To appear: *Proceedings of the 8th ACM International Conference on Supercomputing,* July 1994.

# An Evaluation of Directory Protocols for Medium-Scale Shared-Memory Multiprocessors *

Shubhendu S. Mukherjee and Mark D. Hill
Computer Sciences Department
University of Wisconsin–Madison
1210 West Dayton Street
Madison, WI 53706 USA
shubu@cs.wisc.edu

## Abstract

This paper considers alternative directory protocols for providing cache coherence in shared-memory multiprocessors with 32 to 128 processors, where the state requirements of $Dir_N$ may be considered too large. We consider $Dir_i$B, $i = 1, 2, 4$, $Dir_N$, *Tristate* (also called *superset*), *Coarse Vector*, and three new protocols. The new protocols—*Gray-hardware*, *Gray-software*, *Home*—are optimizations of *Tristate* that use gray coding to favor near-neighbor sharing.

Our results are the first to compare all these protocols with complete applications (and the first evaluation of *Tristate* with a non-synthetic workload). Results for three applications—`ocean` (one-dimensional sharing), `appbt` (three-dimensional sharing), and `barnes` (dynamic sharing)—for 128 processors on the Wisconsin Wind Tunnel show that (a) $Dir_1$B sends 15 to 43 times as many invalidation messages as $Dir_N$, (b) *Gray-software* sends 1.0 to 4.7 times as many messages as $Dir_N$, making it better than *Tristate*, *Gray-hardware*, and *Home*, and (c) the choice between $Dir_i$B, *Coarse Vector*, and *Gray-software* depends on whether one wants to optimize for few sharers ($Dir_i$B), many sharers (*Coarse Vector*), or hedge one's bets between both alternatives (*Gray-software*).

**Keywords:** Shared-memory multiprocessors, cache coherence, directory protocols, and gray code.

0

## 1 Introduction

This paper considers *medium-scale* parallel computers, which we define as having 32 to 128 processors. Small-scale machines differ from medium-scale ones because they can have centralized resources (e.g., main memory) and are often designed primarily to run independent serial programs. In contrast, large-scale machines must use distributed resources (e.g., processor-memory nodes) and are designed for asymptotic scalability, which may compromise performance on small versions of these systems. Medium-scale machines fall in between. They probably use processor-memory nodes to avoid the bottlenecks of small-scale machines, but they may occasionally use unscalable solutions—such as broadcasts—avoided by large-scale machines. Of course, others might pick different numbers for the exact boundaries of medium scale.

We expect that many medium-scale computers will support cache-coherent shared memory in hardware. Relative to message-passing multicomputers, hardware shared memory makes it easier to provide operating system support for multiple users, is a more straightforward target for automatic parallelization of serial programs, and allows programmers of explicitly-parallel programs to use pointers and ignore per-processor memory limits. Per-processor caches reduce average memory latency and bandwidth demand when some locality is present. Hardware cache coherence makes the caches functionally invisible so

* This research was supported in part by NSF PYI Awards CCR-9157366 and MIPS-8957278, NSF Grants CCR-9101035 and MIP-9225097, Univ. of Wisconsin Graduate School Grant, Wisconsin Alumni Research Foundation Fellowship and donations from A.T.&T. Bell Laboratories, Digital Equipment Corporation, Thinking Machine Corporation, and Xerox Corporation. Our Thinking Machines CM-5 was purchased through NSF Institutional Infrastructure Grant No. CDA-9024618 with matching funding from the Univ. of Wisconsin Graduate School.

that compilers and operating systems can optimize for common cases rather than managing worst-case data sharing. For these reasons, we assume cache-coherent shared memory in this paper.

Many protocols have been proposed for implementing cache coherence. We assume that medium-scale computers are too large to rely on snooping a shared bus [2] but small enough that they need not be concerned about asymptotic scalability [10, 12]. A reasonable structure for medium-scale computers is to associate a directory with the memory module in each of the $N$ processor-memory nodes. For each aligned block in memory—say 32 to 128 bytes—a directory entry records the state of the block and the identities of processors which might have copies. We assume a write-invalidate protocol in which the block may be currently uncached, cached writable at one processor, or cached read-only by 1 to $N$ processors. An *invalidation event* occurs when a processor wishes to obtain a writable copy of a block while other cached copies exist. Invalidation events force the coherence protocol to send 1 to $N$ *invalidation messages*.[1] To avoid always sending $N$ invalidation messages, most directory entries include a *sharing code*. When one writable cache copy is outstanding, the sharing code identifies which processor has the block using at least $logN$ bits.[2]

Alternative sharing codes have been proposed for identifying the sharers when multiple read-only copies are outstanding. At one extreme is $Dir_1$B [1] which sends $N$ invalidation messages at each invalidation event with more than one sharer. If there were actually $j$ sharers, $j > 1$, then $N - j$ of the $N$ messages are *unnecessary invalidation messages*. The advantage of $Dir_1$B is that it requires no sharing code beyond the $logN$ bits needed to identify a single writable copy. However, the unnecessary invalidation messages could have three potential negative effects: (a) increased contention in the network, (b) wasted cycles to send the messages, assuming they are sent out one at a time, and (c) wasted cycles to process these messages and increased contention at the directories that do not have a copy of the block. At the other extreme is $Dir_N$ that uses a bit vector to exactly identify the sharers [1]. $Dir_N$ never sends an unnecessary invalidation message, but uses $N$ bits of sharing code. For 128 processors, this sharing code is 50% to 12.5% memory overhead for 32- to 128-byte blocks.

Several proposals also exist that use a smaller sharing code than $Dir_N$, but do not always fall back on broadcast. We call these proposals *multicast* protocols; others have called them *limited broadcast* protocols [1]. The challenge of designing a multicast protocol lies in minimizing both the sharing code size and the number of unnecessary invalidation messages.

In this paper we will study variants of three previously proposed multicast protocols—$Dir_i$B, *Tristate*, and *Coarse Vector*. $Dir_i$B, $1 < i < N$, uses $i \times logN$ bits to exactly identify upto $i$ sharers and broadcasts otherwise [1]. *Coarse Vector* uses $N/K$ bits, where a bit is set if any of the processors in a $K$-processor group cached the block [9]. *Tristate* [1], also called the *superset scheme* by Gupta et al. [9], uses a $logN$ digit code requiring 2 bits per digit. The j-th digit of the code is 0 if the j-th bit of all sharers is 0; the digit is 1 if all sharers have 1; the digit is `both` otherwise. On an invalidation event, *Tristate* sends invalidation messages to all processors covered by its sharing code. 32 processors, for example, require a five-digit code. The code value "`1 both both 1 0`" implies that invalidations must be sent to processors 10010, 10110, 11010, and 11110. In general, if $k$ digits are `both`, then $2^k$ invalidations must be sent.

