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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 March 2001 (08.03.2001)

PCT

(10) International Publication Number
WO 01/16743 A2

(51) International Patent Classification[7]: G06F 9/52

(21) International Application Number: PCT/US00/24298

(22) International Filing Date: 31 August 2000 (31.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/152,151	31 August 1999 (31.08.1999)	US
60/220,974	26 July 2000 (26.07.2000)	US
60/220,748	26 July 2000 (26.07.2000)	US

(71) Applicant *(for all designated States except US)*: **TIMES N SYSTEMS, INC.** [US/US]; Bldg. B, Suite P, 1908 Kramer Lane, Austin, TX 78758 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **MILLER, Chris** [US/US]; 4807 Fieldstone Drive, Austin, TX 78735 (US).

(74) Agent: **BRUCKNER, John, J.**; Fulbright & Jaworski, L.L.P., 600 Congress Avenue, Suite 2400, Austin, TX 78701 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *Without international search report and to be republished upon receipt of that report.*

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

WO 01/16743 A2

(54) Title: SHARED MEMORY DISK

(57) Abstract: Methods, systems and devices are described for a shared memory disk (MEMDISK). A method, includes setting aside a particular range of a shared memory as a MEMDISK; and providing control for each of several operating systems that compose processing nodes coupled to said shared memory such that none of the processing nodes will attempt to utilize pages within said particular region for non-MEMDISK purposes. The methods, systems and devices provide advantages because the speed and scalability of parallel processor systems is enhanced.

SHARED MEMORY DISK

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of computing systems in which multiple processors share memory but in which each is provided with separate access to input-output (I/O) devices such as disks. More particularly, the invention relates to computer science techniques that utilize a shared memory disk (MEMDISK).

2. Discussion of the Related Art

The clustering of workstations is a well-known art. In the most common cases, the clustering involves workstations that operate almost totally independently, utilizing the network only to share such services as a printer, license-limited applications, or shared files.

In more-closely-coupled environments, some software packages (such as NQS) allow a cluster of workstations to share work. In such cases the work arrives, typically as batch jobs, at an entry point to the cluster where it is queued and dispatched to the workstations on the basis of load.

In both of these cases, and all other known cases of clustering, the operating system and cluster subsystem are built around the concept of message-passing. The term message-passing means that a given workstation operates on some portion of a job until communications (to send or receive data, typically) with another workstation is necessary. Then, the first workstation prepares and communicates with the other workstation.

Another well-known art is that of clustering processors within a machine, usually called a Massively Parallel Processor or MPP, in which the techniques are essentially identical to those of clustered workstations. Usually, the bandwidth and latency of the interconnect network of an MPP are more highly optimized, but the system operation is the same.

In the general case, the passing of a message is an extremely expensive operation; expensive in the sense that many CPU cycles in the sender and

receiver are consumed by the process of sending, receiving, bracketing, verifying, and routing the message, CPU cycles that are therefore not available for other operations. A highly streamlined message-passing subsystem can typically require 10,000 to 20,000 CPU cycles or more.

There are specific cases wherein the passing of a message requires significantly less overhead. However, none of these specific cases is adaptable to a general-purpose computer system.

Message-passing parallel processor systems have been offered commercially for years but have failed to capture significant market share because of poor performance and difficulty of programming for typical parallel applications. Message-passing parallel processor systems do have some advantages. In particular, because they share no resources, message-passing parallel processor systems are easier to provide with high-availability features. What is needed is a better approach to parallel processor systems.

There are alternatives to the passing of messages for closely-coupled cluster work. One such alternative is the use of shared memory for inter-processor communication.

Shared-memory systems, have been much more successful at capturing market share than message-passing systems because of the dramatically superior performance of shared-memory systems, up to about four-processor systems. In Search of Clusters, Gregory F. Pfister 2nd ed. (January 1998) Prentice Hall Computer Books, ISBN: 0138997098 describes a computing system with multiple processing nodes in which each processing node is provided with private, local memory and also has access to a range of memory which is shared with other processing nodes. The disclosure of this publication in its entirety is hereby expressly incorporated herein by reference for the purpose of indicating the background of the invention and illustrating the state of the art.

However, providing high availability for traditional shared-memory systems has proved to be an elusive goal. The nature of these systems, which share all code and all data, including that data which controls the shared operating systems, is incompatible with the separation normally required for

high availability. What is needed is an approach to shared-memory systems that improves availability.

Although the use of shared memory for inter-processor communication is a well-known art, prior to the teachings of U.S. Ser. No. 09/273,430, filed March 19, 1999, entitled Shared Memory Apparatus and Method for Multiprocessing Systems, the processors shared a single copy of the operating system. The problem with such systems is that they cannot be efficiently scaled beyond four to eight way systems except in unusual circumstances. All known cases of said unusual circumstances are such that the systems are not good price-performance systems for general-purpose computing.

