Segment Directory Enhancing the Limited Directory Cache Coherence Schemes

Jong Hyuk Choi and Kyu Ho Park
Computer Engineering Research Laboratory
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology (KAIST)
373-1, Kusong-Dong, Yusong-Gu, Taejon 305-701, South Korea
{jhchoi,kpark}@core.kaist.ac.kr

Abstract

We present a new arrangement of directory bits called the segment directory to improve directory storage efficiency: a segment directory can point to several sharing processors with almost the same number of bits as the pointer which can point to only one. Many directory overflows can be eliminated by using the segment directory element in place of the pointer in the limited directory schemes. Also, the segment directory can be implemented without introducing additional hardware overhead and protocol complexity. The detailed execution-driven simulations show that the segment directory always does better than the pointer and eliminates many directory overflows by up to 85%. The resulting improvement in bandwidth and execution time is analyzed in detail for limited directory schemes having different behaviors, with respect to the reduced directory overflows.

1. Introduction

Most scalable shared memory multiprocessors (SSMP) of current generation use directory schemes to ensure cache coherence of shared data. A directory block collocated with a main memory block gives the information on which processors have the cached copies of the given memory block and what actions should be taken upon requests to that memory block. Using this information, we can keep multiple caches coherent via point-to-point communications, avoiding costly broadcast as in the snoopy cache coherence schemes [4]. The representative SSMP systems employing the directory cache coherence are the SGI Origin/2000 [13], the Sequent NUMA-Q [15], and the HAL S1 [21] multiprocessor systems from commercial vendors and the Stanford FLASH [12] and the MIT Alewife [2] from academia.

The full map directory scheme [5] is the base directory cache coherence scheme. It is the most intuitive and efficient method of having a full record of the caching status of all N processors for every memory block. It employs one N-bit full map vector per memory block, each bit of which represents the corresponding processor's block caching status. While the full map directory scheme resolves the performance scaling problem for SSMP systems, they introduce another problem - the directory size explosion.

Many research efforts have been made to resolve the directory size scaling problem. The limited directory schemes [3, 10, 6, 19] keep track of cached copies using a small number of pointers. A pointer is an identifier pointing to a sharing processor. When pointers at memory are running short (directory overflow), special actions have to be taken. In the chained directory schemes [11], cache blocks as well as the memory blocks have pointers. Using these pointers, cached copies are maintained as a distributed linked list. While these schemes reduce the width of a directory block, the sparse directory [10], the directory entry combining [19], and the sectored directory [17] reduce the length of the directory. The directory width and length reduction techniques are orthogonal to each other. We can employ both the techniques simultaneously in one directory design.

The previous directory schemes employ either the full map vector or the pointer as the basic building block of the directory construction. This paper introduces a more efficient arrangement of directory bits as a new building block of cache coherence directory. If we can point to more processors with this arrangement than with the pointer, using almost the same number of bits, we can enhance the performance of the pointer-based limited directory schemes. This arrangement is based on the observation that the pointer makes less efficient use of bits than does the full map vector, whereas the pointer-based directory schemes consume less memory than does the full map directory scheme.

We present a detailed description of a new class of directory called the *segment directory* [7] in this paper. The segment directory is a hybrid of the full map vector and the pointer, and introduces new efficient directory configurations between these two. In fact, the full map vector and the pointer are the special cases of the segment directory as well. A segment directory element consists of a segment vector which is a segment of a full map vector, and a segment pointer which determines the position of the segment within the full map vector.

This paper deals with the segment directory in the limited directory schemes. A segment directory element can point to more processors than a pointer with almost the same number of bits. As a result, the segment directory can delay and eliminate many directory overflows which degrade the performance of limited directory schemes. In addition, it can be used in the same way as the pointer, thereby enabling replacement of the pointer in most pointer-based directory schemes.

The performance of the limited directory schemes with the segment directory is analyzed to show its effectiveness. We show how many directory overflows can be eliminated by using the segment directory and by how much the directory overflows affect the performance of different limited directory schemes. Our results show that the segment directory always does better than the pointer in the four of the limited directory schemes: the limited directory scheme with broadcast [3] / without broadcast [3], the coarse vector directory scheme [10], and the LimitLESS directory scheme [6]. Up to 85% of directory overflows are eliminated, reducing bandwidth requirement and accelerating program execution, accordingly.

The rest of the paper is organized as follows. Section 2 compares the existing two basic building blocks of the directory: the full map vector and the pointer, and presents the background for the segment directory. Section 3 presents the structure and the directory storage overhead of the segment directory. Implementation issues of the segment directory are then discussed in Section 4. In Section 5, performance improvement using the segment directory is analyzed, taking the four limited directory schemes as case studies. Section 6 concludes the paper. In the rest of the paper, we assume the write-invalidate protocol [4]; we also assume that the memory block is 32 bytes long.

2. Full Map Vector vs. Pointer

A full map vector and a pointer are the basic building blocks of directories in the previous directory schemes. Figure 1 shows the directory organization of the full map vector and the pointer for a 32 processor system.

A full map vector has N bits representing the caching status of N processors for a memory block. If the *i*-th bit of

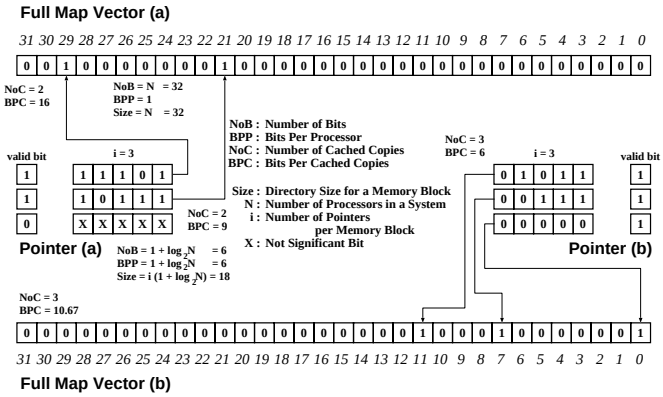


Figure 1. Full Map Vector vs. Pointer.

the full map vector is set, the cache of the *i*-th processor has a copy of the memory block. The full map vector is used in the full map directory scheme [5] where the exact caching status for all memory blocks is maintained.

