PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

MULTIPORT MEMORY ARCHITECTURES

Yuzuru Tanaka

Electrical Engineering Department
Hokkaido University
Sapporo, 060 Japan

ABSTRACT

This paper shows how a k-port memory system can be made of single-port memory devices or multiport memory devices with fewer ports. Access conflicts never occur in such a system unless more than one port tries to write to the same address. A k-port memory system with N words, however, requires $O(k^2)$ devices with N words for each. This large redundancy makes it impractical to select a large k. The access time of a k-port memory system is a sum of the device access time and the time to calculate the parity of k bits. The latter is proportional to $\log_2 k$. For a small k, The latter component can be neglected. Since we are interested in the applications to multiprocessors with a shared memory, the number k of our concern is within the range from 4 to 16, and hence the $\log_2 k$ component of the access time may be neglected. Such a multiport RAM system (MPM) can be used as a multiport cache memory to form a larger multiport memory together with a previously proposed multiport page-memory (MPPM) (Tanaka 1984a, 1984b). This configuration reduces the cost of the multiport memory system from $O(k^2)$ to $O(1)$. Such a multiport cache architecture outperforms parallel cache architectures, especially for $k > 6$.

1 INTRODUCTION

Progress of parallel processing has made us realize the insufficient communication bandwidth between a processing unit and a memory unit (Hwang and Briggs 1985). This bottleneck is sometimes called von Neumann's bottleneck. Super computers are coming to require a multiported vector-register file and a highly interleaved main memory to match the data provision capability of memory systems to the high-speed pipeline computation power of multiple operation pipes. General purpose computers adopt both instruction pipeline and operation pipeline, and require cache and memory interleaving technologies to match them. Some consist of more than two processors sharing a memory space. High-performance computers require more powerful pumps to provide the data processing unit with a sufficient amount of data in a sufficient speed. Parallel database architectures are confronted with a 'disk paradox' indicating that the commercial disk unit development efforts, directed toward more capacity, result in the use of fewer units to store the same database and, hence, lower the maximum data provision power to nullify the parallel processing (Boral and DeWitt 1983).

Multiprocessor architectures are aiming for a shared memory architecture better than parallel cache architectures. While SIMD array processors performs multiple operations with regularly arranged parallelism among them, multiprocessors perform multiple processes having no regular structure among them but sharing a single memory space and/or exchanging messages. Concurrent memory accesses in multiprocessors have no such regular patterns as can be seen in SIMD array processors, which makes it difficult to schedule concurrent memory accesses prior to the execution, therefore requiring a shared memory architecture. Parallel cache architectures have to maintain the cache coherence (Dubois and Briggs 1982, Goodman 1983, Papamarcos and Patel 1984, Katz et al. 1985, McCreight 1984, and Archibald and Baer 1986). Hence they somehow require sequential execution of concurrent write requests to a common single-port memory or a common bus.

All these problems indicate the requirement of larger communication bandwidth between processors and memory banks. The idealistic solution to these problems is the multiplication of the memory I/O ports without deteriorating the access rate of each port. This paper shows that such a multiport memory used as a multiport cache together with a previously proposed multiport page-memory significantly outperforms parallel cache architectures especially for applications to tightly coupled multiprocessors with 4 to 16 processors.

In the current state of the art, however, multiport memory devices with more than 2 ports are only available as small capacity memory devices mainly used as register files. Their applications to shared main memory systems require not only further integration of circuits but also more I/O pins to and from a single chip.

This paper gives an alternative approach to implement multiport memory systems. Instead of circuit-technological methods used in multiport memory devices, we will show architectural methods to build a multiport memory system using memory devices with fewer ports. A trivial example of how we can multiply memory ports is a multiport read-only memory using one copy of the same memory unit for each port (Figure 1.1). This implementation requires as many memory copies as the number of memory ports.

A nontrivial example is a multiport page memory (MPPM) proposed by the present author in 1984 (Tanaka 1984a). In an MPPM, an access unit is not a word but a page. Each access reads or writes a whole page. The size of a page, or the number of words in a page, is restricted to be a multiple of the number of ports. As shown in Figure 1.2, an MPPM consists of a rotary switch and as many memory banks as the number of its ports. The rotary switch repetitively changes the connections between the set of ports and the set of banks so that the $(i+1)$st port may connect to the $(((i+j) \bmod k)+1)$st bank at the $(j+1)$st clock, where k stands for the number of ports. Each page is stored across the k banks as shown in Figure 1.2. A number assigned to each word in a page indicates the displacement of that word in a page. The MPPM architecture

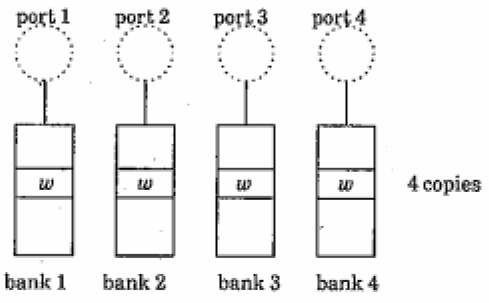


Figure 1.1 A trivial implementation of a 4-port read-only memory.

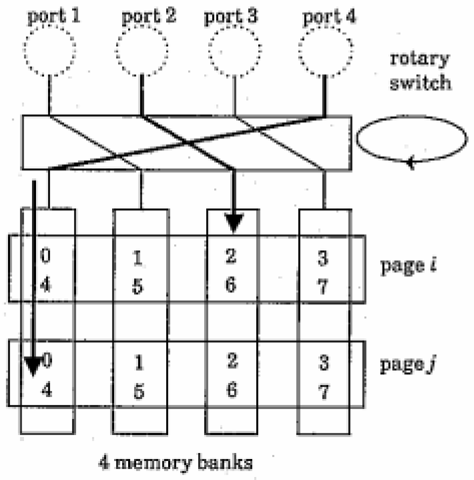


Figure 1.2 An implementation of a 4-port page memory.

is based on the fact that an access of a page does not imply any preferred order of word accesses in the page. Suppose that we want to start the access to page i through the second port when this port is connected to the third bank. Suppose also there are 4 ports, and that the page size is 8. The third bank stores the words at the displacements 2 and 6 in page i. Therefore, the second port can instantaneously access either of these words. At the next clock, this port will be connected to the fourth bank, and will be able to access the word at a displacement of either 3 or 7. Therefore, the second port can access all the words in page i in the order of displacements 2, 3, 4, 5, 6, 7, 0, and 1. These accesses are not influenced by any accesses through other ports. An MPPM requires no redundant storage, and hence, implements a wide range of k, i.e., from a couple of ports to hundreds of ports.

This paper gives another nontrivial class of multiport memory architectures, i.e., a multiport RAM. A multiport RAM, or simply a multiport memory (MPM), allows random accesses of words concurrently requested from multiple ports. Construction of a multiport RAM from memory devices with fewer ports has been believed infeasible. This paper gives an implementation requiring $O(k^2)$ redundant use of memory and $O(\log_2 k)$ time for each access. Because of this cost and access time, our architecture is only applicable to small k. The maximum feasible k is determined by the cost of redundancy rather than by the access time. This limit on k depends on the number of ports of each constituent memory device. If single port memory devices are used, our multiport memory requires k^2 redundant use of memory. In this case, the maximum feasible k may be no larger than 5. If dual port memory devices are used, the redundancy becomes ${}_kC_2$. Now the maximum feasible k may be doubled. If quad port memory devices are used, the redundancy becomes ${}_{(k/2)}C_2$. This may further double the maximum feasible k. We think that the maximum feasible k is roughly no more than $4m$, where m is the number of ports of each constituent memory device. Therefore, our architecture is currently applicable to multiprocessors with no more than 8 processors. The maximum number of processors will be doubled to 16 in a few years.

The architectural implementation gives larger capacity and more ports than current multiport memory devices. The architectural method and the circuit-technological method should not be considered as competitors but complementary methods. When device technology comes to give larger capacity and more ports, the architectural technologies will enable us to assemble those devices to provide a larger number of concurrently accessible ports.

The next section defines multiport memories and access conflicts. The third section shows problems of parallel cache architectures, and their solution by a multiport cache architecture. Section 4 gives a multiport RAM implementation with single-port memories, while Section 5 shows another implementation using dual port memories. Section 6 generalizes these architectures to use multiport memories with an arbitrary number of ports. Section 7 concludes this paper.

2 VARIOUS ACCESS CONFLICTS IN MULTIPORT MEMORIES

A shared memory with k ports allows its k ports to concurrently access information stored in it. An access unit may be a word or a page. A memory with a page as its access unit is called a page memory. Each port of a shared memory may access an access unit at any address. An access of a shared memory from one port may block an access from another port. This situation is called a conflict. A shared memory resolves a conflict among accesses by selecting one of them to proceed with others being kept waiting.

Access conflicts in a shared memory are caused by the physical limitations of its constituent memory devices. For example, an ordinary memory device has only one port and does not allow two simultaneous accesses unless they are read accesses to the same access unit. We call this kind of conflict a physical conflict. Physical conflicts are due to the limited number of concurrently accessible ports provided by a single memory device.

Even if we can remove physical conflicts by providing a memory with a sufficient number of concurrently accessible ports, we have still another type of conflict called a logical conflict. If two write accesses simultaneously occur to the same access unit, their effect can not be logically defined without losing the symmetry between two accessing ports. Suppose, for example, that port i_1 tries to write v_1 while port i_2 tries to write v_2. If port i_1 is given the higher priority, the access unit is updated to v_1. However, this loses the symmetry between two ports because the exchange of port i_1 and port i_2 updates the same access unit to a different value v_2.

Any logical conflict can be resolved if we discard symmetry between ports and give the higher priority to one of them. Even if we resolved physical and logical conflicts, we may still have another type of conflict if the access of an access unit requires more than one access. A page access, for example requires as many memory accesses as the words in one page. If a page is to be updated, this operation should be atomic, i.e., no read nor write of this page should be allowed before the completion of this update. Otherwise, some part of a page may be updated by one port and the remaining part by another port.

Suppose, for example, that a page has two words w_1 and w_2, and that port i_1 and port i_2 try to write (a, b) and (c, d) respectively on (w_1, w_2). We assume that no physical conflicts occur. While a logical conflict prevents these two ports to modify either w_1 or w_2 simultaneously, these two ports can concurrently modify w_1 and w_2 in two different orders, i.e., for example, port i_1 modifies w_1 and w_2 in this order while port i_2 in the reverse order. This concurrent execution modifies (w_1, w_2) to (c, b), which is different from either of the two update results. Similarly, if a page operation is not atomic, a read of a page may get the before value for some part and the after value for the remaining of the same page. Suppose that port i_1 modifies (w_1, w_2) with an initial value (a, b) to (c, d) while port i_2 concurrently reads the same page (w_1, w_2). If the second port accesses the two words in a different order to avoid logical conflicts, the port i_2 may get either (a, d) or (c, b) as read out values. Neither of these values are equal to either the before value (a, b) nor the after value (c, d). Any access to a page under modification by some other port should be forced to wait until the modification is complete. While each page may be concurrently accessed by any number of read operations, it can accept only one write operation with no concurrent read operations. Conflicts caused by this restriction are called structural conflicts.

Among three kinds of conflicts, only physical conflicts are due to hardware restrictions. It is desirable to remove physical conflicts. A multiport memory is a shared memory that causes no physical conflicts. This implies that any access request is immediately accepted and performed without being forced to wait unless it causes any logical or structural conflict. If its access unit is a page, it is called a multiport page memory. A multiport memory with a word as its access unit is called a multiport RAM. A multiport page memory has both logical conflicts and structural conflicts, while a multiport RAM has only logical conflicts.

Logical conflicts as well as structural conflicts are easy to detect. The detection only requires the comparison of addresses sent by the ports. The detection and arbitration circuits can be externally provided independent from the memory. This organization allows a memory itself to allow more than one logically conflicting or structurally conflicting access. The updated result may not be correct if the update is involved in logical or structural conflicts. Such a memory, without external detection, is called a kernel memory. In the following of this paper, 'multiport memory' stands for 'kernel multiport memory' unless otherwise specified. A multiport memory in this sense has no conflicts. From the definition, they have no physical conflicts.