The entire contents of U.S. Patent Applications 09/273,430, filed March 19, 1999 and PCT/US00/01262, filed January 18, 2000 are hereby expressly incorporated by reference herein for all purposes. U.S. Ser. No. 09/273,430, improved upon the concept of shared memory by teaching the concept which will herein be referred to as a tight cluster. The concept of a tight cluster is that of individual computers, each with its own CPU(s), memory, I/O, and operating system, but for which collection of computers there is a portion of memory which is shared by all the computers and via which they can exchange information. U.S. Ser. No. 09/273,430 describes a system in which each processing node is provided with its own private copy of an operating system and in which the connection to shared memory is via a standard bus. The advantage of a tight cluster in comparison to an SMP is "scalability" which means that a much larger number of computers can be attached together via a tight cluster than an SMP with little loss of processing efficiency.

What is needed are improvements to the concept of the tight cluster. What is also needed is an expansion of the concept of the tight cluster.

Another well-known art is the use of memory caches to improve performance. Caches provide such a significant performance boost that most modern computers use them. At the very top of the performance (and price) range all of memory is constructed using cache-memory technologies. However, this is such an expensive approach that few manufacturers use it. All manufacturers of personal computers (PCs) and workstations use caches except

for the very low end of the PC business where caches are omitted for price reasons and performance is, therefore, poor.

Caches, however, present a problem for shared-memory computing systems; the problem of coherence. As a particular processor reads or writes a word of shared memory, that word and usually a number of surrounding words are transferred to that particular processor's cache memory transparently by cache-memory hardware. That word and the surrounding words (if any) are transferred into a portion of the particular processor's cache memory that is called a cache line or cache block.

If the transferred cache line is modified by the particular processor, the representation in the cache memory will become different from the value in shared memory. That cache line within that particular processor's cache memory is, at that point, called a "dirty" line. The particular processor with the dirty line, when accessing that memory address will see the new (modified) value. Other processors, accessing that memory address will see the old (unmodified) value in shared memory. This lack of coherence between such accesses will lead to incorrect results.

Modern computers, workstations, and PCs which provide for multiple processors and shared memory, therefore, also provide high-speed, transparent cache coherence hardware to assure that if a line in one cache changes and another processor subsequently accesses a value which is in that address range, the new values will be transferred back to memory or at least to the requesting processor.

Caches can be maintained coherent by software provided that sufficient cache-management instructions are provided by the manufacturer. However, in many cases, an adequate arsenal of such instructions are not provided. Moreover, even in cases where the instruction set is adequate, the software overhead is so great that no examples of are known of commercially successful machines which use software-managed coherence.

SUMMARY OF THE INVENTION

A goal of the invention is to simultaneously satisfy the above-discussed requirements of improving and expanding the tight cluster concept which, in the case of the prior art, are not satisfied.

One embodiment of the invention is based on a method, comprising: setting aside a particular range of a shared memory as a MEMDISK; and providing control for each of several operating systems that compose processing nodes coupled to said shared memory such that none of the processing nodes will attempt to utilize pages within said particular region for non-MEMDISK purposes. Another embodiment of the invention is based on a system, comprising a multiplicity of processors, each with some private memory and the multiplicity with some shared memory, interconnected and arranged such that memory accesses to a first set of address ranges will be to local, private memory whereas memory accesses to a second set of address ranges will be to shared memory, and arranged such that at least some of said processors are provided with input-output subsystems and that said input-output (I/O) traffic started by one processor for an I/O device attached to another processor will be started by inter-processor signals but continued via use of a portion of shared memory accessed via I/O driver emulation means. Another embodiment of the invention is based on a computer system which provides operating system extensions to perform disk input-output (I/O) functions in a shared-memory environment, where said extensions perform the functions with direct Load and Store operations. Another embodiment of the invention is based on a computer system that provides system-wide registration of shared-memory disk partitions at all of a multiplicity of processing nodes within the system. Another embodiment of the invention is based on a computer system that provides system-wide registration of shared-memory disk access methodologies at all of a multiplicity of processing nodes within the system. Another embodiment of the invention is based on a computer system that provides system-wide status of shared-memory disk operations at all of a multiplicity of processing nodes within the system. Another embodiment of the invention is based on a computer system that provides for multiple instantiations in a shared-memory environment of a disk

to satisfy disk I/O operations for all system members, transparent to the Operating System. Another embodiment of the invention is based on a computer system that provides for caching of data system-wide in a shared-memory environment to satisfy disk I/O functions for all system members, transparent to the Operating System. Another embodiment of the invention is based on a computer system that provides application appropriate access methodologies based on system-wide partitioning of a data store in a shared-memory environment.

These, and other goals and embodiments of the invention will be better appreciated and understood when considered in conjunction with the following description and the accompanying drawings. It should be understood, however, that the following description, while indicating preferred embodiments of the invention and numerous specific details thereof, is given by way of illustration and not of limitation. Many changes and modifications may be made within the scope of the invention without departing from the spirit thereof, and the invention includes all such modifications.