Section 2 proposes three optimizations of *Tristate* that can perform better for near-neighbor sharing. *Gray-hardware* works exactly like *Tristate* except that processors are enumerated using a binary-reflected gray code, so that consecutive processor numbers differ by one bit. *Gray-software* uses the same hardware as *Tristate* but shows how software can redistribute the work so that neighboring work is assigned to processors whose numbers differ in only one bit. Finally, *Home* uses gray-coded processor numbers like *Gray-hardware*, but has a sharing code of only $logN$ bits, where the j-bit is set if the j-th bit of any sharer differs from the j-bit of the home node number. Protocol features are summarized in Table 1.

Section 3 discusses the three benchmarks used in this paper—`ocean` (one-dimensional sharing), `appbt` (three-dimensional sharing), and `barnes` (dynamic sharing), evaluation platform (Wisconsin Wind Tunnel), implementation assumptions (e.g., 32 to 128 processors, notifying protocols, and no special network support for broadcasts or multicasts), and evaluation metric (number of invalidation messages).

Section 4 shows $Dir_1$B sends 15 to 43 times as many invalidation messages as $Dir_N$. Of the closely-related protocols of *Tristate*, *Gray-hardware*, *Gray-software*, and *Home*, we find *Gray-software* per-

[1] The number of invalidation messages is actually $N - 2$ because (a) no invalidation will be sent to the requesting processor initiating the invalidations, and (b) the invalidation to the processor on the home node will be locally absorbed. Of course, if the requesting processor is on the home node, the number of invalidation messages will be $N - 1$. Our discussion ignores these minor effects, but our simulations consider them.

[2] All logarithms in this paper are base two.

| Protocol | Description | Number of bits in sharing code | Invalidation messages consecutive four | worst four |
|---|---|---|---|---|
| *$Dir_N$* | maintains precise identity of sharers | $N$ | 4 | 4 |
| *$Dir_i$*B, $i < N$ | broadcasts invalidations for sharers $> i$ | $i \times logN$ | $N, i < 4$<br>$4, i \geq 4$ | $N, i < 4$<br>$4, i \geq 4$ |
| *Tristate* | each digit in the sharing code represents states 0, 1 and `both` | $2 \times logN$ † | $3 \times logN + 1$ | $N$ |
| *Coarse Vector* | each bit in sharing code represents $K$ processors, $1 < K < N$ | $ceiling(N/K)$ † | $2 \times K$ or $3 \times K$, $K = 2$<br>$K$ or $2 \times K$, $K > 2$ | $4 \times K$ |
| *Gray-hardware* | *Tristate* with gray code in hardware | $2 \times logN$ † | 7 | $N$ |
| *Gray-software* | *Tristate* in hardware, gray code in software | $2 \times logN$ † | 7 | $N$ |
| *Home* | *Gray-hardware* with home id as reference id | $logN$ † | 7 to $N$ | $N$ |

Table 1: Protocols

This table provides a description of the protocols studied in this paper. The three new protocols proposed in this paper are shown in the lower half of the table. Specifically, for *$Dir_i$*B, we have chosen $i$ to be 1, 2, and 4, for our study. Column two provides a brief description of the protocols, column three shows the number of bits necessary for the sharing code, column four expresses the number of invalidation messages sent by the corresponding protocol on an invalidation event when four consecutive processors are involved in sharing, while column five shows the number of invalidation messages if *any* four processors were involved in sharing (worst case). $N$ is the total number of processors in the system.

When four consecutive processors are involved in sharing, the number of invalidations sent on an invalidation event is straightforward for *$Dir_N$* and *$Dir_i$*B. For *Coarse Vector*, when $K = 2$, the number of invalidations is either $2 \times K$, when each bit covers exactly two processors, or $3 \times K$, when the four sharers spans three consecutive bits in the sharing code. When $K > 2$, the number of messages is $K$, if one bit covers all the four sharers, or $2 \times K$, when the sharers span two consecutive bits. For *Tristate*, the expression, $3 \times logN + 1$, can be calculated using reasoning similar to that used to motivate *Gray-hardware* in Section 2.1. Sharers j to j+3 differ in two bits if j mod 4 = 0, which is true for one-quarter of all j's. Otherwise, the bit patterns also differ in one or more higher-order bits. Given this case, the probability of exactly an i-bit difference is $\frac{1}{2^i}$. Thus, the expected number of messages sent for four consecutive sharers is $\frac{1}{4} \times 2^2 + (\frac{3}{8} \times 2^3 + \frac{3}{16} \times 2^4 + ...)$, which sums to $3 \times logN + 1$, for arbitrarily large N. If finite system size is considered, the sum turns out to be $3 \times logN - 2$. For *Gray-hardware*, the expected number of messages for four consecutive sharers is $2^2$ if j mod 4 = 0 and $2^3$ otherwise, which reduces to $7 = \frac{1}{4} \times 2^2 + \frac{3}{4} \times 2^3$. The number of messages for *Gray-software* is identical to *Gray-hardware*. For *Home*, the expected number of invalidation messages for four consecutive sharers is 7, if the home node is one of the sharers; otherwise, it is more than 7 and can be as large as $N$.

For a worst combination of four sharers, *$Dir_N$* and *$Dir_i$*B, $i \geq 4$, will still send only four invalidation messages. *$Dir_i$*B with $i < 4$, *Tristate*, *Gray-hardware*, *Gray-software*, and *Home* will all send $N$ messages. *Coarse Vector* will, however, send only $4 \times K$ messages, because only four separate bits in the sharing code will be set.