3 PROBLEMS OF PARALLEL CACHE ARCHITECTURES AND THEIR SOLUTION

Parallel cache architectures were the only known techniques to implement a shared memory for a multiprocessor system with 4 to 16 processors (Dubois and Briggs 1982, Goodman 1983, Papamarcos and Patel 1984, Katz et al. 1985, McCreight 1984, and Archibald and Baer 1986). Their effective access time, however, significantly increases as the number of ports increases in this range.

A parallel cache has a configuration as shown in Figure 3.1. We assume that each cache is dual-ported. This assumes

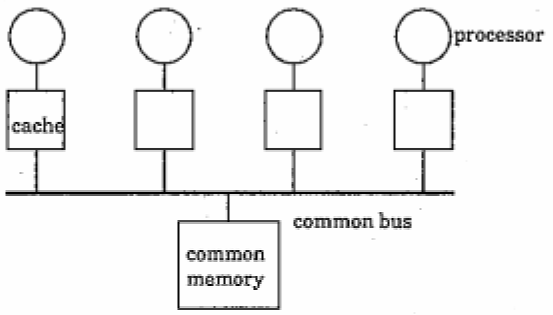


Figure 3.1 A parallel cache architecture.

an idealistic situation. The processor accesses its cache through one of these two ports, while the common bus accesses it through the other. Each read access by one of the processors first accesses its dedicated local cache C. If it finds the word in it, the access completes. Otherwise, it sends out the address to the common bus. The bus accesses every cache through one of its ports to find out this word. If it is found in another cache, the block containing this word is read and its copy is transferred to the local cache C. Otherwise, the block is read from the common memory and sent to the local cache C. Each write access writes the value both in the local cache and the common memory. For simplicity of our analysis, we adopt the write-through policy with the bus-snooping technique for the coherency maintenance. Different from the simple write-through policy, it gives almost as good performance as any write-back policies. In the bus-snooping, each write request is also sent to other cache memories through the common bus to update the copies of the accessed words, and to maintain cache coherence. This uses only one of the two ports of each cache.

In our modeling of a parallel cache, we assume that local cache memories and the common memory have a same access time. We further assume that this access time is same as the clock cycle of the system. Each processor is assumed to take two machine cycles to execute any instruction. It uses the first cycle for the instruction fetch and the second for the execution. Each machine cycle takes one clock cycle at least. All the assumptions above set a rather idealistic situation so that our analysis based on them may approximately give the upper bound of the system performance. Let α denote the proportion of the memory-read machine cycles, β the proportion of the machine cycles without memory accesses, and γ the cache hit rate. Since each instruction fetch reads the shared memory, the value of α is no less than 0.5. The number of local cache memories is denoted by k. The memory space is divided into blocks of B words for each. The bus and the common memory constitute the common resource. An access to this common resource blocks further accesses to itself.

A machine cycle accesses the common resource if and only if it executes either a memory write operation or a memory read operation resulting in a cache miss. In the latter case, the machine cycle is expanded to B clocks to complete the block transfer. We denote by P the probability that a machine cycle accesses the common resource. We obtain P as

$$P=(1-\alpha-\beta)+\alpha(1-\gamma). \qquad (3.1)$$

We denote, by T_C, the mean length of one machine cycle that accesses only the common resource. This value is calculated as

$$T_C=(1/P)((1-\alpha-\beta)+\alpha(1-\gamma)B). \qquad (3.2)$$

In a steady state, we assume W ports are requesting accesses to the common resource. Some of them are requesting a single word access, while others are requesting a block access. We may think that each port is requesting T_C word accesses to the common resource on the average. Therefore, there are, on the average, $T_C W$ word-access requests to the common resource. During the next clock cycle, one of them is performed. At the same time, the remaining $(k-W)$ ports issues access requests to the common resource with the probability P. Therefore, we obtain the following equality;

$$T_C W = T_C W - 1 + (k-W)PT_C, \qquad (3.3)$$

which gives

$$W = k - (1/(PT_C)). \qquad (3.4)$$

This approximation of W is correct if there exists a steady number of access requesting ports, i.e., if $W \geqq 1$.

We denote, by T_P, the mean length of one machine cycle. We obtain this length as

$$T_P = (1-P) + P((W-(1/2))T_C + (((k-W)P+1)/2)T_C) \quad (3.5)$$

for $W \geqq 1$. The first term represents the case where the accessed word is found in the local cache. The second term corresponds to a cache miss. There are W ports requesting accesses to the common resource in the steady state. One of them is under execution. On the average, half of it is executed. Therefore, a new access to the common resource has to wait $(W-(1/2))T_C$ time at least. During the current clock cycle, the remaining $(k-W)$ ports newly request accesses to the common resource with the probability P. Therefore, we have $(k-W)P$ more ports including the port of our concern. The request by the port of our concern will be acknowledged as the $(((k-W)P+1)/2)$-th request on the average. Therefore, the port of our concern has to wait $(((k-W)P+1)/2)T_C$ more time for the completion of its access to the common resource. Summation of these gives T_P as above. Using (3.1) and (3.2), we obtain

$$T_P = (1-\alpha-\beta)(k-(1/2)) + \alpha(1-\gamma)(kB-(1/2)). \qquad (3.6)$$

If $W < 1$, we obtain T_P as

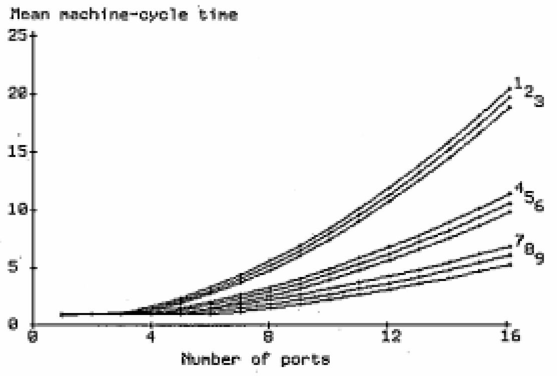
$$T_P = (1-P) + P(kP+1)T_C/2. \qquad (3.7)$$

In such a case, no ports are accessing the common resource when one of them requests access to it. The number of ports that request access to the common resource simultaneously at this same time is approximately kP. Therefore, one of them takes $(kp+1)T_C/2$ time on the average to complete its access to the common resource.

