8-1-1989

Performance Analysis of Hardware Barrier Synchronization

M. O'Keefe
Purdue University

H. Dietz
Purdue University

Performance Analysis Of Hardware Barrier Synchronization

M. O'Keefe
H. Dietz

TR-EE 89-51
August, 1989

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Performance Analysis Of Hardware Barrier Synchronization

M. O'Keefe
H. Dietz

TR-EE 89-51
August, 1989

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Performance Analysis of Hardware Barrier Synchronization

M. O'Keefe and H. Dietz

School of Electrical Engineering
Purdue University
West Lafayette, IN 47907
August 1989

ABSTRACT

Synchronization among cooperating processors is a critical issue in the performance of high speed multiprocessors. For current Multiple Instruction stream Multiple Data stream (MIMD) computers synchronization cost is high. Hence, these architectures can execute only large granularity parallelism efficiently. In this report we study a new hardware synchronization technique, known as a *hardware barrier*. Machines using this technique are known as *barrier MIMDs*. Analytic and simulation studies are employed to show that hardware barrier synchronization can outperform the more common *directed* synchronization techniques. Barrier synchronization can be viewed as a static synchronization mechanism similiar to the implicit synchronization of Very Long Instruction Word architectures (VLIWs). We study two variations of hardware barrier synchronization previously developed, static and dynamic, and suggest a new hybrid approach.

1. Introduction

A barrier is a synchronization point. A processor typically performs the following three steps at a barrier:

[1] Marks itself as present at the barrier.
[2] Waits for all other participating processors to arrive at the barrier.
[3] Proceeds along with the other participating processors past the barrier.

Barrier synchronization has commonly been considered a software technique to provide sequence control, insuring that events happen in the proper order. For example, a `DOALL` loop (all iterations may execute in parallel) requires that all iterations synchronize when completed: execution proceeds past the `DOALL` loop only after all iterations are complete. This can be implemented using `fork` and `join`, but the overhead of spawning and killing processes is high using this approach. Counting semaphores can also be used, although the serialization caused by the mutual exclusion necessary for the semaphores also wastes additional cycles [Axe86],[HeF88]. In addition, the processors are only "approximately" synchronized when using other primitives to implement barriers.

Barrier MIMDs are asynchronous Multiple Instruction stream Multiple Data stream architectures that employ a fast hardware synchronization mechanism known as a *hardware barrier.* They can execute loops, subprograms, and variable-execution time code in parallel like any MIMD architecture. In addition, the hardware barrier synchronization mechanism is fast and efficient, reducing the execution-time cost of synchronization between processes. In this respect, barrier MIMD architectures are similar to Single Instruction stream Multiple Data stream (SIMD) architectures; the execution-time cost of synchronization is essentially zero, and synchronization is implemented statically at compile-time.

The similarities between barrier MIMDs and statically scheduled architectures such as SIMD and VLIW are explored in Dietz and Schwederski [DiS87]. A contiguous spectrum of properties between SIMD and MIMD are also given. We will briefly summarize these properties. The number of *simultaneous operations* specifies how many different operations may be performed on a machine with N processors; the larger this number, the more varied the parallelism that may be executed on a machine. The number of *control flow threads* is the number of independent program counters in a machine of width N. The *relative time synchronization error* specifies the time error known to the compiler between two instructions executing on different processors. In SIMD and VLIW execution, this error is very small, enabling static scheduling of instructions without synchronization overhead. A barrier MIMD also has this property, and it can be instruction scheduled with good efficiency. A barrier can always be inserted to reduce the relative timing error among processors to zero.

An important feature which distinguishes two forms of barrier MIMD is the number of *synchronization control flow threads*, which specifies how many synchronization operations are candidates for current execution. This is the key difference between Static Barrier MIMD (SBM) and Dynamic Barrier MIMD (DBM). SBM barrier execution is characterized by a total ordering imposed on the execution of the barriers that in general will not correspond to the actual ordering that occurs during program execution. Hence, barriers ready to execute may have to wait for other barriers. The hardware mechanism used in Static Barrier MIMD is a queue; barriers are loaded into the queue according to the total ordering of barriers determined at compile-time. Dynamic Barrier MIMD allows the execution of a barrier immediately after the processors associated with the barrier reach it. This requires an associative search of the memory containing the current barriers, and a complex loading mechanism for the associative memory. Hence, the additional performance of the DBM appears to come at significantly higher cost. These issues are considered in more detail in Dietz [DiS87]. This report provides discusses techniques to reduce the delays introduced by Static Barrier MIMD. Both analytic and simulation studies are employed to determine the utility of these techniques.

The last feature considered is whether the synchronization primitives are *directed* or *undirected*[1]. Directed synchronization is more efficient in that it allows one processor to continue execution after it performs the synchronization action; the other processor(s) must wait for the action. Thus, if A is to wait for B and B arrives before A does, B is allowed to continue immediately. Undirected synchronization, such as the hardware barrier discussed in this report, causes all involved processes to wait. Although it is clearly less efficient than directed synchronization, this report will show that under a variety of assumptions the difference is not significant.

The key point made in Dietz et al. [DiS87] concerning barrier machines is that they can synchronize all processors at the clock-cycle level, and this information can be used to satisfy conceptual synchronizations through static code scheduling. In this view, barriers are viewed as a mechanism to reduce the relative-time synchronization error between processes rather than purely as a synchronization technique.