BRIEF DESCRIPTION OF THE DRAWINGS

A clear conception of the advantages and features constituting the invention, and of the components and operation of model systems provided with the invention, will become more readily apparent by referring to the exemplary, and therefore nonlimiting, embodiments illustrated in the drawings accompanying and forming a part of this specification, wherein like reference characters (if they occur in more than one view) designate the same parts. It should be noted that the features illustrated in the drawings are not necessarily drawn to scale.

FIG. 1 illustrates a block schematic view of a system, representing an embodiment of the invention.

FIG. 2 illustrates a block schematic view of another system, representing an embodiment of the invention.

DESCRIPTION OF PREFERRED EMBODIMENTS

The invention and the various features and advantageous details thereof are explained more fully with reference to the nonlimiting embodiments that are illustrated in the accompanying drawings and detailed in the following description of preferred embodiments. Descriptions of well known components and processing techniques are omitted so as not to unnecessarily obscure the invention in detail.

The teachings of U.S. Ser. No. 09/273,430 include a system which is a single entity; one large supercomputer. The invention is also applicable to a cluster of workstations, or even a network.

The invention is applicable to systems of the type of Pfister or the type of U.S. Ser. No. 09/273,430 in which each processing node has its own copy of an operating system. The invention is also applicable to other types of multiple processing node systems.

The context of the invention can include a tight cluster as described in U.S. Ser. No. 09/273,430. A tight cluster is defined as a cluster of workstations or an arrangement within a single, multiple-processor machine in which the processors are connected by a high-speed, low-latency interconnection, and in which some but not all memory is shared among the processors. Within the scope of a given processor, accesses to a first set of ranges of memory addresses will be to local, private memory but accesses to a second set of memory address ranges will be to shared memory. The significant advantage to a tight cluster in comparison to a message-passing cluster is that, assuming the environment has been appropriately established, the exchange of information involves a single STORE instruction by the sending processor and a subsequent single LOAD instruction by the receiving processor.

The establishment of the environment, taught by U.S. Ser. No. 09/273,430 and more fully by companion disclosures (U.S. Provisional Application Ser. No. 60/220,794, filed July 26, 2000; U.S. Provisional Application Ser. No. 60/220,748, filed July 26, 2000; WSGR 15245-711; WSGR 15245-712; WSGR 15245-713; WSGR 15245-715; WSGR 15245-716; WSGR 15245-717; WSGR 15245-719; and WSGR 15245-720, the entire

contents of all which are hereby expressly incorporated herein by reference for all purposes) can be performed in such a way as to require relatively little system overhead, and to be done once for many, many information exchanges. Therefore, a comparison of 10,000 instructions for message-passing to a pair of instructions for tight-clustering, is valid.

The invention can include providing highly-efficient operating system control for a tight cluster system. Among the means of controlling shared memory in such a tight cluster for improved performance is the provision of a shared memory disk (MEMDISK) which can be shared among the various processes and processors of the cluster.

In the context of a computing system in which multiple processing nodes share some memory, and where each node has access to separate input-output (I/O), the invention can include the utilization of shared memory to achieve OS-transparent high-speed access to disk. The invention can also provide the ability to substitute memory accesses for disk accesses under certain circumstances, while maintaining OS-transparency to the substitution.

The invention is applicable to the kind of systems taught by U.S. Ser. No. 09/273,430 and is also applicable to other architectures such as NUMA, CC-NUMA, and other machines in which each processor or processor aggregation is provided with separate I/O. In the case of NUMA and CC-NUMA machines, all of memory is shared and there is one copy of the operating system. In U.S. Ser. No. 09/273,430 only a portion of the memory is shared and each node is provided with a separate copy of the operating system.

In a computing system which is provided with multiple nodes, and in which several of the multiple processing nodes are provided with separate I/O paths, the paths from each of the processing nodes to non-local I/O is generally one of several types. These types include: (1) each of the separate processing nodes restricted from reaching I/O attached to other nodes; (2) each of the processing nodes acting as a surrogate for another node, providing I/O capability to the requesting node in a proxy fashion; (3) in schemes such as NUMA and CC-NUMA, where a processing node may be provided with a path, via directory and cache-coherence means, whereby the single common

operating system can reach I/O on the other nodes; or (4) a system external provision providing a common I/O resource, such as Fibre Channel I/O, a twin-tailed SCSI disk facility, or standard networking facility.

The present invention can be used in the context of the environment described in U.S. Ser. No. 09/273,430 where multiple computers are provided with means to selectively address a first set of memory address ranges which will be to private memory and a second set of memory ranges which will be to shared memory. The invention can include: setting aside a particular range of shared memory as a MEMDISK; providing control means for each of the several operating systems such that none will attempt to utilize pages within this region for normal shared-memory purposes.

A device driver interface can be utilized that is responsive to disk I/O operations to a particular disk and which will translate said I/O operations to LOAD and STORE operations to said region of memory. I/O operations can be intercepted which may otherwise be directed to a physical disk and redirect those operations to said region of memory.