We show, in Figure 3.2, the value of T_P as a function of k for various values of α, β and γ. The size B of a block is selected to be k to compare these results with the performance of our new architecture proposed in the following. These graphs show serious increase of T_P as k increases. The effective performance of a multiprocessor with k processors provided with this parallel cache is no more than k/T_P, which is shown in Figure 3.3 as a function of k. Especially for $k>6$, we see serious performance decreases by the use of additional processors under the assumption that α, β and γ do not change.

For a constant-size block, say $B=16$, the effective number of processors changes as shown in Figure 3.4. This shows performance saturation around four processors. The small bump around each shoulder is due to the rough approximation around $W=1$.

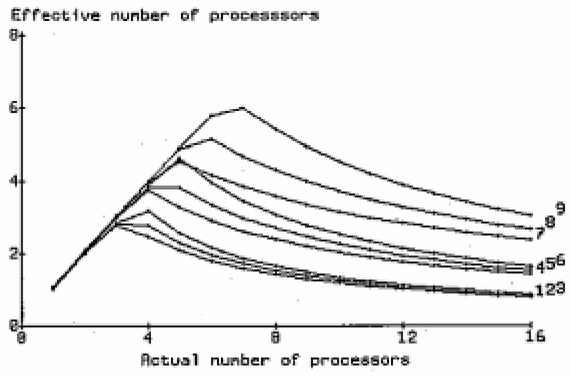
We show in the sequel that a multiport cache significantly outperforms parallel cache architectures for $6 < k \leqq 16$ where we lacked a solution to high-performance shared-memory multiprocessor systems. A k-port cache memory system consists of a k-port RAM and a k-port page memory. Its block size B is selected as a multiple of k. Figure 3.5 shows a configuration of a k-port cache. Its mean length T_M of one



$\alpha=0.7$

#	γ	β	#	γ	β	#	γ	β
1	0.9	0.15	4	0.95	0.15	7	0.975	0.15
2	0.9	0.20	5	0.95	0.20	8	0.975	0.20
3	0.9	0.25	6	0.95	0.25	9	0.975	0.25

Figure 3.2 Mean machine-cycle length T_P of a multiprocessor using a parallel cache architecture ($B=k$).



$\alpha=0.7$

#	γ	β	#	γ	β	#	γ	β
1	0.9	0.15	4	0.95	0.15	7	0.975	0.15
2	0.9	0.20	5	0.95	0.20	8	0.975	0.20
3	0.9	0.25	6	0.95	0.25	9	0.975	0.25

Figure 3.3 Effective multiplicity of processors sharing a parallel cache memory ($B=k$).

machine cycle is equal to that of a single processor with a cache divided into blocks of B words. We assume that the cache and each bank of the MPPM have the same access time that is equal to the clock cycle. We adopt the flagged-write-back strategy in which the MPPM is updated only when a dirty block in the cache has to be replaced with a new block in MPPM. A block in the cache is dirty if it has been modified during its stay in the cache. We denote by δ a probability that we have to replace a dirty block. The MPPM is accessed only when a cache miss occurs. We have to read out the block containing the accessed word from MPPM to the cache after making a room in the cache for this block. This requires a replacement. Therefore, We obtain T_M as

$$T_M = \beta + (1-\beta)(\gamma + (1-\gamma)(1+\delta)B). \qquad (3.8)$$

We show T_M in Figure 3.6 as a function of k for $B=k$, $\beta=0.2$,

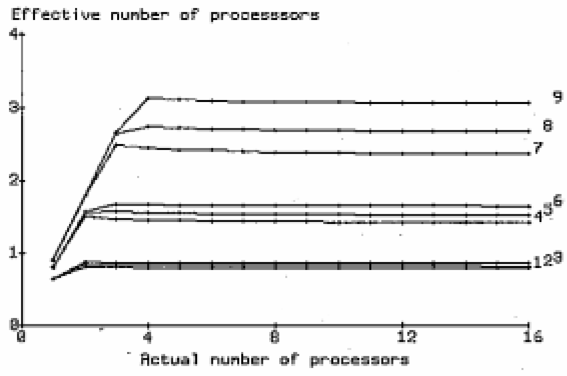


Figure 3.4 Effective multiplicity of processors sharing a parallel cache memory ($B=16$).

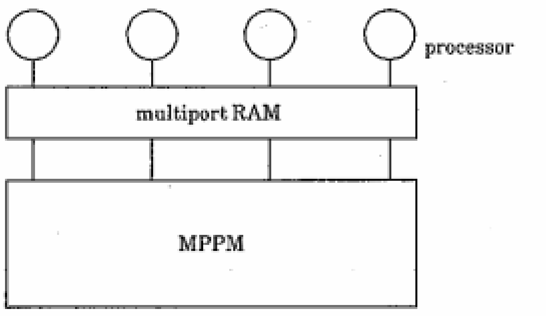
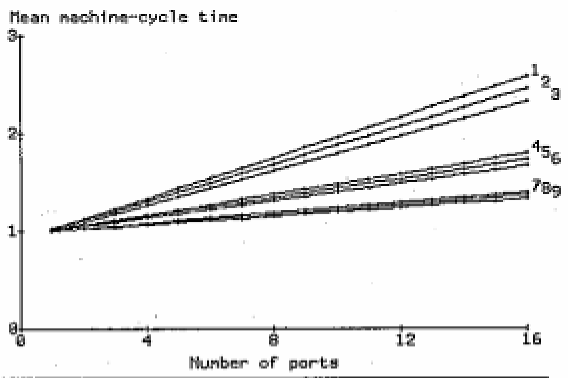


Figure 3.5 A multiport cache architecture.

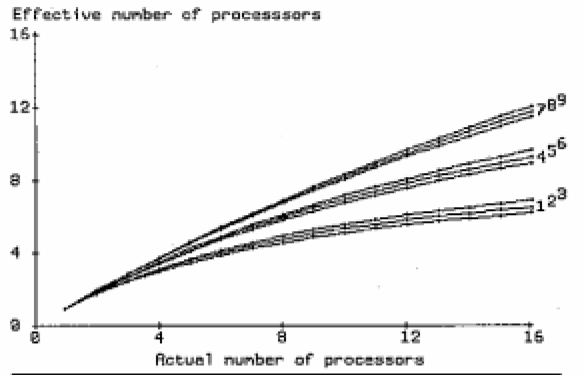


#	γ	δ	#	γ	δ	#	γ	δ
1	0.9	0.3	4	0.95	0.3	7	0.975	0.3
2	0.9	0.2	5	0.95	0.2	8	0.975	0.2
3	0.9	0.1	6	0.95	0.1	9	0.975	0.1

Figure 3.6 Mean machine-cycle length T_M of a multiprocessor using a multiport cache architecture ($B=k$).

and some different values of γ and δ.