† These protocols use an additional $logN$ bits for a counter, since we assume a notifying protocol [6] where only positive acknowledgements are returned on invalidation requests.

forms best of four with the same hardware as *Tristate* . For `ocean`, `appbt`, and `barnes`, respectively, *Gray-software* sends 1.0, 1.3, and 4.7 times as many invalidation messages as *$Dir_N$*. The `barnes` number is large due to a high degree of dynamic sharing. *Coarse Vector* performs better for `barnes` but worse for the other two applications, while *$Dir_2$*B and *$Dir_4$*B perform very poorly whenever there are more than two or four sharers (as is to be expected). Thus, the choice of protocol between *$Dir_i$*B, *Coarse Vector*, and *Gray-software* will depend on whether one wants to optimize for few sharers (*$Dir_i$*B), many sharers (*Coarse Vector* ), or hedge one's bets between both alternatives (*Gray-software*).

We see two key contributions for this paper. First, we introduce three new protocols—*Gray-hardware*, *Gray-software*, and *Home*. Second, we do the first study to compare *$Dir_1$*B, *$Dir_N$*, *Tristate*, *Gray-hardware*, *Gray-software*, *Home*, *Coarse Vector*, *$Dir_2$*B, and *$Dir_4$*B, using the same assumptions, 32 to 128 processors, and running the applications to completion (tens of billions of cycles each). None of the previous studies have done a systematic comparison of the existing multicast protocols for medium-scale shared-memory systems. In particular, *Tristate* has not been evaluated with real benchmarks.

Previous studies of directory protocol performance were limited by systems with smaller number of processors—between 4 and 64. Agarwal et al. [1] evaluated directory protocols for a small bus-based system using four-processor VAX traces less than two million instructions long. In the same paper, they proposed *Tristate* without evaluating it. The MIT Alewife machine uses a $Dir_N$-like protocol, called LimitLESS, which maintains five of the pointers in hardware and the rest in software. Chaiken et al. [5] compared LimitLESS against $Dir_N$, using several applications on 16 and 64 processors with 7 to 30 million references per application. They found that LimitLESS's performance is comparable to $Dir_N$. Gupta et al. [9] compared *Coarse Vector* with *Tristate* using a synthetic benchmark, which randomly picked the processors sharing a block, and concluded that *Coarse Vector* is superior to *Tristate*, which is contradicted by our results based on three non-synthetic benchmarks. In the same paper, Gupta et al. presented invalidation message counts and execution time for four benchmarks having 8 to 22 million shared-memory references on 32 processors using the Tango simulator. They concluded that *Coarse Vector* could be competitive with $Dir_N$. Wood et al. [17] introduced $Dir_1SW^+$—a broadcast protocol similar to $Dir_1$B that traps like LimitLESS to handle invalidations when number of sharers is greater than one. They compared $Dir_1SW^+$ against $Dir_1$B, $Dir_4$B, and $Dir_N$, for a 32-processor system using eight benchmarks by simulating between 1.5 to 25 billion cycles (for each application) on the Wisconsin Wind Tunnel [15]. They concluded that $Dir_1SW^+$'s performance is comparable to $Dir_N$. However, their results could be biased in favor of $Dir_1SW^+$ because their simulations did not accurately model network contention.

# 2 New Multicast Protocols

This section discusses three optimizations of *Tristate*—*Gray-hardware*, *Gray-software*, and *Home*—that can perform better on near-neighbor sharing. The basic idea behind these protocols is to use *gray coding* to reduce the number of `both`'s in the sharing code. We first discuss constructing gray codes for one- and multi-dimensional sharing, and then present the new protocols.

## 2.1 Gray Code

*Tristate* performs non-optimally for near-neighbor sharing between consecutive processors along one dimension, because consecutive processor numbers can differ in $k >> 1$ bits, causing $2^k$ messages to be sent on an invalidation event. Specifically, half of all pairs of consecutive numbers differ in one bit (i.e., even-odd pairs: 0-1, 2-3, 4-5, ...), one-quarter differ in two bits (1-2, 5-6, ...), one-eighth in three bits, etc. The expected number of invalidation messages is $\frac{1}{2} \times 2 + \frac{1}{4} \times 2^2 + \frac{1}{8} \times 2^3 + ... + \frac{1}{2^{logN}} \times 2^{logN} + (\frac{1}{2^{logN}} \times 2^{logN}) = logN$. The final term $(\frac{1}{2^{logN}} \times 2^{logN})$ occurs when processor $N-1$ also shares with processor zero[3]. While $logN$ sounds small, here it means that *Tristate* sends five to seven invalidations (per invalidation event) for 32- to 128-processor systems, while $Dir_N$ needs to send only two invalidations.

If instead we enumerate processors with a binary-reflected gray code, then consecutive processor numbers would always differ in only one bit. In this case, only two invalidation messages would be needed, the same as $Dir_N$. For more consecutive sharers, using a gray code produces results between *Tristate* and $Dir_N$ (see row *Gray-hardware* of Table 1).

While binary-reflected gray coding works well on near-neighbor sharing in one dimension, what can be done to support near-neighbor sharing in multiple dimensions? In the simplest case, one can form a multi-dimensional gray code by concatenating gray codes from each dimension. The multi-dimensional gray code for an $N = 2^4 \times 2^4 \times 2^8$–node mesh uses 16 bits—4 from the first index, 4 from the second, and 8 from the third. Forming a multi-dimensional gray code is more complex, however, if most dimensions are not powers of two.

In general, the problem is equivalent to the following graph embedding problem:

> Given a d-dimensional mesh and the smallest hypercube with as many nodes as there are vertices in the mesh, what is the best possible mapping of vertices of the mesh to the nodes of the hypercube such that neighbors in the mesh are as close to each other as possible [7].

Define *dilation* as the maximum distance between any two mesh neighbors on the hypercube. Alternately, *dilation* can also be defined as the maximum number of bit positions in which any two mesh neighbors differ when mapped to the hypercube. The problem then is to find a mapping with the minimum dilation.

The problem can be solved optimally—with dilation 1—if at least $d-1$ dimensions of a $d$-dimensional mesh are powers of two. This case occurs commonly in parallel applications, because programmers size their data to fit the machine they run on.

For the important case of two-dimensional meshes, Chan [7] shows how to automatically construct an

[3] Without this wrap-around, the series sums to $logN - 1$.

embedding with dilation one or two. Consider a 3 x 5 mesh to be embedded in its smallest hypercube with 16 nodes. Simply taking the gray codes of indices in each dimension—two bits plus three bits—will necessitate a 32-node hypercube. Chan's construction starts with two bits for the first index and two for the second, and then encodes the information missing in the second index in the unused state(s) of the first.

