Journal of Systems Engineering and Electronics
Vol. 25, No. 1, February 2014, pp.158–167

EXCLUSIVELY AVAILABLE VIA
IEEE Xplore® DIGITAL LIBRARY

Elastic pointer directory organization for scalable shared memory multiprocessors

Yuhang Liu[1,2,*], Mingfa Zhu[1,2], and Limin Xiao[1,2]

1. Laboratory of Software Development Environment, Beihang University, Beijing 100191, China;
2. School of Computer Science and Engineering, Beihang University, Beijing 100191, China

Abstract: In the field of supercomputing, one key issue for scalable shared-memory multiprocessors is the design of the directory which denotes the sharing state for a cache block. A good directory design intends to achieve three key attributes: reasonable memory overhead, sharer position precision and implementation complexity. However, researchers often face the problem that gaining one attribute may result in losing another. The paper proposes an elastic pointer directory (EPD) structure based on the analysis of shared-memory applications, taking the fact that the number of sharers for each directory entry is typically small. Analysis results show that for 4 096 nodes, the ratio of memory overhead to the full-map directory is 2.7%. Theoretical analysis and cycle-accurate execution-driven simulations on a 16 and 64-node cache coherence non uniform memory access (CC-NUMA) multiprocessor show that the corresponding pointer overflow probability is reduced significantly. The performance is observed to be better than that of a limited pointers directory and almost identical to the full-map directory, except for the slight implementation complexity. Using the directory cache to explore directory access locality is also studied. The experimental result shows that this is a promising approach to be used in the state-of-the-art high performance computing domain.

Keywords: directory, scalability, memory overhead, positioning precision, overflow, cache coherence non uniform memory access (CC-NUMA).

DOI: 10.1109/JSEE.2014.00019

1. Introduction

As a feature of shared-memory multiprocessors, implicit communication enables programmers to use conventional load and store instructions to issue memory requests. This feature provides better programmability than explicit communication primitives for non-shared-memory computers [1]. As applications have become more diverse in the supercomputing field than the past and the number of non-expert users has grown [2], the easiness of programming is becoming increasingly in demand.

In a distributed multi-processor system with a single address space, the average memory access latency can be reduced with high cache hit rates. However, in multiprocessor environments, the caches for shared data lead to the cache coherence issue, due to multiple copies of shared objects. Solving this issue relies on the coherence protocol, which not only determines the validity of the system, but also affects the system performance, since the protocol's execution induces extra traffic.

For a directory-based cache coherence protocol, the data sharing unit is a cache line, typically in 64 Bytes. This is much smaller than the usual 4 KB page size, thus the occurrence of false sharing is reduced. The directory maintains the shared information for each cache line.

Different directory structures have different memory overheads, sharer positioning accuracy, and implementation complexity. The memory overhead affects the scalability of a directory protocol. Positioning precision for sharers has a direct impact on the traffic generated while executing a coherence protocol. The lower precision the scheme with, the more consistent messages will be sent to the nodes which are not real sharers. Therefore, there are more unnecessary premature invalidations. Meanwhile, implementation complexity has a considerable influence on the reliability and cost of the circuit.

When considering the above three aspects, the conventional directory organizations usually fail to satisfy all requirements simultaneously. For example, the full-map directory scheme is difficult to be applied directly into a large-scale system due to the memory overhead. The overflow processing for the limited pointers scheme degrades the performance. The chained directory scheme results in complex hardware design and long memory access

Manuscript received May 08, 2013.
*Corresponding author.
This work was supported by the National Natural Science Foundation of China (61232009; 61370059), the High Technology Research and Development Program of China (863 Program) (2011AA01A205), and the Fund of the State Key Laboratory of Software Development Environment (SKLSDE-2012ZX-06).

time. The multi-level directory scheme involves implementation problem.

Based on the analysis of shared-memory applications, this paper proposes an elastic pointer directory (EPD). Given that in most cases there are few sharers of each directory entry, this proposed directory structure reduces the probability of overflow, compared with the limited pointers directory. Meanwhile, it reduces the cost of "Link" compared with the dynamic pointer allocation, and increases the degree of operation parallelism since most operations occur in the limited pointers area. Due to the EPD's high positioning precision, its performance is better than the limited pointers directory and almost identical to the full-map directory, with few increases in implementation complexity. The memory overhead is almost identical to that of the limited pointers scheme, and it is significantly reduced when compared with full-map.

For emerging multi-core or many-core processors, how to design scalable shared memory architecture remains a question. On one hand, due to memory overhead, there is no hardware support for the cache coherence in both 48-core single-chip cloud computer (SCC) many-core processor [1] and 80-tile TeraFLOPS processor [3]. On the other hand, some famous scientists such as Martin, Hill, and Sorin predicted that on-chip coherence and the programming convenience are necessary [4]. Furthermore, some explorations have been done. For example, a directory organization based on duplicated tags was proposed [5,6], though it has limited scalability and possible long latencies.