We show in Figure 3.7 the value of k/T_M as a function of k, where B is assumed to be equal to k. This value k/T_M represents the effective multiplicity of processors. Different from parallel cache architectures, the multiport cache architecture allows more than 6 processors to share a memory space without seriously deteriorating the effective number of processors from the actual number of processors.



#	γ	δ	#	γ	δ	#	γ	δ
1	0.9	0.3	4	0.95	0.3	7	0.975	0.3
2	0.9	0.2	5	0.95	0.2	8	0.975	0.2
3	0.9	0.1	6	0.95	0.1	9	0.975	0.1

Figure 3.7 Effective multiplicity of processors sharing a multiport cache memory ($B=k$).

In Figure 3.8, we show the effective multiplicity of processors for the constant block size (i.e., $B=16$). Different from Figure 3.4, we have no performance saturation in this case.

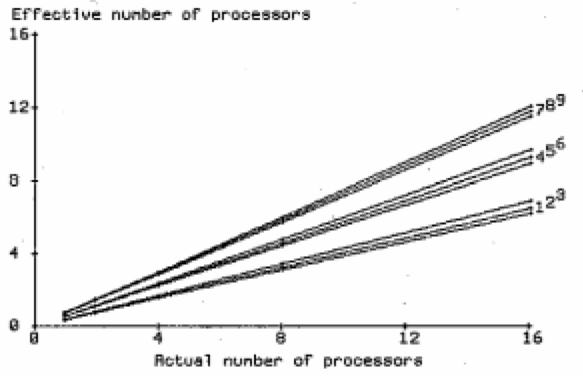


Figure 3.8 Effective multiplicity of processors sharing a multiport cache memory ($B=16$).

The analysis given in this section gives a sufficient reason to develop a multiport RAM architecture providing 4~16 ports.

4. MULTIPORT RAM ARCHITECTURE USING SINGLE-PORT MEMORY

It is well known that a dual port memory can be made of two single port memory banks as shown in Figure 4.1. It uses a a pair of banks. The two ports are always connected to different banks. They exchange the partner banks alternatively. Each read access accesses the current partner bank, while each write access spends two consecutive phases to write the both two memory banks. This architecture, however, can be extended to no more than two ports. The difficulty lies in the fact that the write access cannot be locally performed even if we use redundant storage of information.

Instead of a write access, we consider an inversion of an accessed word. For simplicity, we assume that a word consists of a single bit. This does not result in any loss of generality. Any write operation can be performed by an inversion operation after a read operation. If a bit value read out is

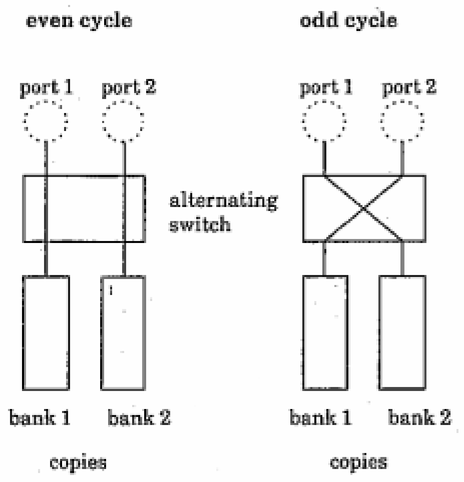


Figure 4.1 A well-known implementation of a dual port memory with two single-port memory banks.

different from a new value to write, an inversion of the addressed word performs this write operation. Otherwise, no operation is needed since the write operation does not change the bit value. While a write operation cannot be locally performed, an inversion can be localized as shown in Figure 4.2. This k-port memory system uses k memory banks. Each

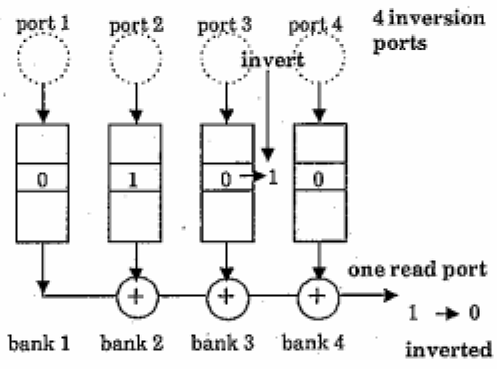


Figure 4.2 Local execution of an inversion operation on a shared data.

one bit word is stored in k one bit words across k different memory banks. The value of this word is defined as the parity of these k corresponding words. Since the parity function of k bits changes its value if only one of these k bits changes its value, each port of the memory in Figure 4.2 can invert any word in the memory by accessing only its dedicated bank. The memory in Figure 4.2, however, has only one read port. To provide sufficient number of read ports, we may multiply this memory system. The result is shown in Figure 4.3. Its access cycle consists of two phases, i.e., a read phase and an inversion phase.

Each read access waits for the next read phase, uses this phase to read the memory, and waits during the next inversion phase. Each write access waits for the next read phase, uses this to read the word, and spends the next inversion phase to invert this word if necessary. A read access from a read port first accesses the same address of k memory banks connected to this read port and calculate the parity of these k bits to obtain the stored value. A write access through

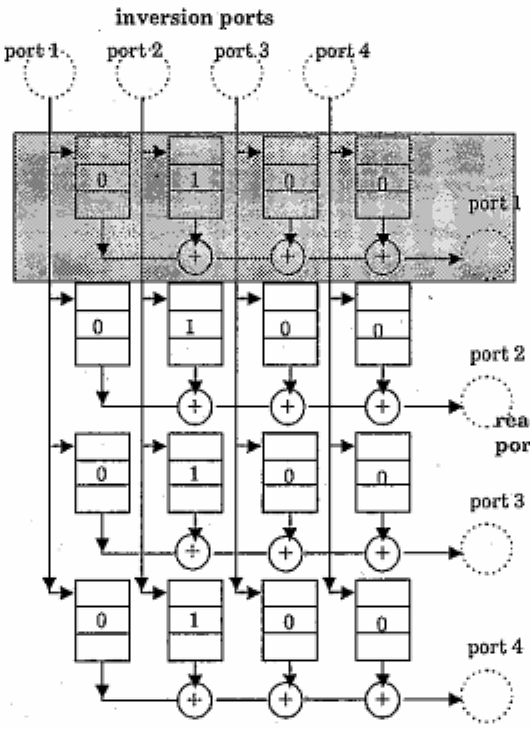


Figure4.3 A 4-port RAM using 16 single port memory banks.

a port A also first performs a read access through the read port A. If the read out bit value is same as the value to write, it spends the next inversion phase without doing anything. Otherwise, it uses the inversion phase to invert k words at the same address of the k memory banks that are connected to the inversion port A.