In a tight cluster, each processor is provided with its own memory and its own operating system or micro-kernel and may be provided with its own I/O subsystem. The present invention is applicable in such a system where more than one such processor is provided with I/O and in which the file system is visible to all or a multiplicity of processors.

When a particular processor develops traffic for an I/O device not on its I/O subsystem, it requests that the processor owning the particular I/O subsystem satisfy the request. In this way, the processors serve as I/O processors to each other.

However, in this invention the requesting process sends to the service process a request sufficient to initiate and completely satisfy the particular I/O block request. Rather than the two processors thereafter interrupting or polling each other during the block transfer at each disk READ or WRITE, the movement is accomplished via the shared MEMDISK.

The MEMDISK, as described here, teaches three major new concepts. First, the purpose is not to minimize access time, although that minimization

occurs, but rather is to minimize interference between a pair of processors so that a first process on a first processor may deliver data to the MEMDISK as it becomes available from the file system and a second process on a second processor may read the data from the shared memory region without interrupting the first processor after the initial START I/O process is begun. MEMDISK WRITES are protected by semaphores to eliminate data corruption, and MEMDISK semaphores are applied on a region-by-region basis so that artificial interference is also eliminated.

The second major concept is that of disk-to-memory dynamic redirection. The initial START I/O to a particular file is delivered, by the device driver, to the processor owning the I/O device. Subsequent I/O to that file is dynamically redirected to the MEMDISK by the device driver and is managed transparent to all other processes including the remainder of the operating system. When the particular file access is completed, subsequent I/O operations to other files will be satisfied by MEMDISK if present there, otherwise subsequent I/O operations to other files will be directed to the appropriate I/O owner by the file subsystem.

A third major teaching of this invention is that of "aging" of data within the MEMDISK region. Once I/O data is placed in the region, it stays there and is available to any process on any processor until the MEMDISK region becomes full, at which time older (least-recently-used) information is replaced by newer information. The portion of the MEMDISK reflecting data on a particular I/O device is kept coherent by the owner of the I/O device.

This invention describes a means outside the operating system but within shared memory to provide any node in a shared-memory computing system access to memory, which is physically attached to another node. In a preferred embodiment, each system is provided with some local, private memory and a separate copy of the operating system. In this preferred embodiment, each of several nodes is provided with its own I/O channel to disks and other I/O units.

Referring to FIGS. 1-2, in a preferred embodiment, the operating system in each node is augmented with external extensions, not part of the operating

system, which provide means by which said extensions have capabilities to reach shared memory and to communicate to other said nodes via Load and Store instructions to shared memory.

In this embodiment, the invention includes other extensions called shared-memory-disk (SMD) extensions, which make use of primitives and by which disk I/O functions which originate in applications and which are then passed to the operating system are processed. Said disk I/O functions, arriving at the Operating System, are processed by said SMD extensions and are translated to shared-memory Load and Store instructions. This effects disk I/O transactions issued by the Operating System into Load and Store transfers via shared memory, and so satisfies the Operating System I/O request transparently.

An additional key teaching of this invention is the system-wide registration of all shared-memory disk partitions at all processing nodes within the system, with an access methodology dependent upon performance requirements. An additional key teaching of this invention is the use of a means herein called "software RAID" to assure high-availability of the disk I/O portion of the computing system. An additional key feature of this system is the provision, in shared memory, for retention of data which passes through shared memory so that subsequent accesses to said retained images can be satisfied solely by memory transfers thus achieving dramatic performance improvements.

Each of the computing nodes that participates in constructing an SMD instantiation contributes a node-local data store to the instantiation hereafter referred to as a SmdBlock. The data store is not limited as to type, and may be private memory, paged virtual memory, local disk I/O, etc. A preferred implementation provides this data store with commodity disk drives. These SmdBlocks are collected and logically bound into a SmdDisk in a repeatable order. This order is subsequently used to base an access methodology most appropriate to use (e.g., ordinally-arranged contiguity for best random access, interleaved contiguity for best sequential access, distributed contiguity for de-clustered failover or software Raid applications, etc.). The SmdDisk and

SmdBlocks are identified within the immediate community of computing nodes by a community-unique signature.

Upon instantiation of a SmdDisk, a known address (or known key to address translation) is searched for a single data structure (hereafter referred to as the SmdAnchor) containing identifying patterns, validity and version information, and a shared-memory pointer to a linked list of SmdDisk control structures. If the SmdAnchor is not yet present, one is created. If the SmdAnchor is present, the SmdDisk list is searched for a matching signature, which informs the instantiation that the supporting shared-memory data structures have been created by another computing node, and it may now access same as appropriate. If the SmdDisk structure is not present, it is created and linked into the SmdAnchor list, for future SmdDisk and SmdBlock instantiations to find. The passive nature of the preferred implementation avoids inter-node messaging and communications overhead, while simplifying state transitions.

