



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

<b>(51) International Patent Classification <sup>6</sup> :</b> <b>G06F 15/16</b>	<b>A1</b>	<b>(11) International Publication Number:</b> <b>WO 95/16967</b>  <b>(43) International Publication Date:</b> 22 June 1995 (22.06.95)
<b>(21) International Application Number:</b> PCT/US94/14349  <b>(22) International Filing Date:</b> 13 December 1994 (13.12.94)  <b>(30) Priority Data:</b> 08/166,443              13 December 1993 (13.12.93)      US  <b>(71) Applicant:</b> CRAY RESEARCH, INC. [US/US]; 655A Lone Oak Drive, Eagan, MN 55121 (US).  <b>(72) Inventors:</b> PASSINT, Randal, S.; Route 5, Box 430BB, Chippewa Falls, WI 54729 (US). OBERLIN, Steven, M.; 20 Peterson Lane, Chippewa Falls, WI 54729 (US). FROMM, Eric, C.; 221 Lake Street, Eau Claire, WI 54703 (US).  <b>(74) Agent:</b> RAASCH, Kevin, W.; Schwegman, Lundberg & Woessner, 3500 IDS Center, 80 South Eighth Street, Minneapolis, MN 55402 (US).	<b>(81) Designated States:</b> CA, JP, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).  <b>Published</b> <i>With international search report.  Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i>	
<b>(54) Title:</b> MESSAGE FACILITY FOR MASSIVELY PARALLEL PROCESSING SYSTEMS		
<b>(57) Abstract</b>		
<p>A messaging facility is described that enables the passing of packets of data from one processing element to another in a globally addressable, distributed memory multiprocessor without having an explicit destination address in the target processing elements memory. A message is a special cache-line-size write that has as its destination a pre-defined queue area in the memory of the receiving processing element. Arriving messages are placed in the queue in the order that they appear at the node by hardware queue management mechanisms. Flow control between processors is usually accomplished by the queue management hardware, with software intervention necessary to deal with the error cases caused by queue overflows, etc.</p>		

**FOR THE PURPOSES OF INFORMATION ONLY**

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo	KR	Republic of Korea	SE	Sweden
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LV	Latvia	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

5

MESSAGE FACILITY FOR MASSIVELY  
PARALLEL PROCESSING SYSTEMS

---

**Field of the Invention**

The invention relates generally to high performance massively parallel processing systems, and more specifically to a messaging facility which  
10 allows efficient emulation of a multicomputing system on a multiprocessing system.

**Background of the Invention**

Massively parallel processing (MPP) systems are computing systems  
15 comprised of hundreds or thousands of processing elements (PEs) individually interconnected by a common high-speed communication network. Many MPPs today are multicomputers, wherein each PE is considered a stand-alone computer with its own central processor, local memory, and associated control logic. In a multicomputer, each PE can only address its own local memory  
20 and cannot directly read or write the local memory associated with another PE. Each PE reads data from another PE's memory by sending a message and I/O-like packet, to the target PE requesting that some data from its memory be formatted and sent back to the requesting PE, or vice versa for writes. Thus  
25 in a multicomputing system, each remote reference is essentially an I/O operation. This style of interprocessor communications is called "message passing." Message passing is a well-known and prevalent MPP programming model because multicomputers are relatively easy to build. The ease of construction of a multicomputer MPP arises from the use of commodity microprocessors in an environment that closely resembles their "natural  
30 habitat" (i.e., that hardware and software implementation envisioned by the microprocessor designers), that is, a network of small autonomous computers.

For many applications, however, a multiprocessor MPP is more desirable than a multicomputing MPP. In a multiprocessor MPP, every PE can directly address all of memory, including the memory of another (remote)  
35 PE, without involving the processor at that PE. Instead of treating PE-to-

remote-memory communications as an I/O operation, reads or writes to another PE's memory are accomplished in the same manner as reads or writes to the local memory.

Multiprocessors have an ease-of-programming advantage over  
5 multicomputers. In a multicomputer, each communication between PE's must be explicitly coordinated and initiated by the programmer. In a  
multiprocessor, however, communications arise naturally and implicitly by the specification of variables and constants in program statements. If the data represented by the variable or constant name happens to reside in the memory  
10 of a remote PE, an access of the data there is automatically initiated the same manner as if the data were in local memory.

Software techniques are known which allow the emulation of a multiprocessor and its associated ease-of-programming on a multicomputer. Such software emulators translate the application program remote reads and  
15 writes into explicit I/O-like message-passing operations in a way that is hidden from the application programmer. Software of a global address space using message passing is very inefficient, however, because I/O-like operations have a large software start-up overhead associated with each message. Successful message passing programming models on  
20 multicomputers rely on relatively large granularity programming, passing large amounts of data with each communication, to amortize the startup overhead over many elements of data.

There is therefore a need in the art for a massively parallel processing system which has the desirable attributes of both a multiprocessing and  
25 multicomputing MPP's, and which further does not have the high-overhead and large granularity restrictions of present software manage of message passing techniques.

#### Summary of the Invention

30 This can conceivably be a very efficient operation, potentially exceeding the performance of a true multicomputer when the granularity of communications is smaller. To support efficient message-passing in a

multiprocessor, it is necessary to provide certain hardware primitives to allow one PE to send data to another PE's message queue and interrupt the target PE upon message arrival. The present invention addresses a need in the art for efficient message-passing emulation in a multiprocessor MPP system.

5           The present invention provides hardware support for a messaging facility in a multiprocessing MPP system to allow emulation of a multicomputing system on a multiprocessing MPP system. The messaging facility of the present invention enables the passing of packets of data from one PE to another without having an explicit destination address in the target  
10 PE's memory. A message is a special cache-line-size remote write that has as its destination a hardware-managed message queue in the memory of the receiving PE. Arriving messages are placed in the message queue in the order that they arrive by hardware mechanisms that also provide the lowest-level communication handshaking protocol service. Flow control between  
15 processors is accomplished by the queue management hardware, with software intervention used as necessary to deal with the error cases caused by queue overflows, etc.

Messages can be transmitted by user-level code. To alert the receiving PE when a message has arrived, all messages set an interrupt to the processor  
20 at the target node. Because the receiving of messages involves responding to the message-arrival interrupt, some system-level code must be executed to receive messages.

The messaging facility can be used to accomplish a remote action or initiate a remote procedure by defining an opcode convention that permits one  
25 processor to send a message containing opcode, address and arguments to another. The destination processor, upon receiving the message after the arrival interrupt, can decode the opcode and perform the indicated action using the argument address and data.

At the start of a program, a head and a tail pointer are initialized to  
30 point to the base of the queue, and a limit counter is initialized to a value representing the amount of available message space in the queue. When a message arrives at a node, it is copied in its entirety into sequential addresses

starting at the address indicated by the tail pointer. The tail pointer is incremented after each message to point to the next message location in the queue. The limit counter, used by the flow control circuits to monitor queue space, is decremented as each message arrives to indicate that the queue is filling. An interrupt to the processor is set each time the tail pointer is incremented by a new arriving message. The receiving processor reads messages in the order they arrived by using the head pointer and must interpret any encoded commands and addresses in software to decipher the action requested by the message. The tail pointer can be read by the processor when servicing an interrupt to determine when the head was caught up (no more messages to service).

The messaging facility provides the primitives for the construction of a message-passing interprocessor communication protocol. The messaging facility allows use of operating system communication and message-passing programming models, such as the defacto-standard Parallel Virtual Machine (PVM) communication protocol.

### **Brief Description of the Drawings**

The foregoing and other objects, features and advantages of the invention, as well as the presently preferred embodiments thereof, will become apparent upon reading and understanding the following detailed description and accompanying drawings in which:

FIGURE 1 shows a simplified block diagram of a representative MPP system with which the present address centrifuge can be used;

FIGURE 2 shows a block diagram of a PE, including a processor and its associated shell circuitry;

FIGURE 3 shows the messaging facility packet format;

FIGURE 4 shows the NACK packet format;

FIGURE 5 shows the format of a message in the message queue;

FIGURE 6 shows a sample message queue before and after writing a message;

FIGURE 7 shows a sample message queue before and after receiving a

message;

FIGURE 8 shows a sample message queue before and after the tail pointer wraps to the first entry of the message queue;

FIGURE 9 shows a sample message queue before and after receiving  
5 an ACK packet;

FIGURE 10 shows a sample message queue before and after receiving a NACK packet;

FIGURE 11 shows a sample message packet before and after the microprocessor reads a message or NACK packet from the queue;

10 FIGURES 12, 13 and 14 show the format of error messages as they are stored in the message queue.

#### **Detailed Description of the Preferred Embodiment**

In the following detailed description of the preferred embodiment,  
15 reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration a specific embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural changes made without departing from the scope of the present invention.

20 The preferred MPP system, for which the present invention provides an address centrifuge, is a MIMD massively parallel multiprocessor with a physically distributed, globally addressable memory. A representative MPP system is shown in FIGURE 1.

FIGURE 2 shows a simplified block diagram of a PE 200. An  
25 individual PE includes a high-performance RISC (reduced instruction set computer) microprocessor 202. In the preferred MPP system, microprocessor 202 is the DECChip 21064-AA RISC microprocessor, available from Digital Equipment Corporation. Each PE is coupled to a local memory 204 that is a distributed portion of the globally-addressable system memory, and includes a  
30 shell of circuitry that implements synchronization and communication functions facilitating interactions between processors.

The shell circuitry includes an interconnection network router 206,

used to connect multiple PEs in the three-dimensional toroidal "fabric". The interconnection network carries all data communicated between PEs and memories that are not local. A block transfer engine 208 in the PE shell circuitry permits asynchronous (i.e., independent of the local processor) movement of data between the local memory 204 and remote memories associated with other PEs, such as block transfers, with flexible addressing modes that permit a high degree of control over the distribution of data between the distributed portions of the system memory. The flexible addressing scheme and address centrifuge are described in the copending and commonly assigned U.S. patent application entitled "ADDRESS CENTRIFUGE FOR DISTRIBUTED MEMORY MASSIVELY PARALLEL PROCESSING SYSTEMS" filed on even date herewith to Oberlin et al. The shell circuitry also includes a data prefetch queue 210 which allows the processor 202 to directly initiate data movement between remote memories and the local processor in a way that can hide the access latency and permit multiple remote memory references to be outstanding.

