Correspondence

A Contention-Based Bus-Control Scheme for Multiprocessor Systems

Jie-Yong Juang and Benjamin W. Wah

***Abstract*—In this paper, we study contention-based bus-control schemes for scheduling processors in using a bit-parallel shared bus. The protocol is designed under the requirements that each processor exhibits a random access behavior, that there is no centralized bus control in the system, and that access must be granted in real time. The proposed scheme is based not only on splitting algorithms used in conventional contention-resolution schemes, but also utilizes two-state information obtained from collision detection. Two versions of the bus-control scheme are studied. The static one resolves contentions of N requesting processors in an average of $O(\log_{W/2}N)$ iterations, where W is the number of bits in the bit-parallel bus. An adaptive version resolves contentions in an average time that is independent of N. The proposed bus-control scheme is extended to support task-dependent priority accesses efficiently.**

***Index Terms*—Bus control, contention resolution, multiprocessor, priority, shared bus.**

I. Introduction

A shared bus provides a common communication path connecting functional units in a computer system. It is becoming increasingly popular in multiprocessors due to its low cost and simple control. Many task-dependent applications, such as resource sharing and load balancing [2], [5], [12], [21], [24], [25], are greatly simplified when a shared bus is used.

In this paper, we study the design of bus-control schemes for a shared bus that interconnects multiple processors in a multiprocessor system. In such an architecture, a bus is usually used to transmit control messages. Unlike bulk data, control messages are generally short, require a fast response, and are generated randomly. Bulk data, on the other hand, may be either transmitted via a different interconnection network or passed via a shared memory. Conventional bus-control schemes, such as daisy chaining, polling, and independent request, are inefficient in the environment under study because they were designed primarily to support a small number of processors making frequent accesses [22]. It is desired that the protocol designed uses distributed control, that its control overhead is small, and that accesses with task-dependent priorities are supported.

We describe a contention-based bus-control scheme evolved from splitting algorithms for contention resolution in CSMA/CD networks [6], [16]. Section II presents the principal operations and the architecture. Comparisons to related contention-resolution schemes are also discussed. Section III shows the adaptation of accesses with task-dependent priority to the proposed bus-control scheme. In Section IV, the performance of the proposed bus-control scheme is evaluated. The design of an adaptive version of the control scheme with load-independent average performance is described and analyzed in Section V. Lastly, concluding remarks are drawn in Section VI.

Manuscript received June 1, 1988; revised October 2, 1990. This work was supported by the National Science Foundation under Grants MIP 88-10584 and IRI 87-09072 and the National Aeronautics and Space Administration under Contract NCC 2-481.

J.-Y. Juang is with the Department of Computer and Information Engineering, National Taiwan University, Taipei, Taiwan.

B. W. Wah is with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

IEEE Log Number 9100086.

II. A Contention-Based Bus-Control Scheme

In a bus-control scheme, a processor is *active* if it has data ready to send, and is *enabled* if it is allowed to transmit to the bus. *No transmission* is observed when all enabled processors are idle, and the bus cycle is wasted during this period.

Central to the contention-based bus-control scheme are a *collision-detection mechanism* and a *transmission-control algorithm.* Each processor is equipped with a collision-detection mechanism, which monitors the bus status, detects simultaneous transmissions from multiple enabled active processors, and signals the processor to stop transmission when collision is detected. A processor runs a transmission-control algorithm to determine if it is in the enabled set.

Extensive research on transmission-control algorithms for local area networks has been carried out. These algorithms assume that each processor knows nothing about the status of other processors except the tristate status of the bus (i.e., successful transmission, idle, or a collision), which can be observed locally. For bit-parallel buses, a better collision-detection mechanism can be designed so that collisions due to overlapped transmissions may provide more useful information to the transmission-control algorithm than the tristate feedback. A collision-detection mechanism and an efficient transmission-control algorithm for bit-parallel buses are described in Sections II-A and II-C, respectively.

A. Collision-Detection Mechanism

A collision-detection mechanism can be implemented by the wired-OR property of a bit-parallel bus. When two or more numbers are transmitted simultaneously in a bit-parallel fashion to the bus from different processors, the result read is simply the bitwise logical OR of these numbers. Collision is detected when the result read is different from what was transmitted. Note that wired-OR can be applied here because functional units in a multiprocessor are located in close proximity to each other.

As an example, assume that there are three active processors, and that each of them transmits a code to the bus. Assume that the following binary codes, $X_1 = 1001, X_2 = 0101$, and $X_3 = 0100$, were transmitted in a bit-parallel fashion. The bitwise logical OR of these codes is 1101, which is different from X_1, X_2, or X_3. Thus, all these processors know that a collision has occurred by comparing this code to the one it transmitted. On the other hand, if only X_2 and X_3 were transmitted, the bitwise logical OR of these two codes is equal to X_2. In this case, the processor that transmitted X_3 detects a collision, but the one that transmitted X_2 does not. Contention is still resolved because the processor detecting a collision will refrain from further transmission.

When multiple codes are superimposed, it is possible that the resulting code is the same as one or more of the original codes. In this case, the processor(s) transmitting the code identical to the superimposed code will not be able to detect collision. Collisions of this type are called *hidden collisions.*

The probability of hidden collisions can be reduced by repeating the same contention procedure using different codes for a few times. Codes from the same code space can be used for this purpose. The probability for two processors transmitting the same code is $1/K$ when the size of the code space is K and only unary codes are used. Suppose that the contention procedure is repeated b times using different codes from the same code space, the probability that a hidden collision remains undetected can be reduced to $(1/K^{b+1})$. Since K is usually large, b can be very small for all practical purposes. Even $b = 0$ may be acceptable in most cases. Statistically, it is impossible to eliminate hidden collisions completely unless other schemes are used, such as including the station identification as part of the code to break ties. However, allowing hidden collisions to exist does not invalidate the proposed scheme, since hidden collisions can be easily captured by error-detection mechanisms when the message is actually transmitted.