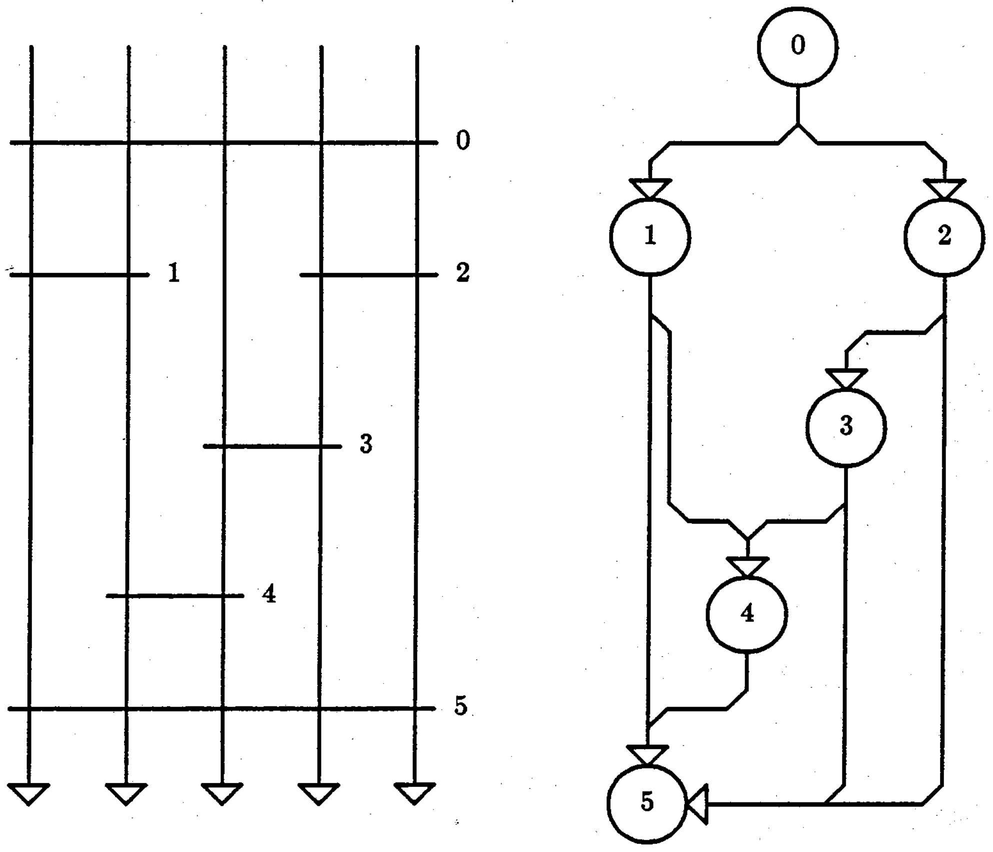
This report examines the performance of the two barrier MIMD architectures proposed in [DiS87]. Both analytic and simulation results are discussed. A barrier simulator has been programmed and executed to gather data on the performance of the barrier architectures compared to directed MIMD machines. It will be shown that the barrier architectures compare favorably to directed MIMD even under the worst case assumptions for the barrier machine performance. This is true independent of the additional

1. A directed synchronization is an operation whereby one processor is forced to wait for some action of another processor, but the processor performing the action need not wait upon performing the action.

benefits that can be realized with the barrier architectures through the appropriate compiler technology. In addition, we will show that the additional delays introduced in SBM execution due to the total ordering of barriers can be reduced through the appropriate static scheduling techniques and a hybrid approach that combines features of the static and dynamic barrier models.

2. Analytic Models

In this report, a barrier configuration will be represented as in figure 1a. The vertical lines represent concurrently executing processors while the horizontal lines represent barriers across the processors they intersect. The time axis is vertical, with time increasing in the downward direction. A barrier configuration may also be represented as a directed acyclic graph (dag), with the graph nodes representing barriers and edges representing the ordering constraints among the barriers. The dag describes the partial ordering among the barriers, which are a partially ordered set (poset). A dag for the barrier configuration shown in figure 1b.



Figures 1a & 1b: Sample Barrier Configuration & DAG

We will consider a simple example to compare the efficiency of barrier and directed synchronization. Observe the barrier configuration in figure 2. The code executed in a given processor between two barrier synchronizations is referred to as a "region" , and is consistent with the definition of a region found in Dietz [Die87].

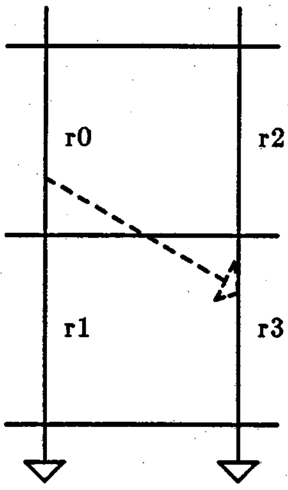


Figure 2: Barrier Implementing Directed Synchronization

Assume the four regions *r0* through *r3* have execution times which are independent and identically distributed. Regions *r0* and *r3* are the producer and consumer of a synchronization, respectively. Assuming that the time to execute the barrier and directed synchronization operations is equal to zero, it is clear that the directed synchronization version should be faster. But how significant is the difference? In the following paragraphs we examine this question.

Let $F_r(r)$ represent the distribution function of the random variable r , the time to execute a region. Let $f_r(r)$ represent the corresponding density function. Also, b_i is a random variable corresponding to the execution time of barrier i . Then the distribution function for b_i is given by

$$F_{b_i}(z) \;=\; F_r^2(z) \tag{1}$$

yielding the density function

$$f_{b_i}(z) \;=\; 2f_r(z)F_r(z) \tag{2}$$

As an example, if $f_r(r)$ is a uniform distribution with a range from 0 to m, we have that the expected value (first moment) of b_1 is $E[b_1] = \frac{2}{3}m$ and, given the linearity of expected values, $E[b_2] = 2E[b_1] = \frac{4}{3}m$.

Let d_2 be the r.v. representing the execution time of barrier two, given that barrier one is implemented using a directed synchronization. Let $p_0 = r_0 + r_1$ and $p_1 = r_2 + r_3$, hence $d_2 = \max(p_0 , p_1)$. Processor zero may start executing region one as soon as it finishes region zero. Clearly, the execution time using directed synchronization will differ from that using barrier synchronization only when $r_0 < r_2$ and $r_1 > r_3$. Given that the r_i are independent and identically distributed (i.i.d.), this occurs with probability 0.25. For this case, we can express the directed sync execution time as

$$d_2 = \max(p_0 , p_1) \tag{3}$$

and then

$$f_{r_0+r_1}(z) = \int f_{r_0}(z-y)f_{r_1}(y)dy \tag{4}$$

Assuming a uniform distribution with range 0 to *m*, it can be shown that

$$E[r_0+r_1] = E[r_2+r_3] = m$$

as expected. The densities for r_0 and r_1 in (4) are actually conditional densities on the event $r_0 < r_2$ and $r_1 > r_3$. The conditional distribution for r_0 is given by

$$\begin{aligned} F_{r_0}(z \mid r_0 < r_2) &= \frac{P[(r_0 \leq z) \cap (r_0 - r_2 \leq 0)]}{P[r_0 < r_2]} \\ &= 2P[(r_0 \leq z) \cap (r_0 - r_2) \leq 0)] \\ &= 2P[(r_0 \leq z)] \cap (r_4 \leq 0)] \end{aligned}$$

where $r_4 = r_0 - r_2$. Recall that for joint distributions

$$F_{xy}(x,y) = P[X \leq x, Y \leq y]$$

hence

$$F_{r_0}(z \mid r_0 < r_2) = 2F_{r_0 r_4}(z , 0)$$

Since r_0 and r_4 are not independent, the joint distribution is not directly available. Instead, we will approximate the conditional density with $f_{r_0}(z)$. To determine $E[d_2]$, we use (2) and again assuming a uniform distribution, we have

$$f_{d_2}(z) = \begin{cases} \dfrac{z^3}{m^4} & 0 \leq z \leq m \\ \dfrac{2}{m^2}(\dfrac{z^3}{2m^2} - \dfrac{3z^2}{m} + 5z - 2m) & m \leq z \leq 2m \end{cases}$$