As shown in Fig. 1, we developed a 16-way high productivity computer based on Godson 3A chip multiprocessor [7,8], which is composed of four 4-way cache coherence non uniform memory access (CC-NUMA) units rather than one 16-way CC-NUMA unit. Due to the full-map directory of the CPU only directly support 16 sharers, the CC-NUMA scale is confined. The practice of the design is of instructive significance for the contributions of this paper, and the research result is promising to be used in state-of-the-art high performance computers.

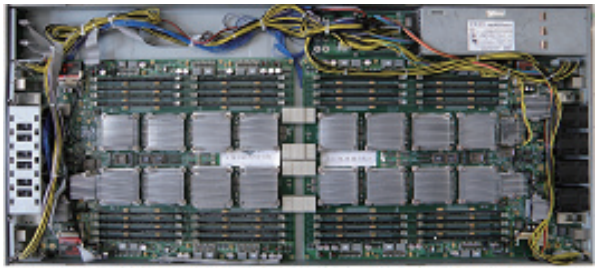


Fig. 1 One rack unit 16-way NUMA high performance computer with four 4-way CC-NUMA unites

The remainder of this paper is organized as follows. In Section 2, a review of related works is presented. The EPD structure and corresponding protocol operations are proposed in Section 3. Analytical evaluation on the memory overhead and the overflow probability of EPD are provided in Section 4. Evaluation via simulation is presented in Section 5 in terms of the application performance improvement. And finally, Section 6 concludes this paper.

2. Related works

In terms of the method of keeping track of the sharers, directory organizations can be divided into following categories: the full-map, limited pointers, chained, coarse vector, one-level hybrid, multi-level, etc. Analyses of memory overhead, position precision, and implementation complexity are presented for each organization, respectively.

Full-map The full-map sharing code (also referred to as bit-vector) uses a presence bit vector to identify the exact sharers of a memory line. Although it is efficient for small-scale systems, its scalability is limited. Its size (in bits) increases linearly with the number of nodes, and therefore, the overall memory overhead is $O(N^2)$.

To reduce the storage requirement, Li et al. [9] proposed a dubbed associative full map directory (*ADir$_p$NB*). Kong et al. [10] also proposed to use physical address mapping on the memory modules to reduce the directory size. However, the corresponding protocols have to be modified and some hardware addition is needed.

Limited pointers The basic idea behind the limited pointers scheme is to use several pointers, each with $\log_2 N$ bits, to encode the unique identities of caches containing the data block, regardless of the scale of the system. However, with only a few pointers in each entry, it is possible to use up the pointers in an entry. This occurs when a read miss occurs and there is no free space remaining in that entry to record the new coming sharer. There are several strategies for handling this situation, such as the broadcast scheme (denoted *Dir$_i$B*), non-broadcast scheme (denoted *Dir$_i$NB*), and super-set scheme. These strategies cause system performance degradation, especially for applications with many simultaneously shared accesses to a memory block.

Chained The idea behind is to distribute directory information across the sharers to simulate the full-map mechanism. In order to get the sharer list, the entire directory link list must be searched which can be the singly or doubly linked list.

The memory overhead of distributed chain directory is $O(N \log_2 N)$. In terms of position precision, it is superior to the limited pointers directory whose performance is impacted by directory overflow.

The disadvantages of the distributed chain directory are as follows. First, it has to maintain a chain for each memory line, causing additional hardware complexity and longer response time. Second, write invalidation has to serially go through the entire sharer list from the list head to the tail, while it can be executed in parallel for the full-map directory. Finally, it requires extra and costly cache capacity, while the full-map can reside in low cost dynamic random access memory (DRAM).

Coarse vector Each bit in the coarse vector directory represents a group of nodes. Due to the coarseness of sharing information, invalidation messages may have to be sent to a group of processors represented by the coarse bit, regardless of whether they are real sharers of the data block. Compared with the full-map scheme, this sharing code scheme reduces memory overhead at the cost of positioning precision of sharers.

Binary tree compressed sharing codes Acacio et al. [11] proposed three compressed sharing schemes. The advantage is that the memory overhead is quite small. The disadvantage is that position precision is lower than the full-map even after improved, and decoding of the compressed sharing code also inevitably introduces some hardware complexities [12].

One-level hybrid If a directory structure is selected among different representations, it is referred to as one-level hybrid structure. For example, the Silicon Graphics (SGI) Origin 2000 and Origin 3000 directory entry can be dynamically interpreted among three formats, i.e. limited pointers, coarse vector, and full-map.