B. Transmission Control: Background

A *contention period* is composed of a sequence of *contention slots.*[1] In each contention slot, processors in the enabled set broadcast codes synchronously on the bus and read the superimposed code from the bus. The contention period ends when the enabled set contains exactly one active processor. It can be reduced by properly choosing the enabled set in each contention slot.

There are two classes of algorithms for choosing the enable set. Backoff algorithms based on random delays [6], [16] are useful to determine the enabled set in a distributed fashion, but are inefficient when the number of active processors is large. Splitting algorithms [1], [3], [9], [10], [17], [18] divide the enabled set into two subsets when a collision occurs: one becomes the next enabled set, and the other will be enabled when the first subset is resolved completely. Since the history of splitting the enabled set is "remembered" and used during contention resolution, this class of algorithms can achieve very high throughput.

Demand assignment multiple access schemes (DAMA) using implicit tokens were proposed to address the bus scheduling problem [8]. In such schemes, some prior information about the network, such as the order of a station in the network and the activity on the channel, is used to determine the processor to enable. The protocol behaves like a logical-ring protocol except that no physical token is actually propagated. In a number of these schemes, dedicated control lines are also needed to simulate token propagation [7], [15]. Although using an implicit token may significantly reduce the bus scheduling overhead, it is less flexible in directly supporting high-level applications in which access priority is task dependent rather than architecture dependent. This problem is illustrated in the resolution of task-dependent priority accesses as follows.

An important objective of our proposed scheme is to support system-wide task-dependent priority accesses. To support such accesses, each packet is associated with a priority level, which reflects the exigency of the task. Channel access is granted to the station holding the packet with the highest priority. Since the priority level of a processor may change dynamically, efficient multiaccess schemes which rely on certain properties of the underlying network architecture, such as the order of processors in a ring, may not support these accesses efficiently. This inefficiency stems mainly from a lack of flexible mapping between the dynamically changing task priorities and the rigid hardware architecture. For example, in DAMA schemes based on scheduling delay, each processor delays its transmission for a period of time to avoid collisions. The delay is uniquely determined by the relative position of the processor in a logical ring. It is difficult to map task priorities into processor addresses in these schemes, since the processor holding the packet with the highest priority may be anywhere in the ring. One may argue that the delay can be determined based on the priority of the local packet in such a way that the highest priority is mapped to the shortest delay. This, however, will turn this class of DAMA schemes into contention-based schemes, since there may be more than one processor holding packets at the highest priority level. The throughput of the resulting scheme will inadvertently be degraded. This is evident in a number of studies in the literature [4], [19], [20], [23], [27]. The same argument carries over to other DAMA schemes as well as the splitting algorithm. For instance, in Capetanakis's tree algorithm, if a task-dependent index is assigned to a processor, then this index may no longer be unique, and there may be one or more processors associated with a given leaf in the tree. In this case, Capetanakis's scheme will not work properly for resolving contention in a priority class. Another contention method has to be used.

In general, task-dependent priority accesses complicates the design of multiaccess schemes. Special provisions must be made to take into account task priorities, usually with a performance penalty. In our proposed scheme, we show that, in a bit-parallel broadcast bus with the collision-detection mechanism described above, contention can be resolved efficiently using task-dependent parameters only [11], [12], [24], [26]. Consequently, every active processor can be a candidate to be enabled whenever the bus becomes free, and high-level scheduling priorities can be used to determine the enabled set. Our method for supporting task-dependent priority accesses is described in Section III. In the next section, a contention-resolution scheme for bit-parallel buses is described.

C. Proposed Transmission Control

Let X_i's be binary-coded numbers chosen from a set S with the following properties.

1) A linear ordering exists among all the elements in the set, that is,

$$\text{if } X_i, \quad X_j \in S, \quad \text{and} \quad i \neq j, \quad \text{then } X_i > X_j \text{ or } X_j > X_i. \tag{1a}$$

2) There exists a function f such that

$$f(X_i \oplus X_2 \oplus \cdots X_N) \leq \max\{X_1, X_2, \cdots, X_N\} \qquad X_i \in S, N \geq 1 \tag{1b}$$

where $\oplus$ is the bitwise logical-OR operator.

The function f is named a *code-deciphering function* in this paper. If function f is applied to the code read from the bus, a threshold can be obtained to partition the currently enabled set into two subsets. One subset consists of processors that have transmitted codes greater than or equal to the threshold, and the other includes the rest. Either one of these subsets can be the enabled set for the next contention slot. In this paper, we choose the enabled set in the next contention slot to be those processors which have transmitted codes greater than or equal to the threshold. Since at least one processor transmitted a code not less than the threshold [(1b)], such an enabled set will never be empty, and the bus will never be idle in a contention slot.

For the code-deciphering technique to work properly, a code space that satisfies (1a) and (1b) must be constructed, and a code deciphering function that can be implemented easily in hardware has to be defined. Mark [15] has proposed a scheme in which each station transmits its address synchronously bit by bit (starting from the most significant bit) over a serial control line. The controller of

[1] In traditional Ethernets, a contention slot represents the roundtrip propagation delay of the network. A contention slot here represents the delay in writing codes to the bit-parallel bus concurrently for all enabled processors, reading the superimposed result, and making a decision for subsequent contention slots.

the serial control line forms a logical OR of the corresponding bits in all addresses transmitted, and broadcasts the result to all stations. For a given bit position, stations which have transmitted a 0 will withdraw from further contention if a 1 is observed on the control line. Since each station is given a unique address, the processor with the highest address will be the only survivor after n contention slots, where n is the number of bits in an address. Note that, in this scheme, a processor with a larger address always has priority over a processor with a smaller address. Mark's scheme is equivalent to choosing codes from a binary code space and has an overhead proportional to the number of bits in the code. We show in this paper that choosing codes from a unary code space can result in better performance.

A *unary code space* of dimension n is the set $S = \{0^a\,1\,0^b | a+b=n-1, a\geq 0, b\geq 0\}$, where 0^k represents a consecutive sequence of k zeros. This set is called a unary code space since there is exactly a "1" in each code in the set. Note that there are a total of n codes in a unary code space of n dimensions.