Three-dimensional meshes can be embedded with dilations of one (two dimensions are powers of two), two (one dimension is a power of two and using Chan's construction for the other two), three (in many cases [4]), and never worse than seven ([8]). Rarely-used higher dimensional meshes can always be embedded with dilation $O(dimension)$ [8].

## 2.2 New Protocols

Here we discuss specific implementation issues related to the three new protocols—*Gray-hardware*, *Gray-software*, and *Home*.

### 2.2.1 Gray-hardware

```
unsigned graycode(unsigned id)
{
  return (id ^ (id >> 1));
}

unsigned inverse_graycode(unsigned graycode)
{
  unsigned i, id = graycode, temp = graycode;

  for (i=1;i<log2(N),i++)
  {
    temp = temp >> 1;
    id ^= temp;
  }
  return id;
}
```

Figure 1: C code for computing binary-reflected gray code and its inverse

This code shows how to do the gray coding in one dimension. The text explains how to do multi-dimensional gray coding.

*Gray-hardware* is optimized for near-neighbor sharing between consecutive processors in one dimension. It works like *Tristate*, except that processor numbers are stored in the sharing state with a binary-reflected gray code. The code can be formed with a shift and exclusive-or (Figure 1). Upon an invalidation event, gray codes are inverted using the naive procedure depicted in Figure 1 or with special parallel prefix hardware.

### 2.2.2 Gray-software

*Gray-software* eliminates two negative aspects of *Gray-hardware*. First, *Gray-software* can be customized to support either one- or multi-dimensional sharing. Second, *Gray-software* eliminates the extra hardware of *Gray-hardware* to use the same hardware as *Tristate*.

| Benchmark | Brief Description | Input Data Set | Cycles ($\times 10^9$) |
|---|---|---|---|
| ocean | 1D stencil | 384 x 384, 2 days | 17.6 |
| appbt | 3D stencil | $32^3$, 4 iter | 77.8 |
| barnes | 8-ary tree | 8192 bodies, 4 iter | 26.4 |

Table 2: Application programs

This table lists the characteristics of the three benchmarks used for simulations in this paper. Column two provides a brief description of the benchmark that is relevant for this paper. Column three lists the input data set. Column four lists the number of cycles (in billions) for 128 processor runs for $Dir_N$.

*Gray-software* supports near-neighbor sharing by asking software to assign neighboring work to processors whose numbers differ by one bit (i.e., are gray codes). Say, for example, a program normally assigns column 3 to processor 3 and column 4 to processor 4. With *Gray-software*, columns 3 and 4 should be assigned to processors 2 and 6, respectively. Alternatively, columns 2 and 7—the columns whose gray codes are 3 and 4—should be assigned to processors 3 and 4.

While this software transformation may sound complex, it can be hidden in a single line change in the many *single-program-multiple-data* (SPMD) programs that calculate what work to do as a function of processor number using something like:

```
my_work = get_my_proc_num().
```

For one-dimension, the above line should be replaced with:

```
my_work = inverse_graycode(get_my_proc_num()).
```

For `dim` dimensions, use:

```
my_work = inverse_multi_graycode(
        dim, n1, n2,..., ndim),
```

where `n1` and `ndim` sizes of each dimension. The inverse functions can be easily provided as library routines.

Changing the mapping from processes to processors—as done by *Gray-software*—may make the processors that share data further away (or closer) in the interconnection network topology of a real machines. We do not expect this movement to have a first-order effect on performance, because with our directory protocols data moves from processor to a directory at an arbitrary node to processor, not directly between processors. Performance could be affected, however, in systems that carefully selected directory nodes to minimize communication distance.

### 2.2.3 Home

*Home* is another multicast protocol that cuts the sharing code size of *Tristate* in half. Instead of using two bits per digit to encode `0`, `1`, and `both`, *Home* uses one bit which is reset only if all sharers have the same bit value as the directory entry's home node. The performance of *Home* is very sensitive to data placement, since it acts like *Tristate* with the home node always participating in the sharing.

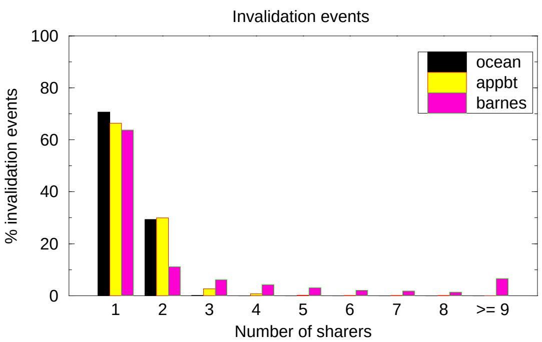


Figure 2: Invalidation events distribution

These measurements are with $Dir_N$ on 128 processors. Measurements with other protocols studied in this paper show negligible difference in the number of invalidation events.

# 3 Methodology

This section discusses the benchmarks used in this paper, the platform to perform evaluations, the parallel system assumptions, and the metric for comparing the different protocols.

## 3.1 Benchmarks

The three benchmarks used in this paper are `ocean` and `barnes` from the SPLASH suite [16], and `appbt`, a NAS serial benchmark [3] that was parallelized by our group. We limited ourselves to three codes—selected as having one-dimensional, three-dimensional, and dynamic sharing—to allow us to focus on qualitative trends and to reduce simulation time. Table 2 summarizes the programs.

`Ocean` is a hydrodynamic simulation of a two-dimensional (2D) cross-section of a cuboidal ocean basin. The principal data structures are two-dimensional arrays. Each processor is assigned a sequence of columns from the 2D arrays. Sharing is between two consecutive processors along the boundary column. Invalidations occur predominantly when the number of sharers is less than equal to two, as shown by Figure 2. Figure 2 shows the invalidation events distribution for the three benchmarks with 128 processors. The horizontal axis shows the number of sharers at an invalidation event, while the vertical shows the percentage of invalidation events occurring with each number of sharers.

`Appbt` is a computational fluid dynamics program, which solves multiple independent systems of non-diagonally dominant, block tridiagonal equations with a 5 x 5 block size. The code is spatially parallelized in three dimensions with each processor assigned the responsibility for updating one 3D sub-block. Sharing is between neighboring processors in 3D along the boundaries of these sub-blocks. The principal sharing occurs along faces, corner columns and corner points of these sub-blocks between two, three, and four processors, respectively. Figure 2 shows the invalidation events distribution with different number of sharers.

