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 $\text{Dir}_N$ may be considered too large. We consider $\text{Dir}_B$, $i = 1, 2, 4$, $\text{Dir}_N$, 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) $\text{Dir}_B$ sends 15 to 43 times as many invalidation messages as $\text{Dir}_N$, (b) Gray-software sends 1.0 to 4.7 times as many messages as $\text{Dir}_N$, making it better than Tristate, Gray-hardware, and Home, and (c) the choice between $\text{Dir}_B$, Coarse Vector, and Gray-software depends on whether one wants to optimize for few sharers ($\text{Dir}_B$), 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.

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 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 DirB [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 DirB 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 DirN that uses a bit vector to exactly identify the sharers [1]. DirN 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 DirN, 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—DirB, Tristate, and Coarse Vector. DirB, 1 ≤ i < N, uses i × logN bits to exactly identify up to 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—onebench (one-dimensional sharing), appb3 (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 DirB sends 15 to 43 times as many invalidation messages as DirN. Of the closely-related protocols of Tristate, Gray-hardware, Gray-software, and Home, we find Gray-software per-
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 DirN. 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 Dir2B and Dir4B perform very poorly whenever there are more than two or four sharers (as is to be expected). Thus, the choice of protocol between Dir2B, Coarse Vector, and Gray-software will depend on whether one wants to optimize for few sharers (Dir2B), 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 Dir1B, DirN, Tristate, Gray-hardware, Gray-software, Home, Coarse Vector, Dir2B, and Dir4B, 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

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
<th>Number of bits in sharing code</th>
<th>Invalidation messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirN</td>
<td>maintains precise identity of sharers</td>
<td>N</td>
<td>4</td>
</tr>
<tr>
<td>Dir2B, i &lt; N</td>
<td>broadcasts invalidations for sharers &gt; i</td>
<td>i × logN</td>
<td>N, i &lt; 4</td>
</tr>
<tr>
<td>Tristate</td>
<td>each digit in the sharing code represents states</td>
<td>2 × logN</td>
<td>3 × logN + 1</td>
</tr>
<tr>
<td>Coarse Vector</td>
<td>each bit in sharing code represents K processors, 1 &lt; K &lt; N</td>
<td>ceiling(N/K) †</td>
<td>4 × K</td>
</tr>
<tr>
<td>Gray-hardware</td>
<td>Tristate with gray code in hardware</td>
<td>2 × logN</td>
<td>7</td>
</tr>
<tr>
<td>Gray-software</td>
<td>Tristate in hardware, gray code in software</td>
<td>2 × logN</td>
<td>7</td>
</tr>
<tr>
<td>Home</td>
<td>Gray-hardware with home id as reference id</td>
<td>logN †</td>
<td>7 to N</td>
</tr>
</tbody>
</table>

