

DYNAMICALLY ALLOCATED LOCAL/GLOBAL STORAGE SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to data processor storage systems and more particularly to dynamic storage systems for multiprocessor systems.

2. Description of the Prior Art

The following are systems representative of the prior art.

U.S. Patent 4,365,295 shows a multiprocessor system including a memory system in which the memory of each processor module is divided into four logical address areas. The memory system includes a map which translates logical addresses to physical addresses and which co-acts with the multiprocessor system to bring pages from secondary memory into primary main memory as required to implement a virtual memory system.

This patent which describes a conventional memory mapping system, does not address the efficient access of memory by single or multiple processors including interleaving storage references by a processor and dynamically directing storage references to global or local portions of each storage module.

U.S. Patent 4,228,496 shows a multiprocessor system including a memory system as above to implement a virtual memory system.

However, this patent which describes a conventional memory mapping system, does not address the efficient access of memory by single or multiple processors including interleaving storage references by a processor and dynamically directing storage references to global or local portions each storage module.

U.S. Patent 4,174,514 shows apparatus for performing neighbourhood transformations on data matrices for image processing and the like achieving processing speeds greater than serial processors within an economy of memory through use of a plurality of serial neighbourhood processors that simultaneously operate upon adjoining partitioned segments of a single data matrix.

This patent shows a multiprocessor system without any provision for access by all processors to a common global storage.

U.S. Patent 4,121,286 shows apparatus for allocating and de-allocating memory space in a multiprocessor environment.

This patent which describes a conventional memory mapping system, does not address the efficient access of memory by single or multiple processors including interleaving storage references by a processor and dynamically directing storage references to global or local portions of each storage module.

U.S. Patent 3,916,383 shows a resource allocation circuit selectively activating individual processors by time slice basis where a time slice has approximately the same time duration as the system storage time. The resource allocation circuit includes a priority network which receives real time common resource utilisation requests from the processors according to the individual processor needs, assigns a priority rating to the received request and alters in response thereto the otherwise sequential activation of the processors. The patent shows a system with several independent data processors within a single central processor which is not a true multiprocessor system in the usual sense.

The present invention relates to a system having independent processors forming a multiprocessor in which a storage system is dynamically partitioned into global storage and local storage.

U.S. Patent 3,820,079 shows a multiprocessing computer structured in modular form around a command control and data bus. Control functions for the various modules are distributed among the modules to facilitate system flexibility. The patent shows a system including conventional memory mapping and interleaving. The memory mapping does not control the interleaving, but the interleaving being the same over all modules for all data.

U.S. Patent, 3,641,505 shows a multiprocessor computing system in which a number of processing units, variable storage units and input/output units may be selectively combined to form one or more independent data processing systems. System partitioning into more than one independent system is controlled alternatively by manual switching or program directed partitioning signals.

This patent which describes a conventional memory mapping system does not address the efficient access of memory by single or multiple processors including interleaving storage references by a processor and dynamically directing storage references to global or local portions each storage module.

U.S. Patent 3,601,812 shows a memory system for buffering several computers to a central storage unit or a computer to several small memory units and a partitioned address scheme for the efficient use thereof. The digits of the address are decomposed into two disjoint subsets one of which is used as a buffer memory address and the other of which is stored with data word to effect identification thereof.

The patent deals with buffering memory data in a multiprocessor and does not show a dynamically partitioned storage system including interleaving storage references by a processor and directing dynamically storage references to global or local portions of storage.

GB-A-1411182 (STC) relates to a fault reconfigurable telephone call processing system in which multiple processors are connected to multiple storage unit on an each to each basis (shown clearly in Fig. 1). In this

(15)

arrangement, communication from one processor to any other part of the system is by way of memory. Insofar as, when the same function is allotted to plural processors, multiple copies of the program determining that function are made available, one to each allotted processor but with the starting address for each processor differing, to avoid clashing, this document has some bearing on interleaving, but only very indirectly so. In the context of the present invention, if interleaving is provided, it is by a uniform portion of a page, a segment, not a functional module of a program as it is the spread of function across parallel processors which dominates the prior art, and the question of multiple copies is essentially alien to the present invention. Interleaving, if used in the present invention, is used to spread and access correspondingly, the segments of a single module across plural storage units, avoiding clashing on the statistical basis of different processors requiring different words of the common module "at the same time" and thus being likely to want access to different (not the same) storage unit "at the same time".