Synchronization circuits in the shell permit synchronization at various different levels of program or data granularity in order to best match the synchronization method that is "natural" for a given parallelization technique. At the finest granularity, data-level synchronization is facilitated by an atomic swap mechanism that permits the locking of data on an element-by-element basis. A more coarse grain data-level synchronization primitive is provided by the present messaging facility, which permits a PE to send a packet of data to another PE and cause an interrupt upon message arrival, providing for the management of message queues and low-level messaging protocol in hardware. Control-level synchronization at the program loop level is provided by a large set of globally accessible fetch-and-increment registers that can be used to dynamically distribute work (in the form of iterations of a loop, for instance) among processors at run time. Yet another form of control-level synchronization, barrier synchronization, is useful to control transitions between major program blocks (i.e., between loops performing actions on the same data sets). The barrier mechanism is described in detail in the



copending and commonly assigned U.S. patent application entitled "BARRIER SYNCHRONIZATION FOR DISTRIBUTED MEMORY MASSIVELY PARALLEL PROCESSING SYSTEMS," filed on even date herewith to Oberlin et al.

5

#### **Messaging Facility Rationale**

The present invention supports a message passing programming paradigm on the preferred MPP multiprocessor system. The messaging facility provides a means to write to another PE without indicating a specific address in the destination PE's memory. The message information is stored in a message queue in the destination PE's local memory. Upon message arrival the PE is interrupted to inform it that a message has arrived. Hardware at the receiving PE decodes a command field in the message, which informs the destination PE of the action which it is to take with respect to the message data payload.

15

The messaging facility has several advantages over conventional multicomputing message-passing mechanisms. First, the message send is extremely lightweight. In a multicomputer, as discussed above, a message send is very similar to an I/O operation. As is well known to those of skill in the art, I/O type requests require a significant amount of message formatting. In addition, substantial overhead is incurred from the call to the operating system required by an I/O type request to establish communications between the transmitting computer and the receiver. However, because the preferred MPP system is a multiprocessor, as opposed to a multicomputer, the target PE can instead be addressed directly and no I/O-type formatting is required. At the same time, normal shared global memory security hardware mechanisms assure that proper permission to communicate, etc., exists between the transmitting and the receiving processors.

20

25

The present messaging facility also incorporates hardware message queue management. Ordinarily, the message queue in memory must be managed and manipulated by software means, adding to the overhead associated with a message and increasing the granularity of the data

30

transmission required to amortize the overhead.

The present messaging facility incorporates hardware flow control (handshaking) at the lowest level. This permits the application to efficiently send messages without consideration for flow control issues in the usual case. Only when message queues fill and an overflow or rejected packet occurs is the flow control responsibility turned over to software. The hardware flow control mechanism is designed so that all such error conditions are recoverable by software. The hardware flow control also reduces the overhead associated with message-passing, allowing still finer communication granularity to be profitably parallelized using message passing programming techniques.

The present messaging facility provides for the efficient passing of short messages by direct transmission. Long messages are passed between processors by direct transmission only of control information. The control information describes the size and location of the long message in the transmitting processor's memory. The receiving processor, upon decoding the control information, can efficiently access the long message where it resides due to the multiprocessor ability to directly address and reference remote memory. This has several advantages over traditional message-passing in a multi-computer: The latency of message transmission (the time for message send until the receiver begins acting on the first elements of the message) is significantly reduced. Only the control information need actually be sent before the receiver is alerted to the "arrival" of the message. The actual moving of the long message data to the receiver can be accomplished under the control of the receiver as part of a computation loop acting upon the data, thus hiding the latency of the message move. Data buffering requirements are also reduced, because the message may not actually ever have to be copied into the memory of the receiving processor.

### 30 Network Packets

FIGURE 3 shows the format of a message packet, which is the unit of information conveyed by the network between processors in the

multiprocessing system. Message packets consist of two parts: A header containing control and address information, and the body, consisting of 4 words of data payload.

The header consists of 4 parts:

- 5 1. Command
- 2. Destination PE number
- 3. Source PE number
- 4. Address

The command field is a twelve-bit field containing the packet type  
 10 (e.g. load, store, message, NACK, etc.). Certain packet types, including the message packet, have an 8 bit subfield in the command that contains a half-word (32-bit) mask indicating valid data in the data field following the header. A message packet will typically carry a command with a mask of all ones (unless the message transmission is interrupted, causing message  
 15 fragmentation, discussed later). Table 1 shows the encoding of the command field.

---

20

Table 1

	2 <sup>11</sup>	2 <sup>10</sup>	2 <sup>9</sup>	2 <sup>8</sup>	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	Packet Type
	1	1	1	1	m	m	m	m	m	m	m	m	Message
	0	0	0	0	1	0	0	x	x	x	x	x	ACK
25	1	1	1	0	m	m	m	m	m	m	m	m	NACK

---

In Table 1, bits 2<sup>11</sup> through 2<sup>9</sup> of the command field indicate the type of packet the command is in. When set to 1, bit 2<sup>8</sup> of the command field indicates the packet is a request. When set to 0, bit 2<sup>8</sup> of the command field  
 30 indicates the packet is a response.

Bits 2<sup>7</sup> through 2<sup>0</sup> of the command field contain an opcode that signals the support circuitry in the destination PE what type of operation to perform. In some cases, the opcode contains mask bits (m) that indicate which 32-bit

halfwords in the data body of the packet contain valid data.

The twelve-bit destination PE number is part of the header, but is not used by the message handling software except as part of an error handling routine. The destination PE number is used by hardware to validate that network packets have been routed to the correct PE. The destination PE number represents a logical PE number in the preferred embodiment.

The source PE number is typically used to route replies back to the requesting PE. The hardware uses the source PE number to construct a new routing tag for the message ACK or NACK (discussed below) to the transmitting processor upon message arrival. The source PE number is twelve bits and preferably represents a logical PE number.

The 24-bit address is the partial physical address the transmitting processor used to send the message. The software opcodes are embedded in the address field. The address is split into two halves for transmission through the network, each 12 bits in size.

### **Message Payload**

A message may in the preferred embodiment contain up to 32 bytes of information. Messages smaller than 32 bytes must be padded out to 32 bytes so that receiving software can distinguish between small messages and message fragments. Message fragmentation can be caused by an interrupt if it occurs during the performance of the 4 stores necessary to fill a write buffer during message transmission.

Messages larger than 32 bytes may be sent as a string of independent 32-byte messages. Above a certain threshold of message size, it will be faster and more efficient to only exchange protocol information using the messaging facility and move the data using other means. For instance, prior to moving a large block of data, a source processor might send a message requesting an allocation of storage at the destination. After allocating space for the large message, the destination PE can return a pointer to the allocated storage, either in a message, or by writing directly to a reply "pigeon hole" in the source PE's memory using a pointer passed as part of the request message.

The sender may then directly copy the large message to the remote node at the allocated location, sending a final short message to indicate when the copy is done. Alternately, large messages can be moved by the receiver using address and size information passed in the protocol and control message sent  
5 by the transmitter detailing where the data resides in the transmitter's memory and how large the data block is.

### **Message Transmission**

Messages are transmitted simply by writing a cache line to a special  
10 remote address. Messages are distinguished from ordinary remote cache-line writes by the mapping of the destination address. Special addresses are reserved in the physical memory address range to be interpreted as message transmission triggers. If a store is performed to the special address, hardware decodes the destination address to indicate that the store is actually a message  
15 send. PE support circuitry then creates a message packet and sends the packet into the interconnection network for routing to the destination PE.

In the preferred MPP system, messages are assembled in circuits on the microprocessor chip called "write buffers." The cache line write to the special address must be performed as sequential stores to addresses within the  
20 same cache line. The write buffers on the microprocessor attempt to assemble full cache lines before requesting a bus cycle and will succeed in most cases if no intervening memory references are made while sending a message. If the processor is interrupted while constructing the cache line message, the message will be sent as two incomplete cache lines in two separate messages  
25 (called message fragments). The half-word mask field in the message header must be examined by software upon receipt of a message to see if the entire message is present. If not, the receiving processor must defer processing of the message fragment until the remaining portions arrive.

In the preferred embodiment, it is important to have an understanding  
30 of the functioning of the microprocessor write buffers in order to ensure proper message transmission. There are 4 buffers, each a cache-line in size. At every store, the microprocessor checks all 4 buffers for a possible match

between the address of the current store and one that may be pending in the write buffers. If a match is found, the write data is merged into its proper place in the cache line as indicated by the word-within-cache-line bits in the address. If there was data at that spot in the write buffer from a previous  
5 store with the same cache line address and word-within-cache-line pointer, the previous data is lost.

The DECChip microprocessor used in the preferred MPP system keeps the data in the write buffers for an indeterminate amount of time, unless specific steps are taken to flush them to the memory subsystem. One or more  
10 write buffers will flush if: A store is made to an address that matches none of the cache-line addresses currently in the write buffers, a memory barrier instruction is issued, or an exception or interrupt occurs. Barring any of these occurrences, the write buffers will still drain eventually, but the timing of the actual write is difficult to predict.

15 It can be seen that if multiple messages are sequentially sent to a particular processor, different special cache line addresses must be used for any five message in a row, or unintentional merging of messages in the write buffers can occur.

Those skilled in the art can see that the message assembly functions of  
20 the write buffers could also easily be accomplished by buffers implemented external to the microprocessor if the microprocessor did not have write buffers or it was not desirable for some reason to use the on-board write buffers for message assembly.

### 25 **Message Arrival**

After receiving the message packet from the network router, the support circuitry in the destination PE attempts to store the message in the message queue. If the message queue can accept the message, the support circuitry stores the message in the queue and sets the message hardware  
30 interrupt for the receiving microprocessor.

The support circuitry in the destination PE then creates a message acknowledge (ACK) packet and sends the packet to the source PE. FIGURE

3 also shows the format of an ACK packet and Table 1 above shows the format of the command field in an ACK packet.