The directory memory overhead of the full map directory scheme is too large even in moderately large scale multiprocessors. For example, when there are 128 processors, 16 bytes of directory memory are required for every memory block. When the memory block is 32 bytes long, the directory memory size is one half the main memory size, which is prohibitive. Further, the size of a full map vector grows linearly with N.

Previous studies [3, 10] on real parallel workloads have shown that most memory blocks are cached by only a small number of processors at a time. Utilizing this result, the limited directory schemes keep only a few pointers per memory block and provide special mechanisms as soon as more processors than the available pointers cache the memory block. A pointer is an identifier having $(1 + \log_2 N)$ bits[1], pointing to a processor which has a cached copy of the memory block. Because the size of a pointer grows logarithmically with N, the directory memory overhead can be kept small in large scale multiprocessors.

Figure 1 shows two full map vector and pointer directory pairs for a 32 processor system. We consider a limited directory scheme having three pointers per memory block. The directories (a) tell us that two processors, P_{21} and P_{29}, have the cached copy of the memory block in concern. The directories (b) identify three processors caching another memory block: P_0, P_7, and P_{11}. In the pointer directory (a), only two pointers are valid. The valid bit of the last pointer describes that it does not contain a valid proces-

[1]The additional one bit is the valid bit for the pointer, which can be removed by reserving a value for empty pointers. To support maximum power-of-two number of processors, however, the valid bit is necessary. Valid bits for a memory block can also be converted to the number of pointers in use [2].

sor identifier.

Definition 1 *The bits per processor (BPP) is the number of bits required to point to a processor with a directory element. We can obtain the BPP by simply dividing the number of bits (NoB) of a directory element by the maximum number of processors it can identify.*

The BPP is the inverse of the static storage efficiency of a directory element. The BPP of a pointer is $(1 + \log_2 N)$, while the BPP of a full map vector is just one. That is, a pointer requires $(1 + \log_2 N)$ bits to point to a processor, while only one bit is sufficient with the full map vector. Clearly, we can say that the full map vector makes more efficient use of bits than does the pointer. Many bits in the full map vector, however, are not used at a given time.

Definition 2 *The bits per cached copy (BPC) is defined as the NoB of a directory block divided by the number of sharing processors the directory block identifies at a given time.*

The BPC can be thought of as the inverse of the dynamic storage efficiency of a directory block at a time. In Figure 1, the pointer directory (a) has the BPC value of nine, because it points to two processors using 18 bits. The pointer directory (b) has the BPC value of six, because each six-bit pointer identifies a processor. By contrast, the BPC values of the full map vectors (a) and (b) are larger than ten. Even though the full map vector has higher static storage efficiency than does the pointer, the pointer has greater dynamic storage efficiency in general.

What makes it possible to keep the dynamic storage efficiency of the pointer high is that the pointer is a small unit that can be used as an arbitrary number of as desired, whereas the full map vector is a large indivisible one that should be used as a whole.

The pointer-based directory schemes reduce the directory memory overhead by keeping only a small number of pointers with various arrangements and mechanisms. Nevertheless, the reduced directory overhead of the pointer-based directory schemes is not without cost. If many shared reads arrive at memory blocks, many directory overflows will result. These overflows and write requests to these memory blocks can cause a substantial increase in memory access latency and traffic, thereby degrading performance.

3. A New Proposal: Segment Directory

The segment directory is a hybrid of the two directory elements. It is a small and independent unit like the pointer, so that its dynamic storage efficiency is high. At the same time, it has very high static storage efficiency as well. The static storage efficiency of the segment directory approaches that of the full map vector.

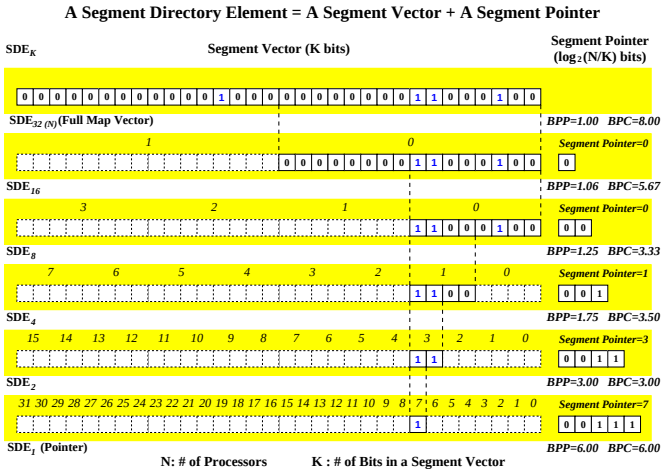


Figure 2. Structure of Segment Directory.

3.1. Structure of Segment Directory

Figure 2 shows the structure of the segment directory elements with different configurations. As shown in the figure, a segment directory element consists of two components: a segment vector and a segment pointer. The segment vector is a K-bit[2] segment of a full map vector; the segment pointer is a $\log_2(N/K)$-bit field keeping the position of the segment vector within the full map vector, aligned in K-bit boundary. A segment directory element with K, denoted as SDE_K, stores the caching status of a segment having consecutive K processors: the segment pointer indicates a segment out of N/K non-overlapping segments; the segment vector is effectively the full map vector for the segment alone.