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 Dir2B, 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 DirN and Dir2B. For Coarse Vector, when K = 2, the number of invalidations in either 2 × K, when each bit covers exactly two processors, or 3 × K, when the four sharers span three consecutive bits in the sharing code. When K > 2, the number of messages in K, if one bit covers all the four sharers, or 2 × K, when the sharers span two consecutive bits. For Tristate, the expression, 3 × 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 \( \frac{1}{4} \). Thus, the expected number of messages sent for four consecutive sharers is \( \frac{1}{4} \times 2^i + (\frac{1}{4} \times 2^i + \frac{3}{4} \times 2^i + ...) \), which sums to 3 × logN + 1, for arbitrarily large N. If finite system size is considered, the sum turns out to be 3 × logN – 2. For Gray-hardware, the expected number of messages for four consecutive sharers is 2i if j mod 4 = 0 and 2i otherwise, which reduces to \( \frac{1}{4} \times 2^i + \frac{3}{4} \times 2^i \). 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, DirN and Dir2B, i ≥ 4, will still send only four invalidation messages. Dir2B with i < 4, Tristate, Gray-hardware, Gray-software, and Home will all send N messages. Coarse Vector will, however, send only 4 × 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.
system using four-processor VAX traces less than two million instructions long. In the same paper, they proposed \textit{Tristate} without evaluating it. The MIT Alewife machine uses a \textit{Dir}_{N}^-\textit{like protocol}, called LimitLESS, which maintains five of the pointers in hardware and the rest in software. Chaiken et al. [5] compared LimitLESS against \textit{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 \textit{Dir}_{N}. Gupta et al. [9] compared \textit{Coarse Vector} with \textit{Tristate} using a synthetic benchmark, which randomly picked the processors sharing a block, and concluded that \textit{Coarse Vector} is superior to \textit{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 \textit{Coarse Vector} could be competitive with \textit{Dir}_{N}. Wood et al. [17] introduced \textit{Dir}_{1}^{SW^+}—a broadcast protocol similar to \textit{Dir}_{1}B that traps like LimitLESS to handle invalidations when number of sharers is greater than one. They compared \textit{Dir}_{1}^{SW^+} against \textit{Dir}_{1}B, \textit{Dir}_1B, and \textit{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 \textit{Dir}_{1}^{SW^+}’s performance is comparable to \textit{Dir}_{N}. However, their results could be biased in favor of \textit{Dir}_{1}^{SW^+} because their simulations did not accurately model network contention.

2 New Multicast Protocols

This section discusses three optimizations of \textit{Tristate—Gray-hardware, Gray-software, and Home}—that can perform better on near-neighbor sharing. The basic idea behind these protocols is to use \textit{gray coding} to reduce the number of \textit{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

\textit{Tristate} performs non-optimally for near-neighbor sharing between consecutive processors along one dimension, because consecutive processor numbers can differ in \(k \gg 1\) 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 \(\frac{1}{2} \times (2 + \frac{1}{2} \times 2^{2} + \frac{1}{4} \times 2^{3} + \ldots + \frac{1}{2^{k-1}} \times 2^{k} + \frac{1}{2^{k}} \times 2^{\log N} = \log N.\) The final term \((\frac{1}{2^{k}} \times 2^{\log N})\) occurs when processor \(N - 1\) also shares with processor zero\(^3\). While \(\log N\) sounds small, here it means that \textit{Tristate} sends five to seven invalidations (per invalidation event) for 32- to 128-processor systems, while \textit{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 \textit{Dir}_{N}. For more consecutive sharers, using a gray code produces results between \textit{Tristate} and \textit{Dir}_{N} (see row \textit{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 \textit{dilation} as the maximum distance between any two mesh neighbors on the hypercube. Alternately, \textit{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

\(^{3}\text{Without this wrap-around, the series sums to } \log N - 1.\)
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 \( O(\text{dimension}) \) [8].

2.2 New Protocols

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

2.2.1 Gray-hardware


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

def 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.

\textit{Gray-hardware} is optimized for near-neighbor sharing between consecutive processors in one dimension. It works like \textit{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

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

\textit{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 \textit{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:

\begin{verbatim}
my_work = get_my_proc_num();
\end{verbatim}

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

\begin{verbatim}
my_work = inverse_graycode(get_my_proc_num());
\end{verbatim}

For \textit{dim} dimensions, use:

\begin{verbatim}
my_work = inverse_multi_graycode(dim, n1, n2, ..., ndim),
\end{verbatim}

where \( n1 \) and \( n\text{dim} \) sizes of each dimension. The inverse functions can be easily provided as library routines.

Changing the mapping from processes to processors—as done by \textit{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.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Brief Description</th>
<th>Input Data Set</th>
<th>Cycles (( \times 10^9 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocean</td>
<td>1D stencil</td>
<td>384 x 384, 2 days</td>
<td>17.6</td>
</tr>
<tr>
<td>appb</td>
<td>3D stencil</td>
<td>32³, 4 iter</td>
<td>77.8</td>
</tr>
<tr>
<td>barnes</td>
<td>8-ary tree</td>
<td>8192 bodies, 4 iter</td>
<td>26.4</td>
</tr>
</tbody>
</table>

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 \( D_{xy} \).
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 Dir7 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 apptb, 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.