For two different codes u and v in a unary code space S, it is easy to verify that (1a) holds if each code is considered as an ordinary binary integer. For any n-bit binary number $X = (x_1 x_2 \cdots x_n)$ a deciphering function f on X can be defined as follows.

$$f(X) = 0^p\,1\,0^{n-p-1} \qquad \text{if } x_{p+1}=1, \quad \text{and} \quad x_j = 0 \text{ for all } 1\leq j\leq p. \tag{2}$$

To verify that the deciphering function satisfies (1b) for codes chosen from the unary code space **S**, we consider N unary codes such that

$$c_i = 0^{a(i)}\,1\,0^{n-a(i)-1}, \qquad i=1,\cdots,N. \tag{3}$$

According to the ordering relation of unary codes,

$$\max(c_1, c_2, \cdots, c_N) = 0^m\,1\,0^{n-m-1} \tag{4}$$

where $m = \min\{a(i)|i=1,2,\cdots,N\}$. If these codes are transmitted to the bus simultaneously, an overlapped variable $Y = (y_1 y_2 \cdots y_n)$ will be received from the bus. The variable Y is the bitwise logical OR of the c_i's, that is,

$$(y_1 y_2 \cdots y_n) = c_1 \oplus c_2 \oplus \cdots \oplus c_N. \tag{5}$$

The Y so obtained retains the following properties:

$$y_{m+1} = 1, \quad \text{and} \quad y_k = 0 \quad \text{for } 1\leq k\leq m.$$

By definition of the deciphering function f, $f(Y) = 0^m\,1\,0^{n-m-1}$. Thus,

$$f(c_1 \oplus c_2 \oplus \cdots \oplus c_N) = \max(c_1, c_2, \cdots, c_N). \tag{6}$$

The deciphering function f defined above essentially searches for the first nonzero bit from the most significant bit of a binary number. It can be implemented in simple hardware. A simple design requires an overhead proportional to N, while a more complex one requires an overhead proportional to $\log_2 N$.

D. A Static Contention-Based Distributed Bus-Control Scheme

Based on the collision-detection mechanism and the code-deciphering function defined above, a contention-based bus-control scheme for a bit-parallel bus is summarized in this section. The scheme described here is *static* because its control is independent of the bus load and the number of active processors.

A bus with a contention-based control scheme alternates between a *contention mode,* which consists of a sequence of *contention slots,* and a *transmission mode.* In the contention mode, active processors contend in using the bus. When the winner is identified, the bus enters the transmission mode, and the winner starts transmitting its data. In other words, a transmission period is always preceded by a contention period, although the contention period may be a degenerate one. The transition between these two operation modes can be synchronized by a status control line which is set to one when the bus is in use. Whenever a processor has data ready to transmit, it senses the bus status by reading the status control line. If the bus is in use, then the processor waits until the control line is reset by the current user at the end of its transmission. Fig. 1 depicts the various periods in transmitting a message.

All the active processors are enabled at the beginning of a contention period. During a contention slot, enabled processors choose a code randomly from the code space S and transmit it to the bus. Each of them then determines whether it is still enabled by reading the resulting code from the bus, computing the deciphered code using the code-deciphering function, and comparing either the code read or the deciphered code to the code generated locally. Three outcomes may result from this comparison: 1) the locally generated code is equal to the code read; 2) the locally generated code is not equal to the code read but is greater than or equal to the deciphered code (general deciphering functions defined in (1b) are not considered here); and 3) the locally generated code is less than the deciphered code. A processor will be eliminated from further contentions in the current period if it detects the third outcome, and will remain enabled if it detects either the first or the second outcome. Using unary codes, only those processors that transmitted the maximum code remain enabled.

More than one potential winner will be identified if multiple processors have generated the same maximum code. To detect this condition, a processory may transmit another code, such as the processor identification, for verifying possible hidden collisions. Should a hidden collision be detected, the contention is not resolved. All the processors in this set repeat the resolution process in the next contention slot.

Note that contention slots for writing codes to the bus, reading codes from the bus, and deciphering the resulting codes read are interleaved with contention slots for resolving hidden collisions. A contention period ends with a contention slot that detects exactly one active processor with no hidden collision. The length of a contention slot depends on implementation factors, such as the clock rate, bus length, device technology, and time needed for deciphering a code.

The processor–bus interface for the contention-based bit-parallel bus-control scheme can be implemented easily with the current technology. It should possess the following functions: 1) sensing the bus-status, 2) reading data from the bus, 3) transmitting data to the bus simultaneously with others, 4) generating random codes, and 5) deciphering codes read from the bus. Functions 1)–3) are similar to those of a conventional bus interface. Function 5) can be implemented simply by a shift register. Function 4) can be implemented by a hardware random number generator.

III. Bus Access with Task-Dependent Priority

Accessing a shared bus with a task-dependent priority scheme is useful in applications such as priority interrupts, task scheduling, resource sharing, and load balancing. In such a priority scheduling discipline, relevant attributes associated with a transmission request are translated into a priority level. An active processor is labeled with the priority level that is the highest among its local pending requests, and the bus is allocated to the processor with the highest priority level in the system.

Before a transmission period starts, the processor(s) with the highest priority message has to be identified. Since the proposed contention-based bus-control scheme identifies the processor with

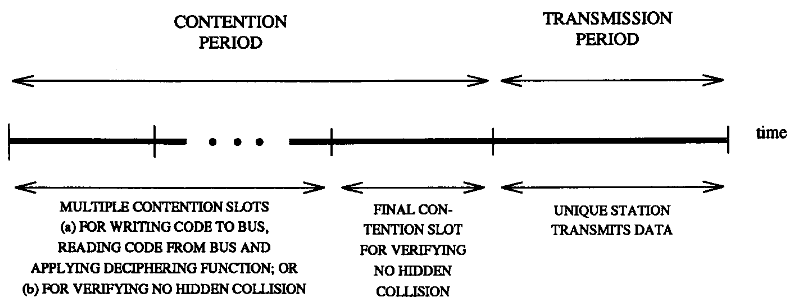
CONTENTION PERIOD
TRANSMISSION PERIOD
time
MULTIPLE CONTENTION SLOTS (a) FOR WRITING CODE TO BUS, READING CODE FROM BUS AND APPLYING DECIPHERING FUNCTION; OR (b) FOR VERIFYING NO HIDDEN COLLISION
FINAL CONTENTION SLOT FOR VERIFYING NO HIDDEN COLLISION
UNIQUE STATION TRANSMITS DATA

Fig. 1. Transmission of a message in the static contention-based distributed bus-control scheme.

the maximum code, a processor can map its priority level into a code and use this number, instead of randomly chosen code, in the course of contention. Through such a mapping, the proposed scheme can easily be modified to support accesses with priority. A simple mapping method is discussed in this section.

