

drive geometry. If the image geometry is equal to the drive geometry **1004** then the data is decompressed **1006** using well known "Run-Length" decompressions schemes. Once the data is decompressed it is written **1007** to the head buffers. After data is written to the head buffer, this part of the process is complete. If the image geometry does not match the drive geometry, of step **1004**, an error message **1005** is displayed.

FIG. **11** depicts further detail of the fill compress buffer from file step of the down load image step of the master computer component of the invention. This function fills the compress buffer by reading a head size (maximum sectors times 512) of data from the image file. If an error occurs during the read, an error message is displayed. First, the data is initialized **1101** to provide data for the process. Next, the compress buffer file size is read **1102** and tested **1103** for error. If an error is encountered, it is displayed **1105**, otherwise a null state **1104** is used and the process returns to the download image process of the invention of FIG. **10**.

FIG. **12** depicts further detail of the get byte from compress buffer step of the down load image step of the master computer component of the invention. This process subprogram functions to retrieve a byte from the compress buffer. When the compress buffer is empty, it is refilled by reading data from the image file. First, the data is initialized **1201** for use in the process. Next the compress pointer is tested **1202** to determine if it is greater than the size of the compress buffer. If it is, the compress buffer is filled **1203** from the image file. The compress pointer is then reset **1204**. The byte pointed to by the pointer/counter is returned **1206** and the process returns to the download image process of FIG. **10**. In the event that the compress pointer does not exceed the size of a buffer, test step **1202**, the process goes through a null state **1205** and returns **1206** the byte pointed to by the compress pointer and returns to the download image process.

FIG. **13** depicts further detail of the write data to had buffer step of the down load image step of the master computer component of the invention. This step functions to take the byte of data passed to it and write it to the head buffer the number of times indicated by the compress counter. This is part of the process of decompressing the image file. When the head buffer is full, it flushes the data to the hard disk drive. First, the data required is initialized **1301**. Next, the data is written into the head buffer **1302**. The head buffer pointer is incremented **1303**. A test **1304** is made to determine whether the head buffer is full. If it is, the head buffer is flushed to disk **1305** and the pointers/counters are adjusted **1306**. A test **1308** is then made to determine whether the compress count is complete. If it is, the process returns to the download image of FIG. **10**. If it is not, then the process returns to step **1302** to move a byte into the head buffer. If the head buffer, of step **1304**, is not full, a null state **1307** is passed through before the compress count test of step **1308**.

FIG. **14** depicts further detail of the flush head buffer step of the write data to head buffer step of the master computer component of the invention. This step functions to flush the head buffer data to disk. The entire head buffer is written in one command. All needed pointers, counters, etc., are updated along with screen information. Also, if the broadcast feature is enabled, the head data is broadcast to the image slaves. First the data is initialized **1401** for use in this process. Next, a test **1402** is made to determine if the process is in the broadcast mode. If it is, the head data is broadcast **1403** to all image slave computers. If not, the process goes through a null state **1404**. The head buffer is then written

1405 to disk and the current head and cylinder data is adjusted **1406** prior to returning to the write data to head buffer process of FIG. **13**.

FIG. **15** depicts a flow chart diagram of the slave component of the invention. is a detailed flow diagram of the current preferred embodiment of the Slave computer process of the invention. The IMGSLAVE is the slave component or process of the invention that provides the parallel disk image process. The slave uses an IPX socket to listen for data from the image master. A special header in the data from the master determines the function the slave will perform. The IMGSLAVE program cannot create or restore an image without the IMGBLSTR program. Its function is to listen to the network for data from the IMGBLSTR, to retrieve data from the network and to write it to the slave's local drive. The slave process begins by initializing **1501** data for processing. When a geometry packet is received **1502** from the master the slave responds with an RSVP **1503**. Next, the slave listens for an RSVP acknowledge **1504** from the master. After the receipt of the RSVP acknowledge, a test **1505** is made to determine whether the register for download is valid, if not, a message indicating that the transfer is unusable is sent **1506**. Alternatively, if the register for download is valid, then the data is downloaded **1507** to the slave. Error checking **1508** is performed and errors are displayed **1509** if detected. If no errors are detected, a download complete message is displayed **1510**.

The following is a listing of the computer source code which is the current best mode preferred embodiment of the invention. The reader can, by consulting this source code, learn all that is necessary to produce and use the invention.

The previous description, including the listed source code, describes the current preferred best mode embodiment of the invention as it is performed on personal computers connected through a computer network with or without a computer server. The software programs which are used to practice this invention typically reside in the memory and/or hard disk storage medium of the networked computers. While the current best mode of the invention is used on personal computers, it is not necessary that it be limited in this way. Any computational device which has a long term storage medium, for example: a disk drive, a tape drive, a CD or optical storage medium; and can be networked to other computational devices. The size, configuration or purpose of the computational device does not limit the use of this invention to image long term stored data from one computational device to another in either a peer-to-peer mode or a client/server mode of operation. Furthermore, while this invention is performed, in its current best mode, by software written in the C programming language, alternative computer languages can equivalently used. The software source code provided as part of the disclosure of this patent application, shows in detail how the functional steps of the invention are performed. Of course, it is contemplated that the inventive concept of this invention may be implemented through other techniques and in other embodiments and in other computer languages. The computer source code is provided to describe the best mode of operation of the invention, such a best mode may evolve and change over time, after the filing of this application without altering the fundamental inventive concept of the method, which is the imaging of computer data from one computer to one or more others over a network using a peer-to-peer method and still remain compatible with a client/server mode of operation, without requiring special purpose network server hardware.

We claim:

1. A method for imaging data between two or more digital computer systems across a computer network the method steps comprising: