be appended it would have room to get bigger while remaining contiguous on the disk. By keeping the data contiguous it will increase the speed at which the disk can read and write the data.

Although the controller will attempt to keep these unused sectors available for expansion of the data, if the disk were to fill up it would use them for new data entries. In this way we can setup the system to achieve greater speed while not sacrificing any disk space.

Setting the allocation size to 1 sector would effectively disable this feature.

Number Of Free Sectors: This is an indication of the number of physical free sectors remaining on the disk.

ID ("Magic) Number: This number identifies this data as a Superblock.

Checksum: This number will change based on the data in the Superblock and is used for error checking. This number is chosen so that when all the words in the Superblock (including the checksum) are added up they will sum to zero.

#### VBT DEFINITION

The Virtual Block Table (VBT) consists of a number of Sector Map entries. One for each grouping of compressed data (or chunks). The VBT may reside anywhere on the

disk. The size of the VBT will depend on how much data is on the disk. Each sector map entry is 8 bytes.

Although there is only one VBT on the disk, each chunk of compressed data will have a copy of its sector map entry in its header. If the VBT were to become corrupted, scanning the disk for all sector maps could create a new one.

### SECTOR MAP DEFINITION

SECTOR MAP	
Type	2 bits
C Type	3 bits
C Info	19 bits
Sector Count	8 bits
LBA	32 bits

Type: This is the sector map type. A value of "00" will correspond to this sector map definition. Other values are reserved for future redefinitions of the sector map.

C Type: This is the compression type. A value of "000" will correspond to no compression. Other values are to be defined.

C Info: This is the compression information needed for the given compression type. These values are to be defined.

Sector Count: This is the number of physical sectors on the disk that are used for this chunk of compressed data.

LBA: The logical block address (or physical disk address) for this chunk of compressed data.

### DATA

Each data chunk will consist of a header and compressed data. The data chunk may up anywhere from 1 to 256 sectors on the disk.

### DATA BLOCK HEADER

Each compressed chunk of data will be preceded on the disk by a header. The header will contain the following information:

- Sector Map
- VBI

//

- ID (“Magic”) Number
- Checksum

Sector Map: This is a copy of the sector map entry in the VBT for this data chunk.

VBI: This is the Virtual Block Index. It is the index into the VBT that corresponds to this data chunk.

ID (“Magic”) Number: This number identifies this data as a data block header.

Checksum: This number will change based on the data in the header and is used for error checking. This number is chosen so that when all the words in the header (including the checksum) are added up they will sum to zero.

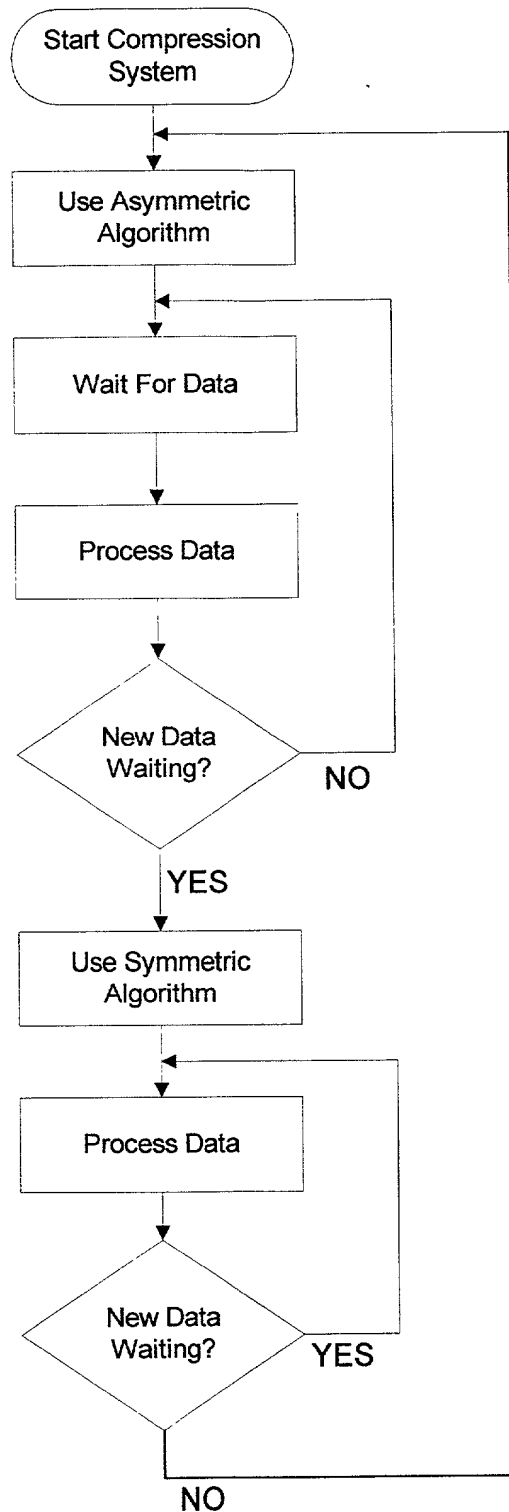
#### NOTE ON PARAMETER SELECTIONS

The virtual size of the disk will depend on the physical size of the disk, the compress size selected, and the expected compression ratio.