Then,

$$E[d_2] = \int_{-\infty}^{+\infty} z f_{d_2}(z)dz = \frac{37}{30}m \simeq 1.23m$$

Recalling the theorem of total probability

$$f(x) = f(x \mid A_1)P(A_1) + \cdots + f(x \mid A_n)P(A_n) \tag{4}$$

where the events $A_1, A_2, \cdots, A_n$ form a partition of the event space, and $f(x \mid A_i)$ represents the conditional density of x given event A_i, and $P(A_i)$ represents the probability of event A_i. From (4) it follows that

$$E[x] = E[x \mid A_1]\, P(A_1) + E[x \mid A_2]\, P(A_2) + \cdots + E[x \mid A_n]\, P(A_n) \tag{5}$$

and hence $E[d_2] = (\frac{37}{30}m)(0.25) + (\frac{4}{3}m)(0.75) \simeq 1.3083m$ compared to $E[b_2] = \frac{4}{3}m \simeq 1.3333$. Thus, directed synchronization is only 2.7% faster than barrier synchronization, under the given assumptions. Using the simulator, we obtained statistics for the expected values of the barrier and directed synchronization execution times for uniform distributions and found the difference to be less than 4%. Clearly, this difference is not large.

Let us consider the same question given an exponential distribution for the regions $f_{r_i}(r_i) = \lambda e^{-\lambda z}$ where the mean $m = \frac{1}{\lambda}$. From (4) we know that

$$f_{b_1}(z) = 2f_{r_0}(z)F_{r_0}(z) = 2\lambda e^{-\lambda z}(1 - e^{-\lambda x})$$

The expected value of the execution time of b_1 is then

$$E[b_1] = \int_{-\infty}^{+\infty} z f_{b_1}(z)dz = \frac{3}{2\lambda}$$

from which it follows that

$$E[b_2] = 2E[b_1] = \frac{3}{\lambda} = 3m$$

If we ignore the conditional nature of the densities for the directed case the resulting densities for p_0 and p_1 are given by

$$f_{p_0}(z) = f_{p_1}(z) = \lambda^2 z e^{-\lambda z}, \quad z > 0 \tag{6}$$

for which $E[p_0] = 2m$. The density of d_2 can be determined using (4) and (6), and $E[d_2] = 2.75m$, under the given assumptions, and (5) yields only a 2% difference between the barrier and directed performance. Simulation results showed a difference of slightly less than 4%. A similar approach can be applied assuming Gaussian distributions, but the differences are again very small, less than 4%.

We have seen that for a very simple case, the execution times of barrier and directed synchronization are quite close for a variety of distributions representing the behavior of different kinds of code. The exponential distribution approximates the execution time of a loop with a data-dependent exit test. The normal distribution would approximate the execution of straight-line code, given the occurrence of variable-time instructions and memory references. The additional wait time caused by the barrier is typically quite low. It should be noted that the assumption that the barrier and directed synchronizations execute in the same amount of time is quite generous to directed synchronization. As shown in the following sections, the hardware barrier can execute in a few clock cycles, compared to several hundred clock cycles for the fastest directed synchronization implementations. On the other hand, in this very simple case there cannot be any delays due to the total ordering in the SBM queue since the three barriers will always execute in the same order as they appear in the queue. In the next section, we quantify the delays caused by the static barrier queue.

3. Effect of SBM Delays

To understand the potential impact of delays imposed by the total ordering in the queue we will consider the following example, shown in figure 3. In this barrier configuration, n barriers are *unordered*[2] and there are $n!$ possible orderings. The worst case for the SBM occurs when the code regions between barriers have the same expected execution times; in that case, no assumptions concerning the execution ordering of the barriers can made, and the placement of the barriers at compile-time is essentially a random selection.

We will first characterize the number of barriers that are delayed by a particular SBM queue ordering, and show that these delays are equivalent to "combining" the delayed and delaying barriers into several larger barriers or even a single barrier. After characterizing the percentage of barriers combined for a given schedule, it is possible to estimate the delay caused by this combining phenomena.

Consider the case with $n = 3$. There are six possible execution time orderings of barriers 1, 2, and 3[3]. Consider execution ordering $3 \rightarrow 2 \rightarrow 1$: barriers 3 and 2 are forced to wait on barrier 1, and the effect is equivalent to the three barriers being combined into a single barrier. This ordering is shown in figure 4.

2. Barriers are *unordered* if there are no constraints on the order in which they may execute.
3. Note that in this discussion, the numbering scheme for the barriers corresponds directly to their ordering in the SBM queue. Hence, barrier 1 is first in the queue, barrier 2 is second, etc.

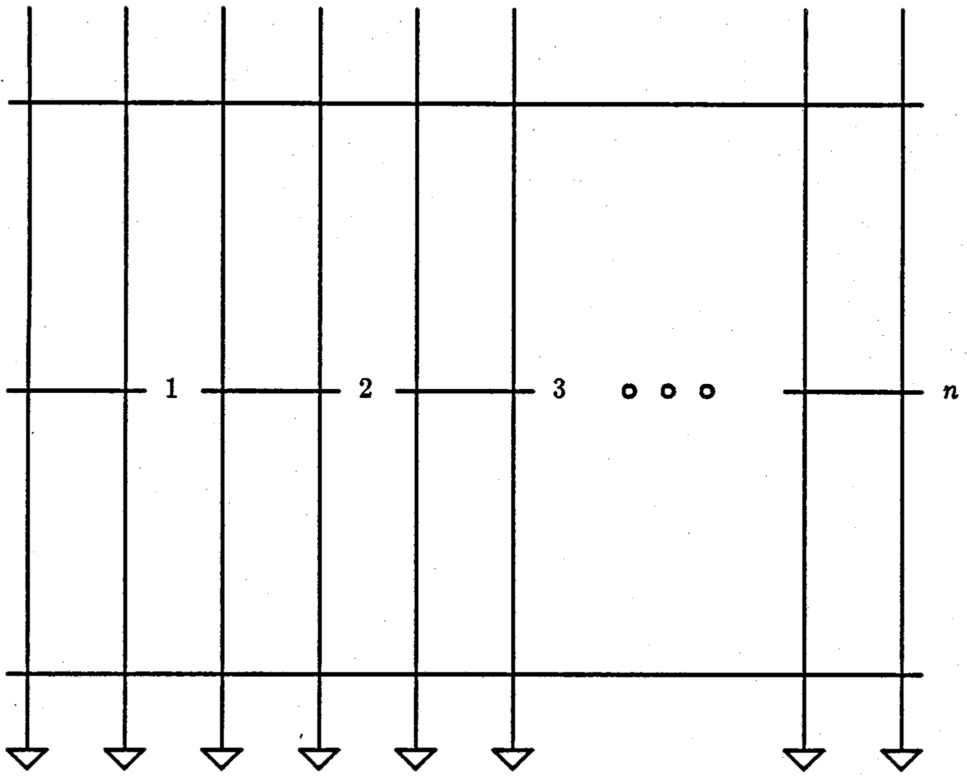


Figure 3: Configuration with n Unordered barriers

If the execution ordering is $2 \rightarrow 1 \rightarrow 3$, barrier 2 is forced to wait for barrier 1 to execute, and these two barriers are, in effect, combined. The different execution orderings can be represented as a tree, shown in figure 5.