The advantage is that it is possible to exploit the benefits of one more directory structures adapting to different situations. The disadvantage is that extra hardware is required in determining shared status, so as to choose a suitable form to denote it. Once directory state changes, the special hardware is needed for the conversion between the three directory formats. These disadvantages increase the hardware overhead and degrade the system performance, which should be avoided while designing a one-level hybrid directory in future.

Multi-level Acacio [11] and Pan [13] proposed a two-level directory respectively, consisting of a small full-map first-level and a the compressed second-level. Decoding of the compressed sharing code and the transformation between the code and full-map increases the implementation complexity.

Unfortunately, when considering memory overheads, positioning precision for sharers, and implementation complexity, current directory structures shown in Fig. 2 usually fail to satisfy all requirements simultaneously.

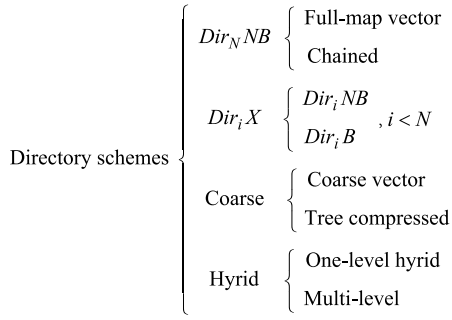
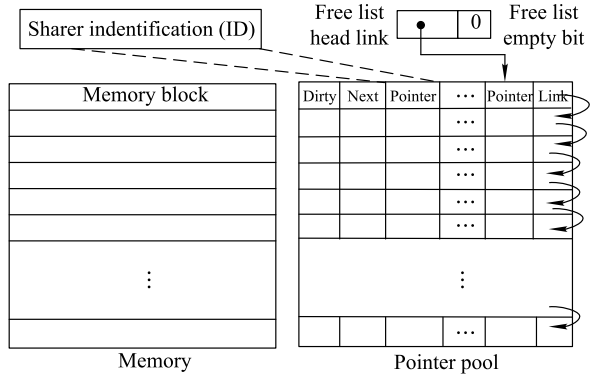


Fig. 2 Directory schemes classification

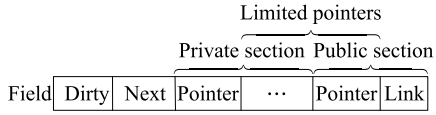
3. The EPD

As in most cases there are few sharers for each data entry, it is not necessary to assign a bit for each potential possible sharer, and a limited pointers-like structure may be enough. However, the overflow problem of the limited pointers scheme should be considered. Due to many pointers in limited pointers directory are not used at the same time, it is possible to organize the public pointer section from each directory entry to constitute a common pool. This is proved effective to reduce the probability of overflow compared with limited pointers.

With the aid of the clue for architecture conceiving and innovating [14] to balance between the comprehensive opposites, an EPD structure is proposed and illustrated in Fig. 3, including a private pointers zone and a public pointers zone. Essentially, the directory is a pointers pool. In terms of memory, the directory is divided into a limited pointers zone and a public pointers zone.



(a) Memory and corresponding EPD



(b) Entry structure of EPD

Fig. 3 EPD

As shown in Fig. 3, each data block corresponds with a line in the EPD structure, beginning with a "Dirty" field

and a "Next" field. The "Next" field, initialized as -1, stores a link that points to a certain line in the directory; the link is allocated when overflowing. Following the "Next" field is the "limited pointers" field, containing a number of pointers (as does the limited pointers directory). Note that a "Pointer/Link" pair fills the rest of the line. The "Link" field in the "Pointer/Link" pair identifies the next "Pointer/Link" pair, which it links to, allowing the pairs to be organized as a singly linked list. All unused pairs make up a list as the free public pointers zone, while a "Free list head link" register is used to store the head pointer of the list. The "Free list empty bit" indicates whether the list is empty or not.

Initially, the EPD structure behaves as a limited pointers directory structure. However, when the limited pointers field in EPD overflows, the free public pointer zone allocates an unused "Pointer/Link" pair (from free list head) to store the new pointer. While new sharers continue coming, EPD uses the Link field to link to the newly allocated "Pointer/Link" pair. After allocating each pair, the free list head register points to the next pair in the public pointer zone (stored in "Next" field of allocated pair as initialized before) in order to maintain the free pair list.

For a write request, the limited pointers field of the line is cleared, except the requesting cache ID. Following this, the pair list is retrieved by the free public pointers zone, saving extra pointers. That is, the retrieved pairs are inserted into the head of the free public pointers list.