If the message queue in the destination PE cannot accept the message, the support circuitry returns the message to the requesting PE by creating a  
5 no-acknowledge (NACK) packet. A NACK packet contains the same information as the message packet; however, the NACK packet has the destination node and source phits exchanged, and bit 28 of the command field set to 0, as shown in FIGURE 4. A phit is a minimal size packet. After  
10 receiving a NACK packet, the support circuitry in the PE that created the message stores the NACK in its own message queue. The message queue always contains room for NACK packets, as discussed below, and never rejects a NACK packet.

The processor that sent the message can then read the NACK from the message queue. Because the NACK contains all of the information that the  
15 original message did, the transmitting microprocessor can reconstruct and resend the original message, if desired. This process may repeat until the message is accepted and the transmitting support circuitry receives an ACK packet (instead of a NACK) which completes the message write.

## 20 **Message Queue**

After receiving a message packet, the support circuitry in a PE places the message in a designated location in local memory called the message queue, where arriving messages are stored prior to processing. The preferred message queue stores up to 4080 message, NACK, or error packets plus 16  
25 reserved locations for a small amount of overflow (total of 256K bytes of information), although any desired size message queue could be selected in other implementations. The support circuitry places message packets in the message queue in the order that they are received. The message queue in the preferred embodiment is a first-in, first-out (FIFO) circular buffer.

30 When a message is transmitted, the support circuitry at the transmitting node reserves a location in its own local message queue. By reserving a message location in the local queue, the support circuitry guarantees that there

will be room in the message queue in the event of receiving a NACK.

When a message is received from a remote node, the support circuitry stores the message in the next sequential location in the message queue. Likewise, when a NACK is received, the support circuitry stores the NACK in the message queue, consuming the space that was reserved when the original  
5 message was sent.

When an ACK is received, the support circuitry releases the space in the message queue reserved for a NACK. The reserved space is no longer needed after receiving an ACK.

10 In the preferred MPP system, the support circuitry contains a hardware counter and two memory-mapped registers used to control the message queue. The hardware counter called the limit counter (MQ\_LC), is a 13-bit up/down counter that indicates how many message slots in the message queue do not contain information or are not reserved for possible NACK packets. The limit  
15 counter cannot be read or written directly, but can be reset to 4079 when software writes any value to the message queue tail pointer (MQ\_TP) memory-mapped register. The MQ\_LC is the indicator of how much space is available in the message queue. The MQ\_LC is initialized with the maximum message count for the selected queue size (minus some slack to allow for  
20 pipeline run-on after interrupt and error messages). Incrementing or decrementing the MQ\_LC by one advances or retards the count of available space by one message. The MQ\_LC is decremented by hardware each time a message is transmitted and incremented each time a message acknowledgement arrives. The MQ\_LC is incremented by software for each  
25 message dequeued. If there is a simultaneous software-triggered increment and hardware triggered increment, the counter does nothing. If there is a simultaneous software-triggered increment and hardware triggered increment, the MQ\_LC increments by two.

The MQ\_TP register is a 12-bit, readable and writable hardware  
30 counter that contains the tail pointer. The tail pointer is a value that represents the slot in the message queue where the next message that arrives will be stored. The actual memory offset represented by the tail pointer is



calculated by multiplying the tail pointer by 64 and adding the result to the message queue base address. The value of the MQ\_TP may be read and may be reset to 0. To reset the MQ\_TP register to 0, software must write any value to the MQ\_TP register.

- 5           When the MQ\_TP is read, the MQ\_TP returns the current value of the MQ\_TP. When any value is written to the MQ\_TP, the support circuitry resets the value of the MQ\_TP to 0 and resets the message queue limit counter to 4079.

10           Table 2 shows the bit format of the MQ\_TP register and describes the bits in the register.

          The message queue limit increment register (MQ\_LIR) is a memory-mapped register that software uses to manipulate the limit counter. The MQ\_LIR is a write-only pseudoregister that has no content. The MQ\_LIR is used by software to advance the MQ\_LC. Any store to the MQ\_LIR causes  
15           the MQ\_LC to increment by one message. The value stored to the MQ\_LIR is irrelevant: The act of writing to that location triggers an atomic increment of the limit counter. When software writes any value to the MQ\_LIR, the value of the limit counter increments by one. When software reads the value of the MQ\_LIR, bit 212 of the MQ\_LIR register contains the sign bit of the  
20           limit counter and the remaining bits are not defined.

          Table 3 shows the bit format of the MQ\_LIR register when it is read from and describes the bits in the register.

          In addition to these hardware registers and counters, the messaging facility software must maintain a 12 bit head pointer. The head pointer  
25           indicates the location of the next message in the message queue that will be read by the microprocessor.

          The support circuitry also controls the message and error hardware interrupts. The message interrupt indicates to the microprocessor that one or more message, NACK, or error packets have been stored in the message  
30           queue. The error interrupt indicates to the microprocessor that one or more errors have occurred.

The following sections describe how the registers, counters, and signals are used to control the messaging facility.

### **Message Queue Transmission Protocol**

5           When the microprocessor signals the support circuitry that the microprocessor is writing a message, the support circuitry checks the value of the message queue limit counter. If the limit counter is greater than or equal to zero, the support circuitry decrements the limit counter by one. A transmitting PE must maintain room for returned messages in the message  
10 queue in the event that the destination message queue is full. The support circuitry in the source PE decrements the value of the limit counter to reserve a location in the message queue for each message transmitted. Figure 6 shows a sample source message queue before and after sending a message. After decrementing the limit counter, the support circuitry sends a message  
15 packet to the destination PE. If the processor should attempt to transmit a message when the limit counter equals zero (no room for NACKs), the processor is interrupted with an error. This is referred to as a message queue full error.

          Although the message queue full error exists, the support circuitry still  
20 decrements the limit counter by one and sends a message packet to the destination PE. If a message queue full error occurs, the operating system should immediately dispose of unread messages stored in the message queue to provide more space in the message queue.

### **25 Message Queue Arrival Protocol**

          When the support circuitry in a destination PE receives a message packet from the network router, the support circuitry first checks the value of the limit counter in the message queue. If the value of the limit counter is greater than zero, the support circuitry accepts the message. If the value of  
30 the limit counter is less than or equal to zero, the message queue is full and the support circuitry rejects the message.

          Figure 7 shows a sample destination message queue before and after

receiving a message. When the support circuitry accepts a message, it stores the message in the message queue at the location pointed to by the tail pointer. After storing the message in the queue, the support circuitry increments the tail pointer, decrements the limit counter, and sets the Message hardware interrupt to the microprocessor.

Every message arrival interrupts the target processor for service. An arriving message sets a latch that asserts the Message Interrupt input to the EV-4. The latch remains set until it is cleared by a write to the MQ\_LIR register.

In the normal course of operation, the processor would remain in the message handler routine until the head index pointer equaled the MQ\_TP, meaning that the queue had been emptied, before returning to the user.

To avoid inadvertently missing a message arrival, the MQ\_LIR write should be performed before the read of the MQ\_TP and the comparison with the software head index pointer.

The tail pointer only increments. Because of this characteristic, if the tail pointer reaches the last entry in the message queue, the tail pointer automatically wraps to the first entry of the message queue when the support circuitry accepts another message. Figure 8 shows a sample destination message queue before and after the tail pointer wraps to the first entry of the message queue. When the head pointer reaches the last entry in the message queue, software must reset the value of the head pointer to the beginning of the message queue.

After accepting the message, the support circuitry creates a message acknowledge (ACK) packet. The support circuitry sends a message ACK packet to the PE that sent the original message.

After receiving the ACK packet, the support circuitry in the PE that sent the message increments the value of the limit counter to deallocate the slot on the message queue reserved for a possible NACK. Essentially decrementing the limit counter reserves space in the message queue for possible rejections, ACKs or NACKs release the reserved space. For example, Figure 9 shows a sample source message queue before and after

receiving an ACK packet.

By incrementing the value of the limit counter, the support circuitry frees up space in the message queue for sending or receiving another message. This action completes a successful message transfer from one PE to another.

5 As previously stated, when the support circuitry receives a message packet, the support circuitry first checks the value of the limit counter in the message queue. If the value of the limit counter is less than or equal to zero, the support circuitry rejects the message.

10 When the support circuitry in the destination PE rejects a message, it converts the message into a NACK packet. After converting the packet, the support circuitry sends the NACK packet to the PE that sent the original message.

15 Figure 10 shows a sample source message queue before and after receiving a NACK packet. After receiving the NACK packet, the support circuitry in the PE that sent the message stores the NACK packet in the message queue at the location pointed to by the tail pointer. After storing the NACK packet in the queue, the support circuitry increments the tail pointer and sets the Message hardware interrupt to the microprocessor.

20 When the microprocessor sent the original message, the support circuitry decremented the value of the limit counter. Because the limit counter was decremented at that time, the support circuitry does not decrement the limit counter when the support circuitry receives a NACK packet.

25 After the support circuitry sets the Message interrupt to the microprocessor, the microprocessor reads the NACK information from the message queue. The microprocessor then sends the message again to the destination PE.

#### **Destination Message Processing**

30 After the support circuitry stores a message packet, NACK packet, or error message in the message queue, the support circuitry sets the Message hardware interrupt to the microprocessor. This signals the microprocessor that

one or more message facility packets have arrived and are stored in the message queue.

The microprocessor may then read the message, NACK, or error message packet from the message queue and interpret the software code, encoded commands, or addresses to decipher the action requested by the message. Figure 11 shows a sample message queue before and after the microprocessor reads a message or NACK packet from the queue. After reading a packet from the message queue, the microprocessor increments the software-controlled head pointer and increments the limit counter by writing any value to the MQ\_LIR register. When the microprocessor increments the limit counter, the support circuitry clears the Message hardware interrupt.

Normally, after the microprocessor receives the Message hardware interrupt, the microprocessor continues to read messages from the message queue until the value of the head pointer equals the value of the tail pointer. This indicates to the microprocessor that there are no more messages or NACK packets in the message queue.

To avoid missing a message arrival, the microprocessor should increment the limit counter before comparing the value of the tail pointer and the head pointer. The limit counter is incremented by writing any value to the MQ\_LIR register.