This memory configuration removes physical access conflicts from concurrent read and write accesses. A k-port configuration of capacity W, however, requires k^2 memory banks each of which requires the same capacity W. The parity computation of k bits requires $O(\log_2 k)$ time. If k is small, this time is much shorter than the access time of the memory banks, and hence, can be neglected.

5. MULTIPORT RAM ARCHITECTURE USING DUAL PORT MEMORY DEVICES

It is also important to consider how a multiport memory can be made of fewer-port memory devices. Dual port memory devices are already available, and quadport memory devices are expected to come on the market in the near future.

This section shows how dual port memory devices can be assembled to implement a multiport memory system. We assume that each word consists of a single bit. This does not cause any loss of generality. A write access is again implemented by an inversion operation after a read access to localize its access to the memory bank dedicated to each port. Every pair of ports has to share the result of any write access issued by any of these two ports. To achieve this goal, one dual port memory bank is provided for each pair of ports. Since a set of k ports has ${}_kC_2$ different pairs of ports, this requires the same number of dual port memory banks. For any $1 \leqq i < j \leqq k$, we denote a bank for the ith and jth ports by M_{ij}. The ith port uses the first port of M_{ij}, while the jth port uses its second port.

Each of the k ports may be paired with other k-1 ports. Therefore, each port is connected to k-1 dual port memory banks. Each port accesses a word that is distributively stored in k-1 words across the k-1 dual port memory banks connected to this port. For the time being, we assume that k is an even number. In this case, each port is connected to an odd number of dual port memory banks, i.e., each one bit word is represented by an odd number of bits stored across k-1 memory banks.

We define the value of each one bit word as the parity of its associated k-1 bits. Then, as shown in Figure 5.1, an inversion

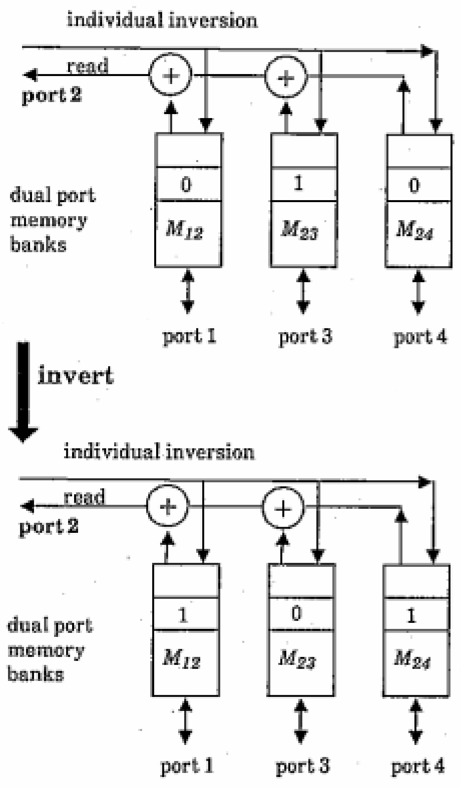


Figure 5.1 Local execution of an inversion operation through dual port memory banks.

of a word can be performed by individually inverting these k-1 bits by accessing k-1 different memory banks. Since we are assuming that logical conflicts are removed by an external hardware, no two ports simultaneously perform an inversion operation of the same word. Suppose that an inversion of a word W was performed by the ith port. For any j different from i, the jth port shares only one dual port memory with the ith port. Let this memory bank be M. If $i<j$, M is M_{ij}. Otherwise, it is M_{ji}. An inversion of a word through the ith port inverts the word in M, but does not change the words at the same address in other memory banks connected to the jth port. Since the value of each word is represented as the parity of its k-1 values stored across the k-1 memory banks connected to the accessing port, this word value is inverted by an inversion of one of these k-1 values. Therefore, An inversion of a word through any one of these ports also inverts the value of this word at every other port.

Figure 5.2 schematically shows a configuration of a 6 port memory system. Each horizontal line denotes an access line of each port, while each vertical line segment represents a dual

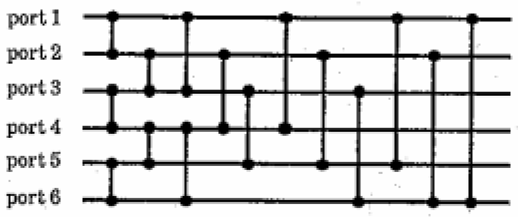


Figure 5.2 A 6-port memory configuration using 15 dual port memory banks.

port memory bank with two ports marked by dark circles. All the bits in each memory bank are assumed to be initially reset to zero. A read and a write access respectively take one and two memory cycles. Logical conflicts are externally avoided. Each read access reads the specified address from the k-1 dual port memory banks that are connected to the accessing port, and gets their parity as a read out value. While the calculation of the parity takes $O(\log_2(k\text{-}1))$ time, it may be neglected compared to the memory access time if k is in the range of our concern, i.e., $4 \leqq k \leqq 16$. A write access reads out the value of the specified word in the first cycle, and in the second cycle, if the value is different from the value to write, inverts this word in each of the k-1 memory banks connected to the accessing port. If the old and the new values are found to be the same in the first cycle, the write access completes its operation in the first cycle and saves the next cycle for the next access.