Fig. 4 lists the pseudo-code executed at the directory controller for a read miss to a clean block. A new pointer identifying the new requester should be inserted into the corresponding directory entry of the requested block.

```
Procedure insert(Cache) {
// EPD
  if (Next ! = −1){
       ptr = the value in free list head reg;
       free list head reg = ptr->link;
       ptr->pointer = Cache;
       ptr->link = Next;
       Next = ptr;
  }
 else{
       Insert to limited pointers section of this memory line
  }
}
```

Fig. 4 Procedures used in the proposed scheme to implement insert operation

The operation required for a write hit to a clean block is shown in Fig. 5. Invalidations must be sent to all of the sharers indicated by the directory, except for the cache that issued the request.

```
Procedure invalidate(Cache) {
// EPD
// In limited pointers section for this memory line, send
invalidation to Cache indicated by pointer
 for (every pointer entry in the directory line){
    send invalidation to Cache indicated by ptr data
    clear the ptr data
}
//if there are extra pointers in public pointer zone
 if (Next != −1) {
    ptr = Next;
    while (ptr != −1) {
       send invalidation to Cache indicated by ptr
       ptr->pointers = −1;   // clear the ptr data
       // recycle the pointer
       ptr->link = free_list_head;
       free_list_head = ptr;
    }
  }
}
```

Fig. 5 Procedures used in the proposed scheme to implement invalidate operation

4. Evaluation via analysis

Both the memory overhead and positioning precision for sharers are the key factors that determine the scalability of the shared memory system.

4.1 Analysis of memory overhead

Assume that the system scale is N, local shared memory of a single-node is M Bytes(B), size of memory line is L B, bit width of the "Dirty" field is 1, bit widths of "Next" and "Link" are both $\log_2(M/L)$, bit width of "Pointer" is $\log_2 N$, and number of pointers in a directory entry is i. Then, according to these assumptions, total bit width of each directory entry is $1 + 2\log_2(M/L) + i\log_2 N$, the overhead ratio when compared with data is $\dfrac{\left\lceil (1 + 2\log_2 \dfrac{M}{L} + i\log_2 N)/8 \right\rceil}{L} \times 100\%$, and the overhead ratio when compared with full-map is $\dfrac{\left\lceil (1 + 2\log_2 \dfrac{M}{L} + i\log_2 N)/8 \right\rceil}{N/8} \times 100\%$.

Suppose the size of a memory module within a node is 1 GB, in which the number of memory line is 1 GB/64 B = 2^{24} (so "Next" and "Link" fields are set as 24 bits respectively), then the public area initially includes 2^{24} pointers. Suppose the private cache capacity is 64 MB, and the size

of a cache line is 64 B, then the number of elements in the pointer pool is 16 times as many as that of the cache lines within a private cache.

Taking the system scale $N = 4\ 096$ for instance, the format of the directory entry is shown in Fig. 6. That is, the "Dirty" field bit width is 1, "Next" and "Link" field bit widths are both 24, the "Pointer" field bit width is 12, and 5 pointers are configured in an entry, so the width of the entire entry is 109 bits. The percent memory overhead is

$$\frac{\lceil 109/8 \rceil}{64} \times 100\% = 21.8\%.$$

The memory overhead ratio to full-map is

$$\frac{\lceil 109/8 \rceil}{4096/8} \times 100\% = 2.7\%.$$

As shown in Fig. 6, first, the percent memory overhead relative to the data size is increasing linearly with the number of pointers used in a directory entry. However, even when 8 pointers are incorporated in a directory entry, and with large scale $N = 16\ 384$, the percent memory overhead is only 35.9%, which is small compared with that of the full-map scheme. Furthermore, involving even fewer pointers without overflow (analyzed in the next section), the percent memory overhead is more favorable. Finally, the percent memory overhead is not sensitive to system scale N, which means that the directory structure has good scalability in terms of memory overhead.

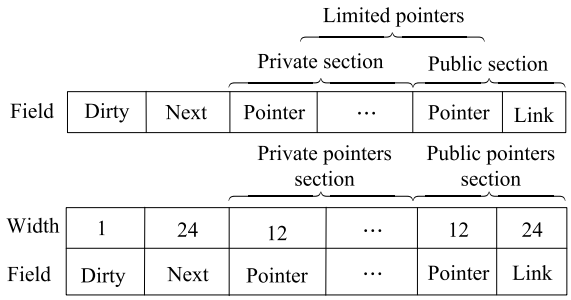


Fig. 6 Instance of an EPD entry

In Fig. 7, the horizontal axis represents the number of pointers in a directory entry. The vertical axis reflects the memory overhead ratio of EPD to full-map. First, the memory overhead ratio is relatively small, and this trend is more evident as the system becomes larger. Second, as the number of pointers in an entry increases, the memory overhead will increase gradually yet slowly. That means, even if the configuration contains more pointers, the memory overhead advantage of EPD relative to full-map remains stable, which is shown in Fig. 8.

The above analysis is on the proposed directory within the memory. Furthermore, in order to reduce the memory overhead and time to access the directory, Ros et al. [15] moved directory information from the main memory to the L2 cache level. The same technique can also be used for the proposed directory, which may further reduce the memory overhead of EPD.