US-A-4,414,624 (Summer et al) relates to a special purpose, plural processor system with a unitary shared storage module and local storage units for each processor. All storage is accessible via a shared memory bus under the control of a control processor and an arbitration unit. In this arrangement, contention is for the shared memory bus and the object of the whole system is to execute a single program using plural processors in parallel, with each processor performing its own sub-function, with a common task manager in each processor. "Global memory" is shared but, here again, duplication is used to minimise contention for the shared memory bus, the duplicated sections of global memory being resident in respective local storage units. READ is essentially local and WRITE is essentially global and in parallel and, to accommodate this, the local storage units are dual ported. While

(16)

there is time frame control exercised by the control processor, no particular control is exercised by the addressing scheme. The present invention does not have anything corresponding to the unitary shared storage module, any counterpart function being provided by the composite of the local storage units and control thereof is by the addressing organisation.

US-A-4,285,040 (Carlson et al) relates to access to local and shared storage under addressing control wherein, on virtual to real address translation, a marker appears which indicates, solely, that the location required is resident in local storage, or not. If not, an interrupt is taken and the translation operation is aborted, presumably, to fetch the "page" into local storage and change the marker thereafter so that processing can continue. A further marker also appears which indicates contiguous or dislocated segments within a "page", the manner of translation, when possible, changing dependent on this second marker.

The present invention relates to a plural processor complex in which the processor and the storage units attach to the ports of a network and in which, by means of a co-ordinated virtual addressing scheme and the translation thereof, performed independently by each processor, the real location - local address AND port - of the required data is determined and by which, in one option, the locations of a "page" can be given systematically distributed port addresses despite having the same high order virtual address values. No markers are used to indicate local or remote residence as this all derives directly from the address translation and, thus, no translation abort is required.

According to the present invention there is provided data processing apparatus comprising a communications network having plural individually addressable ports with a processor and a storage unit connected to at least some, and potentially all, of the ports providing addressable processing nodes, the network being arranged to support



node address containing message passing from any originating processing node and the addressed node, each processor incorporating a table driven, virtual to real address translation facility and an interface mechanism, in part controlled thereby, providing bi-directional communication between the processor, its connected port and the local storage unit, the translation facility responding to a virtual address from the processor to derive a processing node address which, if it is the local processing node address, causes the interface to connect the processor and the local storage unit and, if it is not the local processing node address, to connect the processor and the network via the local port and a message containing the generated other node address, and hence, indirectly, to the addressed processing node, the interface mechanism at the addressed processing node connecting the port to the storage unit local to that port, the translation control tables being writable at run time to match the requirements of applications, whereby the real address space of any particular processor is variably divided between its local storage unit and global storage in the form of the aggregate of the storage units in other processing nodes, and the local storage unit is equally variably divided between local storage and global storage, directly by the translation defined by the local translation control tables.

The present invention will be described further by way of example with reference to embodiments thereof as illustrated in the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a multiprocessor system according to the present invention.

FIG. 2 is a block diagram of a processing node according to the present invention.

FIG. 3 shows the physical address space of the multiprocessor, that is, the address of each word in all the memories in the multiprocessor. The most significant part of the address specifies the node number. The least significant part of the address is the offset within one of the memories.

FIG. 4 shows the entire contents of a virtual page as it would be stored with an interleave amount of zero; that is, sequentially within one of the node's memories.

FIG. 5 shows the entire contents of a virtual page as it would be stored with an interleave amount of log base 2 of the number of nodes; that is, fully interleaved across all of the memories.

FIG. 6 shows how an operating system would probably chose to allocate a set of sequential and interleaved pages within memory.

FIG. 7 is a block diagram of a Map/Interleave block shown in FIG. 2 according to the present invention.

FIG. 8 is a block diagram of a Network/Storage Interface block shown in FIG. 2 according to the present invention.

In the drawing, like elements are designated with similar reference numbers, and identical elements in different specific embodiments are designated by identical reference numbers.

DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

The present invention allows the main store of a multiple processor computer to be dynamically partitioned, at run time, between storage

6

(19)

local to each processor and storage globally accessible by all processors.