Upon instantiation of a SmdBlock, the SmdAnchor, and its corresponding SmdDisk list, is searched for a matching signature. If one is not found, the search is repeated ad infinitum, with an appropriate delay to allow for another computing node or computing process to create the SmdDisk structure. If and when one is found, the SmdBlock inserts its computing node-unique information into the SmdDisk structure. When all SmdBlocks that contribute to a SmdDisk have inserted their information into the control structure, the SmdDisk is ready for use, and thusly describes a distributed data store.

When an I/O from the Operating System arrives, the SmdDisk structure can be examined and the operation broken into a list of transactions matching the SmdBlock node-locality and requirements of the SmdDisk data store access methodology. These transactions are then sent to each of the SmdBlock nodes for fulfillment using shared-memory Load and Store operations to transfer the data between nodes.

Referring to FIG. 2, on read transactions, the SmdDisk allocates shared-memory data areas for each of the transactions and sends read commands to the SmdBlock nodes referencing said data areas. The SmdBlock nodes perform the

node-local data store reads, moving the data into said areas, then return status to the original node. When the originating node collects all SmdBlock transaction statuses (either passively or via the mechanisms described in [2]), the operation is considered complete and the data contained in the shared-memory data areas can be used to fulfill the original request.

Still referring to FIG. 2, on write transactions, the SmdDisk allocates shared-memory data areas for each of the transactions and moves the write data into said areas, according to the access methodology, then sends write commands to the appropriate SmdBlock nodes. The SmdBlock nodes perform the data store writes, moving the data from said areas into their data store, then return status to the originating node. When the originating node collects all SmdBlock transaction statuses, the operation is complete.

On both read and write transactions, the shared-memory areas used to transfer the data can be kept resident in shared memory, and subsequently used to satisfy read requests by any node connected to the shared-memory system. This implements a shared-memory cache of the shared and distributed disk. The management of a shared-memory cache requires the use of a shared-memory mutual exclusion mechanisms to maintain coherency. This shared nature allows multiple nodes to access, and contribute to, the shared cache, resulting in being able to completely satisfy Disk I/O operations with Load and Store operations. The control structures to manage said shared-memory cache can be kept within the SmdDisk control structure, allowing the above mentioned search and access methods to be used. This cache can result in a significant performance enhancements, as the latency and transfer time for a node to deliver data into shared memory, as well as the access of the physical data store, is eliminated.

An extension of the invention allows different access methodologies to be implemented for differing requirements, without affecting the Operating System perceived implementation. In a preferred implementation for optimal random access, the data stores contributed by all SmdBlock nodes can be ordinally arranged as contiguous data stores using commodity disk drives. This allows the head movement latency to be mitigated across the SmdBlock contributors, providing greatly reduced access latency and improved

performance. In a preferred implementation for optimal sequential data access, the data stores can be arranged in a striping, or Raid0 configuration, thus improving throughput by effecting concurrent media access. In a preferred implementation for high availability, the SmdBlocks can implement a "software RAID" by striping in a Chained Declustering methodology allowing head movement mitigation and concurrent access (at the expense of duplicate data store space). In another preferred implementation for high availability, the SmdBlocks can implement another "software RAID" by striping the SmdBlock nodes' contributions in a RAID5 methodology, balancing the computing cost of data parity generation and checking with the improvements provided by head movement mitigation and concurrent access. In the preferred embodiment, the shared-memory caching is used in conjunction with a "software RAID" methodology, for a complete high performance fault tolerant implementation. The Operating System perceived implementation of the data store remains, in all cases, transparent.

While not being limited to any particular performance indicator or diagnostic identifier, preferred embodiments of the invention can be identified one at a time by testing for the substantially highest performance. The test for the substantially highest performance can be carried out without undue experimentation by the use of a simple and conventional benchmark (speed) experiment.

The term substantially, as used herein, is defined as at least approaching a given state (e.g., preferably within 10% of, more preferably within 1% of, and most preferably within 0.1% of). The term coupled, as used herein, is defined as connected, although not necessarily directly, and not necessarily mechanically. The term means, as used herein, is defined as hardware, firmware and/or software for achieving a result. The term program or phrase computer program, as used herein, is defined as a sequence of instructions designed for execution on a computer system. A program may include a subroutine, a function, a procedure, an object method, an object implementation, an executable application, an applet, a servlet, a source code, an object code, and/or other sequence of instructions designed for execution on a computer system.

Practical Applications of the Invention

A practical application of the invention that has value within the technological arts is waveform transformation. Further, the invention is useful in conjunction with data input and transformation (such as are used for the purpose of speech recognition), or in conjunction with transforming the appearance of a display (such as are used for the purpose of video games), or the like. There are virtually innumerable uses for the invention, all of which need not be detailed here.

Advantages of the Invention

A system, representing an embodiment of the invention, can be cost effective and advantageous for at least the following reasons. The invention improves the speed of parallel computing systems. The invention improves the scalability of parallel computing systems.