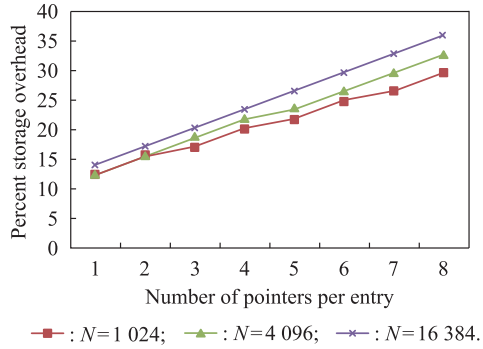


Fig. 7 Percent memory overhead in terms of different number of pointers per entry

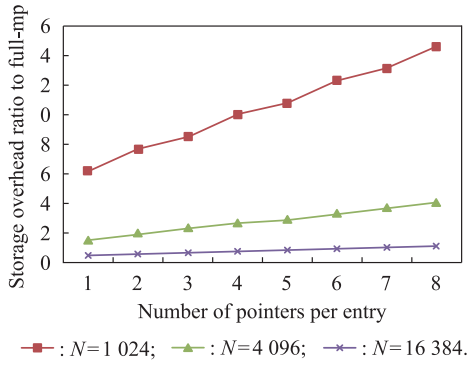


Fig. 8 Memory overhead ratio to full-map in terms of different number of pointers per entry

4.2 Analysis of overflow probability

Given a best-case scenario, data in each private cache are cached in uniform distribution from the memory modules within the CC-NUMA system. In such cases, it is impossible to overflow if the number of elements in the pointer pool is no less than that of the cache lines in the private cache.

However, depending on applications, the cached data from the memory may not be complete in uniform distribution. In a worst-case scenario, all of the private caches are full with data which are cached from a given memory module. The number of pointers needed to avoid overflow is the sum of the number of lines in all the private caches. Such occurrences are rare. Otherwise, the accessed module will be overloaded and become a hotspot, resulting in significant performance degradation. At this time, the discussion on whether directory overflow would occur can be left behind.

Below, the probability that the overflow of pointer pool is close to 0 is demonstrated. Therefore, it is unnecessary

to set the pointer pool as large as the sum of the number of lines in all the private caches in order to deal with the worst-case.

For EPD, event A "pointer pool overflow" is equivalent to "pointers in public pointer area have been used up". M/L is the initial amount of pointers in the public pointer area, and it is also the maximum number. X_j is the number of the current sharers for memory line j. There are i pointers in each directory entry, of which $(i-1)$ is in the limited pointers area and there is one in the public pointer area. The possibility of event A is

$$P(A) = P\{\sum_{j=1}^{M/L} \sigma[X_j - (i-1)] > M/L\} \quad (1)$$

$$\sigma(x) = \begin{cases} x, & x \geqslant 0 \\ 0, & x < 0 \end{cases}. \quad (2)$$

For the limited pointers scheme, there are i pointers in each directory entry, and the expectation of overflow times within a memory module is

$$E = \sum_{j=1}^{M/L} P\{X_j > i\} = \sum_{j=1}^{M/L} P\{X_j - (i-1) > 1\} \quad (3)$$

X_j	0	1	2	3	4	5	6	7
$P/(\%)$	4	50	28	10	5	2	0.6	0.4

(4)

The above is the distribution law for the number of sharers in each directory entry for an application when the system scale is 64. The probability values, which are less than 0.2%, have been omitted.

Values are set to be representative for different applications, according to the experimental results from the simulation.

$$M/L = 1\ \text{GB}/64\ \text{B} = 2^{24}, \quad i = 5$$

The distribution law of $Y_j = \sigma(X_j - 4)$ is

Y_j	0	1	2	3
$P/(\%)$	97	2	0.6	0.4

then

$$P(A) = P\{\sum_{j=1}^{2^{24}} Y_j > 2^{24}\}. \quad (5)$$

2^{24} can be considered as a very large number.

$$P(A) \approx \lim_{n\to\infty} P\{\sum_{j=1}^{n} Y_j > n\} =$$

$$\lim_{n\to\infty} P\{\frac{1}{n}\sum_{j=1}^{n} Y_j > 1\} =$$

$$1 - \lim_{n\to\infty} P\{\frac{1}{n}\sum_{j=1}^{n} Y_j \leqslant 1\} \quad (6)$$