Prior art multiprocessor systems provide either

1. only local, and no global storage
2. only global, and no local storage
3. global storage and a fixed amount of local storage

Some of the systems of type 2. have a fixed amount of local storage in the form of a cache to effectively reduce global storage latency; as will be noted, the present invention does not preclude the use of a cache or, in general, the use of a storage hierarchy.

Unlike the above systems, the invention described here allows the storage configuration to be dynamically altered to fit the needs of the user resulting in substantially improved performance over a wider range of applications. Efficient passing of messages between processors, achieved in systems of type 1. above by special hardware, is also supported by this invention.

Configuration

As shown in Fig. 1, the machine organisation needed consists of N processing nodes 20 connected by some communications network 10. The processors and main storage of the system are contained in the nodes (see Fig. 2). Any network providing communication among all the nodes may be used.

Network Description

Figure 1 shows an interconnection network (ICN) 10 which connects the various nodes 20 together. This invention does not require any specific

50

9

interconnection network design, but such network must necessarily have as a minimum the following capabilities:

Messages which originate at any one node 20 can be reliably routed through network 10 to any other node 20.

The routing of a message is based upon addressing information contained within a "Node #" field of the message.

The message-routing functions of the ICN 10, when coupled with those of the various nodes 20, must enable any processor to access any memory location at any node 20 merely by specifying the correct absolute address.

The memory-mapping mechanisms of this invention provide each processor with the capability of generating such absolute addresses.

Fig. 2 shows the contents of a node. Addresses for storage references issued by the processor (PROC) 22 are mapped by the MAP/INTERLEAVE (M/I) 24 as described below.

A cache 26 is used to satisfy some storage references after mapping. The invention described here does not require the use of a cache nor does it restrict the placement of the cache. For example the cache 26 could reside between the processor 22 and M/I block 24.

References not satisfied by the cache 26 (or all references, if there is no cache) are directed by the network/storage interface (NET/STORE INTF. (NSI)) 28 to either the portion of main store 30 at that node or through the network 10 to store 30 of another node.

The NSI 28 also receives reference requests from other nodes and directs them to the storage of a node to be satisfied. This effectively makes the node's storage 30 dual-ported. Close to the same increase in

efficiency, at lower cost, can be obtained by locally interleaving a node's storage 30 and overlapping the processing of interleaved requests.

Local/Global Mapping

M/I 24 performs the usual two-level segment/page mapping of virtual addresses produced by processor 22 to real addresses, under the direction of some form of segment/page tables held in the main store 30. The real addresses produced uniquely identify every word or byte in all the nodes' stores: the high-order bits specify the node number, and the low-order bits specify the word or byte within a node's store. This is illustrated in Fig. 3.

In this invention, M/I 24 may also perform an interleaving transformation on the address. Whether it does so or not is specified by an additional field, unique to this invention, that is added to entries in the the segment and/or page tables. The effect of this transformation is to make a page of real storage a sequential block of addresses completely contained within a node (see Fig. 4); or a block of addresses that is scattered across several nodes' stores (see Fig. 5).

A sequential page can thus be guaranteed to be in a node's own store 30, local to that processor 22 and quickly accessible, providing the function of a local storage. Since an interleaved page is spread across many storage blocks, the probability of storage conflicts when multiple processors reference it is greatly reduced; this provides efficient globally-accessible storage.

To further reduce the probability of conflicts, the interleaving transformation may also "hash" the node number portion of the address, for example, by XOR-ing (exclusive-OR-ing) the node number portion of the address with other address bits. This would reduce the probability of conflict when regular patterns of access occur.

22

11

The degree of interleaving used -- the number of nodes across which an interleaved page is spread -- may be specified by the additional field added to the segment and/or page tables. This field may also specify characteristics of the "hashing" used.

By having some pages mapped sequentially, and some interleaved, part of main store 30 may be "local" and part "global." The amount that is local vs. global is under control of the storage mapping tables, and thus may be changed at run time to match the requirements of applications.

An example of the kind of main store use that this invention makes possible is illustrated in Fig. 6. This shows global storage allocated from one end of all nodes' storage 30, local storage from the other. While this is not the only way of using the invention described here, it illustrates how the invention allows the proportions of storage used for global and local storage to change in the course of running applications.

Message Passing

In addition to the communication afforded by global storage, direct inter-processor message passing is supported by this invention: Direct main storage data movement instructions (e.g., "MVCL" IBM System 370 Principles of Operation) can be used to move data from a sequential page in one processor to a sequential page in another processor, without disturbing or requiring use of any other node's storage.