If the range of possible priority levels is less than the number of codes in the code space, then the code space can be partitioned into subspaces so that each subspace corresponds to a priority level. The mapping from priority levels to codes should preserve the order of priorities, that is,

$$\text{If } c_x \in \boldsymbol{S}_i, \quad c_y \in \boldsymbol{S}_j, \quad \text{and} \quad i > j, \quad \text{then } c_x > c_y, \tag{7a}$$

where $\boldsymbol{S}_i$ and $\boldsymbol{S}_j$ are subspaces associated with priority levels i and j, respectively. Such a mapping is called *code-space partitioning.* A priority level can be mapped to a randomly chosen code in the corresponding subspace. This allows contentions among the same priority class to be resolved at the same time with the identification of the highest priority level.

If the range of possible priority levels is large, then more than one priority level should be mapped to the same code. Again, the mapping should preserve the order of priorities, that is,

$$\text{If } x > y \quad \text{then } C(x) > C(y) \tag{7b}$$

where $C(x)$ and $C(y)$ represent the codes associated with priority levels x and y, respectively. Any strictly increasing function can be used for this purpose. Such a function partitions priority levels into subranges so that each subrange is associated with a code. This scheme is called *priority-space partitioning.*

Using code-space and priority-space partitioning, the processor with the highest priority message can be identified by the proposed bus-control scheme. In the beginning of a contention period, priority-space partitioning is used if the range of priority levels is large. After a contention slot, the subrange which is mapped to the maximum code is identified. A new mapping which maps this subrange into the entire code space is again defined. Since the subrange of the priority levels is reduced after each contention slot, the number of priority levels in the range eventually becomes less than the number of codes in the code space. Code-spacing partitioning is then used. The process continues until the processor with the highest priority level is identified.

In each contention period, many mappings are valid. Among them, the one that minimizes the number of contention slots should be used. The optimal mapping can be determined based on the distribution of priority levels generated by active processors [12], [14].

IV. Evaluation of the Contention-Based Bus Control Scheme

The performance of the proposed bus-control scheme is evaluated by analysis and verified by simulations. The codes used in the evaluations are assumed to be uniformly distributed. It is also easy to transform a set of independent and identically distributed random numbers in any given distribution into random numbers that are uniformly distributed.

Let N be the number of active processors at the beginning of a contention period, and K be the size of the unary code space. Note that K is less than or equal to the bus width W if all the bits of a code are transmitted to the bus in parallel. Since the number of active stations may not be known exactly, it may have to be estimated. A method for estimating N is discussed in Section V-B. In the static case, it is assumed that N is the same as the total number of (active and inactive) processors and that $K = W$.

Assume that all codes are generated randomly; and that a processor generates a given code c_i $(i = 1, 2, \cdots, N)$ with probability $1/K$. Designate the maximum among the N codes generated as $c_{\max}$, where $c_{\max} = 0^{K-m}\,1\,0^{m-1}$ and the codes generated by the other $N-1$ processors are smaller than $c_{\max}$. Contention is resolved if exactly one processor generates code $c_{\max}$. The probability for this event is

$$q(m|N,K) = N\left(\frac{1}{K}\right)\left(\frac{m-1}{K}\right)^{N-1}. \tag{8}$$

Since the maximum code generated can acquire one of the K possible values, $P_{K,N,W}$, the probability that contention is resolved in one contention slot, is

$$P_{K,N,W} = \sum_{m=1}^{K} q(m|N,K) = \frac{N}{K^N}\sum_{u=1}^{K-1} u^{N-1} \tag{9}$$

where $K = W$.

$P_{K,N,W}$ can be plotted against N/W, the normalized number of enabled processors. Such a figure reveals that the probability of success in one attempt is higher if the code space (equal to the bud width) is larger and the number of enabled processors is kept constant (that is, N/W is smaller). It also shows that when W is fixed, $P_{K,N,W}$ is a strictly decreasing function of N and decreases to zero when N is large. This means that the proposed bus-control scheme is unable to resolve contention in one slot (not counting the extra contention slots to reduce the probability of hidden collision) when the traffic is heavy. However, many enabled processors will be eliminated in each contention slot. The probability of success is increased as contention proceeds because the number of survivors is reduced.

It can be shown that, on the average, the fraction of contending processors that survive a contention is less than $2/K$. The exact analysis is not shown here due to space limitation.

Let $N_t, t = 1, 2, \cdots$, be the average number of processors that participate in the t-th contention slot during a given contention period.

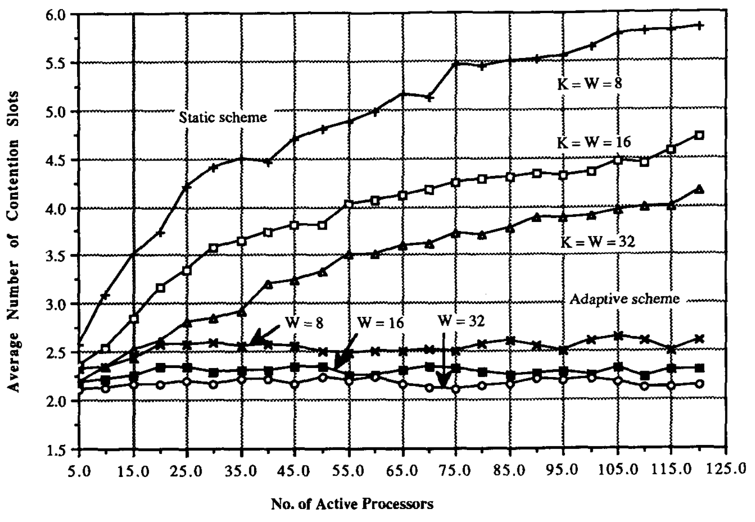

Fig. 2. Average number of contention slots for resolving contention of N processors for both static and adaptive contention-based bus-control scheme. (Each point shown in the curves using adaptive control is the minimum among a wide range of code-expansion factors simulated.)