A break-down of the actions involved in messaging, and the effect on the Head, Tail, and Limit counters follows (T = Tail, H = Head, L = Limit, + 1 or -1 means increment by message size):

WRITE (send message):  
 If (L <= 0) FAULT -- interrupt local processor  
 L = L - 1  
 MESSAGE ARRIVAL:  
 If (L <= 0) NACK -- return message to sender  
 Else T = T + 1 -- set message interrupt  
 L = L - 1  
 RECEIVE MESSAGE (process message interrupt):  
 Read data (H)

20

$H = H + 1$  (wrap if necessary)

$L = L + 1$

ACK ARRIVAL:

$L = L + 1$

5 NACK ARRIVAL (rejected message):

$L = L + 1$

$T = T + 1$  -- set message interrupt

### Error Handling

10 Three types of error conditions can be checked for in the preferred embodiment. First, message packet information could be corrupted during transit of the message from the source to the destination PE. Second, a PE could attempt to write a message into its message queue when the message queue is already full. Finally, a message could be misrouted through the torus  
15 network and arrive at the wrong destination PE.

### Message Memory Error Checking

Figure 5 illustrates the placement of the 12-bit command, source, destination, or address fields in the 16 bit parcels as stored in the message  
20 queue. The body of the packet is stored in memory locations  $n$  through  $n+3$ . Several copies of the header phits of the packet are stored in memory locations  $n+4$  through  $n+7$ . The message payload data is preferably stored into the local memory in the message queue locations with valid Single Error Correction, Double Error Detection (SECDED) check bits as generated by the  
25 transmitting processor. If a memory error occurs in the message queue DRAM that holds the data, it will be detectable when the processor loads the data.

The message header is not accompanied by SECDED check bits. Instead, each 16-bit parcel of the header has two bits of parity in bit positions  
30  $2^{12}$  and  $2^{13}$ . Bit  $2^{15}$  is normally a zero. Bit  $2^{14}$  is always zero. Bit  $2^{15}$  is set to a one in the command parcels to indicate when a network error has occurred, to distinguish between messages and network error messages.

Although the parity bits in the header phits of the message packet enable the network interface to check if an error occurred during a transfer, the microprocessor does not use the parity bits to check if an error occurred while reading the header phits from the message queue. To allow the  
5 microprocessor to detect errors when reading header phits from the message queue, the support circuitry places several copies of the header phits in the message queue.

To check for errors in the header phits, message handling software compares the word of header phit information stored at location  $n+4$  with the  
10 word of header phit information stored at location  $N+6$  using a bit-wise exclusive - OR. Likewise, message handling software compares the word of header phit information stored at location  $N+5$  with the word of header phit information stored at location  $N+7$  (refer again to Figure 5).

If the word in location  $N+4$  matches the word in location  $N+6$  and the  
15 word in location  $N+5$  matches the word in location  $N+7$ , the header phits of the packet were not corrupted when transferring from the message queue to the microprocessor. If the words do not match, one or more of the header phit bits changed value while being transferred from the message queue to the microprocessor.

20 If the message handling software detects an error, the software must determine which header phits contain an error. To do this, the software examines each header phit in the word read from the message queue and generates a new set of parity bits for the header phit. If the new parity bits match the parity bits read from the message queue, the header phit is not  
25 corrupted. If the new parity bits do not match the parity bits read from the message queue, the header phit is corrupted.

When the microprocessor reads the body of the message from the message queue, the microprocessor signals the support circuitry that it wants to perform SECDED on the words. The support circuitry then sends the body  
30 of the message to the microprocessor with a data acknowledge code that causes the SECDED hardware in the microprocessor to be enabled to detect and/or correct errors in the message data.

### Queue-Full Error

It is possible for the microprocessor to be notified of a message queue full error when there are still up to four messages in the write buffer. When the microprocessor receives an interrupt that indicates the queue full error, the  
5 microprocessor empties the write buffer.

As the support circuitry receives each of the four message writes from the microprocessor, the support circuitry decrements the message queue limit counter in preparation for possible NACK packets and sends the messages to the destination PE's. In this case, the value of the limit counter continues to  
10 go negative with each message sent. The negative value indicates that the message queue is fully subscribed and any NACK packet received will be stored in one of the 16 reserved locations for overflow packets.

In addition to possible NACK packets from messages that were stored in the write buffer, the support circuitry may receive an error message at any  
15 time. When the support circuitry receives an error message and the message queue is full, the support circuitry still stores the error message in the message queue and decrements the limit counter by one. Because the limit counter is all ready negative, this also uses one of the reserved 16 slots in the message queue for overflow packets.

Another condition that could possibly occur is when a PE issues 4080 message writes and has not yet received any ACK packets. In this case the message queue is full but empty. If this occurs, software must periodically read the value of the MQ\_LIR register and examine bit 212 of the register. If  
20 this bit is set to 1, the value of the limit counter is negative and no more message writes should be issued. If this bit is set to 0, the value of the limit counter is positive and message writes may be issued.

If a queue full condition exists ( $MQ\_LC \leq 0$ ) and a processor attempts to send a message, the Error Interrupt.

Because the processor could have message waiting to be sent at the  
30 time the queue-full condition is detect, surplus room in the local queue ("slack") is maintained to allow the allocation of space for possible NACKs from the waiting messages.



Slack is established by initializing the MQ\_LC with a value slightly smaller than the maximum available queue space. MQ\_LC is allowed to go negative when a queue full transmission is detected, but only to allow for the transmission of the backed-up messages. Messages arriving from external nodes are rejected when the MQ\_LC is less than or equal to zero. NACKs and ACKs may arrive at any time regardless of the value of the MQ\_LC.

The amount of slack required to account for queuefull pipeline run-on can be computed by adding up the processor write buffers, which hold outgoing message data before they are transmitted through the torus network (4 in the preferred microprocessor), plus the erroring message at the external bus interface (1), plus additional writes that may be in the load/store pipeline behind the write-buffers (3 stores, equal to one more message fragment) for a total of 6 in the preferred embodiment.

Additional slack needs to be included to provide room for network error messages (see the next section).

### Network Error Messages

There are two types of network packet errors: a misrouted packet error and a packet parity error. If the network interface receives a packet and the value of the destination phit does not match the value for the destination PE, a misrouted packet error occurs. If the network interface detects a parity error in the header phit of a packet, a packet parity error occurs. The header parcel parity bits are checked by the network interface hardware upon the arrival of any packet type. If any parity does not agree with the transmitted parity bits, the packet is converted by hardware into a network error message and stored in the message queue.

The message handling software can determine if a message is an error message by testing bit  $2^{15}$  of the command parcel.

In the case of a misroute packet error, the receiving PE can attempt to retransmit the message to the appropriate destination PE. Thus in this case, the error need not necessarily result in a system failure.

Virtually all network failures that are detected using the header parity

are fatal. The information captured in the message queue is intended to be an aid to troubleshooting rather than provide a means for overall system recovery.

5 If either network packet error occurs, the network interface turns the packet it received into an error message by setting bit 2<sup>15</sup> of the command phit to 1. The network interface then sends the error message to the appropriate PE in the node.

10 The error message is treated as if it were an arriving message, regardless of the packet type. It is stored into the message queue by the network interface hardware and the error interrupt to the processor asserted. No ACK is sent.

Only the first error packet is captured. Further error messages are simply discarded until the error condition is cleared by software.

15 Because a network error message may occur at any time, including when a queue full condition already exists, room for an additional message must be provided for in the slack computation to allow the error message to be absorbed. This brings the slack total to 7 messages.

20 The network interface may create an error message out of any of the seven packet types, the message packet, ACK packet, or the NACK packet. Figure 12, Figure 13, and Figure 14 show the format of error messages as they are stored in the message queue.

## WHAT IS CLAIMED IS:

1. A messaging facility in a multiprocessor computer system having a plurality of processing elements interconnected by a n-dimensional interconnect network, each processing element including a processor and a local memory, wherein globally addressable portions of local memory of each processing element form a distributed memory, the messaging facility comprising:
  - buffer means in a source processing element for assembling a message to be sent from the source processing element to a destination processing element based on information provided from the source processing element's processor;
  - a network router for transmitting the assembled message from the source processing element to the destination processing element via the interconnect network;
  - a message queue in a designated area of the local memory of the destination processing element for storing the transmitted message;
  - tail pointer means for indexing into the message queue to indicate a location where the transmitted message is to be stored in the message queue;
  - interrupt means for providing an interrupt to the destination processing element's processor in response to the message being stored in the message queue; and
  - head pointer means for indexing into the message queue to indicate a location where the message is stored in the message queue for reading by the destination processing element's processor in response to the interrupt.
2. The messaging facility of claim 1 further comprising:
  - a queue limit counter to provide a limit count representative of the remaining available locations in the message queue.
3. The messaging facility of claim 2 further comprising:
  - means for generating an acknowledged message based on the limit count indicating that there is an available location for storing the transmitted

message and for generating a not acknowledged message based on the limit count indicating that there are no available locations for storing the transmitted message.

- 5 4. The messaging facility of claim 2 further comprising:  
means for causing the queue limit counter to increment the limit count as messages are read from the message queue; and  
means for causing queue limit counter to decrement the queue limit count as messages are stored in the message queue.

10

5. The messaging facility of claim 3 wherein the source processing element includes a message queue and a queue limit counter and wherein the messaging facility further comprises:  
means for causing the queue limit counter of the source processing  
15 element to decrement the limit count after the message is assembled in the buffer means to thereby reserve a location in the message queue of the source processing element for a not acknowledged message generated by the destination processing element in response to the transmitted message.

- 20 6. The messaging facility of claim 5 further comprising:  
means for causing the queue limit counter of the source processing element to increment the limit count in response to receiving an acknowledged message from the destination processing element indicating that the destination processing element accepted the transmitted message.