`Barnes` performs a gravitational N-body simulation using the Barnes-Hut algorithm. The main data structure is an 8-ary tree, which is partitioned contiguously among processors. We have used the *cost-zones* partitioning scheme described in detail by Singh et al. [16]. The allocation in this scheme is such that contiguity of partitions in the tree does not guarantee contiguity in space. The sharing pattern is dynamic and irregular and the frequency of many dynamic sharers grows with the number of processors.

## 3.2 Evaluation Platform

Our measurements were done on the *Wisconsin Wind Tunnel* [15]. It runs parallel shared-memory programs on a parallel message-passing computer (a Thinking Machines CM-5) and uses a distributed, discrete-event simulation to concurrently calculate the programs' execution times on a proposed target machine. The Wisconsin Wind Tunnel simulates one or more target nodes (processors) per host node. Physical memory limitations restrict us to simulations of 128 processors or less on a 32-node CM-5. All protocols were implemented as variants of the base $Dir_1SW^+$ [17] protocol module.

## 3.3 System Assumptions

We assume cache-coherent shared-memory multiprocessors of 32, 64, or 128 processor-memory nodes, where each node contains a processor, shared-memory module, cache, and network interface. Processors

| **Benchmark** | **Processors** | **Invalidation messages (millions)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $Dir_N$ | *Tristate* | *Coarse Vector* | *Gray−hardware* | *Gray−software* | *Home* | $Dir_4$B | $Dir_2$B | $Dir_1$B |
| ocean | 32 | 0.45 | 0.87 | 0.70 | 0.45 | 0.48 | 1.09 | 0.45 | 0.45 | 4.40 |
| | 64 | 1.86 | 5.44 | 6.00 | 1.93 | 1.93 | 13.63 | 1.86 | 1.93 | 55.02 |
| | 128 | 7.31 | 19.97 | 28.91 | 7.51 | 7.50 | 53.53 | 7.31 | 8.03 | 314.65 |
| appbt | 32 | 0.72 | 1.01 | 1.10 | 1.07 | 0.96 | 1.93 | 0.72 | 0.72 | 3.82 |
| | 64 | 1.02 | 1.53 | 2.77 | 1.67 | 1.30 | 4.99 | 1.02 | 2.09 | 15.11 |
| | 128 | 1.94 | 3.61 | 9.77 | 3.47 | 2.52 | 18.49 | 1.94 | 9.54 | 78.85 |
| barnes | 32 | 0.49 | 1.25 | 0.83 | 1.19 | 1.21 | 1.81 | 0.98 | 1.80 | 2.92 |
| | 64 | 0.68 | 2.46 | 1.69 | 2.28 | 2.33 | 3.72 | 2.44 | 4.34 | 6.52 |
| | 128 | 0.95 | 4.76 | 3.14 | 4.41 | 4.41 | 7.49 | 6.01 | 9.90 | 14.16 |

Table 3: Total invalidation messages

This table lists the raw invalidation message count (in millions) for all the protocols studied in this paper. Minor differences in invalidation messages between *Gray-hardware* and *Gray-software* for `ocean` and `barnes` are due to differences in the number of local invalidations at the processor-memory nodes, which do not generate invalidation messages.

execute SPARC binaries. Memory locations other than stack references and instructions are cached in a node's cache (256 KB, 4-way set-associative, 32-byte blocks). A cache miss invokes a coherence protocol that sends messages, accesses a directory entry etc.

We assume the network supports only point-to-point messages—i.e., *there is no special support for broadcasts or multicasts.* Network topology is ignored and all messages are assumed a fixed latency of 100 processor cycles. Finally, our protocol implementations assume that a directory entry logically keeps a count of the outstanding copies of a block, a processor always notifies the directory when it replaces a block (called *notifying* [6]), and only positive acknowledgements are collected at an invalidation event.

### 3.4 Evaluation Metric

The ultimate measure of performance is total program execution time. The Wisconsin Wind Tunnel allows us to calculate total program execution time for a 100-cycle network that ignores contention. The latency seen by a message in an real network, however, depends on the network topology, link capacity, and message contention encountered while the message traverses the network. We distrust the execution time results of the Wisconsin Wind Tunnel for this study, because our results show that $Dir_1$B can send 40 times the number of invalidation messages as $Dir_N$. Since invalidation messages come in bursts, they may encounter considerable contention that affects execution time, but is not modeled by this version of the Wisconsin Wind Tunnel.

For this reason, this paper will compare protocols using the number of invalidation messages sent divided by the number sent by $Dir_N$—a metric not affected by contention. A further benefit of this metric is that it focuses on exactly the place where the protocols differ, much like miss ratio highlights how caches differ even when a program's execution time is the bottom line. Finally, this metric does not tie results to specific assumptions for network topology and link capacity.

## 4 Results

This section discusses the results for $Dir_1$B, $Dir_N$, *Tristate*, *Gray-hardware*, *Gray-software*, *Home*, *Coarse Vector*, $Dir_2$B and $Dir_4$B. Table 3 gives raw invalidation message counts for most runs presented in this section. Figure 3 is an example of a graph triple we will use several times. The horizontal axis shows the number of processors, while the vertical axis shows the total number of invalidation messages with a protocol divided by the total number of invalidations for $Dir_N$.