In figure 5.3, we show how two consecutive write accesses

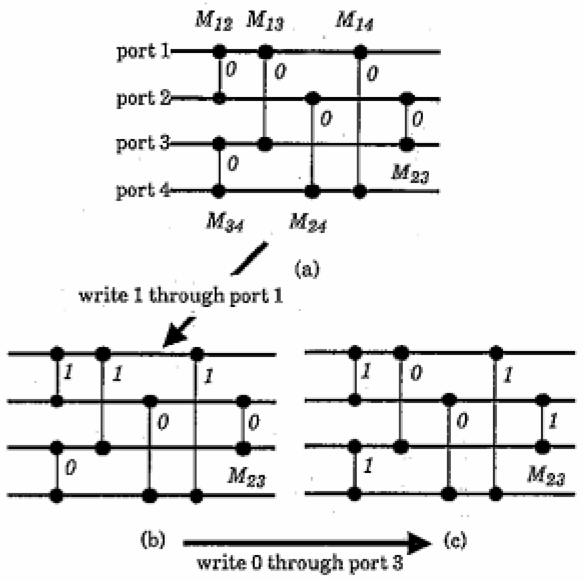


Figure 5.3 Two consecutive write operations in a 4-port memory system.

from different ports to a same address change the values of this address in 6 dual port memory banks that constitute a 4 port memory system. Initially, all the values of this word are reset to zero. After the first port has written 1, the values of this word in M_{12}, M_{13}, and M_{14} are changed to 1 as shown in (b). At this situation, each port reads out 1 as the value of this word. For example, the second port reads out 1, 0, 0 respectively from M_{12}, M_{23}, and M_{24} to get 1 as their parity. The second write operation is issued from the third port. It requests to write 0 to the same word. The third port first reads out this word to find that an inversion is necessary. It inverts the values of this word in M_{13}, M_{23}, and M_{34} as shown in (c). Their values become 0, 1, and 1 respectively. This makes any port to read out 0 from this word. For example, the second port may read out 1, 1, and 0 from M_{12}, M_{23}, and M_{24} to get 0 as their parity. The fourth port may read out 1, 0, and 1 respectively from M_{14}, M_{24}, and M_{34} to get 0 as their parity.

For any odd k, a k-port memory system with W words is implemented by the same configuration except that each port is also connected to a dedicated single port memory bank with W words. Figure 5.4 shows the case with $k=5$. Without

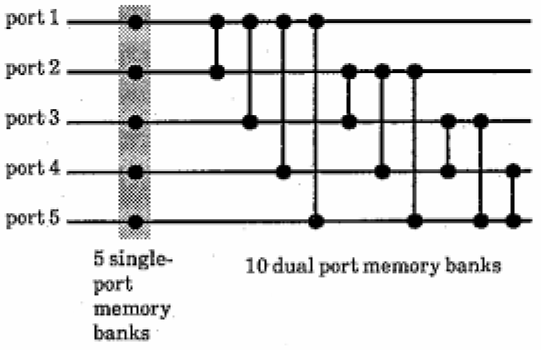


Figure 5.4 A 5-port memory requires 10 dual port memories and 5 single-port memories.

additional single port memory banks, an inversion of a word from one port P inverts its value accessed by every other port, but not the value accessible from the port P itself. In this case, k-1 becomes even. Individual inversion of an even number of bits does not change their parity. The added single port memory banks solve this problem. They are involved in each parity calculation and in each inversion operation.

6. MULTIPORT MEMORY USING MEMORY DEVICES WITH MORE THAN TWO PORTS

The multiport memory system configurations shown in the preceding chapter can be applied also to systems using memory devices with more than two ports. Multiport memory devices with 4 to 8 ports are technologically within a scope of consideration. Three approaches are now studied. The first is an extension of each memory cell to allow multiple accesses. The second is the provision of two buffers on the same chip of a single port memory respectively for an access request queue and for a result queue to make these three components to work in a pipeline fashion. The memory component may adopt memory interleaving technology. The result is a single port pipeline memory that provides sufficient access rate to use it as a memory with several ports. The third approach is a combination of the previous two approaches. Our concern here is how we can use a set of such multiport memory devices to implement a memory system with a larger number of ports.

We show in Figure 6.1 an 8 port memory system constructed with six memory devices with 4 ports. As in chapter 5, each vertical line represents a memory bank with four ports marked by dark circles. Each horizontal line represents an access line of a port. A dark circle at the cross of an access line of a port P and a memory bank M means that this port P accesses this memory bank M through one port of this memory. Read and write accesses are performed in a same way as described in the previous chapter.

For the same reason described in Chapter 5, each port must access an odd number of memory banks. This condition,

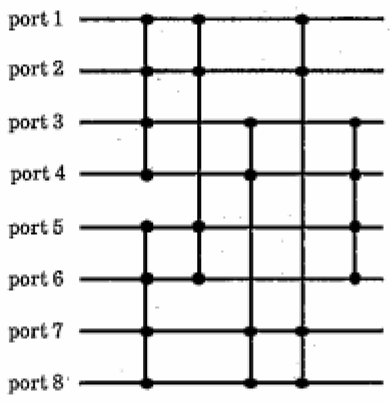


Figure 6.1 An 8-port memory using 6 4-port memories.

however, is not a sufficient condition for the configuration to work as a multiport memory system. The effect of an inversion through a port is shared by another port through those memory banks connected to both of these ports. When dual port memory banks are used, there is only one memory bank connected to a pair of ports. Here, however, there may be more than one memory bank connected to a pair of ports. In Figure 6.1 for example, each pair of ports share three memory banks. If a pair of ports share an even number of memory banks, an inversion through one of this pair does not change the parity value at the other port. Therefore, we need the following two conditions for a configuration to correctly work as a multiport memory.

(1) Each port must access an odd number of memory banks.

(2) Each pair of ports must share an odd number of memory banks.

Obviously, these two conditions also form a sufficient condition for a configuration to work as a multiport memory.

An implementation of a k-port memory system using the minimum number of memory banks with m ports is a block design problem. We have no general method to solve this problem for any k and m satisfying $k > m$. Figure 6.2 shows an

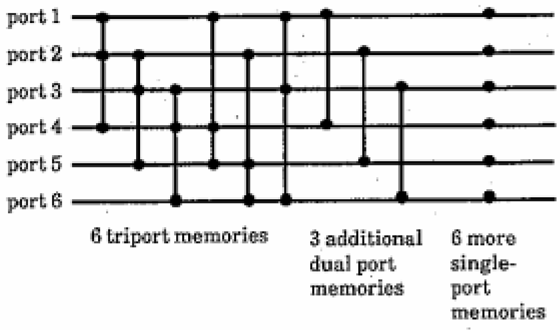


Figure 6.2 A stepwise implementation of a 6 port memory using triport memories.

example implementation of a 6 port memory system using 6 triport memory banks. It requires 3 additional dual port memory banks to satisfy the above condition (2). This addition violates the condition (1), and requires 6 additional single port memory banks to satisfy it. The result is not guaranteed to be a minimum implementation. In fact, a configuration in figure

6.3 uses 4 triport memories and 3 dual port memories to implement a 6-port memory.