25

7. The messaging facility of claim 1 wherein the tail pointer means circularly indexes the message queue.

8. The messaging facility of claim 7 wherein the message queue is a  
30 first-in, first-out circular buffer.

9. The messaging facility of claim 1 wherein the tail pointer means is implemented in hardware.
10. The messaging facility of claim 1 wherein the head pointer means is  
5 implemented in software.
11. The messaging facility of claim 1 further comprising:  
means for detecting possible error conditions and for providing an  
error interrupt to the destination processing element's processor in response to  
10 detecting an error condition.
12. The messaging facility of claim 11 wherein one type of possible error  
condition results from a message being corrupted during the transmission from  
the source processing element to the destination processing element and  
15 wherein the messaging facility further comprises:  
means for responding to an error condition indicating that the  
transmitted message is corrupted to transform the corrupted message into an  
error message to be stored in the message queue for retrieval during software  
error recovery.  
20
13. A method of passing messages in a multiprocessor computer system  
having a plurality of processing elements interconnected by a n-dimensional  
interconnect network, each processing element including a processor and a  
local memory, wherein globally addressable portions of local memory of each  
25 processing element form a distributed memory, the method comprising:  
assembling a message to be sent from the source processing element to  
a destination processing element based on information provided from the  
source processing element's processor;  
transmitting the assembled message from the source processing  
30 element to the destination processing element via the interconnect network;

indexing into a designated area of the local memory of the destination processing element's local memory to indicate a location where the transmitted message is to be stored;

storing the transmitted message in the indexed location in the  
5 designated area of the local memory of the destination processing element;

providing an interrupt to the destination processing element's processor in response to the message being stored in the indexed location; and

indexing into the designated area of the local memory of the destination processing element's local memory to indicate a location where the  
10 message is stored for reading by the destination processing element's processor in response to the interrupt.

14. A multiprocessor computer system comprising:

a plurality of processing elements, each processing element including a  
15 processor, a local memory having a designated message queue area, and shell circuitry, each shell circuitry including:

buffer means for assembling a message to be sent from a source processing element to a destination processing element based on  
20 information provided from the source processing element's processor,

a network router for transmitting the assembled messages from  
the source processing element to the destination processing element via the interconnect network to be stored in the designated message queue  
area of the local memory of the destination processing element,

tail pointer circuitry to index into the designated message queue  
25 area of the local memory of the destination processing element to indicate a location where the transmitted message is to be stored, and

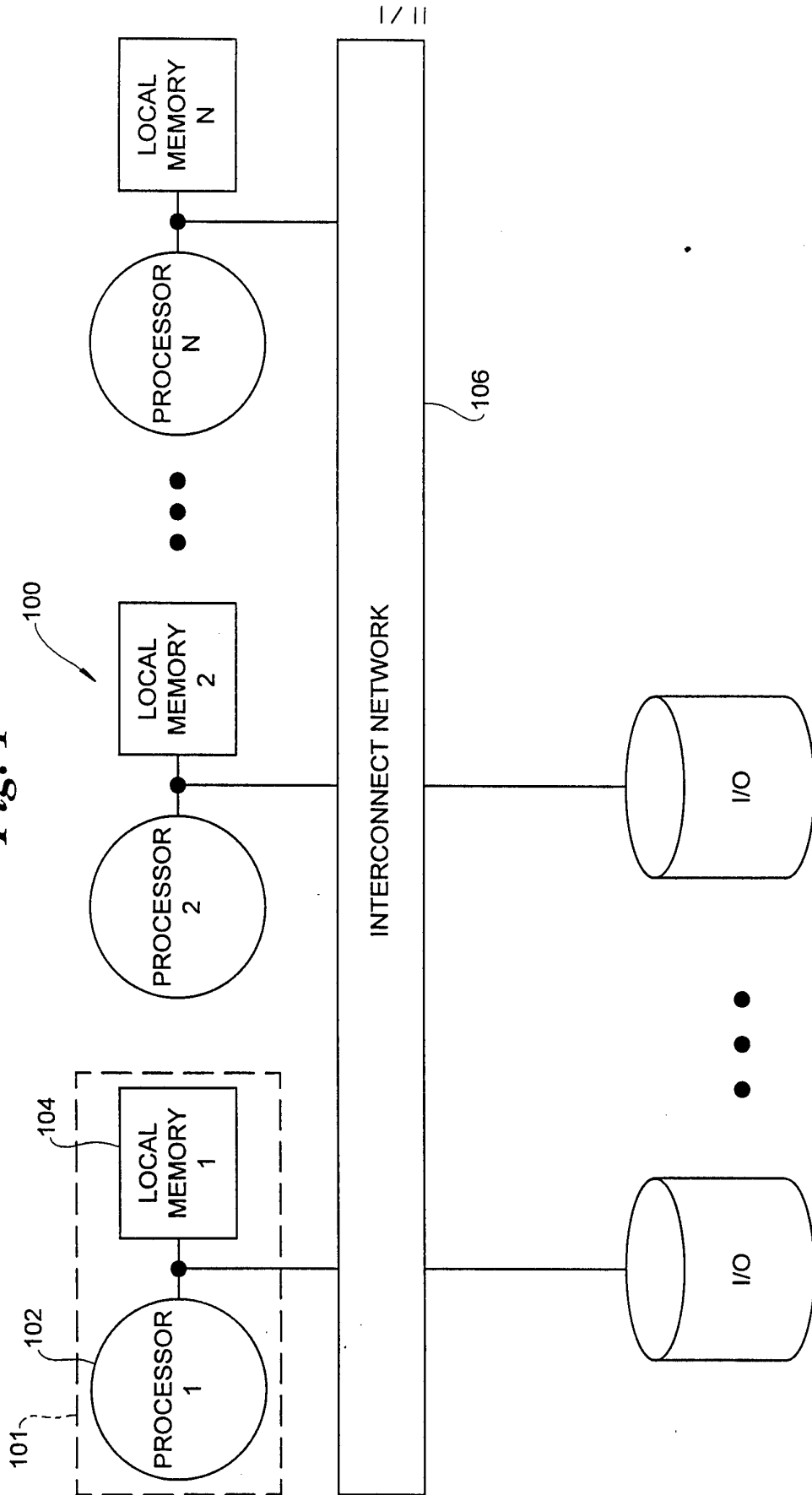
interrupt circuitry providing an interrupt to the destination processing element's processor in response to the message being stored  
in the designated message queue area;

30 messaging facility software providing head pointers to indicate locations where messages are stored in the designated message queue areas of the local memories of the destination processing elements for reading by the

destination processing elements' processors in response to the interrupts from the destination processing elements' interrupt circuitry; and

a n-dimensional interconnect network interconnecting the plurality of processing elements.