Taking a system with 32 processors, as for example in Figure 2, the SDE_4 consists of a 4-bit segment vector and a 3-bit $(= \log_2(32/4))$ segment pointer. Thus, the SDE_4, consuming seven bits, can point to a maximum of four processors. The SDE_4 in Figure 2 has a segment pointer value of one. Thus, the SDE_4 stores the caching status of the four processors in the segment 1. These are processors 4, 5, 6, and 7. The segment vector indicates that processors 6 and 7 cache the memory block locally.

Theorem 1 *The segment directory is a unified framework for directory element construction.*

Proof. When K is equal to N, the segment vector becomes the full map vector. The segment pointer is not required, since there is only one possible position of the segment within the full map vector. Therefore, the SDE_N is the full map vector.

[2]We assume that $K = 2^i$ for some integer i, where $1 \leq K \leq N$.

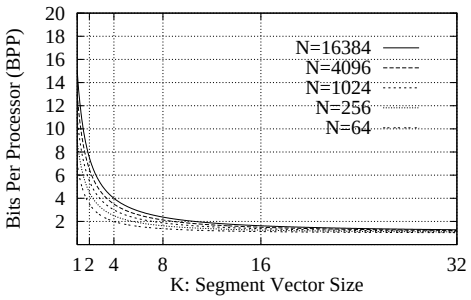


Figure 3. BPP (Bits Per Processor) of Segment Directory.

When K is equal to 1, at the other extreme, the segment vector has only one bit and the segment pointer has $\log_2 N$ bits. The segment pointer indicates the position of the one bit out of N possible positions. The segment pointer is the conventional pointer and the one bit segment vector is the valid bit for the pointer. Therefore, the SDE_1 is the pointer.

As a result, we can conclude that the segment vector includes the full map vector and the pointer within its framework of directory element construction. Because these directory elements are the only directory elements identifying sharing processors, the segment directory can be thought of as a unified approach to directory element construction □.

The number of bits (NoB) required for an SDE_K is represented as

$$NoB\,(SDE_K) = K + \log_2(\frac{N}{K}), \tag{1}$$

where the first term is for the segment vector and the second for the segment pointer. The BPP of the SDE_K is represented as

$$BPP\,(SDE_K) = \frac{NoB\,(SDE_K)}{K} = \frac{K + \log_2 \frac{N}{K}}{K}. \tag{2}$$

Corollary 1 *An SDE_2 consumes the same number of bits as a pointer, while having double the static storage efficiency of the pointer.*

Proof. They occupy the same number of bits, since

$$\begin{aligned} NoB\,(SDE_2) &= 2 + \log_2(\frac{N}{2}) \\ &= 1 + \log_2 N = NoB\,(SDE_1). \end{aligned} \tag{3}$$

An SDE_2 can point to two processors, whereas a pointer can point to only one, Therefore, the static storage efficiency of the SDE_2 is twice as that of the pointer □.

Corollary 2 *The static storage efficiency of the segment directory is monotonically increasing with increasing K.*

Proof. The BPP(SDE_K) is always smaller than the BPP(SDE_{K-1}), since $BPP\,(SDE_K) - BPP\,(SDE_{K-1})$ is always negative for Ks from 2 to N. Therefore, the BPP is monotonically decreasing with increasing K. The corollary holds because the static storage efficiency is the inverse of the BPP □.

Theorem 2 *All non-pointer segment directory elements have higher static storage efficiency than the pointer.*

Proof. It can be easily proved from the induction using the Corollary 1 (induction base) and the the Corollary 2 (induction step) □.

Taking more SDE_Ks as examples: the SDE_4 can point to four processors, consuming only one more bit than does the pointer: $(2 + \log_2 N)$ bits; the SDE_8 can point to eight consuming four more bits than the pointer: $(5 + \log_2 N)$ bits; the SDE_{16} can point to 16 with $(12 + \log_2 N)$ bits.

Figure 3 shows how fast the BPP of the segment directory decreases with increasing K. The BPP of the SDE_K decreases very rapidly and becomes close to that of the full map vector, even with small Ks. Thus, it is sufficient to use the SDEs with small K to increase the static storage efficiency. In Figure 2, the BPP values of the several segment directory elements are shown. The BPP values of the SDE_K with $K \geq 4$ are less than 2, whereas the BPP of the pointer is as high as 6.

Because the SDE with small K is a small unit and it is possible to use an arbitrary number of SDEs, similar to the pointer, the segment directory has high dynamic storage efficiency as well. Figure 2 also shows the BPC values of several segment directory elements. As you can see, the BPC of the segment directory is less than that of the pointer even when only a few bits of the segment vector are set to one. Therefore, its dynamic storage efficiency is usually higher than that of the pointer, which leads to the reduced number of directory overflows in the limited directory schemes.

4. Implementation of Segment Directory

In scalable multiprocessor of current generation, the directory coherence controller is prone to be a system bottleneck [16]. In this section, we show that the segment directory can be implemented without increasing hardware complexity, memory access latency, and directory controller occupancy.

4.1. Translation of Segment Directory Element

The translation logic between the SDE and the processor ID (PID) is different from those of the pointer and the full map vector. Figure 4 shows the translation methods between the PID and three kinds of directory elements.

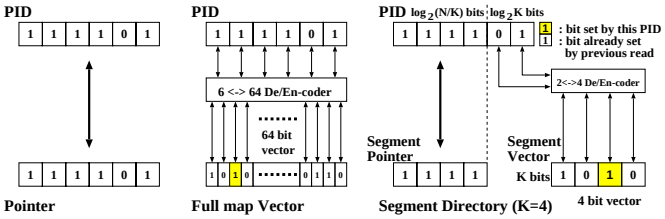


Figure 4. Translation between the Processor ID (PID) and the Three Directory Elements.