Then

$$N_t \le \left(\frac{2}{K}\right)^t N \qquad t = 1, 2, \cdots \tag{10}$$

Contention is resolved when $N_t = 1$. t_s, the number of contention slots for resolving contentions of N processors, satisfies the following inequality.

$$1 \le \left(\frac{2}{K}\right)^{t_s} N. \tag{11}$$

This inequality yields

$$t_s \le \log_{K/2} N. \tag{12}$$

Therefore, the average number of contention slots for resolving the contention of N processors is bounded by $\log_{K/2} N$, where $K = W$. The above bound does not include the extra contention slots for reducing the probability of hidden collisions.

Simulations have been conducted to evaluate the performance of the proposed scheme. The bit-parallel bus and its wired-OR property relevant to contention resolution are simulated by a procedure that examines every bit of the codes generated. The results obtained are plotted in Fig. 2.

Analysis and simulation results both indicate that the proposed scheme performs better when the bus width is large. In this case, the corresponding code space is large and the probability of collisions is small.

The code-deciphering function in (2) also applies to a binary code space. Nevertheless, contentions using binary codes may result in a longer contention period because the contention-resolution scheme can only detect the most significant bit that is mismatched among the codes generated by the contending processors. As a result, only half of the stations on the average are eliminated in each contention slot. With a similar analysis, it can be shown that the average number of contention slots for resolving the contention of N processors is increased to $\log_2 N$.

V. An Adaptive Contention-Based Distributed Bus-Control Scheme

Since the number of contention slots grows logarithmically with the total number of processors, the scheme with a fixed code space is inefficient when either the number of processors is large or the bus width is small. A possible solution to this problem is to choose a sufficiently large code space. The difficulty of having a large code space in the static bus-control scheme lies in the constraint $K \le W$; that is, the number of bits of a code should be less than or equal to that of the bus. This constraint can be removed by introducing a *virtual code space* in which the number of bits in a code can be greater than the number of bits in the bus. When a code is transmitted, only its most significant bits are transmitted. In other words, if K is greater than W, then only the first W bits of a code are transmitted. However, if K is not greater than W, then the entire code is transmitted, and K can be set to W to obtain a higher success probability of resolving collisions.

With a virtual code space, a code space of an arbitrary size can be used. To maximize the probability that exactly one processor generates the maximum code, the size of the code space should vary with the number of active processors. A large code space should be used for a large number of active processors. Code space can be updated at the beginning of each contention period. This approach results in an *adaptive scheme* in the sense that its control depends on the bus load.

For the adaptive scheme to work properly, the number of active processors has to be known. This number is not readily available and has to be estimated. An efficient method for estimating it is discussed in Section V-C.

A. Performance of the Adaptive Bus-Control Scheme

To determine the proper size of the code space, the performance of the adaptive bus-control scheme is first evaluated by assuming that N can be estimated accurately. In Sections V-B and V-C, the effects of errors in estimating N are studied.

Let r, the *code-expansion factor,* be defined as the ratio of K and N. The probability that contention is resolved in one slot [refer to (8) and (9)] (not including the slots for verifying the absence of hidden collision), given that $c_{\max}\left(= 0^{K-m}\,1\,0^{m-1}\right)$, is

$$P_{K,N,W} = \sum_{m=K-W+1}^{K} q(m \mid N, K)$$

$$= \sum_{m=K-W+1}^{K} N\left(\frac{1}{K}\right)\left(\frac{m-1}{K}\right)^{N-1}$$
$$= \frac{N}{(rN)^N} \sum_{u=K-W}^{K-1} u^{N-1}. \qquad (13)$$

Note that this equation only sums the last W terms of $q(m|N,K)$, while all possible terms associated with m are summed in (9). A plot of $P_{K,N,W}$ with respect to different values of r and W shows that the success probability of an adaptive scheme is substantially higher than that of the static one. When N is large, the success probability of the static scheme approaches zero, while that of the adaptive scheme remains high. The following theorem shows that this success probability is asymptotically bounded.

Theorem:

$$P_{K,N,W} \geq \alpha(r,W) = \lim_{N\to\infty} P_{K,N,W} = \frac{1-e^{-W/t}}{r(e^{1/t}-1)}. \qquad (14)$$

The proof of the theorem is straightforward and is omitted due to space limitation.

Note that the lower bound of $P_{K,N,W}$ depends on the bus width W and the code-expansion factor r. For example, it is equal to 0.924 when $W = 32$ and $r = 8$, and equal to 0.762 when $W = 8$ and $r = 2$. When r is chosen properly, $P_{K,N,W}$ is close to its lower bound for most values of N. Therefore, the lower bound provides a good approximation for the success probability. With this approximation, the probability distribution of the contention being resolved in the ith contention slot (not counting the final contention slot to verify that no hidden collision is present) can be approximated as a geometric distribution, or

$$p(i) = \alpha(r,W)(\alpha(r,W))^{i-1} \qquad i = 1,2,3,\cdots \qquad (15)$$

when $\alpha(r,W)$ is the lower bound of $P_{K,N,W}$. C, the approximate average number of contention slots, is then equal to the mean of this distribution, that is,

$$C = \sum_{i=1}^{\infty} i \times p(i) = \frac{1}{\alpha(r,W)}. \qquad (16)$$

Since $\alpha(r,W)$ is independent of N, so is the average number of contention slots. The independence of N is illustrated by the simulation results depicted in Fig. 2.