Given $Y_1, Y_2, \ldots, Y_j$ are independent from each other, and have the same mathematical expectation and variance, the expectation is

$$\mu = (1\times 2 + 2\times 0.6 + 3\times 0.4)\times 1\% = 0.044 < 1, \quad (7)$$

and the variance is d. For any small positive real number ε,

$$1 - \frac{d^2/n}{\varepsilon^2} \leqslant P\{\left|\frac{1}{n}\sum_{j=1}^{n} Y_j - \mu\right| \leqslant \varepsilon\} =$$

$$P\{\mu - \varepsilon \leqslant \frac{1}{n}\sum_{j=1}^{n} Y_j \leqslant \mu + \varepsilon\} \leqslant$$

$$P\{\frac{1}{n}\sum_{j=1}^{n} Y_j \leqslant \mu + \varepsilon\} \leqslant P\{\frac{1}{n}\sum_{j=1}^{n} Y_j \leqslant 1\} \leqslant 1. \quad (8)$$

If $n \to \infty$ for the above formula,

$$\lim_{n\to\infty} P\{\frac{1}{n}\sum_{j=1}^{n} Y_j \leqslant 1\} = 1, \quad (9)$$

so we can conclude that

$$P(A) \approx 1 - \lim_{n\to\infty} P\{\frac{1}{n}\sum_{j=1}^{n} Y_j \leqslant 1\} = 0. \quad (10)$$

The expectation of the number of the overflow for limited pointers directory is

$$E = \sum_{j=1}^{M/L} P\{X_j > i\} = 2^{24}\times 1\%. \quad (11)$$

Therefore, $P(A) << E$. Results are explained as follows: the overflow probability for each directory entry is low ($0 \sim 10\%$); when the overflow of a number of directory entries occurs simultaneously, the probability is even smaller; furthermore, many directory entries overflow simultaneously and run out of the public pointer area, which consists of M/L pointers, this case rarely occurs. In sum, $P(A) \approx 0$, hence the proposed EPD is elastic and of little overflow.

5. Evaluation via simulation

With the respective analysis of memory overhead and overflow probability in the above two sections, the evaluation of the performance through simulation is given as follows.

5.1 Simulation environment and results

Simulation technology can be used to compensate for the mathematical analysis for cycle-to-cycle application running [16]. We have modified execution-driven Rice simulator for ILP multiprocessors (RSIM) simulator for

the present study in order to model the target system [17]. The modeled system has a CC-NUMA with 16 or 64 nodes. These nodes are interconnected with a two-dimensional (2D) wormhole routing mesh and an invalidation-based mod-shared-invalid directory cache-coherent protocol is implemented. The nodes also contain features such as out-of-order (OoO), superscalar, multi-stage pipeline, and multi-level cache hierarchy.

The architecture parameters of the target system are shown in Table 1, the values of which have been chosen to be similar with the parameters of emerging shared memory multiprocessors.

Table 1 System architecture parameters

16 or 64-Node system	
ILP processor	
Processor speed	2.2 GHz
Max fetch/retire rate	4
Instruction window	64
Cache Parameters	
Cache line size	64 B
L1 cache write through (WT)	Direct mapped, 64 KB
L1 request ports	2
L1 access latency	1 cycles
L2 cache write back (WB)	4-way associative, 1 MB
L2 request ports	1
L2 tag access latency	3 cycles
L2 data access latency	5 cycles
Number of miss status holding registers (MSHRs)	8 per cache
Directory parameters	
Directory access time	21 cycles
First coherence message creation time	12 cycles
Next coherence message creation time	6 cycles
Memory Parameters	
Memory access time	18 cycles
Memory interleaving	4
Internal Bus Parameters	
Bus speed	3 cycles
Bus width	32 B
Interconnection Network Parameters	
Topology	2D mesh
Flit size	8 B
Flit delay at network switches	4 cycles
Arbitration delay at network multiplexers	4 cycles

Table 2 summarizes the applications used in this study to evaluate the benefits of its proposals. Several scientific applications covering a variety of computation and communication patterns have been selected. Barnes indicates an implementation of the Barnes-Hut N-Body algorithm, solving a problem in galaxy evolution. *fft* represents the fast Fourier transformation, which is a key kernel of applications that use spectral methods. *lu* indicates an LU factorization of a dense matrix and is representative of many dense linear algebra computations. *ocean* simulates the influence of eddy and boundary currents on large-scale flow in the ocean. *radix* is a radix sort program and *water* is a molecular dynamics simulation of water. These are from the SPLASH-2 benchmark suite [18]. *em3d* is a shared memory implementation of the Split-C benchmark [19]. Wind tunnel simulation application *mp3d* is drawn from the SPLASH suite. Finally, "unstructured" represents a type of fluid dynamics applications.