*Fig. 1*





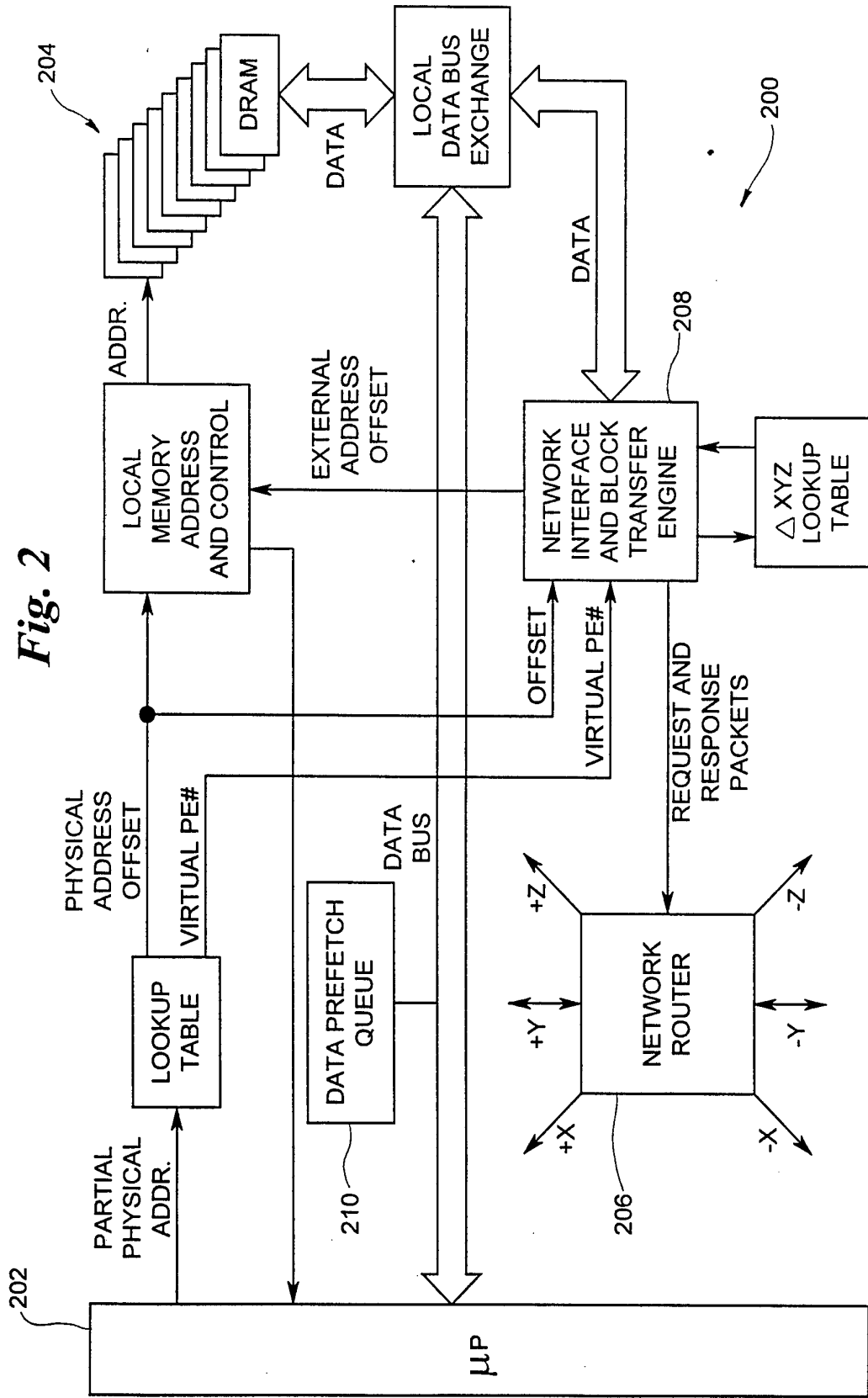
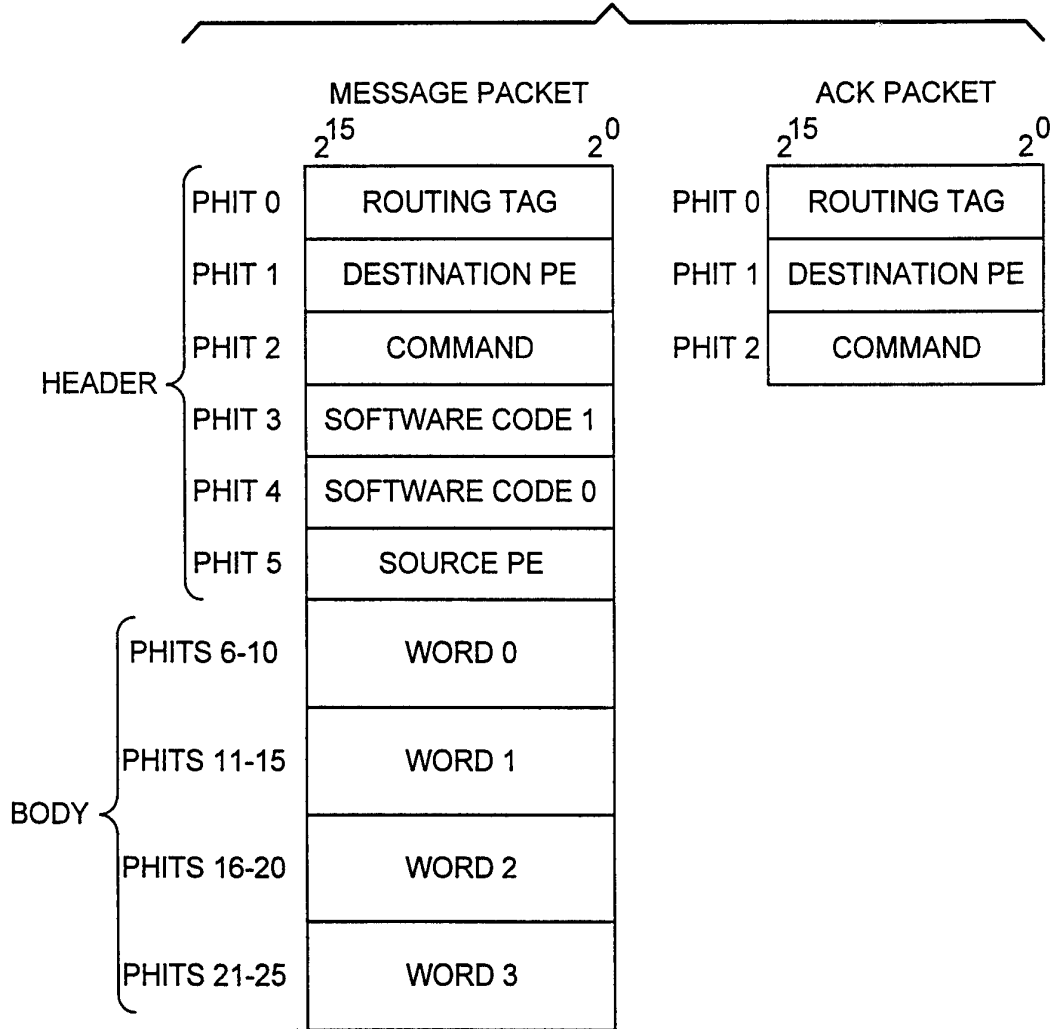
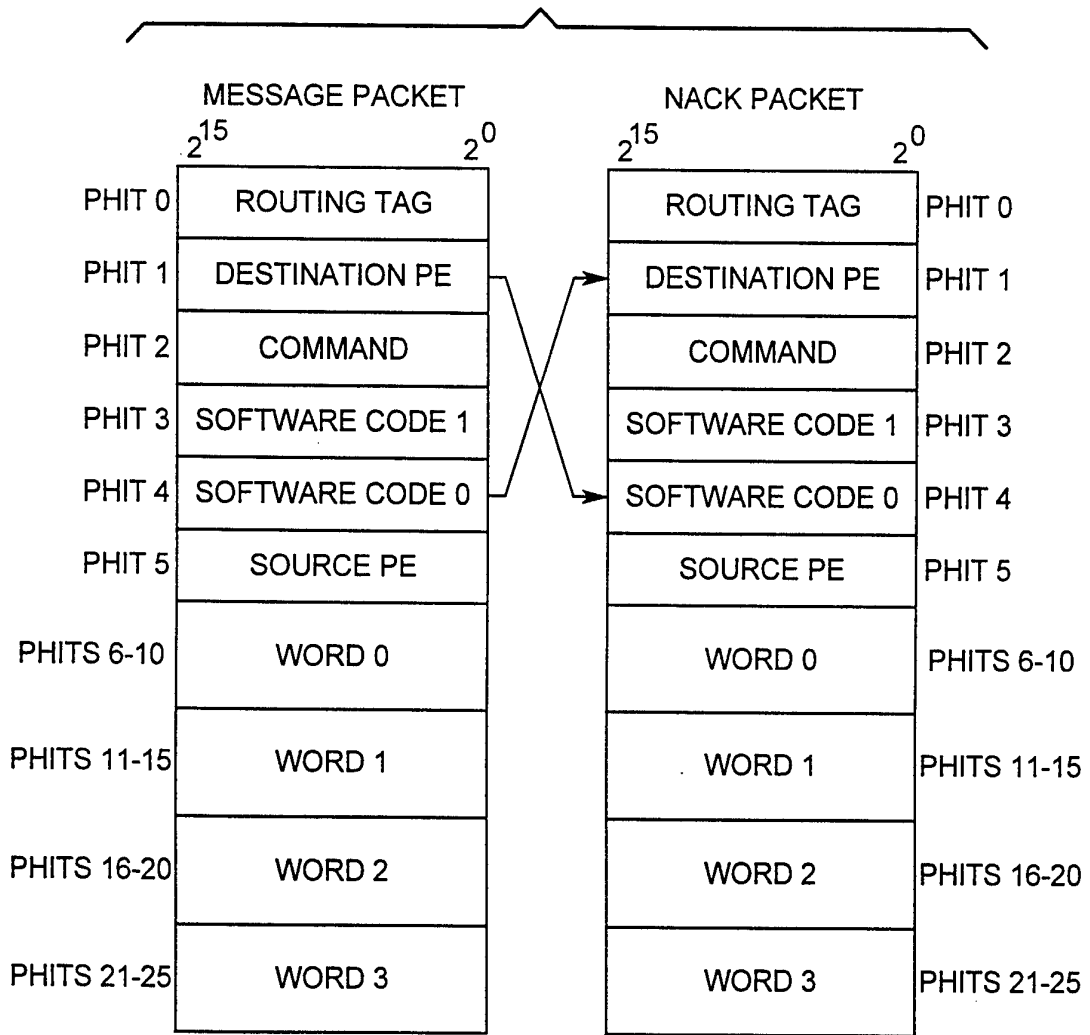


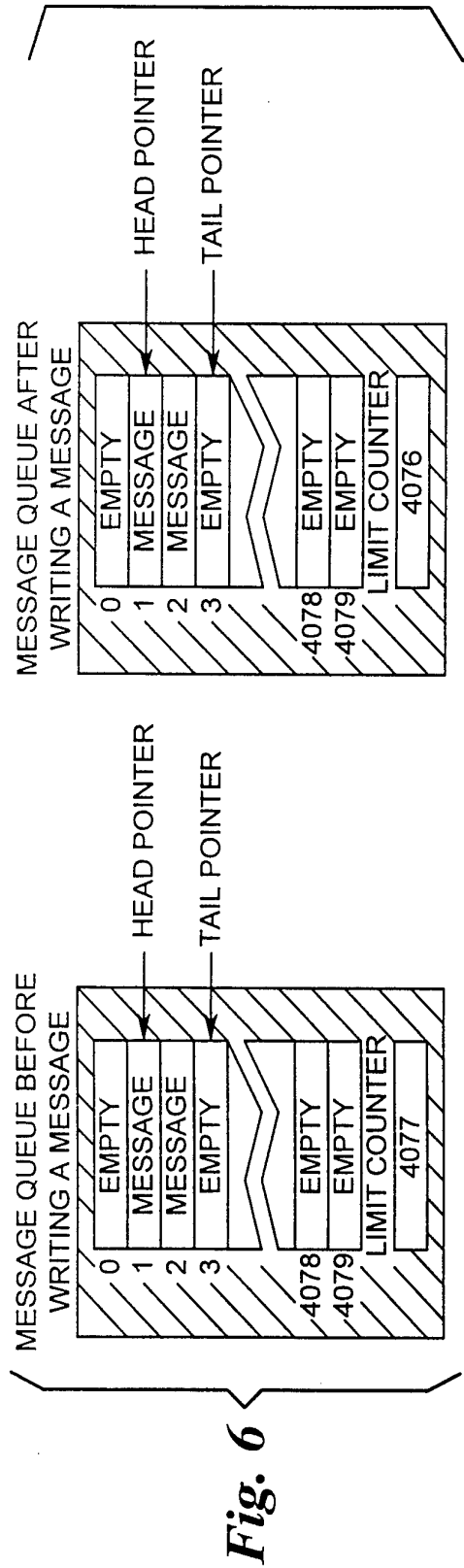
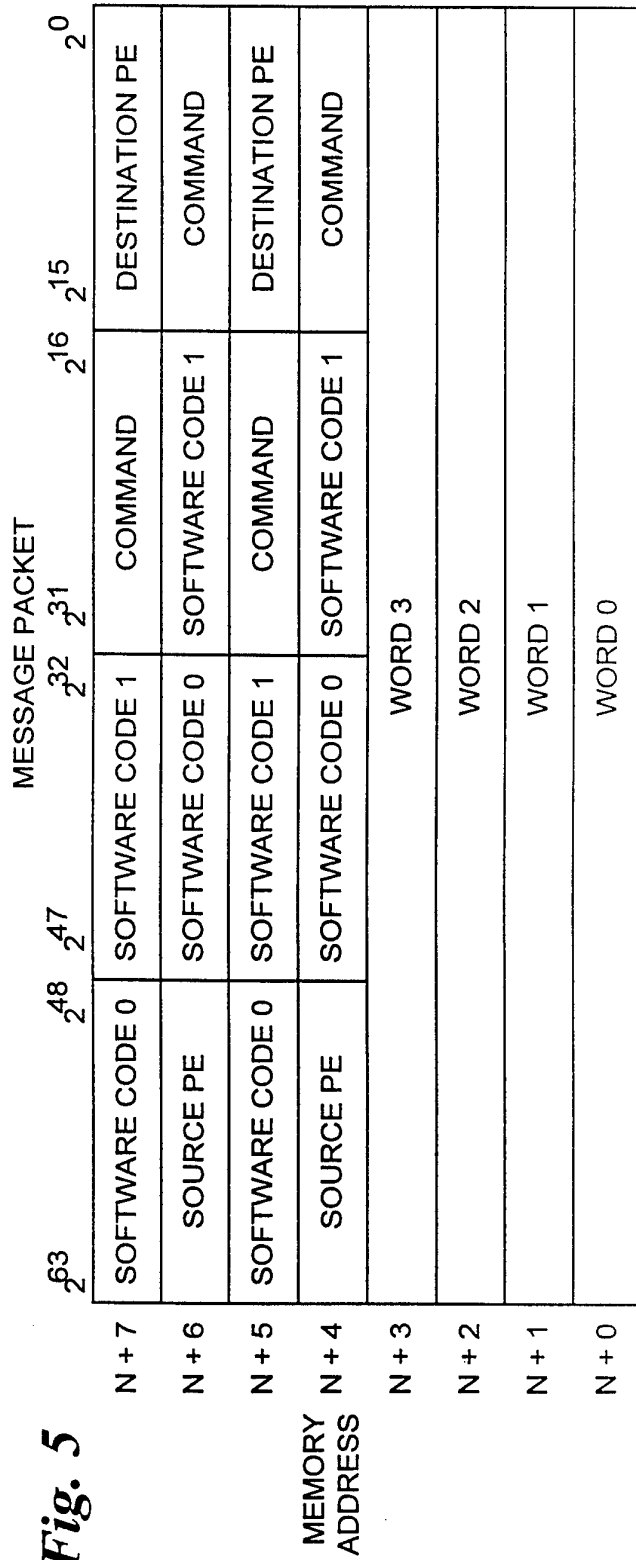
Fig. 2