Each level of the tree corresponds to the firing of a particular barrier. The leaves of the tree have been annotated with the number of barriers that are delayed given the particular execution ordering. We can determine the expected value for the percentage of barriers delayed (combined), which we will call the *combining quotient,* by weighting the number of barriers delayed by the appropriate probability. Under our assumptions, all execution orderings are equiprobable. Hence, the probability that p barriers are delayed is given by $\frac{\kappa_n(p)}{n!}$, where $\kappa_n(p)$ corresponds to the number of execution orderings with p barrier delays given n barriers in the queue. It can be shown that

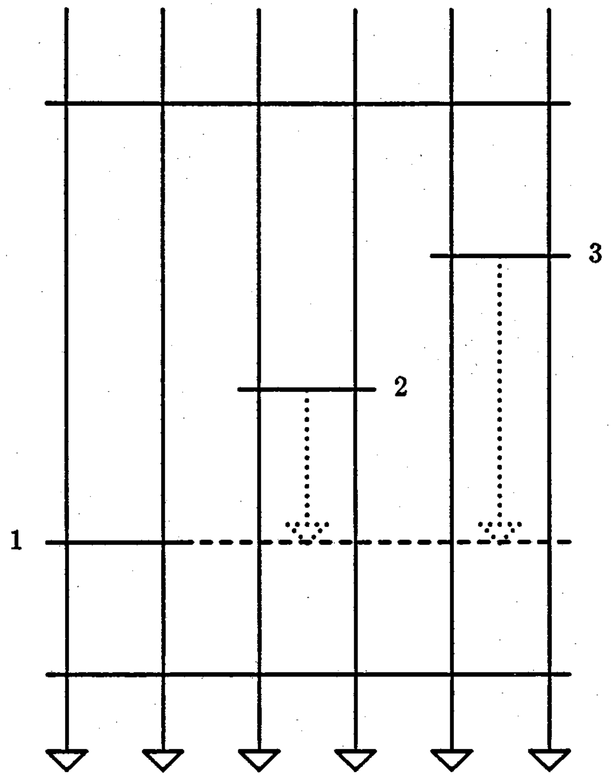


Figure 4: Effect of "Bad" Static Barrier Order

$$\kappa_n(i) \;=\; \kappa_{n-1}(i) + n\,\kappa_n(i-1) \tag{7}$$

This recurrence has been used to generate Table 1.

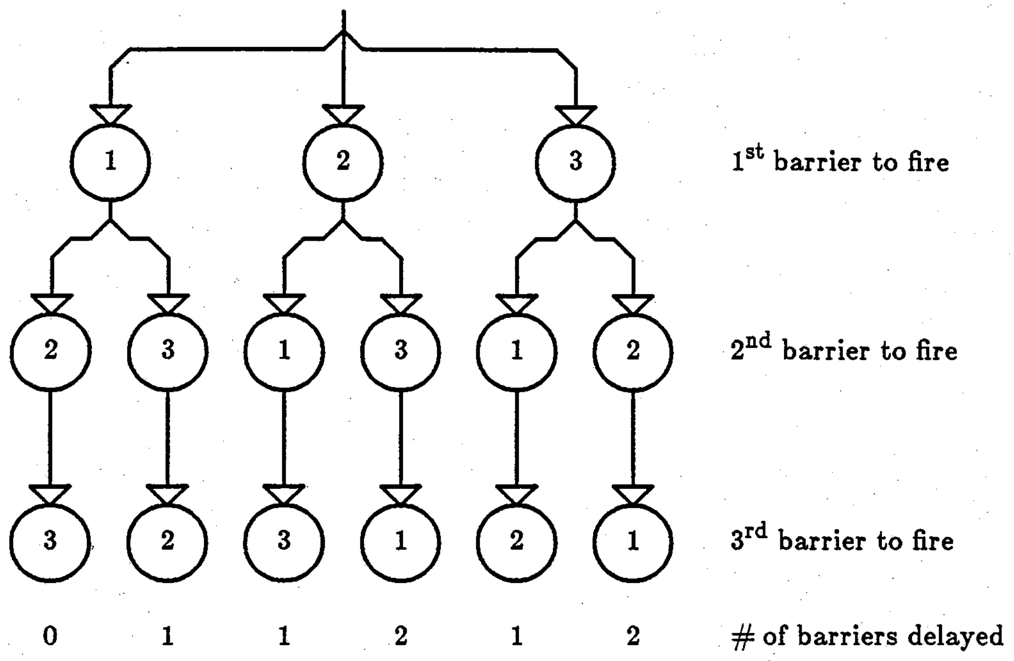


Figure 5: Tree Representing All Possible Execution Orders

n	E[delays]	$\frac{E[delays]}{n}$	0	1	2	3	4	5	6	7	8	9	10	11
2	0.50	0.50	0.50	0.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	1.17	0.58	0.17	0.50	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	1.92	0.64	0.04	0.25	0.46	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	2.72	0.68	0.01	0.08	0.29	0.42	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	3.55	0.71	0.00	0.02	0.12	0.31	0.38	0.17	0.00	0.00	0.00	0.00	0.00	0.00
7	4.41	0.73	0.00	0.00	0.03	0.15	0.32	0.35	0.14	0.00	0.00	0.00	0.00	0.00
8	5.28	0.75	0.00	0.00	0.01	0.05	0.17	0.33	0.32	0.13	0.00	0.00	0.00	0.00
9	6.17	0.77	0.00	0.00	0.00	0.01	0.06	0.19	0.33	0.30	0.11	0.00	0.00	0.00
10	7.07	0.79	0.00	0.00	0.00	0.00	0.02	0.07	0.20	0.32	0.28	0.10	0.00	0.00
11	7.98	0.80	0.00	0.00	0.00	0.00	0.00	0.02	0.09	0.21	0.32	0.27	0.09	0.00
12	8.90	0.81	0.00	0.00	0.00	0.00	0.00	0.01	0.03	0.10	0.22	0.32	0.25	0.08

Table 1: Combining Probabilities for Unordered Static Barriers

The table shows that as the number of unordered barriers in the queue increases the combining quotient, given in the third column, increases asymptotically. The combining quotient versus number of unordered barriers is given in figure 6. Each row in table 1 corresponds to a certain number of unordered barriers, and the columns labeled 0 through

15 contain the probabilities that p barriers will be combined, where p is the column number. For example, with 11 unordered barriers in the queue, the probability that 7 barriers are combined is 0.21. It can be seen from figure 6 that over 80% of the barriers are combined when there are more than 11 unordered barriers in the queue. The percentage is less for smaller numbers of barriers. When the number is from two to five, less than 70% of the barriers are combined.