Apptb 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 Dir7 SW+ [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
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 DirB can send 40 times the number of invalidation messages as DirN. 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 DirN—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 DirB, DirN, Tristate, Gray-hardware, Gray-software, Home, Coarse Vector, Dir2B and Dir1B. 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 DirN.

#### 4.1 DirN and Dir1B

Table 3 shows that for DirN 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 appb, 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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Processors</th>
<th>Dir1B</th>
<th>DirN</th>
<th>Coarse Vector</th>
<th>Gray-hardware</th>
<th>Gray-software</th>
<th>Home</th>
<th>Dir2B</th>
<th>Dir1B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocean</td>
<td>32</td>
<td>7.46</td>
<td>9.81</td>
<td>6.76</td>
<td>5.25</td>
<td>7.96</td>
<td>5.25</td>
<td>9.81</td>
<td>6.76</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1.96</td>
<td>5.44</td>
<td>6.00</td>
<td>1.93</td>
<td>3.92</td>
<td>4.36</td>
<td>1.93</td>
<td>3.92</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1.80</td>
<td>7.82</td>
<td>28.93</td>
<td>7.51</td>
<td>7.52</td>
<td>52.03</td>
<td>7.51</td>
<td>52.03</td>
</tr>
<tr>
<td>appb</td>
<td>32</td>
<td>5.32</td>
<td>1.04</td>
<td>4.10</td>
<td>2.07</td>
<td>0.96</td>
<td>3.01</td>
<td>0.96</td>
<td>3.01</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1.02</td>
<td>1.03</td>
<td>2.72</td>
<td>1.07</td>
<td>2.36</td>
<td>4.95</td>
<td>1.07</td>
<td>2.36</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1.94</td>
<td>3.62</td>
<td>9.57</td>
<td>3.67</td>
<td>2.52</td>
<td>10.49</td>
<td>3.67</td>
<td>10.49</td>
</tr>
<tr>
<td>barnes</td>
<td>32</td>
<td>6.28</td>
<td>1.26</td>
<td>9.93</td>
<td>1.19</td>
<td>1.31</td>
<td>1.81</td>
<td>1.36</td>
<td>1.81</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>1.04</td>
<td>2.46</td>
<td>3.09</td>
<td>2.28</td>
<td>2.33</td>
<td>3.72</td>
<td>2.33</td>
<td>3.72</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>9.00</td>
<td>4.70</td>
<td>3.14</td>
<td>4.41</td>
<td>4.41</td>
<td>7.40</td>
<td>4.41</td>
<td>7.40</td>
</tr>
</tbody>
</table>

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.
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_B increases rapidly with increasing number of processors. For 32 processors, the number of invalidation messages for Dir_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_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_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_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 70% 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
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 Dir1B 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, Dir2B, and Dir4B

Figure 5 displays the results for Coarse Vector, Dir2B, and Dir4B 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 × log N. 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 Dir1B 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 DirN. 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 Dir2B and Dir4B (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 Dir2B and Dir4B successfully capture this. For appbt, Dir2B results in a factor of 4.9 increase in invalidation messages over DirN, while Dir4B is identical to DirN. This is because the number of sharers sometimes goes beyond two but stays below five almost all the time. For barnes, both Dir2B and Dir4B 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. Dir1B uses a sharing code of only log N bits, but must send invalidation messages to all N processors. At the other extreme, DirN uses an N-bit vector to encode exactly who has the data to avoid sending unnecessary invalidation messages. The goal of other
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-dimension 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 $\log N$ bits (from Tristate’s $2 \times \log N$) 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$_i$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$_B$ 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$_3$B, and occurs significantly in barnes for both Dir$_2$B and Dir$_3$B.

Thus, the choice of protocol between Dir$_B$, Coarse Vector, and Gray-software will depend on whether one wants to optimize for few sharers (Dir$_B$), 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 compromised the generality of our study. We did not examine Dir$_N$B 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