All the disclosed embodiments of the invention described herein can be realized and practiced without undue experimentation. Although the best mode of carrying out the invention contemplated by the inventors is disclosed above, practice of the invention is not limited thereto. Accordingly, it will be appreciated by those skilled in the art that the invention may be practiced otherwise than as specifically described herein.

For example, although the shared memory disk described herein can be a separate module, it will be manifest that the shared memory disk may be integrated into the system with which it is associated. Furthermore, all the disclosed elements and features of each disclosed embodiment can be combined with, or substituted for, the disclosed elements and features of every other disclosed embodiment except where such elements or features are mutually exclusive.

It will be manifest that various additions, modifications and rearrangements of the features of the invention may be made without deviating from the spirit and scope of the underlying inventive concept. It is intended that the scope of the invention as defined by the appended claims and their equivalents cover all such additions, modifications, and rearrangements.

The appended claims are not to be interpreted as including means-plus-function limitations, unless such a limitation is explicitly recited in a given claim using the phrase "means for." Expedient embodiments of the invention are differentiated by the appended subclaims.

CLAIMS

What is claimed is:

1. A method, comprising:

setting aside a particular range of a shared memory as a MEMDISK; and

providing control for each of several operating systems that compose processing nodes coupled to said shared memory such that none of the processing nodes will attempt to utilize pages within said particular region for non-MEMDISK purposes.

2. The method of claim 1, wherein a first process on a first processor delivers data to the MEMDISK as data becomes available from a file system and a second process on a second processor reads data from the MEMDISK without interrupting the first processor after an initial START I/O process is begun.

3. The method of claim 2, wherein MEMDISK WRITES are protected by semaphores to eliminate data corruption.

4. The method of claim 3, wherein MEMDISK semaphores are applied on a region-by-region basis so that artificial interference is eliminated.

5. The method of claim 1, wherein an initial START I/O to a particular file is delivered, by a device driver, to a processor owning the I/O device and subsequent I/O to said particular file is dynamically redirected to the MEMDISK by the device driver and is managed transparent to all other processes including a remainder of the operating system.

6. The method of claim 5, wherein, when an access to the particular file is completed, subsequent I/O operations to other files will be satisfied by

MEMDISK if present there, otherwise subsequent I/O operations to other files will be directed to an appropriate I/O owner by a file subsystem.

7. The method of claim 1, wherein once I/O data is placed in the MEMDISK, data stays there and is available to a process on a processor until the MEMDISK becomes full, at which time older information is replaced by newer information.

8. The method of claim 7, wherein a portion of the MEMDISK reflecting data on a particular I/O device is kept coherent by an owner of the particular I/O device.

9. An electronic media, comprising: a computer program adapted to set aside a particular range of a shared memory as a MEMDISK; and provide control for each of several operating systems that compose processing nodes coupled to said shared memory such that none of the processing nodes will attempt to utilize pages within said particular region for non-MEMDISK purposes.

10. A computer program comprising computer program means adapted to perform the steps of setting aside a particular range of a shared memory as a MEMDISK; and providing control for each of several operating systems that compose processing nodes coupled to said shared memory such that none of the processing nodes will attempt to utilize pages within said particular region for non-MEMDISK purposes when said computer program is run on a computer.

11. A computer program as claimed in claim 10, embodied on a computer-readable medium.

12. A system, comprising a multiplicity of processors, each with some private memory and the multiplicity with some shared memory, interconnected and arranged such that memory accesses to a first set of address ranges will be

to local, private memory whereas memory accesses to a second set of address ranges will be to shared memory, and arranged such that at least some of said processors are provided with input-output subsystems and that said input-output (I/O) traffic started by one processor for an I/O device attached to another processor will be started by inter-processor signals but continued via use of a portion of shared memory accessed via I/O driver emulation means.

13. The system of claim 12, wherein the transition from physical I/O to memory-converted I/O is performed automatically at the I/O driver level so that all application and all other operating system interfaces are maintained so that it is fully transparent.

14. The system of claim 12, wherein said flow of I/O via shared memory is such that the two processors involved do not need to interrupt each other for the satisfying of a given I/O request after it is started and until it is complete.

15. The system of claim 12, wherein said shared-memory region (MEMDISK) retains the information placed therein for use by other processes.

16. The system of claim 15, wherein said information is replaced on a least-recently-used basis as the MEMDISK becomes full.

17. A computer system which provides operating system extensions to perform disk input-output (I/O) functions in a shared-memory environment, where said extensions perform the functions with direct Load and Store operations.

18. The computer system of claim 17, wherein each of said multiplicity of processors includes a separate operating system and a separate input-output.

19. A computer system that provides system-wide registration of shared-memory disk partitions at all of a multiplicity of processing nodes within the system.

20. The computer system of claim 19, wherein each of said multiplicity of processors includes a separate operating system and a separate input-output.

21. A computer system that provides system-wide registration of shared-memory disk access methodologies at all of a multiplicity of processing nodes within the system.