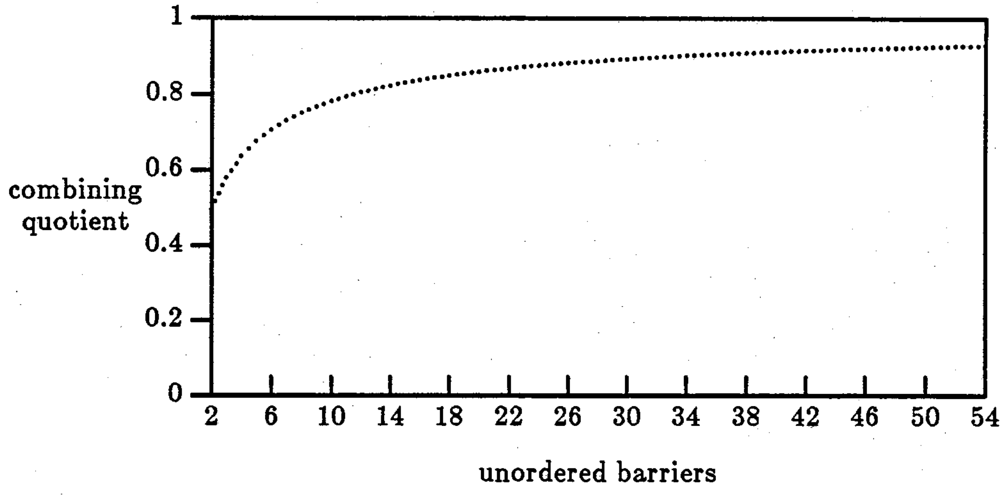


Figure 6: Combining Quotient Vs. No. of Unordered Barriers

The analysis suggests that if the barriers are expected to execute at about the same time, many of the barriers are combined. This phenomena is undesirable in that many processes are forced to wait for the slower processes. To obtain an estimate of these effects, we will determine the expected execution times for processes forced to wait on a barrier, focusing on worst case upper bounds.

Since a large percentage of the barriers in the SBM queue are combined, it is important to know how this affects the execution time of a given program. The upper bound on the additional delays introduced can be determined by assuming that all barriers are combined into a single barrier across all processors. This simplifies the analysis considerably. We will determine the expected execution time for this case for several different distributions of region execution times, but first we give a general upper bound found in Hwang [HwB84]. If random variables $x_1, x_2, \ldots, x_n$ are independent and identically distributed (i.i.d.) with mean m and standard deviation s, then

$$E\left\{\max(x_i)\right\} \leq m + \frac{n-1}{\sqrt{2n-1}} s$$

Hence, we see that the growth in the expected value of the barrier execution time is $O(\sqrt{n}\,)$. It is possible to obtain tighter bounds if particular distributions are assumed.

Let us assume that the region execution times are identical and uniformly distributed, with mean $\frac{m}{2}$ and ranging from 0 to m, and there are n regions that all participate in the barrier. First, we note that the distribution of the max function is given by $F_{\max}(z) = F_u^n(z)$, where $F_u(z)$ is the uniform distribution function. Hence,

$$f_{\max}(z) = \frac{n}{m}(\frac{z}{m})^n$$

which yields

$$E(z) = \frac{n}{m^n}\int_0^m z^n dz = m(\frac{n}{n+1})$$

and $E(z) \rightarrow m$ as $n \rightarrow \infty$, as expected. If we assume that the n regions have an exponential distribution, we can use results concerning *mean time to failure* for parallel systems [Tri82]. In this case,

$$E(z) = m \sum_{i=1}^{n} \frac{1}{i} \simeq m \log_e n$$

where m is the mean of the exponential distribution.

Now let us assume gaussian distributions with mean m and standard deviation s. This yields the following equation for the random variable representing the barrier execution time:

$$z = \max(sz_1+m,\ sz_2+m,\ \cdots,\ sz_n+m) = s\max(z_1,\ z_2,\ \cdots, z_n) + m$$

where the z_i are i.i.d. gaussian random variables with mean 0 and standard deviation 1. Clearly, $E(z) = \sigma E(\max(z_i)) + m$. Rather than evaluating this equation directly, we will employ an important idea from order statistics discussed in [KrW84] and [Gum58]. Given i.i.d. random variables $(x_1, x_2, \cdots, x_n)$, each having distribution function $G(x)$, their *characteristic maximum value* m_n is the solution of the equation

$$1 - G(m_n) = \frac{1}{n} \tag{9}$$

It turns out that m_n is a good estimate of $\max(x_1, x_2, \cdots, x_n)$ for large values of n. Kruskal and Weiss [Tri82] show that for a normal distribution

$$m_n \simeq m + s\sqrt{2 \log_e n} \tag{10}$$

These bounds, developed for a several different distributions, suggest that even in the worst case, when all barriers are combined into a single barrier, the growth in completion time is fairly restrained. Figure 7 shows a plot of these equations for the three different distributions considered here. The number of unordered barriers was scaled by the combining coefficient to get a more accurate bound. We assume that the barriers are combined into a single barrier, and that this single barrier executes last. Note that for the normal distribution, the expected execution time is only $2\frac{1}{2}$ standard deviations away from the mean.

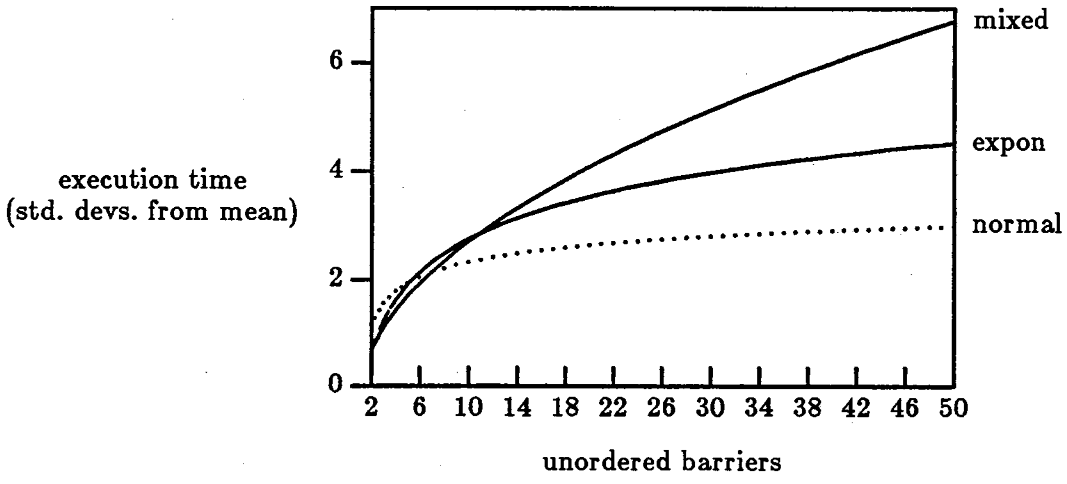


Figure 7: Execution Time Vs. No. of Unordered Barriers

Simulation results discussed in the next section support these analytic results. In addition, two techniques, one related to static scheduling and the other concerning the barrier hardware, are proposed to reduce the delays due to combining effects.