*Fig. 3*



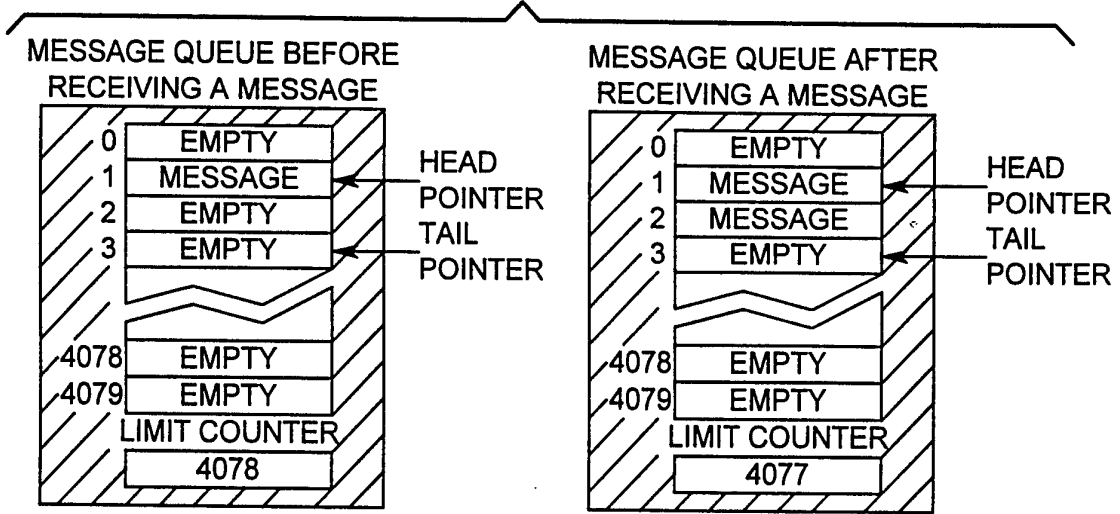
*Fig. 4*



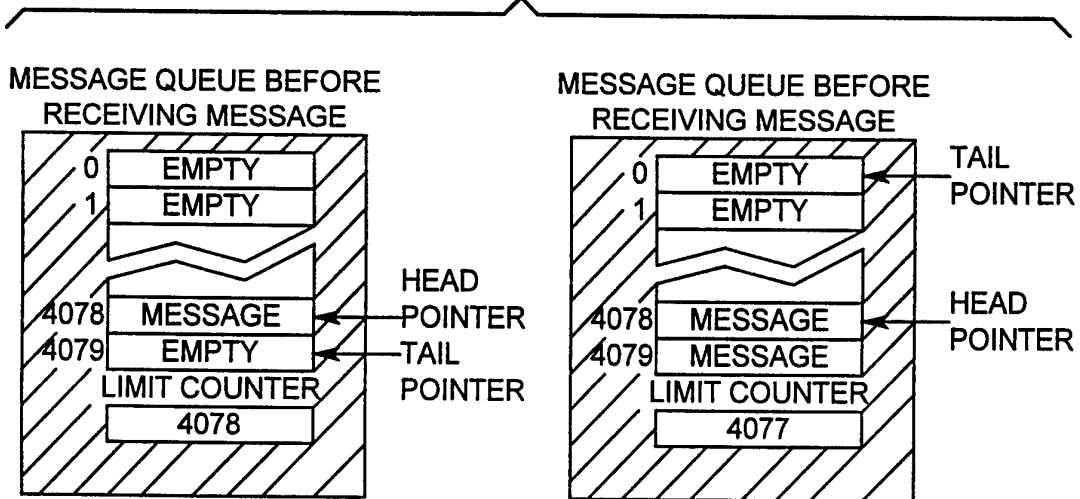


6/11

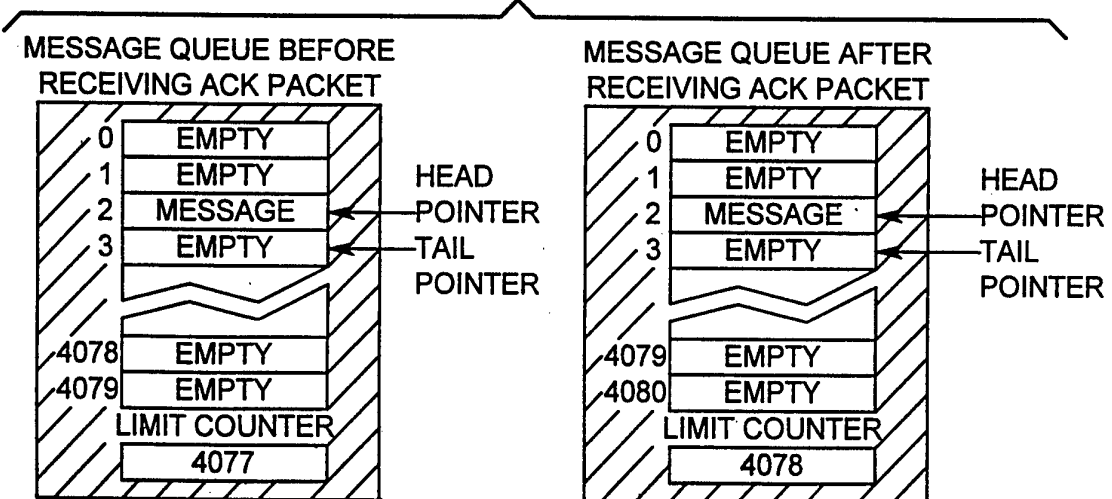
**Fig. 7**



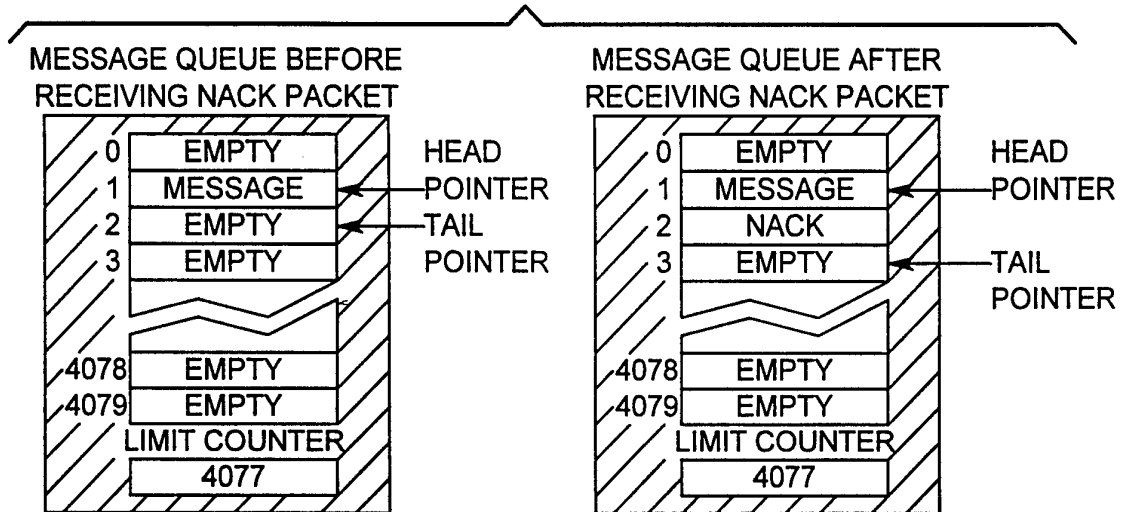
**Fig. 8**



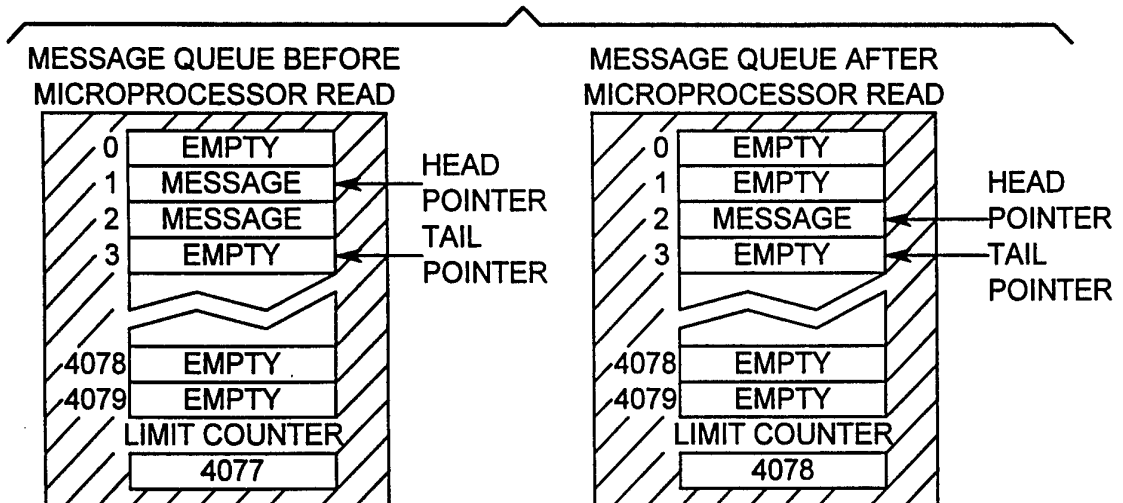
**Fig. 9**



**Fig. 10**



**Fig. 11**



**Fig. 12a**

PACKET TYPE 0		2 <sup>32</sup> 2 <sup>31</sup>	2 <sup>16</sup> 2 <sup>15</sup>	2 <sup>0</sup>
N + 7	NOT VALID DATA	NOT VALID DATA	† COMMAND	DESTINATION PE
N + 6	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	COMMAND
N + 5	NOT VALID DATA	NOT VALID DATA	† COMMAND	DESTINATION PE
N + 4	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	COMMAND
N + 3		WORD 3 (DOES NOT CONTAIN VALID DATA)		
N + 2		WORD 2 (DOES NOT CONTAIN VALID DATA)		
N + 1		WORD 1 (DOES NOT CONTAIN VALID DATA)		
N + 0		WORD 0 (DOES NOT CONTAIN VALID DATA)		

MEMORY ADDRESS

THE ERROR BIT (BIT 2<sup>15</sup>) IS NOT SET IN THESE COMMAND PHITS

8/11

**Fig. 12b**

PACKET TYPE 1		2 <sup>32</sup> 2 <sup>31</sup>	2 <sup>16</sup> 2 <sup>15</sup>	2 <sup>0</sup>
N + 7	REQUEST ADDRESS 0	REQUEST ADDRESS 1	† COMMAND	DESTINATION PE
N + 6	SOURCE PE	REQUEST ADDRESS 1	REQUEST ADDRESS 1	COMMAND
N + 5	REQUEST ADDRESS 0	REQUEST ADDRESS 1	† COMMAND	DESTINATION PE
N + 4	SOURCE PE	REQUEST ADDRESS 1	REQUEST ADDRESS 1	COMMAND
N + 3		WORD 3 (DOES NOT CONTAIN VALID DATA)		
N + 2		WORD 2 (DOES NOT CONTAIN VALID DATA)		
N + 1		WORD 1 (DOES NOT CONTAIN VALID DATA)		
N + 0		WORD 0 (DOES NOT CONTAIN VALID DATA)		

MEMORY ADDRESS

† THE ERROR BIT (BIT 2<sup>15</sup>) IS NOT SET IN THESE COMMAND PHITS

**Fig. 12c**

PACKET TYPE 2

63	2	48	2	47	2	32	2	31	2	16	2	15	2	0
N + 7	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1
N + 6	SOURCE PE	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	DESTINATION PE
N + 5	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	COMMAND
N + 4	SOURCE PE	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	REQUEST ADDRESS 0	REQUEST ADDRESS 1	DESTINATION PE
N + 3	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	COMMAND
N + 2	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	RESPONSE ADDRESS 1
N + 1	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	RESPONSE ADDRESS 1
N + 0	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	RESPONSE ADDRESS 1

THE ERROR BIT (BIT 2<sup>15</sup>) IS NOT SET IN THESE COMMAND PHITS

MEMORY ADDRESS

SUBSTITUTE SHEET (RULE 26)

9 / 11

**Fig. 13a**

PACKET TYPE 3

63	2	48	2	47	2	32	2	31	2	16	2	15	2	0
N + 7	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	DESTINATION PE
N + 6	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	COMMAND
N + 5	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	DESTINATION PE
N + 4	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	NOT VALID DATA	COMMAND
N + 3	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	
N + 2	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	
N + 1	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	
N + 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	WORD 0	

†THE ERROR BIT (BIT 2<sup>15</sup>) IS NOT SET IN THESE COMMAND PHITS

MEMORY ADDRESS



**Fig. 13b**

PACKET TYPE 4

63	48	47	32	31	16	15	2	0
N+7	REQ/RES ADDRESS 0	REQ/RES ADDRESS 1	COMMAND	REQ/RES ADDRESS 1	COMMAND	DESTINATION PE		
N+6	SOURCE PE	REQ/RES ADDRESS 0	COMMAND	REQ/RES ADDRESS 1	COMMAND	DESTINATION PE		
N+5	REQ/RES ADDRESS 0	REQ/RES ADDRESS 1	COMMAND	REQ/RES ADDRESS 1	COMMAND	DESTINATION PE		
N+4	SOURCE PE	REQ/RES ADDRESS 0	COMMAND	REQ/RES ADDRESS 1	COMMAND	DESTINATION PE		