Since the pointer keeps the PID, no translation is required between them: direct copy between them suffices. For the full map vector, the bit location set to 1 is the PID. The translation between them requires a decoder (a PID to the full map vector bit) and a priority encoder (the most / least significant active full map vector bit to the PID).

The segment directory requires both the translation methods of the pointer and the full map vector: the segment pointer and the upper $\log_2(N/K)$-bits of the PID are directly copied between each other; a segment vector bit is translated into the lower $\log_2 K$ bits of the PID using a priority encoder and the lower $\log_2 K$ bits of the PID into a segment vector bit using a decoder.

Because the size of the segment vector is much smaller than that of the full map vector, the decoder and the priority encoder of the segment directory are much smaller and faster than those required for the full map vector. Furthermore, the number of bits of the segment vector can be kept constant, whereas the size of the full map vector increases linearly with the number of processors.

The resulting segment directory translation hardware is very simple and scalable: it can be as efficient as that of the pointer; it can be more efficient than that of the full map vector, especially in large scale multiprocessors. The reduced hardware complexity, then, results in the faster translation of the segment directory than the full map vector translation. Therefore, the segment directory can reduce the memory access latency and the directory controller occupancy under the level attained by the full map vector.

4.2. Segment Directory in Limited Directory Schemes

The segment directory can be used as an alternative to the pointer in most pointer-based directory schemes. In this paper, we apply the segment directory to four limited directory schemes. They are the limited directory schemes with broadcast [3] / without broadcast [3], the coarse vector directory scheme [10], and the LimitLESS directory scheme[3] [6]. These directory schemes are identical until directory overflow occurs. They store the PIDs of the processors caching the memory block into the pointers. They take different actions when directory overflow occurs.

The limited broadcast scheme sets the broadcast bit in the directory block when the directory overflow occurs. When a later write request to the memory block arrives, invalidations will be broadcast to all processors in the system. The CV changes the semantics of the directory from the pointers to the coarse vector. The coarse vector is a bit vector, each bit of which points to a region of processors rather than to a processor. The later write to the block causes invalidations to be multicast to all processors in the regions whose corresponding coarse vector bits are set. In these two schemes, once a directory overflow occurs, additional directory overflows are not generated at the memory block until invalidation time.

The limited non-broadcast scheme disallows the directory overflow by making one of the pointers available. It does so by invalidating the cached copy pointed to by the pointer. In the LimitLESS, the directory overflow generates an interrupt to the processor. The interrupt handler flushes the contents of the pointers into the directory data structure in the main memory and makes all the pointers available. The later write to this block also generates an interrupt so that the interrupt handler sends invalidations using the stored directory. In these two schemes, unlike the limited broadcast and the coarse vector schemes, successive overflows will be generated when more read requests come to the overflowed memory block: in limited non-broadcast, more directory overflow will be generated upon every read request after directory overflow; in LimitLESS, successive overflows will be generated when more than i subsequent read requests to the block arrive after each overflow.

The segment directory can be used in these limited directory schemes in place of the pointer with only minor protocol changes. First, when sending invalidations in response to the `exclusive` / `read-exclusive` request [19], a pointer generates only one invalidation. On the other hand, an SDE_K can generate up to K invalidations. An SDE is converted to several PIDs using the translation hardware shown in Figure 4.

Second, to store a new PID into the segment directory in response to the `read` request [19], we should search the memory block's directory for an element with the same segment pointer field. If such a directory element is found, the (PID mod K)-th bit of the corresponding segment vector is set to 1. Otherwise, a new directory element should be

[3]The LimitLESS directory scheme is categorized into the extended directory schemes in some classifications as given in [14]. Because the LimitLESS scheme only extends the directory overflow strategy of the limited directory schemes to the software overflow handler, we classified it as a limited directory scheme, following the classification given in [8].

Table 1. The Directory Configurations under Evaluation ($N = 64$).

Directory Configuration	Number of Bits per Directory Element	Number of Directory Elements	Number of Bits per Directory Block
(4,1 (,r))	7	4	28
(4,2 (,r))	7	4	28
(4,4 (,r))	8	4	32
(5,1 (,r))	7	5	35
(1,16 (,r))	18	1	18
(1,32 (,r))	33	1	33
(2,8 (,r))	11	2	22
(3,8 (,r))	11	3	33
(5,2 (,r))	7	5	35

Table 2. Benchmark Programs.

Program	Description	Data Sets
LU	LU-decomposition of a dense matrix	256x256 matrix
Radix	Radix sort	256K keys radix of 1K
Barnes	3D hierarchical N body simulation	4k bodies
FMM	2D hierarchical N body simulation	4k bodies

allocated, and its corresponding bit is set to 1.

The search for the matching segment pointer can be made in parallel using as many comparators per node as there are segment directory elements in a memory block. Because the number of directory elements for a memory block is usually small in the limited directory schemes, additional hardware is minimal. Through careful design of datapaths within the directory controller, the search delay can be kept minimal. Further, the search can be overlapped with the reply data transmission as well (read-ahead optimization [2]). Thus, it is possible to search the directory elements without increasing the latency of the read requests.

5. Performance Evaluation

In this section, we show the effectiveness of the segment directory quantitatively. The reduction in directory overflows in limited directory schemes and their impact on the system performance are analyzed for an example 64 processor SSMP system.

Table 3. Memory Access Latency (Read) of the Simulated Computer.