22. The computer system of claim 21, wherein each of said multiplicity of processors includes a separate operating system and a separate input-output.

23. A computer system that provides system-wide status of shared-memory disk operations at all of a multiplicity of processing nodes within the system.

24. The computer system of claim 23, wherein each of said multiplicity of processors includes a separate operating system and a separate input-output.

25. A computer system that provides for multiple instantiations in a shared-memory environment of a disk to satisfy disk I/O operations for all system members, transparent to the Operating System.

26. The computer system of claim 25, wherein each of a multiplicity of processors includes a separate operating system and a separate input-output.

27. A computer system that provides for caching of data system-wide in a shared-memory environment to satisfy disk I/O functions for all system members, transparent to the Operating System.

28. The computer system of claim 27, wherein each of a multiplicity of processors includes a separate operating system and a separate input-output.

29. A computer system that provides application appropriate access methodologies based on system-wide partitioning of a data store in a shared-memory environment.

30. The computer system of claim 29, wherein each of a multiplicity of processors includes a separate operating system and a separate input-output.

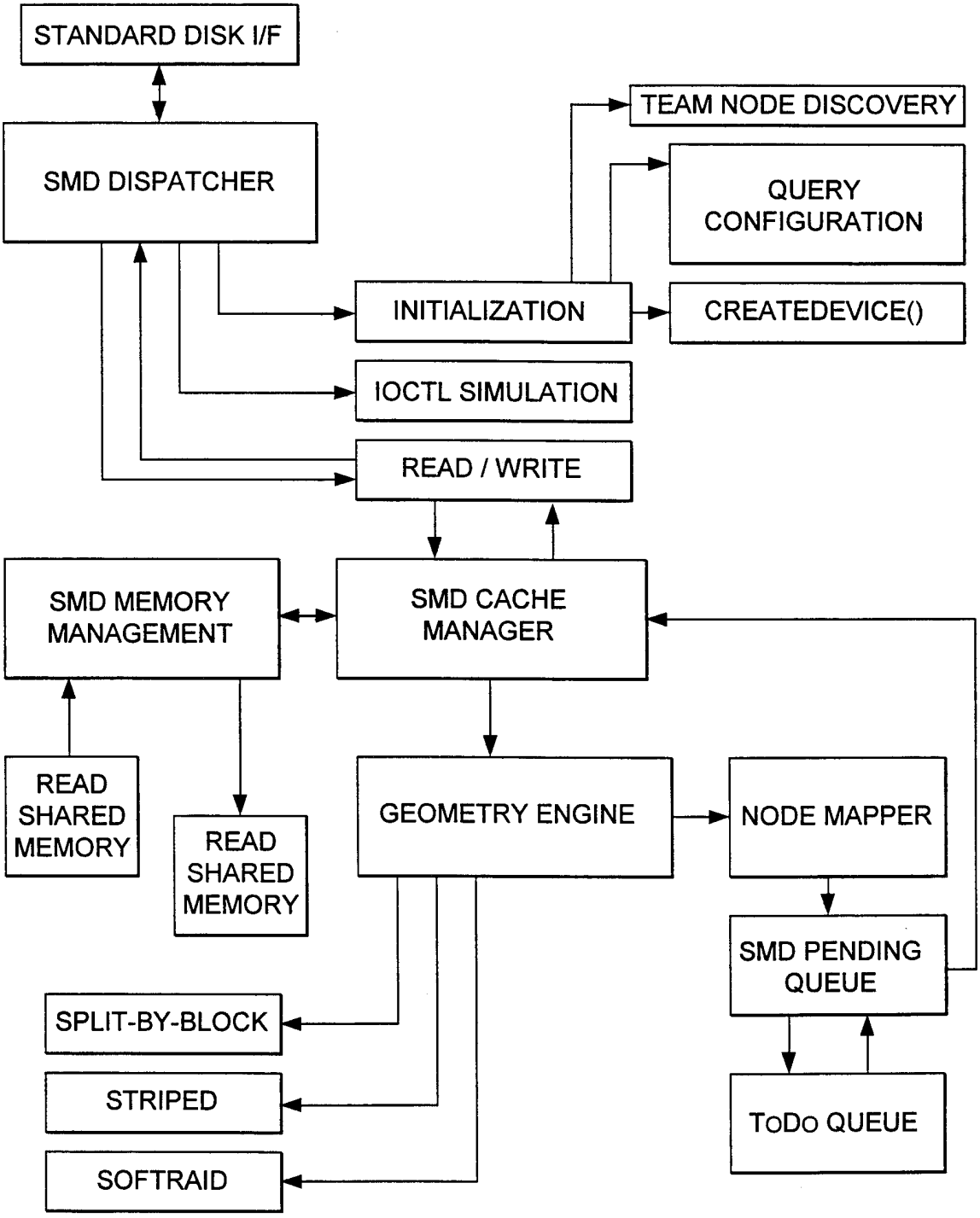
STANDARD DISK I/F
SMD DISPATCHER
TEAM NODE DISCOVERY
QUERY CONFIGURATION
INITIALIZATION
CREATEDEVICE()
IOCTL SIMULATION
READ / WRITE
SMD MEMORY MANAGEMENT
SMD CACHE MANAGER
READ SHARED MEMORY
READ SHARED MEMORY
GEOMETRY ENGINE
NODE MAPPER
SMD PENDING QUEUE
SPLIT-BY-BLOCK
STRIPED
SOFTRAID
ToDo QUEUE