4. Staggered Barrier Scheduling

The analysis given in the previous section made the worst case assumption that the unordered barriers where scheduled such that they all had the same expected execution time. In this situation, the compiler has no useful information concerning the ordering of the barriers in the SBM queue. Any random ordering of the barriers would be expected to perform just as well as any other ordering. We now introduce the concept of *staggered* barrier scheduling. This refers to scheduling barriers so that the expected execution time of a set of unordered barriers $\left\{b_1, b_2, \cdots, b_i, \cdots, b_n\right\}$ is a monotone nondecreasing function. Let $E(b_i)$ be the expected execution time of barrier b_i. Then the following equation

$$E(b_{i+\phi}) - E(b_i) = \delta E(b_i) \qquad (11)$$

defines the *stagger coefficient* δ and the integral *stagger distance* ϕ. We say that two barriers b_j and b_k are *adjacent* if $|j-k| = \phi$. The stagger coefficient δ refers to the percentage difference between the expected execution times of adjacent barriers. Figure 8 shows a schedule of four barriers with a stagger coefficient $\delta = 0.10$ and stagger distance $\phi = 1$.

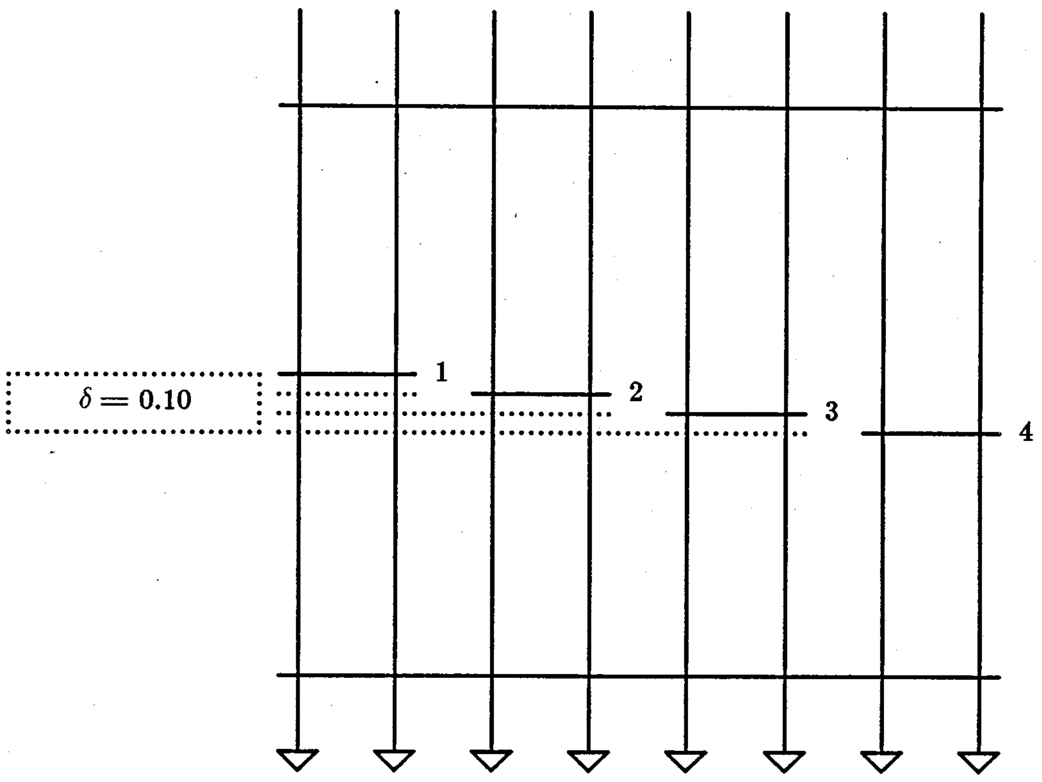


Figure 8: Staggered Barrier Schedule (ϕ=1, δ=0.10)

Figure 9 shows a similar schedule of four barriers, except the stagger distance $\phi = 2$.

The advantage of staggered scheduling is that the barriers can now be expected to execute in a particular order with a higher probability than if there was no staggering. This "expected" execution ordering can then be used as the ordering of the barriers in the SBM queue. Let us consider an example. Let X_i represent the random variable for the execution time of barrier b_i. We wish determine $P[\, X_{i+m\phi} > X_i \,]$, the probability that barrier $b_{i+m\phi}$ executes after b_i. The former barrer is staggered $m\delta$ percent from the latter. We have