Memory Location	Latency (cycles)
Cache Hit	1
Local Memory	39
Remote Memory (SHARED)	107
Remote Memory (DIRTY)	205

In the following discussions, the limited broadcast, the limited non-broadcast, and the LimitLESS directory schemes are denoted by $\text{LimB}_{i,k}$, $\text{LimNB}_{i,k}$, and $\text{LimLESS}_{i,k}$, respectively, where i is the number of directory elements for a memory block and k is the segment vector size. The coarse vector scheme is denoted by $\text{CV}_{i,k,r}$, where r is the size of a region pointed to by a coarse vector bit. The schemes with $k = 1$ are the original limited directory schemes with the pointer (SDE_1), and those with $k \geq 2$ are the schemes with the new segment directories. The full map directory scheme is denoted as FM. When we refer to all four limited directory schemes collectively, we use the $(i, k\,(, r))$ format, where i, k, and r have the same meaning as the above.

Table 1 shows various directory configurations under evaluation. The performance of $(4, 1\,(, r))$, $(4, 2\,(, r))$, $(4, 4\,(, r))$, and $(5, 1\,(, r))$ schemes among them are compared mainly. The $(4, 1\,(, r))$ schemes use four pointers. To improve performance, the $(4, 2\,(, r))$ schemes use four SDE_2s and the $(4, 4\,(, r))$ schemes use four SDE_4s. Alternatively, the $(5, 1\,(, r))$ schemes add one more pointer to each memory block of the $(4, 1\,(, r))$ schemes to improve performance. We will also analyze the directory overflow characteristics for the LimLESS schemes with all the configurations listed in the table including the above four.

Table 1 also shows how many bits a directory block consumes for directory configurations under evaluation. Both the $(4, 1\,(, r))$ schemes and the $(4, 2\,(, r))$ schemes consume 28 bits of directory memory per directory block. In the $(4, 4\,(, r))$ schemes, each directory block consumes 32 bits. Finally, one more pointer of $(5, 1\,(, r))$ makes the directory block occupy 35 bits.

5.1. Simulation Methodology

We used the Tango Lite [9] simulation environment with four programs from the SPLASH-2 [22] benchmark suites to show the effectiveness of the segment directory. Table 2 describes the four benchmark programs used as workloads.

The simulated computer is a 64-node SSMP system interconnected via a mesh wormhole routing network. A simple MSI protocol [19] is used for the LimNB. With some protocol changes, it is also used for the LimB, the CV, and the LimLESS.

It is assumed that the cache is infinite-sized to concentrate on coherence misses and traffic[4]. The cache and the

[4]With respect to the directory block utilization, the infinite-sized cache assumption does not distort the result by much, unless the `replacement-hint` messages [19] are sent to home upon eviction of shared cache lines.

Table 4. Number of Directory Overflows.
% is normalized to the number of overflows of the $(4, 1\ (, r))$ schemes.

Scheme	LU	Radix	Barnes	FMM
$\text{LimNB}_{4,1}$	375,922 (100%)	1,028,326 (100%)	10,512,270 (100%)	14,845,965 (100%)
$\text{LimNB}_{4,2}$	263,099 (70%)	638,649 (62%)	8,865,877 (84%)	13,905,931 (94%)
$\text{LimNB}_{4,4}$	231,068 (61%)	302,258 (29%)	6,670,326 (63%)	11,867,545 (80%)
$\text{LimNB}_{5,1}$	262,409 (70%)	925,314 (90%)	9,400,768 (89%)	14,474,844 (98%)
$\text{LimB}_{4,1} \cdot \text{CV}_{4,1,r}$	15,660 (100%)	15,650 (100%)	33,178 (100%)	51,769 (100%)
$\text{LimB}_{4,2} \cdot \text{CV}_{4,2,r}$	13,774 (88%)	14,880 (95%)	27,768 (84%)	36,544 (71%)
$\text{LimB}_{4,4} \cdot \text{CV}_{4,4,r}$	8,497 (54%)	13,595 (87%)	20,583 (62%)	21,113 (41%)
$\text{LimB}_{5,1} \cdot \text{CV}_{5,1,r}$	15,050 (96%)	15,393 (98%)	25,254 (76%)	39,936 (77%)
$\text{LimLESS}_{4,1}$	16,695 (100%)	127,031 (100%)	79,716 (100%)	93,305 (100%)
$\text{LimLESS}_{4,2}$	15,722 (94%)	77,534 (61%)	69,931 (88%)	67,061 (72%)
$\text{LimLESS}_{4,4}$	9,061 (54%)	40,355 (32%)	53,709 (67%)	38,639 (41%)
$\text{LimLESS}_{5,1}$	15,669 (94%)	100,082 (79%)	57,361 (72%)	65,505 (70%)

memory blocks are 32 bytes in length. A node computer consists of a processor, a cache controller, a directory controller, and a network interface. The node computer is organized based on one in [19], and the interconnection network is modeled based on the Agarwal's network model [1]. The assumed memory consistency model is the weak ordering [18]. Representative memory access latencies of the simulated computer are shown in Table 3.

5.2. Reduction in Directory Overflows

Table 4 shows the reduction in directory overflows. The LimB and the CV have the same directory overflow characteristics. In fact, the LimB may be thought of as the CV with $r = N$. Their actions taken upon and after the directory overflows are the only differences.