FIG. 1

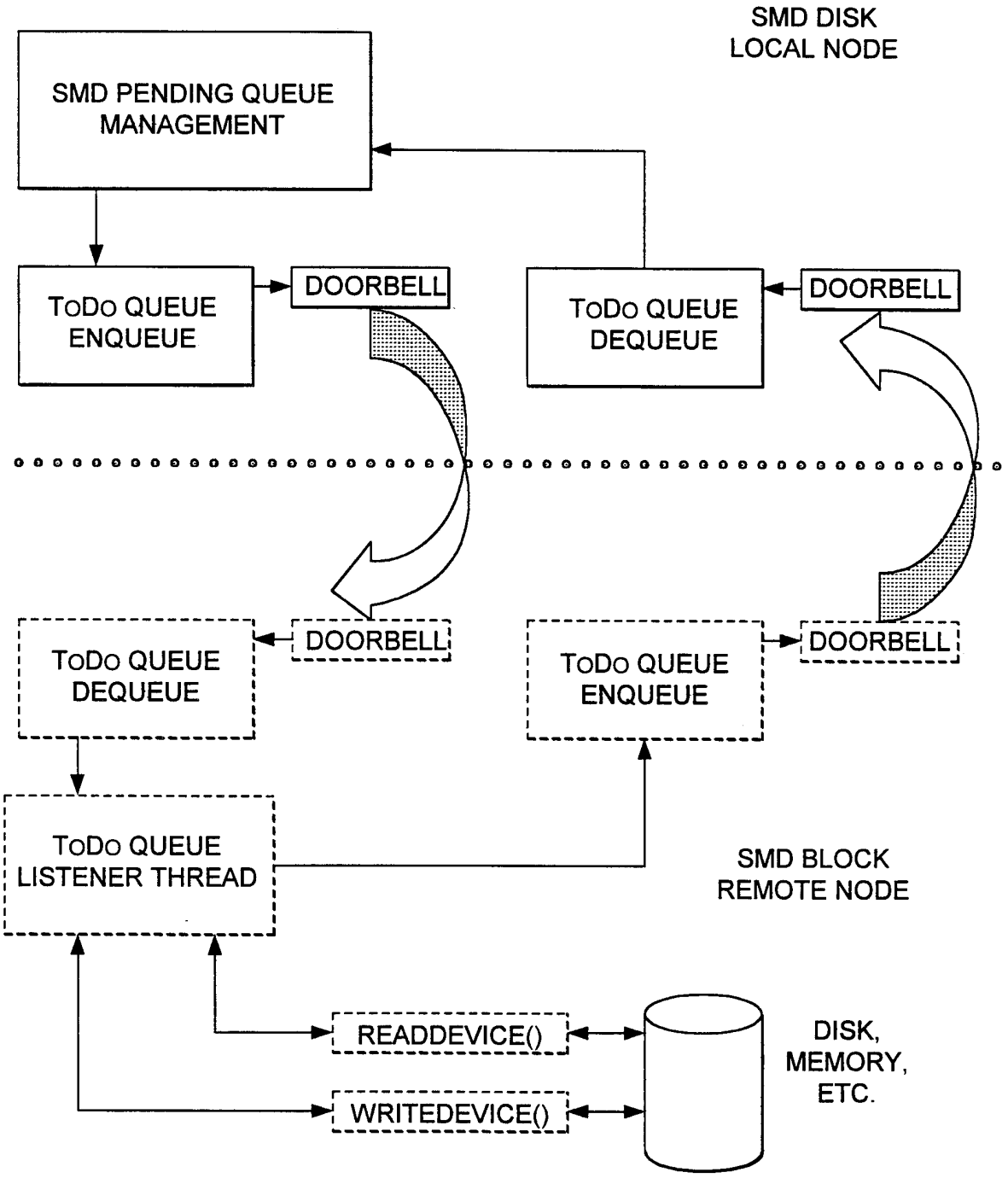
SMD DISK
LOCAL NODE
SMD PENDING QUEUE
MANAGEMENT
ToDo QUEUE
ENQUEUE
DOORBELL
ToDo QUEUE
DEQUEUE
DOORBELL
ToDo QUEUE
DEQUEUE
DOORBELL
ToDo QUEUE
ENQUEUE
DOORBELL
ToDo QUEUE
LISTENER THREAD
SMD BLOCK
REMOTE NODE
READDEVICE()
WRITEDEVICE()
DISK,
MEMORY,
ETC.

FIG. 2