Description of Storage Mapping Tables

The storage mapping tables are used by the M/I. They define the mapping performed by the M/I between the address issued by the processor and the address accessed in memory. Specifically, and unique to this invention, they define whether an interleaving transformation is to be applied to an address or not, and may specify what interleaving transformation if any is to be applied. The tables themselves may reside in the M/I itself; or in the main memory of the system (either global or local storage),

referenced by the M/I; or in both. Wherever they reside, they are modifiable by software running on the system's processors. It will often be convenient to combine the definition of interleaving in these tables with a virtual memory mapping of some form, e.g., page mapping, segment mapping, or two-level segment and page mapping ((reference: Baer, J., "Computer Systems Architecture", Computer Science Press, Rockville, MD, 1980)) by extending the usual contents of such tables to include a field of at least one bit containing information determining the interleaving and/or remapping. This has been done in the preferred embodiment described here, but is not required by this invention, which only requires that the existence and/or amount of the interleave be controlled by each processor. Other mechanisms for doing this include: extending the processors' instruction set to have interleaved and non-interleaved data access instructions; by instruction set extension or I/O instruction control, have instructions that turn interleaving on or off for data and/or instruction fetch.

Description of the Operation of the M/I 24

Figure 7 illustrates the operation of the Map/Interleave (M/I) for the case where memory mapping and low-order remapping are both incorporated. The top of the figure shows a virtual address as received from the processor and stored in VAR 242. This is subdivided, as shown, into a segment and/or page index (S/P I) 244, a page offset (PO) 246, and a word offset (WO) 248. These fields have the conventional meanings in memory mapping systems. The WO, which specifies which byte in an addressed word (or word in a larger minimal unit of addressing) is to be accessed is passed through the entire mapping process unchanged (as shown), and will not be mentioned further.

The S/P I is used in a conventional way as an index into the storage mapping tables, as shown. From the storage mapping tables, the real Segment/Page offset (S/P O) 250 is derived in a conventional way by Table Lookup to form a Real Address as shown. Unique to this invention, the Table Lookup also produces an interleave amount (shown) associated with

54

13

each segment and/or page which is specified in the storage mapping tables.

After the Real Address is derived, the low-order Remap 252 may be applied to produce a Remapped Address in RAR 254. This may also be applied as part of the variable amount variable-width right rotate described below, or may be omitted, in which case the Real Address is passed through unchanged to the next stage. The low-order Remap operates on a field LR to produce a new address field LR' of the same width, using the rest of the Real Address (field labelled HR) as shown. The width of LR (and LR') may be any value between two extremes: at largest, it is equal in width to the page offset (PO); at smallest, it is the maximum allowed interleave amount, i.e., if the width is N, the maximum number of memory modules is 2^N . Fig. 7 shows it at an intermediate point between these two extremes. The purpose of the low-order Remap is to randomise successive addresses that are to be interleaved across a subset of memory modules, i.e. accessed in different sequences. This lowers the probability of many processors accessing the same memory module simultaneously when the data structures being accessed have a size that is an integer multiple of the amount of storage in one interleaved sweep across all the memories. The maximum size of LR arises from the need to keep pages addressed in contiguously-addressed blocks; the minimum size is the minimum needed to effectively perform the function described above. The low-order Remap is one-to-one, i.e., every possible value of LR must be mapped into a different value of LR'. One possible low-order Remap is the following: Let the bits of LR be named LR0, LR1, ... LRn from right to left; and the bits of HR and LR' be named similarly. Then, using "xor" to represent the conventional exclusive-or logic function, a suitable low-order remap is: $LR'0 = LR0 \text{ xor } HR0$; $LR'1 = LR1 \text{ xor } HR1$; ... $LR'n = LRn \text{ xor } HRn$.