### 4.1 $Dir_N$ and $Dir_1$B

Table 3 shows that for $Dir_N$ the number of invalidation messages sent grows with the number of processors. For `ocean`, the increase is roughly a factor of four, when we double the number of processors. The first factor of two comes from having near-neighbor sharing of twice as many boundary elements, because columns are now divided between twice as many processors. The second factor of two occurs because using more processors maps less data to each per-processor cache. Data not replaced by finite cache effects must instead be recalled with invalidation messages. When we double the number of processors and halve the cache size—not shown—`ocean`'s invalidations just double. For `appbt`, the number of invalidation messages grows with the number of processors because although the sharing pattern distribution does not change, we have increased number of boundary elements because the same 3D grid is divided into greater number of processors. For `barnes`, the frequency of many dynamic sharers increases with

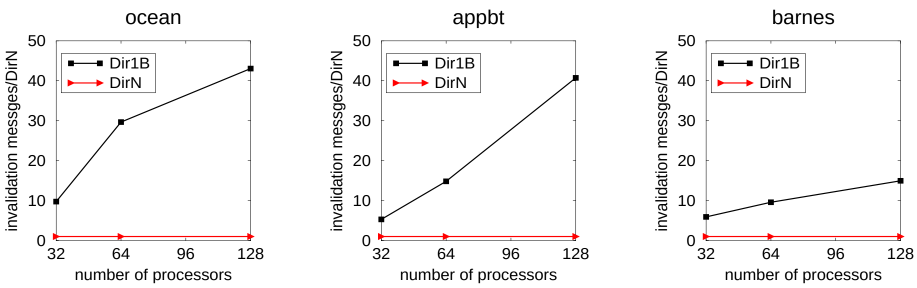


Figure 3: Invalidation messages for $Dir_1$B and $Dir_N$

The vertical axis shows number of invalidation messages for $Dir_1$B and $Dir_N$ relative to $Dir_N$ itself for the same number of processors.

the number of processors, resulting in the increase in the absolute number of invalidation messages for $Dir_N$.

Figure 3 shows that relative to $Dir_N$ the number of invalidation messages for $Dir_1$B increases rapidly with increasing number of processors. For 32 processors, the number of invalidation messages for $Dir_1$B is five to ten times $Dir_N$, while for 128 processors, the invalidation messages blow up to 40 times $Dir_N$ for `ocean` and `appbt`. In `ocean`, sharing is predominantly between two neighboring processors, while in `appbt` it is primarily between a maximum of three processors. Since the number of sharers does not increase with the number of processors, $Dir_1$B sends more invalidation messages than necessary for a greater number of processors. In `barnes`, the frequency of many dynamic sharers increase with the number of processors. As a result, $Dir_1$B sends fewer unnecessary messages relative to $Dir_N$, resulting in a 15-times increase for 128 processors.

## 4.2 Tristate

The question now is—can the multicast protocols get close to $Dir_N$ with much less state? Figure 4 displays the answer. Note that the vertical axis in this figure extends to 10 rather than 50, as in Figure 3. Figure 4 shows that *Tristate* is successful in keeping the invalidation message count closer to $Dir_N$. Unlike $Dir_1$B the number of messages does not grow rapidly with increasing number of processors. For 128 processors, *Tristate* results in less than four and two times the invalidation messages of $Dir_N$ for `ocean` and `appbt`, respectively. Results are relatively good, because these benchmarks have a low degree of sharing for which *Tristate* is optimized. Interestingly, for a dynamic benchmark like `barnes` with a possibility of random sharing patterns which could degrade the performance of *Tristate*, the invalidation messages are within a factor of five more for 128 processors. It appears that sharing in `barnes` is not completely random in practice, and that the sharers are largely consecutive processors.

## 4.3 Gray-hardware, Gray-software, and Home

*Gray-hardware* improves upon *Tristate* when neighboring processors are involved in sharing (Section 2.2). This effect is predominant in `ocean` (Figure 4), where two consecutive processors share a column (Section 3). *Gray-hardware* reduces the number of messages sent by *Tristate* by a factor of two to three and is almost identical to the number of messages sent by $Dir_N$. For `appbt`, sharing is between neighboring processors in three dimensions (Section 3). Since *Gray-hardware* is targeted towards sharing in one dimension, it does not show any spectacular improvement over *Tristate* in this case. The improvement is about 4% over *Tristate* for 128 processors. In `barnes`, we have two effects - (a) the frequency of many sharers grows with the number of processors, and (b) the sharing pattern is dynamic. These imply that the sharers might not always be neighboring processors. Figure 4 shows that the improvement is roughly 8% for 128 processors.

*Gray-software* sends almost the same or fewer invalidation messages than *Gray-hardware*. Thus, the extra hardware for gray coding and taking its inverse can be eliminated. For `ocean`, *Gray-software* is almost identical to *Gray-hardware* because both the protocols use one-dimensional gray coding. The results are more interesting for `appbt`, where three-dimensional gray coding is achieved in software, which exploits the 3D near-neighbor sharing pattern of the benchmark. Here for 128 processors, *Gray-hardware* sends 79% more invalidation messages than $Dir_N$. *Gray-software* closes almost two-thirds of this gap to use only 30% more invalidation messages than $Dir_N$. For `barnes`, there was no direct way to

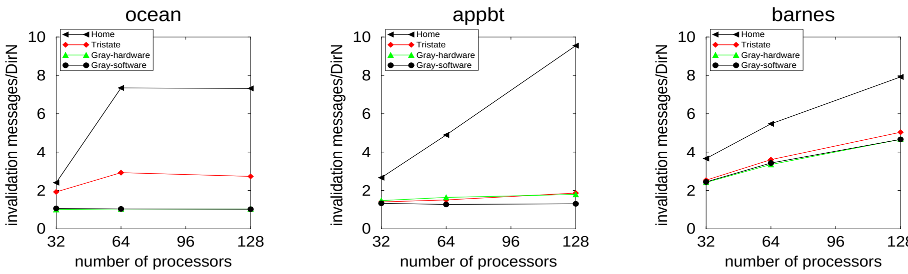


Figure 4: Invalidation messages for *Home*, *Tristate*, *Gray-hardware*, and *Gray-software*

The vertical axis shows number of invalidation messages for *Home*, *Tristate*, *Gray-hardware*, and *Gray-software*, relative to $Dir_N$ for the same number of processors. Note the change in the vertical axis from 50 in Figure 3 to 10 in this figure. We had used a scale of 50 in Figure 3 to accommodate $Dir_1$B within the graph.

determine the sharing pattern. Hence, we chose to use *Gray-software* in one dimension. The results are similar to `ocean`, in that there is almost no difference in the invalidation messages with *Gray-hardware*.

*Home* uses the same number of bits for the sharing code as in $Dir_1$B by using the home node number of a block as its reference number to do the encoding. *Home* can perform as well as *Tristate* or *Gray-software* if data is placed so that the home node is one of the sharers, but *Home* will perform worse otherwise. Since we did not control data placement, Figure 4 displays the latter case. Results show *Home* should not be used when data placement is not controlled.