N+3		WORD 0						
N+2		WORD 0						
N+1		WORD 0						
N+0		WORD 0						

MEMORY ADDRESS

THE ERROR BIT (BIT 2<sup>15</sup>) IS NOT SET IN THESE COMMAND PHITS

SUBSTITUTE SHEET (RULE 26)

**Fig. 13c**

PACKET TYPE 5

63	48	47	32	31	16	15	2	0
N+7	NOT VALID DATA	NOT VALID DATA	COMMAND	NOT VALID DATA	COMMAND	DESTINATION PE		
N+6	NOT VALID DATA	NOT VALID DATA	COMMAND	NOT VALID DATA	COMMAND	DESTINATION PE		
N+5	NOT VALID DATA	NOT VALID DATA	COMMAND	NOT VALID DATA	COMMAND	DESTINATION PE		
N+4	NOT VALID DATA	NOT VALID DATA	COMMAND	NOT VALID DATA	COMMAND	DESTINATION PE		
N+3		WORD 3						
N+2		WORD 2						
N+1		WORD 1						
N+0		WORD 0						

MEMORY ADDRESS

THE ERROR BIT (BIT 2<sup>15</sup>) IS NOT SET IN THESE COMMAND PHITS

**Fig. 14**

PACKET TYPE 6, MESSAGE PACKET, OR NACK PACKET

	$2^{63}$	$2^{48}$	$2^{47}$	$2^{32}$	$2^{31}$	$2^{16}$	$2^{15}$	$2^0$
N + 7	REQ/RES ADDRESS 0	REQ/RES ADDRESS 1	†COMMAND	DESTINATION PE				
N + 6	SOURCE PE	REQ/RES ADDRESS 0	REQ/RES ADDRESS 1	COMMAND				
N + 5	REQ/RES ADDRESS 0	REQ/RES ADDRESS 1	†COMMAND	DESTINATION PE				
N + 4	SOURCE PE	REQ/RES ADDRESS 0	REQ/RES ADDRESS 1	COMMAND				
N + 3	WORD 3							
N + 2	WORD 2							
N + 1	WORD 1							
N + 0	WORD 0							

MEMORY ADDRESS

†THE ERROR BIT (BIT  $2^{15}$ ) IS NOT SET IN THESE COMMAND PHITS

INTERNATIONAL SEARCH REPORT

International Application No  
PCT/US 94/14349

A. CLASSIFICATION OF SUBJECT MATTER  
IPC 6 G06F15/16

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)  
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP,A,0 460 599 (THINKING MACHINE CORPORATION) 11 December 1991 see column 5, line 48 - column 7, line 55 see column 11, line 3 - column 16, line 22 ---	1-14
A	WO,A,88 08652 (D. HILLIS) 3 November 1988 see page 7, line 12 - line 34 see page 12, line 8 - page 13, line 10 see page 14, line 8 - line 34 see page 16, line 6 - page 17, line 33 see page 20, line 8 - line 23 ---	1,13,14
A	COMPCON SPRING '93, 22 February 1993, SAN FRANCISCO, CA, US pages 312 - 320 BRADLEY R. CARLILE 'The CRAY APP shared memory system' see the whole document -----	1-14

Further documents are listed in the continuation of box C.

Patent family members are listed in annex.

\* Special categories of cited documents :

- "A" document defining the general state of the art which is not considered to be of particular relevance
- "E" earlier document but published on or after the international filing date
- "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- "O" document referring to an oral disclosure, use, exhibition or other means
- "P" document published prior to the international filing date but later than the priority date claimed

- "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- "&" document member of the same patent family

Date of the actual completion of the international search

31 March 1995

Date of mailing of the international search report

18. 04. 95

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2  
NL - 2280 HV Rijswijk  
Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl,  
Fax: (+ 31-70) 340-3016

Authorized officer

Soler, J

**INTERNATIONAL SEARCH REPORT**

Information on patent family members

International Application No

**PCT/US 94/14349**

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
EP-A-0460599	11-12-91	CA-A- 2043505		07-12-91
		JP-A- 4232561		20-08-92
-----				
WO-A-8808652	03-11-88	US-A- 4984235		08-01-91
		EP-A- 0381671		16-08-90
		JP-T- 2503244		04-10-90
		US-A- 5117420		26-05-92
-----				