B. Estimation of N

The adaptive bus-control scheme requires N, the number of active stations, to be known. Although N cannot be found exactly without a lot of overhead, it can be estimated from the outcomes of contention observed during the contention process in the proposed bus-control scheme.

Recall that the bus alternates between the contention and the transmission modes. Suppose a code space of size K_{i-1} was used in the $(i-1)$th contention period, and it was observed in that period that no transmission took place in the first $(t-1)$ contention slots, and that there were B bits set to one in the result read in the tth slot. Assume that there are N contending processors in the $(i-1)$th contention period. The codes chosen by these N processors spread over a space of K_{i-1} codes. These K_{i-1} codes are mapped to K_{i-1} bits of which W bits are transmitted in each contention slot, starting with the most significant bits. According to the above observation, B bits in $t \times W$ bits were found set to one. Thus, a rough estimate of B/N can be approximated by $t \times W/K_{i-1}$, and $\hat{N}_i$, an estimate for N is the beginning of the ith contention period, can be obtained as follows.

$$\hat{N}_i = \frac{B \times K_{i-1}}{t-W} - 1 + \nu \qquad (17)$$

where ν is the average number of idle processors that became active since the $(i-1)$th contention period. A one is subtracted in the estimate because the processor that transmitted in the $(i-1)$th period is assumed to become idle. To estimate ν, the arrival rate of bus requests has to be estimated. In steady state, the arrival rate is approximately equal to the number of messages transmitted per time unit. The estimate in (17) will systematically underestimate the actual value of N since there may be more than one processor that generates the same code. Therefore, a slightly larger value of r, the code-expansion factor, may have to be used to compensate the error. Note that the estimation method does not take priority into account. It has to be modified properly to use in task-dependent applications.

According to simulations, the number of contention slots required (including the final contention slot to verify the absence of hidden collision) is increased by about 5% when N is estimated (see Fig. 3). This degradation in performance can be reduced by a moving average estimate of N [12], [24]. A moving average is obtained by a weighted sum of several recent estimations. Alternatively, an estimate that couples the observation from the bus and the distribution of N can be derived [13]. Such a scheme is more complex, while the performance improvement in this particular application is limited.

C. Choice of the Code-Expansion Factor

Since the success probability is an increasing function of W, better performance can be expected for larger W. However, W is usually determined by other requirements and should be treated as a fixed parameter in the bus-control scheme. The code-expansion factor, on the other hand, provides a leverage for tuning the performance of the adaptive scheme. There is an optimal choice of the code-expansion factor to minimize the average number of contention slots. For a small value of r, the size of code space is small, and the probability of collisions is increased. For a large value of r, the probability of no code with a one in the first W significant bits being generated is high, which will result in no transmission in the first contention slot. Optimization based on Markovian decision processes or dynamic programming can be formulated, but the algorithms are too complex to be practical. Two heuristic methods are proposed below.

A good heuristic is to choose r that maximizes $P_{K,N,W}$, the success probability. Since $P_{K,N,W}$ depends on N, the r so determined also depends on N. This requires r to be updated whenever N changes and may complicate the implementation of the scheme. As indicated in (16), the lower bound of success probability is crucial to the performance of the adaptive bus-control scheme. Maximizing the lower bound would have a significant effect on the number of contention slots required, which can be achieved by maximizing (14) with respect to r. Though no closed-form solution was found for this optimization, numerical evaluations of the expression provide a good approximation. These evaluations allow us to choose the best r for a given W. For example, when W is 16, the best choice of r is 4.7. Note that although there is an optimal choice of r, the lower bound of success probability does not change significantly for a wide range of r.

A second alternative is to conduct simulations to determine the r that leads to the best performance. Since closed-form mathematical expressions for the proper choice of r cannot be obtained, simulation is a practical method here. A simple and efficient simulator can be developed to reflect the details of the bus-control scheme due to the

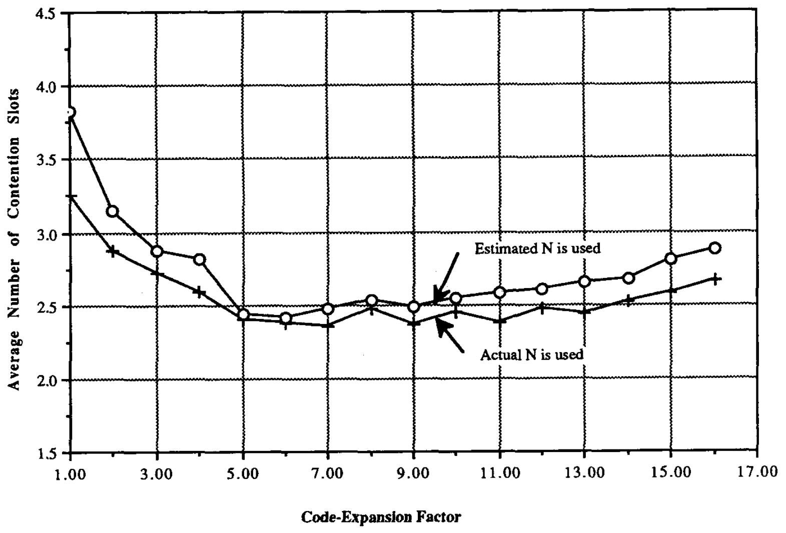

Fig. 3. Degradation in performance due to the estimation of N in the adaptive bus-control scheme ($N = 50, W = 16$).

simple control required. The simulator we developed simulates the collision-detection mechanism by scanning all the codes generated by the contending processors in each contention slot. The transmission-control and the load-estimation procedures were coded exactly.

Table I shows the simulation results. The values shown include the final contention slot for verifying the absence of hidden collisions. The best choice of r is around 6 for $W = 16$, which is slightly larger than what we obtained from the maximization of the lower bound of success probability. The difference here is due to the underestimation of N using (17). A larger r will compensate this underestimation to obtain the same size of code space. Simulation results also indicate that the average number of contention slots does not change substantially for a wide range of r, which indicates that the adaptive bus-control scheme is robust and that a complex model for optimizing the code-expansion factor r may not be necessary. It also suggests that the code space does not have to be updated unless the number of active processors changes dramatically.