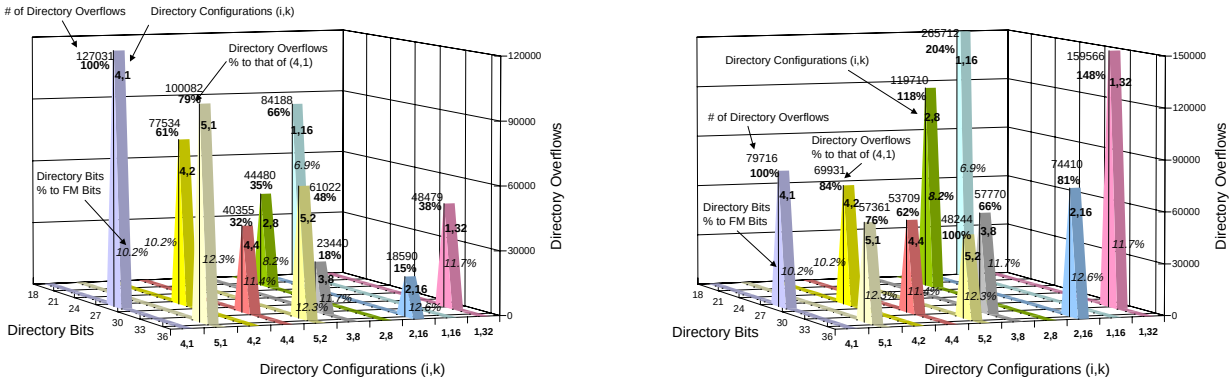
In the $(4, 2\ (, r))$ schemes, up to 39% of the directory overflows are eliminated from those of the $(4, 1\ (, r))$ schemes. Because both the $(4, 1\ (, r))$ and the $(4, 2\ (, r))$ consume 28 bits of directory memory per memory block and the number of directory overflows is reduced in all applications, the dynamic storage efficiency of the directory is improved. The $(4, 4\ (, r))$ schemes consume 4-bit more memory per block than do the $(4, 1\ (, r))$ schemes, while the directory overflows are reduced by up to 71%. In contrast, the $(5, 1\ (, r))$ schemes consume 7-bit more memory per block than do the $(4, 1\ (, r))$ schemes. Reduction in overflows using one more pointer is 30% at the most. The $(4, 4\ (, r))$ schemes have fewer directory overflows than do the $(5, 1\ (, r))$ schemes in all cases. Therefore, using the segment directory is much more effective in reducing directory overflows than is simply adding a pointer.

Table 4 illustrates that the LimNB has the largest number of directory overflows. The miss rate of the LimNB is increased by invalidating cached copies at the directory overflows. The increased miss rate, then, increases the number of directory overflows again. The table also shows that the directory overflows of the LimLESS are more than those of the LimB / CV. This results prove the qualitative description of the limited directory schemes given in Section 4.2.

Figure 5 shows more directory overflow results for all directory configurations listed in Table 1. The reduction in directory overflows of the LimLESS scheme is shown for the Radix and the Barnes. For Radix, a small number of large segment directory elements are more effective than a large number of small segment directory elements. The directory overflows of the $\text{LimLESS}_{2,16}$ are very small compared with those of the conventional $\text{LimLESS}_{4,1}$. More than 85% of directory overflows are eliminated. For Barnes, in contrast, small segment directory elements are more effective. Using many of them is more beneficial than using a small number of large elements. Generally, the reduction in overflows greatly depends on the workload characteristics: if there is a lot special type of locality in the application memory access pattern that could be beneficial to the segment directory, then the segment directory of large K is very effective in reducing overflows; otherwise, the segment directory of small K is more beneficial.

5.3. Performance of Limited Directory Schemes with Segment Directory

Figures 6, 7, and Table 5 show the performance improvement of the four limited directory schemes by the segment directory. The total traffic is divided into request (processor request including `writeback`), reply (`data` / `exclusive reply`, `copyback` / `flush request`, and `copyback` / `flush reply`), and invalidation+acknowledgement (including invalidations done) traffic [10, 19]. The traffic is measured in number of bytes; the



(a) Radix. (b) Barnes.

Figure 5. Directory Overflows of Various LimLESS$_{i,k}$ Directory Configurations.

execution time is measured in number of cycles.

Figure 6 shows the traffic and execution time of the workloads for the LimNB normalized to those of the FM. As shown, the performance of the LimNB is not stable. The poor performance of the LimNB has also been shown in [10, 20, 14]. In fact, this poor performance is a result of the largest directory overflows of all the limited directory schemes. Figure 6 shows by how much the repeated ping-pong sequence of directory overflows and the resulting cache misses can degrade the performance. The FMM runs 39 times slower with the LimNB$_{4,1}$ than with the FM.

As shown in Figure 6, the segment directory reduces the traffic and program execution time for the LimNB. The Radix with the LimNB$_{4,4}$ runs twice as fast as that with the LimNB$_{4,1}$. Still, however, it runs 2.56 times slower than the Radix with the FM. Even though the segment directory reduces traffic and execution time for the LimNB, the LimNB still exhibits low performance.

Figure 7 shows the normalized traffic and execution time of the workloads for the LimB and the CV. We can see that the amount of request and reply traffic for all schemes is essentially the same. As the LimB and the CV do not increase the miss ratio, only the invalidation traffic is increased over that of the FM. In the LimNB, by contrast, the miss ratio is increased as explained above, so that all components of traffic are increased.