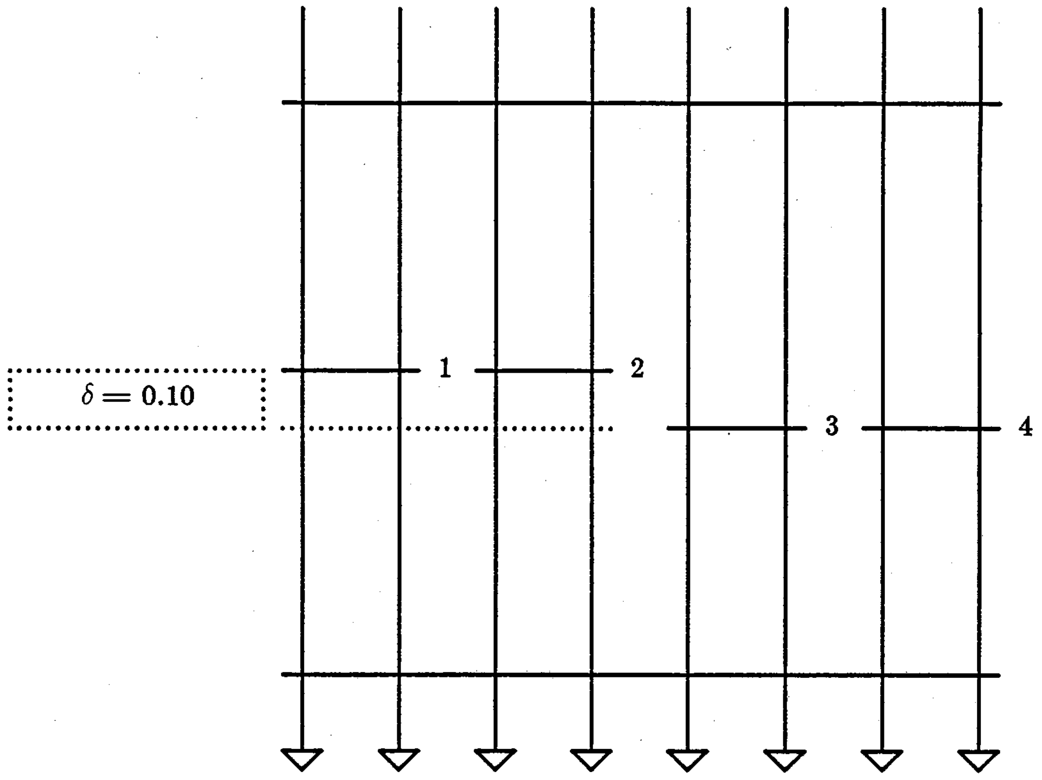


Figure 9: Staggered Barrier Schedule (ϕ=1, δ=0.10)

$$\mathrm{P}[\, X_{i+m\phi} > X_i \,] = \mathrm{P}[\, X_{i+m\phi} - X_i > 0 \,] = 1 - \mathrm{P}[\, X_{i+m\phi} - X_i < 0 \,] = 1 - F_{X_{i+m\phi}-X_i}(0)$$

and if exponential distributions are assumed

$$\mathrm{P}[\, X_{i+m\phi} > X_i \,] = \frac{(1.0+m\delta)\lambda}{\lambda + (1.0+m\delta)\lambda}$$

Simulations results show that staggered scheduling reduces the delay caused by *queue waits*, i.e. waits caused solely by the SBM queue ordering. Figure 10 shows the simulation results assuming that region execution times have a normal distribution with μ=100 and s=20, ϕ=1 and δ set to 0.0, 0.05, and 0.10.

It is evident from figure 10 that staggering the barriers can significantly reduce the accumulated delays caused by queue waits.

5. Hybrid Barrier Mechanism

Recall that the static barrier hardware design (a queue) was much simpler and less expensive than the dynamic barrier design (an associative memory). When first proposed [DiS87], static barrier and dynamic barrier were considered as two extremes; the Static

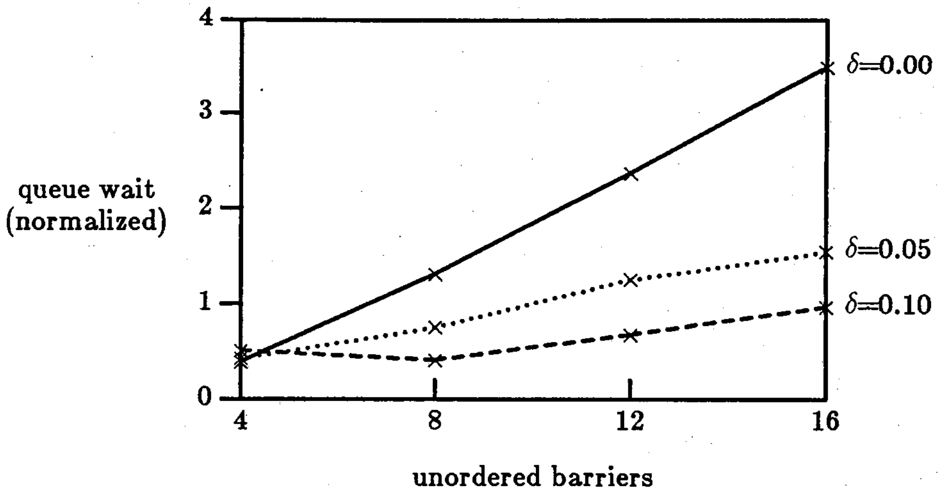


Figure 10: Effect of Staggering on Queue Wait Time

Barrier MIMD design was favored due to the smaller hardware requirements, although it could introduce delays not present in a dynamic barrier scheme. In this section, we propose a hybrid barrier mechanism, shown in figure 11, that combines the associative buffer of dynamic barrier with the queue of static barrier.

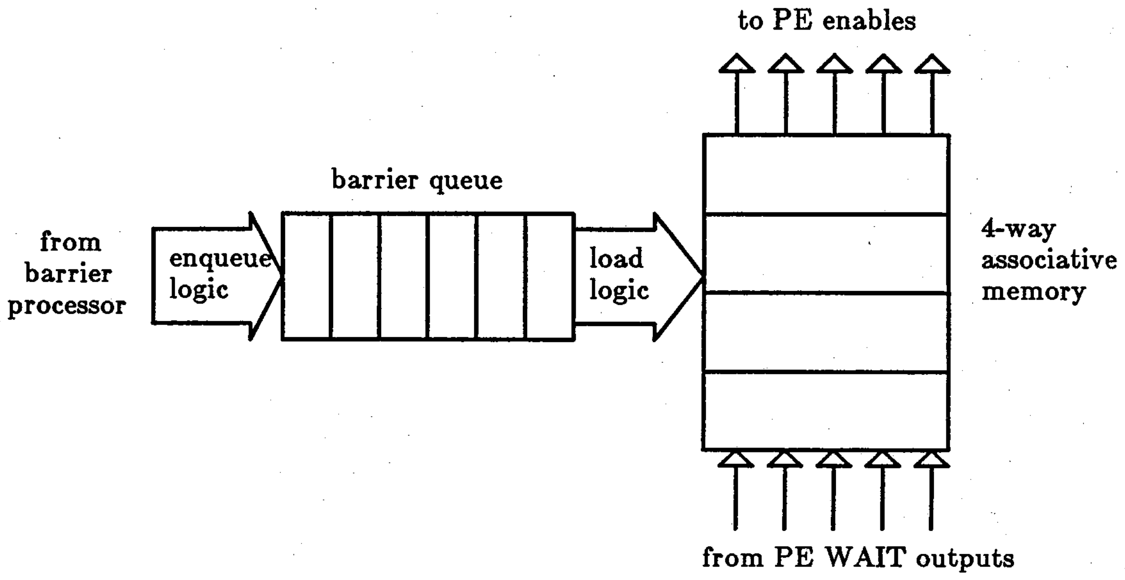


Figure 11: Hybrid Static/Dynamic Barrier Architecture

The basic idea is to use the queue to load the barriers into a very small associative memory (in figure 11 the associative memory has four cells). Preliminary simulation results have shown that the associative memory in the hybrid barrier architecture need be no larger than four to five cells to reduce delays caused by the barrier hardware mechanism to almost zero.

Preliminary simulation results are displayed in Figures 12 and 13. The horizontal axis indicates the number of unordered barriers that are to be executed, while the vertical axis represents the total barrier delay, normalized to μ. The region execution times are taken from a normal distribution with $\mu = 100$ and $s = 20$ before staggering is applied.