Using a proper value of r, the performance of the adaptive bus-control scheme was evaluated by simulations. The results obtained are plotted in Fig. 2. For a fair comparison to the static scheme, N was assumed known.

TABLE I
SIMULATION RESULTS SHOWING AVERAGE NUMBER OF SLOTS FOR RESOLVING CONTENTIONS OF N PROCESSORS USING AN ADAPTIVE BUS-CONTROL SCHEME WITH CODE-EXPANSION FACTOR r (N IS ESTIMATED IN OUR CONTROL SCHEME)

Bus Width W = 8 Bits										
N r	1	2	3	4	5	6	7	8	9	10
10	3.09	2.79	**2.50**	2.55	2.63	2.70	2.89	3.00	3.14	3.33
20	3.74	2.90	2.61	**2.60**	2.66	2.77	2.86	2.98	3.18	3.26
30	4.41	3.07	2.85	**2.73**	**2.73**	2.77	2.97	3.09	3.4	3.13
40	4.46	3.02	**2.67**	2.73	2.70	2.70	2.80	2.93	3.14	3.43
50	4.58	3.08	2.82	2.76	2.74	**2.69**	2.87	2.94	3.24	3.49
60	4.78	3.43	2.92	2.85	**2.73**	2.74	2.85	2.98	3.22	3.45
Bus Width W = 16 Bits										
N r	1	3	5	6	7	8	10	12	14	16
10	2.55	2.43	2.35	**2.30**	2.39	2.47	2.53	2.50	2.76	2.93
20	3.16	2.51	2.40	**2.39**	**2.39**	2.46	2.51	2.65	2.76	2.91
30	3.57	2.59	2.49	**2.42**	2.46	**2.42**	2.62	2.64	2.78	2.97
40	3.85	2.83	**2.42**	2.63	2.53	2.43	2.51	2.60	2.68	2.82
50	3.82	2.88	2.44	**2.41**	2.47	2.53	2.54	2.60	2.67	2.87
60	4.07	2.72	2.64	2.45	**2.43**	2.51	2.59	2.61	2.73	2.91
Bus Width W = 32 Bits										
N r	1	7	8	9	10	11	12	16	24	32
20	2.61	2.35	**2.25**	2.30	**2.25**	2.31	2.31	2.32	2.59	2.85
40	3.19	2.39	2.52	**2.30**	2.36	2.36	2.31	2.34	2.55	2.81
60	3.49	2.30	2.31	2.39	**2.29**	2.30	**2.29**	2.35	2.47	2.88
80	3.71	2.46	2.36	2.36	2.31	**2.28**	**2.28**	2.35	2.48	3.12
100	3.92	2.43	2.43	2.45	2.38	2.35	**2.33**	**2.33**	2.54	3.01
120	3.94	2.48	2.41	2.38	2.35	2.30	2.29	**2.28**	2.64	2.90

VI. CONCLUSION

In this paper, we have studied a contention-based bus-control scheme for a shared bit-parallel broadcast bus that connects a large number of processors in a multiprocessor system. Two versions of the contention-based bus-control scheme are studied. The static version resolves contentions of N processors in an average of $O(\log_{W/2} N)$ contention slots, where W is the number of bits in the bus. When the bus traffic is light, the static version is simple and effective. The adaptive version estimates N and uses a code space of variable size depending on N. We show that contentions can be resolved in a time that is load-independent on the average. When the code-expansion factor is properly chosen, an adaptive version can resolve contention in two contention slots most of the time. A simple yet effective method for estimating N is proposed.

The proposed bus-control scheme belongs to a broad class of reservation schemes using splitting algorithms. It achieves high performance by taking advantage of the multiple-bit two-state feedback in a bit-parallel broadcast bus and the properties of unary codes. Without constraining on architecture-dependent parameters, it can efficiently support task-dependent priority accesses, a desirable feature that is difficult to implement in conventional bus-control schemes with distributed control.

REFERENCES

[1] D. Bersekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[2] S. H. Bokhari, "Finding maximum on an array processor with a global bus," *IEEE Trans. Comput.*, vol. C-33, pp. 133–139, Feb. 1984.

[3] J. I. Capetanakis, "Tree algorithms for packet broadcast channels," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 505–515, Sept. 1979.

[4] I. Chlamtac and W. R. Franta, "Message-based priority access to local networks," *Comput. Commun.*, vol. 3, no. 2, pp. 77–84, Apr. 1980.

[5] R. Dechter and L. Kleinrock, "Broadcast Communications and Distributed Algorithms," *IEEE Trans. Comput.*, vol. C-35, pp. 210–219, Mar. 1986.

[6] Institute of Electrical and Electronics Engineers, "Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Local Area Network," ANSI/IEEE Std. 802.3, 1985.

[7] K. P. Eswaran, V. C. Hamacher, and G. S. Shedler, "Collision-free access control for computer communication bus networks," *IEEE Trans. Software Eng.*, vol. SE-7, Nov. 1981.

[8] M. Fine and F. A. Tobagi, "Demand assignment multiple access schemes in broadcast bus local area networks," *IEEE Trans. Comput.*, vol. C-33, pp. 1130–1159, Dec. 1984.

[9] J. F. Hayes, *Modeling and Analysis of Computer Communication Networks*. New York: Plenum, 1984.

[10] M. G. Hluchyj, "Multiple access communication: The finite user population problem," Tech. Rep. LIDS-TH-1162, Massachusetts Instit. of Technol., Cambridge, MA, Nov. 1981.

[11] J. Y. Juang and B. W. Wah, "A multiaccess bus-arbitration scheme for VLSI-densed distributed systems", in *Proc. Nat. Comput. Conf.*, July 1984, pp. 13–22, AFIPS Press.