Table 2 Benchmarks and input sizes

Benchmark	Input
barnes	16 384 bodies, 4 time steps
em3d	38 400 nodes, degree 2, 15% remote and 25 time steps
fft	256 K complex doubles
lu	256×256, block 8
mp3d	50 000 nodes, 8 time steps
ocean	258×258 ocean
radix	524 288 keys, 1 024 radix
unstructured	mesh.2k, 5 time steps
water	512 molecules

In terms of execution times of the benchmarks, a performance comparison between the proposed EPD and limited pointers is presented in Fig. 9. The X-coordinate denotes the used benchmarks, and the Y-coordinate indicates performance improvement when using the scheme proposed, as compared with *limited pointers*.

As can be observed from Fig. 9, the performance of the proposed EPD is close to that of full-map, even when only 2 to 3 pointers are configured in each directory entry. While memory overheads are nearly all the same, the EPD has a significant performance improvement over limited pointers, since the number of overflow for the proposed scheme is far less than that of limited pointers. This result is in good agreement with the theoretical analysis presented in Section 4.

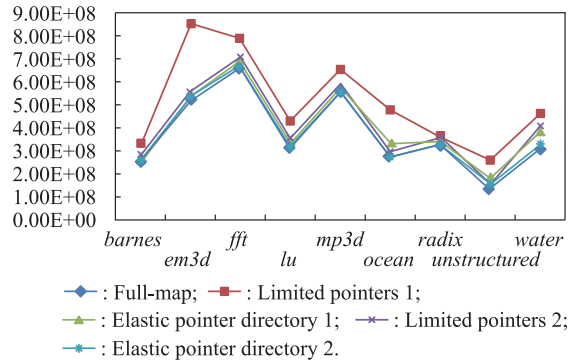


Fig. 9 Performance improvement ratio of EPD compared to limited pointers and full-map (for limited pointers i, i is the number of pointers in each directory entry)

Furthermore, the application performance is directly related to broadcast frequency and coherence delay. Broadcast frequency that affects memory stall time, depends on several factors: the distribution of the number of sharers in the directory entry for a given application, the number of simultaneous directory entry overflows, application input size, number of processors, etc. Coherence delay is

mainly associated with interconnection network and directory controllers.

5.2 Review from more general perspectives

In the following, the scalablity will be analyzed from a more general point of view, that is from application and architecture perspectives.

From the architecture perspective, the efficient use of directory constitutes the first basis of the scalability of CC-NUMA in terms of real performance. K denotes the number of network transactions on the critical path of operations when the invalidation event occurs. The value of K depends on the directory organization and the real number of sharers for a given application. The cycles per instruction (CPI) [20] in the CC-NUMA architecture is evaluated for different K values. For random sharing mode application, CPI is sensitive to the values of K, so the directory organization proposed with K being 1 outperforms the link-list that with K being more than 1.

From the architecture perspective, the innterconnection network architecture is also an impacting factor, but with less sensitivity compared with the directory architectue where the value of K is small.

From the application perspective, the efficient use of on-chip cache constitutes the second basis of the CC-NUMA scalability in terms of the real performance.

In the following, we give more discussions on the efficient use of directory which is the most important factor for scalablity. The basic idea is to intergrat a cache to accelerate the access of directory.

As shown in Fig. 10, based on the proposed partly shared directory, a directory cache is proposed, which is capable of forming a two-level directory architecture combing of EPD and directory cache for exploring directory access locality.

The first-level directory consists of a small set of directory entries that are frequently accessed, each one containing a full-map sharer indicator. Essentially, it is a directory cache, stored in static random access memory (SRAM) with small access latency.

In the second level, using the proposed directory sharer indicator, a directory entry is assigned to each memory line. It is stored in synchronous dynamic RAM (SDRAM) with higher access latency but larger capacity compared with SRAM.

When a read or write request is issued, the home node of the corresponding memory line will simultaneously access the two-level directory. The accessed address consists of three parts: tag, set index, and offset. First, set index is used to access the first level to select the proper set. Second, if one of the tags for the indexed set equals the tag in the access address, and valid bit "V" is set, then the "AND" gate outputs a "true" value. That means a first-level access hit has occurred, and the "MUX" selects the hit entry from the first level and outputs it. A "false" output is considered as a miss, and the following operations are conducted.

(i) Since a directory entry is assigned to each memory line, the access to the second level directory uses the whole address as an index, where no misses occur. For the EPD structure, the second level directory entry should be converted into the full-map format, and the "MUX" is used to select from the two- level results to output.

(ii) Assuming a multi-way set associative directory cache with write-back policy is used, one directory cache item is replaced and wrote back is enabled. Note, this will occur if all of the ways in the corresponding set have been used.

(iii) The N-bit full-map vector output by MUX in step (i) is put at the corresponding position of the replaced item in the first level directory.