### 4.4 Coarse Vector, $Dir_2$B, and $Dir_4$B

Figure 5 displays the results for *Coarse Vector*, $Dir_2$B, and $Dir_4$B versus the just-discussed *Gray-software*. To be fair, we use the same number of bits for the sharing code of *Coarse Vector* as in *Gray-software*—$2 \times logN$. For regular applications with well-defined sharing patterns and low number of sharers like `ocean` and `appbt`, *Coarse Vector* is worse than *Gray-software* (Figure 5), and the difference grows with increasing number of processors. For 128 processors, the deterioration is around a factor of four for these benchmarks. However, for dynamic sharing patterns like in `barnes`, with a large number of sharers, *Coarse Vector* shows a slower degradation rate with increasing number of processors, and is consistently better than *Gray-software* (Figure 5). We found that for `barnes` (not shown), *Coarse Vector* is worse than *Gray-software* when the number of sharers equals two. But it becomes progressively better than *Gray-software* as the number of sharers increase. Even though *Coarse Vector* does better than *Gray-software* for `barnes`, both perform much worse than $Dir_N$ due to the high degree of sharing.

The accuracy of *Coarse Vector* in tracking the number of actual sharers increases with increasing number of bits for the sharing code. Decreasing the number of bits for the sharing code (and hence increasing the number of processors per bit), results in increasing number of invalidation messages, as shown by Figure 6. The horizontal axis shows the number of bits devoted to the sharing code for *Coarse Vector*, while the vertical axis shows the corresponding number of invalidation messages relative to $Dir_N$. Increasing the number of bits for the sharing code from 12 to 28 bits cuts down the number of invalidations by a factor of 1.6–1.9 for the three benchmarks.

Finally, we compare *Coarse Vector* and *Gray-software* with $Dir_2$B and $Dir_4$B (Figure 5). The results confirm the fact that the broadcast protocols become unstable when the number of sharers exceeds the number of explicit pointers maintained by these protocols. For `ocean`, the number of sharers is predominantly two. Both $Dir_2$B and $Dir_4$B successfully capture this. For `appbt`, $Dir_2$B results in a factor of 4.9 increase in invalidation messages over $Dir_N$, while $Dir_4$B is identical to $Dir_N$. This is because the number of sharers sometimes goes beyond two but stays below five almost all the time. For `barnes`, both $Dir_2$B and $Dir_4$B are worse than the two multicast protocols because number of sharers can exceed four.

## 5 Conclusion

This paper considers alternative directory protocols for providing cache coherence in medium-scale shared-memory multiprocessors. The protocols we compare differ primarily in their *sharing code*. $Dir_1$B uses a sharing code of only $logN$ bits, but must send *invalidation messages* to all $N$ processors. At the other extreme, $Dir_N$ uses an $N$-bit vector to encode exactly who has the data to avoid sending unnecessary invalidation messages. The goal of other

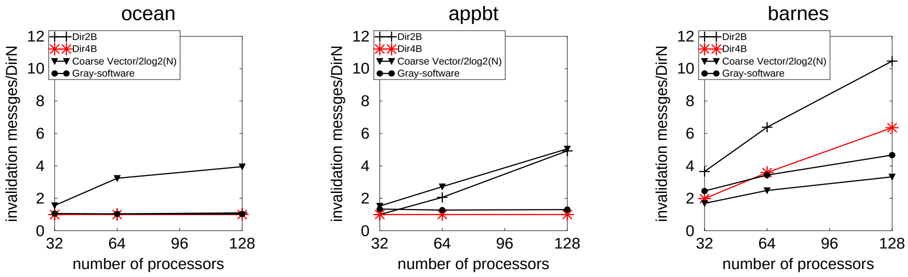


Figure 5: Invalidation messages for $Dir_2$B, $Dir_4$B, *Coarse Vector*, and *Gray-software*.

The vertical axis shows number of invalidation messages for $Dir_2$B, $Dir_4$B, *Coarse Vector*, and *Gray-software*, relative to $Dir_N$ for the same number of processors.

protocols—$Dir_i$B, $1 < i < N$, *Coarse Vector*, and *Tristate* (also called *superset*)—are to use a sharing code near the size of $Dir_1$B's, but still send few unnecessary invalidation messages, like $Dir_N$.

To optimize for near-neighbor sharing, we propose three new protocols that are optimizations to *Tristate*. *Gray-hardware* enumerates processors with a binary-reflected gray code so that neighboring processors in one-dimenision differ by only one bit. *Gray-software* pushes the gray coding into software so that multi-dimensional near-neighbor sharing can be accommodated with *Tristate* hardware. Finally, *Home* cuts the sharing code size down to $logN$ bits (from *Tristate*'s $2 \times logN$) at the expense of more unnecessary invalidation messages in the absence of careful data placement.

We gathered results for three benchmarks—`ocean` (one-dimensional sharing), `appbt` (three-dimensional sharing), and `barnes` (dynamic sharing)—using the Wisconsin Wind Tunnel to simulate 32-, 64-, and 128-processor systems. We assume notifying protocols and no special network support for broadcasts or multicasts. We measure performance using the total number of invalidation messages rather than total execution time to focus on how the protocols differ to avoid having to vary network topology and link capacity assumptions.

Results for 128 processors, for example, show $Dir_1$B sends 43 (`ocean`), 40 (`appbt`), and 15 (`barnes`) times as many invalidation messages as $Dir_N$, providing a large window of opportunity for the other protocols. *Tristate* exploits much of this opportunity by sending 2.7, 1.9, and 5 times as many invalidation message as $Dir_N$. It appears *Tristate* performs better here than it did for Gupta et al. [9], because sharing in our benchmarks was less random than in their synthetic one.

Of the closely-related protocols of *Tristate*, *Gray-hardware*, *Gray-software*, and *Home*, we recommend *Gray-software*. *Gray-software* performs as well or better as the others in all cases, requires the same hardware as *Tristate*, and is not as sensitive to data placement as *Home*. For 128 processors, *Gray-software* sends the same number of invalidation messages as $Dir_N$ for `ocean`, and 1.3 and 4.7 as many invalidation message as $Dir_N$ for `appbt` and `barnes`, respectively.