The actual interleaving transformation is then performed by a variable amount right rotate on a variable-width bit-field device 256, producing the actual Absolute Address used to access the system's storage modules. This uses the Interleave Amount derived earlier, and operates on the real

address after remapping (if remapping is done) excluding the word offset (WO). The width of the field to be rotated and the amount the field is to be rotated are specified by the interleave amount. The operation of the right rotate is as follows: Let HS be numbered similarly as LS above. Given an interleave amount of Q , the width of the field to be rotated is $HSq-1$ through LRO. The number of bit positions the field is rotated is Q . Instead of a variable amount variable-Width right rotate, a conventional bitwise rotation of the combined HS, CS, and LS fields by Q could be used. However, the scheme presented allows systems to be constructed with fewer than the maximum number of processing nodes because it retains, in the Absolute Address Reg 258, high-order (leftmost) 0s that appeared the Remapped Address in RAR 254. Conventional rotation would not do this, and therefore the fact that all possible values of LS must be allowed forces addressing of all possible nodes.

In the absolute address, the final HS' field designates the processing node whose storage module contains the data to be accessed (Node #); the combined CS and LS' fields indicate the offset in that storage module where the data word is to be found (Storage Offset); and the WO field indicates which byte or sub-word is desired.

Note that when the interleave amount is 0, the variable amount Variable-Width right rotate leaves HS' equal to HS, and LS' equal to LS. This leaves the Absolute Address the same as the Remapped Address, thus providing direct sequential addressing. This provides the sequential addressing described above. Appropriate values in the Storage Mapping Tables allow this to be storage local to the node generating the addresses, or storage entirely contained in other nodes (the latter useful for message passing and other operations).

Note also that the use of less than the maximum possible interleaving effectively restricts the processors across which global memory is allocated. This can be used in several ways, e.g.: (a) to allow the system to continue to operate, although in a degraded mode, if some of

the storage modules are inoperative due to their failure, the failure of the network, etc., (b) to effectively partition the system, allowing parts to have their own global and local memory allocation independent of other parts, thus reducing interference between these parts -- either to run several independent problems, or a well-partitioned single problem.

The arrangement described above can function with or without a private cache memory 26. The cache can be positioned as indicated in Figure 2 or between the processor and NI. The function of cache memory is to reduce memory access time for those memory accesses which occur repeatedly in time or at contiguous memory addresses. For cache coherence to be maintained in a multiprocessor configuration, it is necessary for such a cache to have an additional capability which would not ordinarily be implemented on a uniprocessor cache. If for example one processor can read one memory location at approximately the same time that another processor is writing in the same location, it is required that neither processor satisfy such memory references in its own cache. This additional capability can be provided by a variety of different means, such as cross-interrogation between different cache memories, or by specifying certain memory locations to be non-cachable. Any such caching scheme (or none at all) can be applied within this context.

The arrangement described includes a Network-Storage interface (NSI) 28 whose operation is illustrated in Figure 8. The routing functions of this unit (as described below) are necessary for the proper functioning of this embodiment. Any hardware configuration which provides a comparable coherent set of message-routing functions could be employed instead, and its implementation should be straightforward for anyone skilled in the art. Such a unit is associated with each processor node, as illustrated in Figure 2. The function of this unit is to route messages between the associated processor, the associated memory-controller, and other processor-nodes on the network. The types of messages sent include, but are not limited to

Load requests issued by the local processor.

Store requests issued by the local processor.

Cache-line load requests issued by the local cache, resulting from cache-misses on storage requests by the local processor.

Cache-line store requests issued by the local cache, resulting from cache-misses on storage requests by the local processor.

Responses to storage load or store requests by the local processor and/or cache.

Load or store requests issued by other processors or caches, referencing memory locations contained in the memory of the local processor node.

Responses to storage requests issued by other processors or caches, being returned from the memory of the local processor node.

Messages from the local processor to remote processors, or from remote processor nodes to the local processor.

Synchronisation requests (such as test-and-set, etc.) issued by the local processor, to be performed at the local memory or at remote memory locations.

Responses to synchronisation requests.

All such messages must contain information sufficient to identify the type of the message.