[12] J. Y. Juang, "Resource allocation in computer networks," Ph.D dissertation, School of Elec. Eng., Purdue Univ., West Lafayette, IN, Aug. 1985.

[13] J. Y. Juang and C. C. Lee, "A maximum a priori estimation of channel laod," in *Proc. Princeton Conf. Inform. Sci. Syst.*, Mar. 1986.

[14] J. Y. Juang and B. W. Wah, "A unified minimum-search method for resolving contentions in multiaccess networks with ternary feedback," *Inform. Sci.*, vol. 48, no. 3, pp. 253–287, 1989.

[15] J. W. Mark, "Distributed scheduling conflict-free multiple access for local area communications networks," *IEEE Trans. Commun.*, vol. COM-28, pp. 1968–1976, Dec. 1980.

[16] R. Metcalfe and D. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, no. 7, pp. 395–404, 1976.

[17] K. Mittal and A. Venetsanopoulos, "On the dynamic control of the urn scheme for multiple access broadcast communication systems," *IEEE Trans. Commun.*, vol. COM-29, pp. 962–970, July 1981.

[18] J. Moseley and P. Humblet, "A class of efficient contention resolution algorithms for multiple access channels," *IEEE Trans. Commun.*, vol. C-35, pp. 145–157, Feb. 1985.

[19] L. M. Ni and X. Li, "A prioritized packet transmission in local multiaccess networks," in *Proc. 8th Data Commun. Symp.*, Oct. 1983, pp. 234–244.

[20] R. Rom and F. A. Tobagi, "Message-based priority function in local multiaccess communication systems," *Comput. Networks*, pp. 273–286, 1981.

[21] Q. F. Stout, "Mesh connected computers with broadcasting," *IEEE Trans. Comput.*, vol. C-32, pp. 826–830, Sept. 1983.

[22] K. J. Thurber *et al.*, "A systematic approach to the design of digital bussing structures," in *Proc. AFIPS Conf.*, vol. 41, AFIPS Press, 1972, pp. 719–740.

[23] F. A. Tobagi, "Carrier sense multiple access with message-based priority functions," *IEEE Trans. Commun.*, vol. COM-30, no. 1, pp. 185–200, Jan. 1982.

[24] B. W. Wah and J. Y. Juang, "Resource scheduling for local computer systems with a multiaccess network," *IEEE Trans. Comput.*, vol. C-34, pp. 1144–1157, Dec. 1985.

[25] B. W. Wah and Y. N. Lien, "Design of distributed databases on local computer systems with a multiaccess network," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 606–619, July 1985.

[26] B. W. Wah and J. Y. Juang, "An efficient contention resolution protocol for local multiaccess networks," Patent 4630264, Filed Sept. 21, 1984, Granted Dec. 16, 1986.

[27] T. Znati and L. M. Ni, "A prioritized multiaccess protocol for distributed real-time applications," in *Proc. COMPSAC*, 1987, pp. 324–331.

VLSI Architectures for Multidimensional Transforms

Chaitali Chakrabarti and Joseph JáJá

Abstract—**In this paper we propose a family of VLSI architectures with area–time tradeoffs for computing $(N \times N \times \cdots \times N)$ d-dimensional linear separable transforms. For fixed-precision arithmetic with b bits, the architectures have an area $A = O(N^{d+2a})$, computation time $T = O(d\,N^{d/2-a}\,b)$ and achieve the AT^2 bound of $AT^2 = O(n^2b^2)$ for constant d, where $n = N^d$ and $0 < a \leq \frac{d}{2}$.**

Index Terms— **Area–time tradeoffs, AT^2 optimal, multidimensional transforms, rotator unit, subblock rotator unit, subblock transpose unit, VLSI architecture.**

I. Introduction

Multidimensional transforms are a powerful tool for analyzing multidimensional signals. The 2-D discrete Fourier transform (DFT) is widely used in spectrum analysis, speech processing, and image processing. The 3-D and 4-D DFT's are used to represent dynamic patterns in computer vision and pattern analysis. The number of computations involved in these transforms is very large, and consequently optimal architectures with efficient computational schemes have to be developed.

In this paper, we propose a family of regular and simple architectures for computing $(N \times N \times \cdots \times N)$-point d-dimensional linear separable transforms with area–time tradeoffs. Our architecture consists of one-dimensional $DXT(N)$ computation units which compute $DXT(N)$ over one index, and permutation units which order data so that in the next iteration, $DXT(N)$ can be computed over the next index. The architecture has an area $A = O(N^{d+2a})$ and computation time $T = O(dN^{d/2-a}b)$ for all a in the range $\frac{1}{2}\log_N b \leq a \leq \frac{d}{2}$, where $b = O(\log N)$ is the precision. Thus, there exists a family of architectures with $AT^2 = O(d^2n^2b^2)$ for different values of a, $n = N^d$. For the case when d is a small constant, our architectures are optimal. To the best of the author's knowledge, $d \leq 4$ for almost all known applications requiring multidimensional transforms.

The rest of the paper is organized as follows. In Section II, we state the definitions and assumptions for the computation of multidimensional linear transforms. We briefly review the existing architectures in Section III. In Section IV, we first describe an architecture when the input is in a single file and then develop a family of architectures with area–time tradeoffs.

II. Preliminaries

Let n be the total number of data elements that are to be organized in a d-dimensional (d-D) data cube. Then each element can be represented by d indexes $n_1, n_2, \cdots, n_d$. Any d-D linear separable transform is defined by

Manuscript received July 29, 1988; revised October 24, 1990. This work was supported in part by NSA Contract MDA-904-85H-0015, NSF Grant DCR-86-00378, and by the Systems Research Center under Contract OIR-85-00108.

C. Chakrabarti is with the Department of Electrical Engineering, Center for Solid State Electronics Research, Center for Systems Science and Engineering, Arizona State University, Tempe, AZ 85287.

J. JáJá is with the Department of Electrical Engineering, Institute for Advanced Computer Studies, Systems Research Center, University of Maryland, College Park, MD 20742.

IEEE Log Number 9100085.