For example if we have a 10GB disk and given the selected compress size are expecting a 3:1 compression ratio then the virtual disk size would be 30GB. This will be the maximum amount of uncompressed data that the file system will be able to store on the disk.

If the number chosen is too small then the entire disk will not be utilized. Consider the above example where a system is setup with a 10GB disk and a 30GB virtual size. Assume that in actuality during operation the average compression ratio obtained is 5:1. Whereas this could theoretically allow us to put 50GB on the 10GB disk, in practice we would still only be able to put 30GB of data on disk before we get a “disk full” message. With a 5:1 compression ratio the 30GB of data would only take up 6GB on the disk leaving 4GB unused. Since the operating system would think the disk is full it would not attempt to write any more information to the disk.

If the number chosen is too large then the disk will fill up when the operating system would still indicate that there was space available on the disk. Again consider the above example where a system is setup with a 10GB disk and a 30GB virtual size. Assume that in actuality during operation the average compression ratio obtained is only 2:1. In this case the physical disk would be full after writing 20GB to it but the operating system would still think there are 10GB remaining. If the operating system tried to write more information to the disk an error would occur.



## File System Control

Fig 1

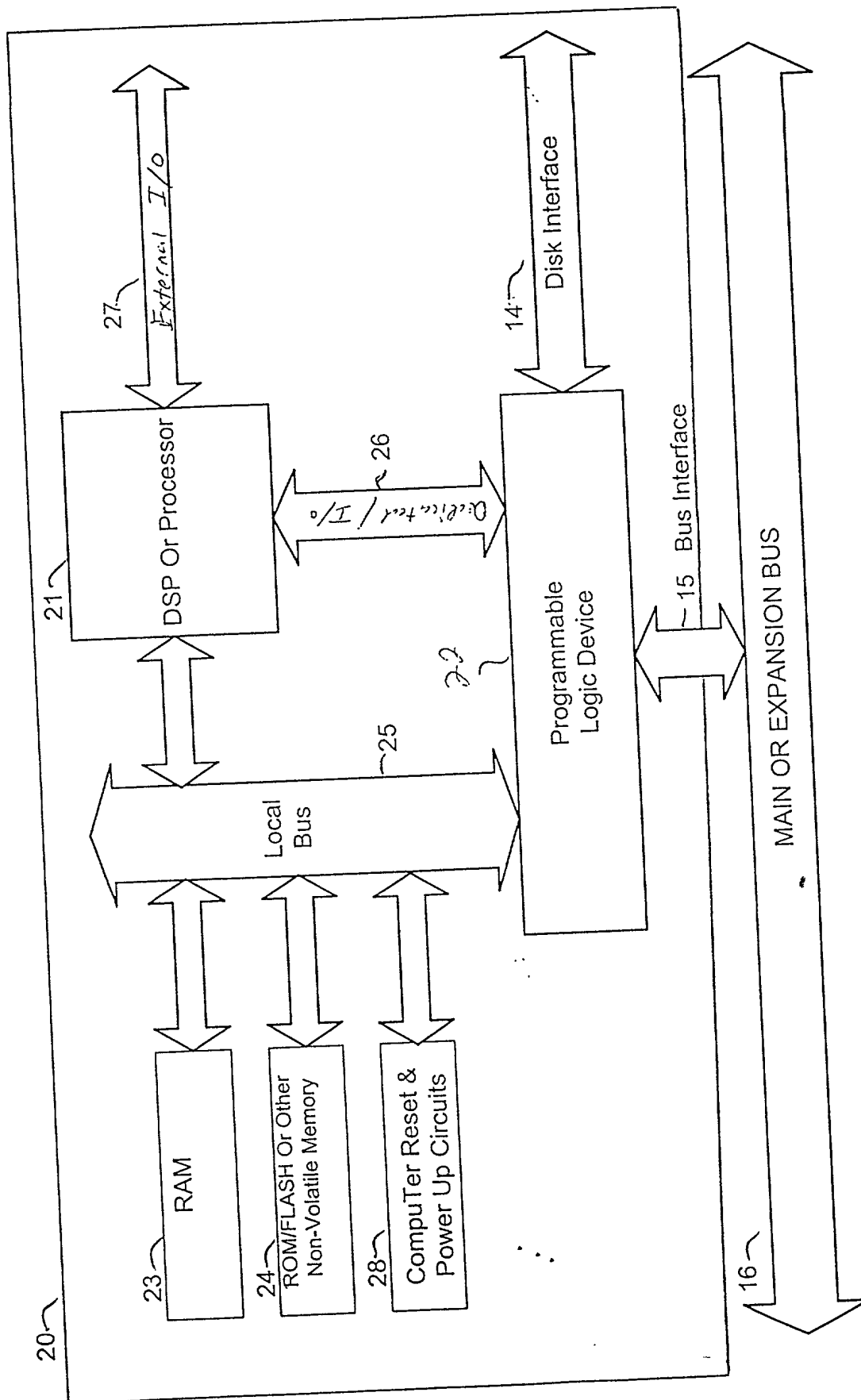


FIGURE 2