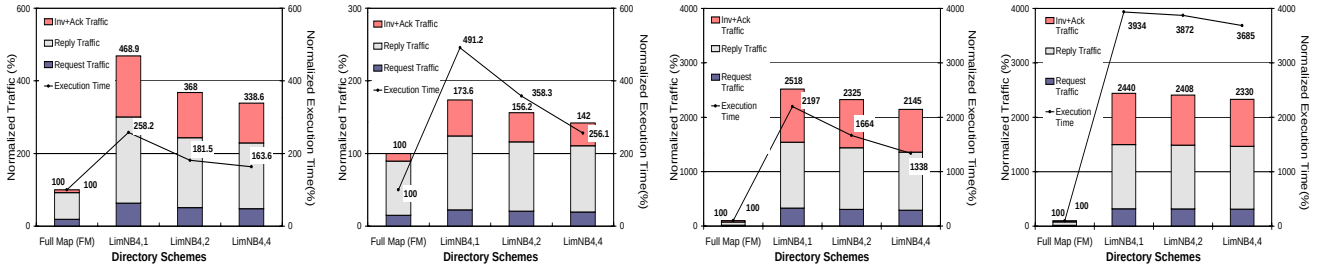
The invalidation distributions of the LU and the Radix show that more than 96% of the invalidating writes produce only one invalidation. Even though there are considerable amount of directory overflows, there are a few invalidating writes to the blocks which experience directory overflows. Thus, the LimB and the CV already exhibit good performance with the pointer. The performance improvement using the segment directory is not large, although the segment directory eliminates up to 46% and 13% of the directory overflows of the LimB$_{4,1}$ / the CV$_{4,1,r}$ for the LU and the Radix, respectively.

The Barnes and the FMM exhibit different characteristics. These applications are characterized by many mostly-read and small migratory memory blocks. Between each computational phases, there are invalidating writes to the memory blocks of mostly-read nature that result in large invalidations. For this type of programs, the performance degradation of the limited directory schemes is not small.

The increase in traffic for the LimB$_{4,1}$ over that for the FM is over 85%, and that in execution time is up to 15.6%. The segment directory reduces traffic by up to 47% and execution time by up to 8%.

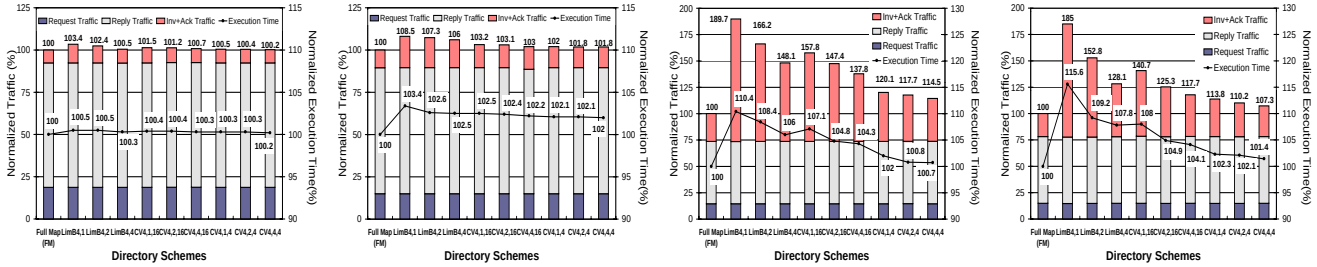
For the CV, we first let the region size r = 4, having the 64-processor system in mind. Because the performance of the CV$_{4,1,4}$ is already close to that of the FM, performance improvement using the segment directory is not substantial. Traffic is reduced by about 6%, and execution time by less than 2%. In a 512-processor system, however, a coarse vector with $r = 4$ consumes 128 bits, or half of the 32-byte memory block. To reduce storage overhead, we must increase the r. The coarse vector with $r = 16$ consumes 32 bits. Performance of the CV$_{4,k,16}$ for the 64-processor system is also shown in Figure 7. Traffic is reduced by as much as 23% and the execution time by up to 4%.

By the above results, the segment directory is proved to improve the performance of the LimB and the CV for applications generating large invalidations such as the Barnes and the FMM. From Figure 7, we can see that the LimB$_{4,4}$ even outperforms the CV$_{4,1,16}$ for both the Barnes and the FMM. With the interconnection networks having less bandwidth, higher contention probability, and higher channel / port occupancy than the simulated one, the performance of



(a) LU (b) Radix (c) Barnes (d) FMM

Figure 6. Performance of LimNB with Segment Directory.



(a) LU (b) Radix (c) Barnes (d) FMM

Figure 7. Performance of LimB and CV with Segment Directory.

Table 5. Normalized Execution Time of the LimLESS with Segment Directory.

Scheme	LU	Radix	Barnes	FMM
$\text{LimLESS}_{4,1}$	111.9	140.1	103.4	109.9
$\text{LimLESS}_{4,2}$	109.9	129.8	103.3	108.2
$\text{LimLESS}_{4,4}$	103.3	124.5	102.1	105.9
$\text{LimLESS}_{5,1}$	109.2	133.0	102.2	107.9

the LimB and the CV would be degraded further. In this case, the segment directory would be more beneficial.

Table 5 shows the performance of the LimLESS with the segment directory. Because the LimLESS simulate the FM by software extension, the traffic is essentially the same as that of the FM. Thus, we only show the execution time results. Because increase in the execution time of the LimLESS comes from the processor interruption and the interrupt handling latency[5] at the directory overflow, execution time of the LimLESS is more related to the number of directory overflows than it is to the number of invalidating writes. Increases in execution time of the LU and the Radix are not small, in contrast to those for the LimB and the CV.

[5]The interrupt handling latency is set based on that of the Alewife Machine [2].

Using the segment directory, the execution time of all the applications except the Radix is reduced under 106%. The large execution time of the Radix comes from the large directory overflows (Table 4). The segment directory speeds up the Radix by as much as 15.6%. The execution time of the LimLESS is made more robust by the segment directory.

Table 5 also shows that it is more effective to use the segment directory than it is to add pointers. $\text{LimLESS}_{5,1}$ consistently shows lower performance than does the $\text{LimLESS}_{4,4}$, although $\text{LimLESS}_{5,1}$ consumes more memory than does the $\text{LimLESS}_{4,4}$.

As a result, the segment directory makes limited directory schemes more competitive with the FM: the performance approaches that of the FM, while at the same time consuming much less memory.