In addition, all such messages arriving at the NSI 28 must contain information sufficient to determine whether the message is to be routed to the local processor/cache 26, the local store 30, or to the

28

17

interconnection network 10. In the case of storage requests, by a processor or cache, such information is contained in the "Node #" field of the memory address. If the value of the "Node #" field coincides with the number of the local node, such requests are routed to the local memory 30; otherwise they are routed to the interconnection network 10. The memory-mapping scheme described above ensures that the required interleaving is thereby performed. Similarly, responses to storage requests are routed either to the local processor 22 or to the interconnection network 10, so as to return to the processor node which originated the message. Other messages also must contain "Node #" fields and message-type identifying codes, which uniquely identify such messages in order to be properly routed by NSI 28. The NSI is capable of routing messages from any of the three sources to any of the other two outputs, based on information contained in fields within the messages. In particular, the devices shown in the figure can operate to perform such routing as follows:

The PE router (PE RTE) 282 receives messages from the PE 22. If the "Node #" indicates the current node, the PE RTE 282 sends the message to the local store 30 via the local memory concentrator (LM CON) 284; otherwise, it sends it to the network via the network concentrator (NET CON) 286.

The local memory router (LM RTE) 288 receives response messages from the local store 30. If the "Node #" indicates the current node, the LM RTE 288 sends the message to the local PE 22 via the PE concentrator (PE CON) 290; otherwise, it sends it to the network via the network concentrator (NET CON) 286.

The network router (NET RTE) 292 receives messages from the network, and on the basis of the type of each message determines whether it is (a) a request from another processor for access to the local memory module; or (b) a reply from another node containing information requested by the current node from another node's local memory. In case (a), the message is sent to the local

memory via the LM CON 284; otherwise, it is sent to the local PE 22 via the PE CON 290.

The network concentrator 286 receives messages (either requests or replies) from either the PE 22, via the PE RTE 282; or the LM 30, via the LM RTE 288. It passes both to the network 10 for routing to another node based on the message's "Node #".

The PE concentrator 290 receives reply messages from either the local store 30, via the LM RTE 288; or the network 10, via NET RTE 292. It passes them to the PE 22 (and/or cache 26).

The local memory concentrator 284 receives request messages from either the local PE 22, via the PE RTE 282; or network 10, via NET RTE 292. It passes them to local store 30.

In addition to paths for data communication, the routers and concentrators indicated above must communicate control information indicating when data is valid (from the router to the mul and when it can be accepted (from the concentrator to the router)).

A two-ported memory could be used instead of the LM RTE 288 and LM CON 284 devices.

Thus, while the invention has been described with reference to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made without departing from the scope of the invention.

CLAIMS

1. Data processing apparatus comprising a communications network having plural individually addressable ports with a processor and a storage unit connected to at least some, and potentially all, of the ports providing addressable processing nodes, the network being arranged to support node address containing message passing from any originating processing node and the addressed node, each processor incorporating a table driven, virtual to real address translation facility and an interface mechanism, in part controlled thereby, providing bi-directional communication between the processor, its connected port and the local storage unit, the translation facility responding to a virtual address from the processor to derive a processing node address which, if it is the local processing node address, causes the interface to connect the processor and the local storage unit and, if it is not the local processing node address, to connect the processor and the network via the local port and a message containing the generated other node address, and hence, indirectly, to the addressed processing node, the interface mechanism at the addressed processing node connecting the port to the storage unit local to that port, the translation control tables being writable at run time to match the requirements of applications, whereby the real address space of any particular processor is variably divided between its local storage unit and global storage in the form of the aggregate of the storage units in other processing nodes, and the local storage unit is equally variably divided between local storage and global storage, directly by the translation defined by the local translation control tables.

2. Data processing apparatus as claimed in claim 1, wherein the interface mechanism provides independent bi-directional interconnection between the processor and the local storage, the local storage and the local port and the local port and the processor.



31

3. Data processing apparatus as claimed in either preceding claim, wherein the entries in the translation tables each contain a field which specifies the number, which can be zero, of other node addresses which contain locations contributing to the page required, the translation facility, when the number is other than zero, cyclically modifying the node address in response to that number whereby the locations of a distributed page are interleaved over a corresponding plurality of local storage units in other processing nodes.
4. Data processing apparatus as claimed in claim 3, wherein the field also specifies the manner of cyclic node address modification.
5. Data processing apparatus as claimed in claim 3, wherein the portion of the local storage unit corresponding to virtual addresses translating into the local node address, the number being, per force, zero, is allocated on page boundaries and the remainder of the local storage unit is allocated on variable segment boundaries.
6. Data processing apparatus as claimed in any preceding claim, wherein the number is a power of 2, not exceeding the number of processing nodes current connected to the network ports.
7. Data processing apparatus as claimed in claim 6, wherein the node address modification is performed by a variable amount rotation of a variable width address field.
-