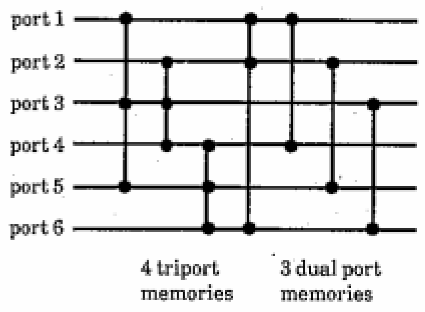


Figure 6.3 Another implementation of a 6-port memory using fewer triport and dual port memories.

Unfortunately, we have no general method, for arbitrarily given k and m, that gives an minimum implementation of a k-port memory using memories with no more than m ports. For an even m and its multiple k, however, the problem reduces to a minimum implementation of a $2k/m$-port memory with dual port memories. Its k ports are divided into $2k/m$ groups. Each group consists of $m/2$ ports. The m ports of each m-port memory are also divided into two groups, $m/2$ ports for each. Each group is considered as a single port. This reduces the original problem to a minimum implementation of a $2k/m$-port memory with dual port memories. In the solution to a reduced problem, each port accesses an odd number of memory banks, and each pair of ports share only one memory bank. Therefore, in the corresponding solution to the original problem, each port accesses an odd number of memory banks. Each pair of ports in the same group share an odd number of memory banks, while each pair of ports in different groups shares exactly one memory bank. We show an example minimum implementation of such a case in Figure 6.4. Such an

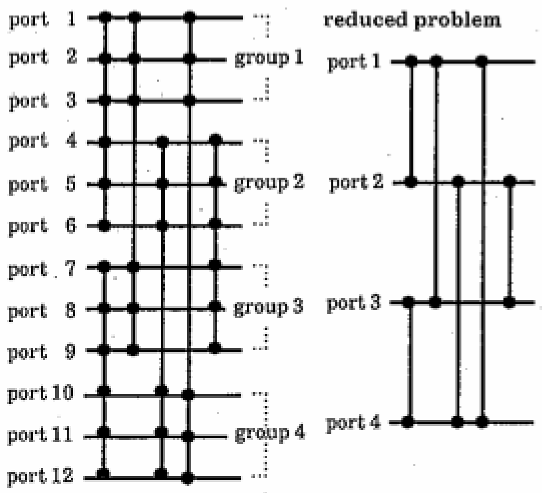


Figure 6.4 A minimum implementation of a 12-port memory using 6-port memories and its corresponding reduced problem.

implementation requires ${}_{(2k/m)}C_2$ of m-port memories, and $\log_2((2k/m)-1)$ access time.

Similarly, if m is a multiple of h and k is a multiple of m, an implementation of a k-port memory using memories with no more than m ports is reduced to an implementation of a hk/m-port memory using memories with no more than h ports.

7. Conclusion

We have given architectural solutions to implement a multiport RAM with 4~16 ports. A k-port memory uses $O(k^2)$ storage redundancy, and hence, costs $O(k^2)$ times as much as a single port memory of the same size. While its access requires $O(\log_2 k)$ time, it may be considered the same as the access time of a single memory for $k \leqq 16$. The maximum feasible k depends on the number of ports of the constituent memory devices. If we can use m port memory devices, the maximum feasible k is roughly $4m$. Therefore, an implementation of an 8-port memory is now within the scope of a consideration. In a couple of years, we may consider implementations of 16-port memories.

The most important application of multiport memories is a shared memory for multiprocessor systems with 4~16 processors. We have shown under reasonable assumptions that parallel cache architectures are not applicable to multiprocessors with more than 6 processors. The multiport cache architecture consisting of a multiport RAM as a cache and an MPPM as a main memory solves the cost problem of the multiport memory architectures, and significantly outperforms parallel cache architectures for $6 < k \leqq 16$.

This multiport cache architecture will solve the bottle neck between parallel processors and their shared memory space.

REFERENCES

[1] J. Archibald and J. L. Baer, 'Cache coherence protocols: Evaluation using a multiprocessor simulation model,' *ACM Trans. on Computers*, 4, no.4, pp.273-298, 1986.

[2] H. Boral and D. J. DeWitt, 'Database machines: An idea whose time has passed?' *Database Machines*, Springer-Verlag, pp.166-187, 1983.

[3] M. Dubois and F. A. Briggs, 'Effects of cache coherence in multiprocessor systems,' *IEEE Trans. on Computers*, C-31, no.11, pp.1083-1099, 1982.

[4] J. Goodman, 'Using cache memory to reduce processor-memory traffic,' *Proc. 10th Int'l Sympo. on Computer Architecture*, IEEE, New York, pp.124-131, 1983.

[5] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, Inc., 1985.

[6] R. Katz, S. Eggers, D. A. Wood, C. Perkins, and R. G. Sheldon, 'Implementing a cache consistency protocol,' *Proc. 12th Int'l Sympo. on Computer Architecture*, IEEE, New York, pp. 276-283, 1985.

[7] E. McCreight, *The Dragon Computer System An early overview*, Xerox Corp., Sept. 1984.

[8] M. Papamarcos and J. Patel, 'A low overhead coherence solution for multiprocessors with private cache memories,' *Proc. 11th Int'l Sympo. on Computer Architecture*, IEEE, New York, pp. 348-354, 1984.

[9] Y. Tanaka,'A multiport page-memory architecture and a multiport disk-cache system,' *New Generation Computing*, 2, no.3, pp.241-260, 1984.

[10]Y. Tanaka, 'MPDC: Massive parallel architecture for very large databases,' *Proc. Int'l Conf. on Fifth Generation Computer Systems 1984*, pp.113-137, Tokyo, Nov., 1984.