6. Conclusions

This paper presented the segment directory as an improvement to the pointer. The segment directory is a hybrid of the full map vector and the pointer, unifying them within its framework and introducing more efficient directory configurations between them. The segment directory elements can point to more processors than can the pointer, and the element can be as small as the pointer. The segment directory can be used in place of the pointer to more efficiently utilize bits than does the pointer. Also, we have shown that

the segment directory can be implemented with little additional hardware overhead and protocol complexity. The hardware overhead can even be smaller than that for the full map vector. The directory controller for the segment directory can also be implemented simple and fast, so that the memory access latency and the controller occupancy can be kept low with the segment directory.

We applied the segment directory to four limited directory schemes and evaluated the performance improvement. Using the segment directory is proved more effective than adding more pointers. The segment directory reduces directory overflows and improves performance in all four schemes for all the evaluated workloads. Many directory overflows are removed, thereby reducing traffic requirements and memory access latency of the multiprocessors equipped with limited directory schemes. Moreover, the segment directory can be easily applied to limited directory schemes with only minor protocol changes.

In the performance evaluation, we did not modify the benchmarks to take advantage of a specialized form of data locality that may be helpful to the segment directory. Identifying and utilizing those locality of reference for the segment directory is left as a further work.

Acknowledgements

This work is in support of MOST, Korea. Jong Hyuk Choi has been in support of LG Electronics and IBM Research. He thanks K. Ekanadham and P. Pattnaik for their support.

References

[1] A. Agarwal. Limits on interconnection network performance. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):398–412, Oct. 1991.

[2] A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung. The MIT Alewife Machine: Architecture and performance. In *Proc. of the 22nd Annual Int'l Symp. on Computer Architecture (ISCA'95)*, pages 2–13, June 1995.

[3] A. Agarwal, R. Simoni, J. L. Hennessy, and M. Horowitz. An evaluation of directory scheme for cache coherence. In *Proc. of the 15th Annual Int'l Symp. on Computer Architecture (ISCA'88)*, pages 280–289, May 1988.

[4] J. K. Archibald and J.-L. Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Trans. on Computer Systems*, 4(4):273–298, Nov. 1986.

[5] L. Censier and P. Feautrier. A new solution to coherence problems in multicache systems. *IEEE Trans. Comput.*, C-27(12):1112–1118, Dec. 1978.

[6] D. Chaiken, J. Kubiatowicz, and A. Agarwal. LimitLESS directories: A scalable cache coherence scheme. In *Proc. of the 4th Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, pages 224–234, Apr. 1991.

[7] J. H. Choi and K. H. Park. A new efficient arrangement of directory bits for cache coherence. In *Proc. of the 7th Workshop on Scalable Shared-Memory Multiprocessors*, pages 13–14, 1998. http://www.research.ibm.com/people/a/ashwini/ssmm7.html.

[8] D. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware / Software Approach*. Morgan Kaufman Publishers, 1998.

[9] S. Goldschmidt. *Simulation of Multiprocessors: Accuracy and Performance*. PhD thesis, Stanford University, 1993.

[10] A. Gupta, W.-D. Weber, and T. Mowry. Reducing memory and traffic requirements for scalable directory-based cache coherence schemes. In *Proc. of the 1990 Int'l Conf. on Parallel Processing (ICPP'90)*, pages I–312–321, 1990.

[11] D. V. James, A. T. Laundrie, S. Gjessing, and G. S. Sohi. Distributed-directory scheme: Scalable coherent interface. *IEEE Computer*, 23(6):74–77, June 1990.

[12] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. L. Hennessy. The Stanford FLASH multiprocessor. In *Proc. of the 21st Annual Int'l Symp. on Computer Architecture (ISCA'94)*, pages 302–313, Apr. 1994.

[13] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA highly scalable server. In *Proc. of the 24th Annual Int'l Symp. on Computer Architecture (ISCA'97)*, pages 241–251, June 1997.

[14] D. E. Lenoski and W.-D. Weber. *Scalable Shared Memory Multiprocessing*. Morgan Kaufmann Publishers, 1995.

[15] T. Lovett and R. Clapp. STiNG: A CC-NUMA computer system for the commercial marketplace. In *Proc. of the 23rd Annual Int'l Symp. on Computer Architecture (ISCA'96)*, pages 308–317, May 1996.

[16] M. M. Michael, A. K. Nanda, B.-H. Lim, and M. L. Scott. Coherence controller architectures for SMP-based CC-NUMA multiprocessors. In *Proc. of the 24th Annual Int'l Symp. on Computer Architecture (ISCA'97)*, pages 219–228, June 1997.

[17] B. W. O'Krafka and A. R. Newton. An empirical evaluation of two memory-efficient directory methods. In *Proc. of the 17th Annual Int'l Symp. on Computer Architecture (ISCA'90)*, pages 138–147, 1990.

[18] C. Scheurich and M. Dubois. Correct memory operation of cache-based multiprocessors. In *Proc. of the 14th Annual Int'l Symp. on Computer Architecture (ISCA'87)*, pages 234–243, June 1987.

[19] R. T. Simoni. *Cache Coherence Directories for Scalable Multiprocessors*. PhD thesis, Stanford University, 1995.

[20] W.-D. Weber. *Scalable Directories for Cache-Coherent Shared-Memory Multiprocessors*. PhD thesis, Stanford University, 1993.

[21] W.-D. Weber, S. Gold, P. Helland, T. Shimizu, T. Wicki, and W. Wilcke. The mercury interconnect architecture: A cost-effective infrastructure for high-performance servers. In *Proc. of the 24th Annual Int'l Symp. on Computer Architecture (ISCA'97)*, pages 98–107, 1997.

[22] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. of the 22nd Annual Int'l Symp. on Computer Architecture (ISCA'95)*, pages 24–36, 1995.