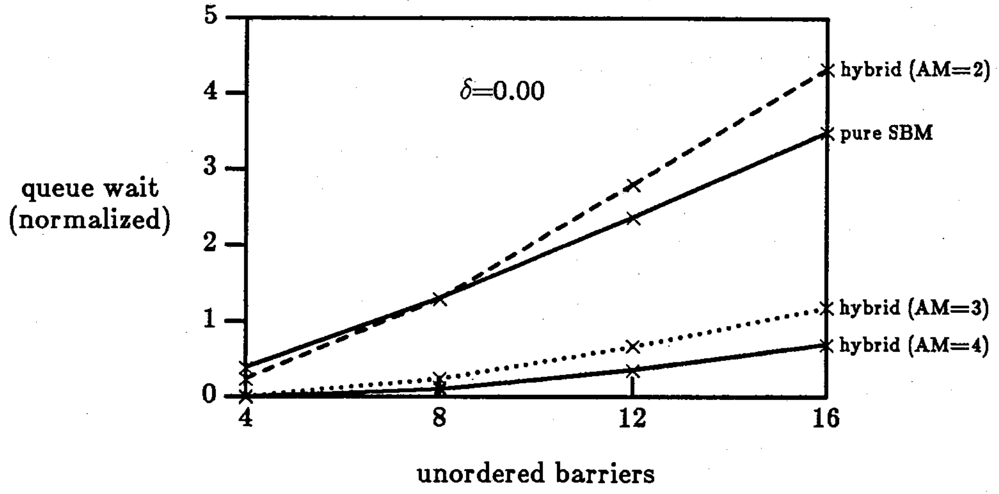


Figure 12: Effect of Hybrid Architecture on Queue Wait Time

From Figure 12, it is evident that the hybrid barrier scheme reduces barrier delays almost to zero for small associative buffer sizes. There is an anomaly here for an associative buffer size of two: in this case, the barrier delays are greater that those of the pure static barrier scheme when the number of barriers is greater than about eight. The reasons for this anomaly are currently under investigation, but no clear answer is currently available. This anomaly is of more theoretical than practical significance.

Figure 13 shows the results when staggered scheduling is employed with $\delta = 0.10$ and $\phi = 1$. The effects of staggering alone reduce the delays significantly.

6. Conclusions and Further Work

This report has discussed the hardware barrier, a new technique for fast synchronization in parallel processors. We have shown that in some cases the hardware barrier can compete with directed sychronization even under assumptions very favorable to directed. Whether this is true for the general case is currently under study, but simulations to date suggest that barriers can compete with directed. The effects of static barrier delays have been quantified, and upper bounds developed. Two techniques for reducing these delays were developed: staggered barrier scheduling and the hybrid barrier mechanism. Simulations run to date have shown these techniques to be especially effective when used together. Additional simulations need to be performed to verify these preliminary results.

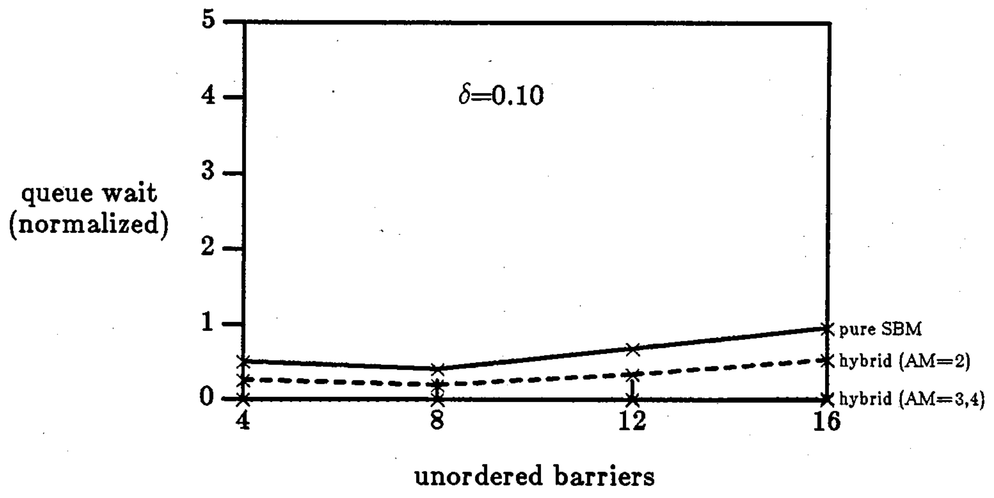


Figure 13: Effects of Staggering + Hybrid Architecture on Queue Wait Time

Various stagger coefficients and stagger distances should be simulated and compared.

The performance analysis studies to date have concentrated almost wholly on scheduling large-grain tasks. However, the initial proposal for the hardware barrier mechanism focused on the small granularity parallelism made available by this fast synchronization technique. Hence, the next phase of the performance analysis will concentrate on fine-grained scheduling. Various heuristics are currently being considered for scheduling barrier MIMDs, and an interface between the code scheduler and the simulator will be constructed to test the results of the code scheduler using the various heuristics.

The overhead and implementation requirements for different synchronization techniques need to be examined. As described in this report, barriers can be designed to execute at the clock cycle level. This is not possible with other techniques which must access shared memory, registers, or a even a combining network. In addition, unlike a barrier, these techniques do not provide a *statically* predictable time to perform the synchonization. Barrier synchronization requires additional hardware, including a synchronization network a barrier processor, enable flags, and other logic at each processor.

References

[DiS87] H.G. Dietz and T. Schwederski, "Extending Static Synchronization Beyond SIMD and VLIW," Technical Report TR-EE 88-25, School of Electrical Engineering, Purdue University, June 1988.

[Die87] H.G. Dietz, *The Refined-Language Approach to Compiling For Parallel Supercomputers*, Ph.D. Dissertation, Polytechnic University, June 1987.

[Axe86] T.S. Axelrod, "Effects of Synchronization Barriers on Multiprocessor Performance," *Parallel Computing*, Vol. 3, pp. 129-140, 1986.

[HeF88] D. Hensgen, R. Finkel, and U. Manber, "Two Algorithms for Barrier Synchronization," *Int. Journal of Parallel Programming*, Vol. 17, No. 1, pp. 1-17.

[KrW84] C. P. Kruskal and A. Weiss, "Allocating Independent Subtasks on Parallel Processors," *Int. Conf. on Parallel Processing*, pp. 236-240, 1984.

[HwB84] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill:New York, 1984, pg. 611.

[Tri82] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall: Englewood Cliffs, NJ, 1982, pp. 217-219.

[Gum58] E.J. Gumbel, *Statistics of Extremes*, Columbia University Press: New York, 1958.