*Coarse Vector* performs worse than *Gray-software* for `ocean` and `appbt` that have few dynamic sharers, but better for `barnes` that more frequently has many dynamic sharers. This "more stable" behavior of *Coarse Vector* occurs, because it never sends more than $(K-1) \times i$ unnecessary invalidation messages for $i$ sharers with each bit representing $K$ processors. Not surprisingly, $Dir_i$B is less stable than both *Coarse Vector* and *Gray-software*, because it sends $N$ messages when there are more than $i$ sharers. This rarely occurs in `ocean`, occurs significantly in `appbt` for $Dir_2$B but not for $Dir_4$B, and occurs significantly in `barnes` for both $Dir_2$B and $Dir_4$B.

Thus, the choice of protocol between $Dir_i$B, *Coarse Vector*, and *Gray-software* will depend on whether one wants to optimize for few sharers ($Dir_i$B), many sharers (*Coarse Vector*), or hedge one's bets between both alternatives (*Gray-software*).

The scope of any experimental study is finite. Our study compares directory protocols that have very similar implementations. Specifically, we examined implementations that differ primarily in how the sharing code is encoded. We chose to exclude protocols that use traps [5, 11], distributed directories [10], directory caching [9], and several other optimizations [13, 14], because setting the plethora of implementation assumptions needed for these alternatives would have compromized the generality of our study. We did not examine $Dir_i NB$ because it performs poorly without a special mechanism for handling read-only data [17]. Nevertheless, in some situations, the protocols we did not study may perform better than the ones we did study.

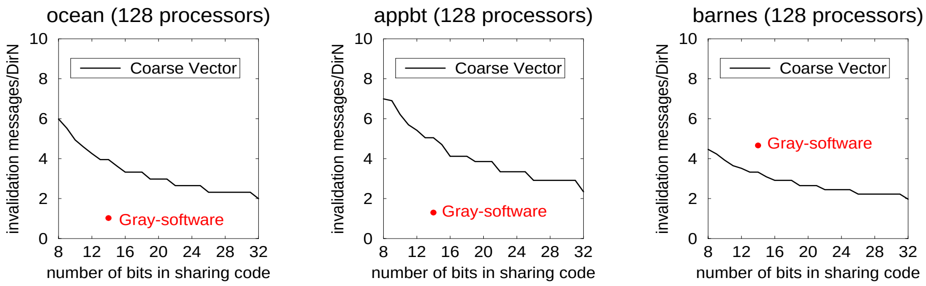


Figure 6: Invalidation messages for *Coarse Vector* and *Gray-software*

This figure shows the number of invalidation messages sent by *Coarse Vector* as the number of bits devoted for the sharing code is increased from 8 to 32 bits for 128 processors. The sharing code of j bits has each bit represent ceiling(128/j) processors. The number of bits used by *Gray-software* is fixed at 14 bits.

# 6 Acknowledgements

Members of the Wisconsin Wind Tunnel group provided invaluable support with the Wind Tunnel, CM-5, and the benchmarks. Singh et al. [16] wrote and distributed the SPLASH benchmarks. Bailey et al. [3] wrote and distributed the sequential version of the NAS benchmarks. Eric Bach and Suresh Chalasani helped with the literature on mesh-to-hypercube mapping. Doug Burger and David Wood helped develop the initial ideas in this paper. Finally, Satish Chandra, Jim Larus, Guri Sohi, Madhusudhan Talluri, and David Wood provided invaluable comments on the initial drafts of this paper.

# References

[1] Anant Agarwal, Richard Simoni, Mark Horowitz, and John Hennessy. An Evaluation of Directory Schemes for Cache Coherence. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 280–289, 1988.

[2] J. Archibald and J.-L. Baer. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. *ACM Transactions on Computer Systems*, 4(4):273–298, 1986.

[3] David Bailey, John Barton, Thomas Lasinski, and Horst Simon. The NAS Parallel Benchmarks. Report RNR-91-002 Revision 2, Ames Research Center, August 1991.

[4] Said Bettayeb, Zevi Miller, and I. Hal Sudborough. Embedding Grids in Hypercubes. *Journal of Computer and System Sciences*, (45):340–366, 1992.

[5] David Chaiken, John Kubiatowics, and Anant Agarwal. LimitLESS Directories: A Scalable Cache Coherence Scheme. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, pages 224–234, April 1991.

[6] David Lars Chaiken. Cache Coherence Protocols for Large-Scale Multiprocessors. Technical Report MIT/LCS/TR-489, MIT Laboratory for Computer Science, September 1990.

[7] Mee-Yee Chan. Dilation-2 Embeddings of Grids Into Hypercubes. In *Proceedings of the 1988 International Conference on Parallel Processing (Vol. III)*, pages 295–298, 1988.

[8] Mee-Yee Chan. Embedding of d-Dimensional Grids Into Optimal Hyerpcubes. In *Proceedings of the First ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 52–56, 1989.

[9] Anoop Gupta, Wolf-Dietrich Weber, and Todd Mowry. Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes. In *Proceedings of the 1990 International Conference on Parallel Processing (Vol. I Architecture)*, pages 312–321, 1990.

[10] David B. Gustavson. The Scalable Coherent Interface and Related Standards Projects. *IEEE Micro*, 12(2):10–22, February 1992.

[11] Mark D. Hill, James R. Larus, Steven K. Reinhardt, and David A. Wood. Cooperative Shared Memory: Software and Hardware for Scalable Multiprocessors. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS V)*, pages 262–273, October 1992.

[12] Kendall Square Research. Kendall Square Research Technical Summary, 1992.

[13] Wisam Michael. A Scalable Coherent Cache System With A Dynamic Pointer Scheme. In *Proceedings of the 1992 International Conference on Supercomputing*, pages 358–367, 1992.

[14] Brian W. O'Krafka and A. Richard Newton. An Empirical Evaluation of Two Memory-Efficient Directory Methods. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 138–147, 1990.

[15] Steven K. Reinhardt, Mark D. Hill, James R. Larus, Alvin R. Lebeck, James C. Lewis, and David A. Wood. The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers. In *Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 48–60, May 1993.

[16] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford Parallel Applications for Shared Memory. *Computer Architecture News*, 20(1):5–44, March 1992.

[17] David A. Wood, Satish Chandra, Babak Falsafi, Mark D. Hill, James R. Larus, Alvin R. Lebeck, James C. Lewis, Shubhendu S. Mukherjee, Subbarao Palacharla, and Steven K. Reinhardt. Mechanisms for Cooperative Shared Memory. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 156–168, May 1993.