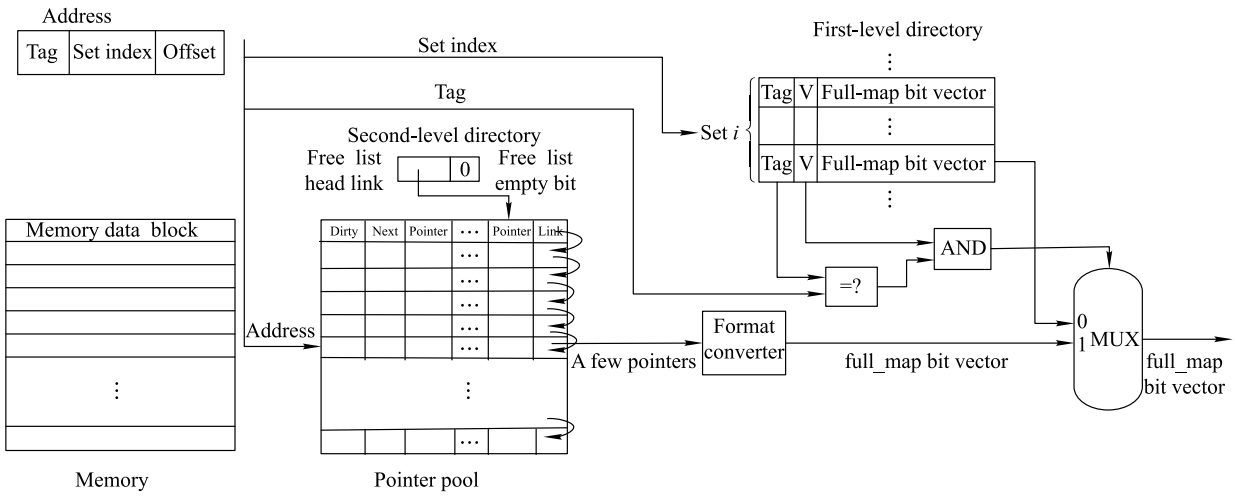


Fig. 10 Two-level directory structure based on the proposed directory organization

The above operations are different with the scheme presented by Pan et al. [12,13]. For the scheme where limited-pointers sharer indicator is used in the second level directory, overflow occurs when the number of sharers is more than the limited number of pointers in the second-level entry. Some memory stall time is needed to resolve this issue. Suppose the hit ratio of the first-level is h ($h \in [0,1]$), the access latencies of the two levels are T_{L1} and T_{L2} respectively. Since the first-level is implemented in the SRAM, hit time and access time are much less than that of the second-level, $T_{L2} = \lambda \cdot T_{L1}$, $\lambda > 1$, $\lambda \in R$. The speedup relative to single-level directory is

$$\text{Speedup} = \frac{T_{L2}}{T_{L1} + (1-h)(T_{L1} + T_{L2})} = \frac{1}{\frac{1}{\lambda} + (1-h)(1 + \frac{1}{\lambda})}. \tag{12}$$

The formula reflects the relationship between the speedup and directory cache hit ratio. One can infer from the formula that while the hit rate is 30% or more, the acceleration effect is at least 14.9%, given the first level is ten times as fast as the second.

6. Conclusions

The scalability of directory mechanisms is a hot research topic, and a core issue that the designers of a large-scale shared-memory multiprocessor must deal with. Based on an analysis of the strengths and weaknesses of the various directory schemes, in terms of memory overhead, position precision, and implementation complexity, the EPD organization is proposed. EPD achieves good performances on all these key aspects simultaneously with quality efficiency. Both the analytical and experimental results show that EPD is a promising technique for the state-of-the-art high performance shared memory multiprocessors.

Acknowledgments

The authors would like to express sincere gratitude to the Supercomputing Research Center of Beihang University and Computer Network Information Center of Chinese Academy of Sciences, for providing quality experiment facilities and giving good comments on the original idea of this work.

References

[1] T. G. Mattson, M. Riepen, T. Lehnig, et al. The 48-core SCC processor: the programmer's view. *Proc. of the ACM/ IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010: 1–11.

[2] P. Kogge, K. Bergman, S. Campbell, et al. *Exascale computing study: technology challenges in achieving exascale systems*. USA: The Defense Advanced Research Projects Agency, 2008.

[3] S. R. Vangal, J. Howard, G. Ruhl, et al. An 80-tile sub-100-w TeraFLOPS processor in 65-nm CMOS. *Journal of Solid State Circuits*, 2008, 43(1): 29–41.

[4] M. M. Martin, M. D. Hill, D. J. Sorin. Why on-chip cache coherence is here to stay. *Communications of the ACM,* 2012, 55(7): 78–89.

[5] A. Ros, M. E. Acacio, J. M. García. A scalable organization for distributed directories. *Journal of Systems Architecture*, 2010, 56(2/3): 77–87.

[6] A. Ros, M. E. Acacio, J. M. García. Scalable directory organization for tiled CMP architectures. *Proc. of the International Conference for Computer Design*, 2008: 112–118.

[7] X. Gao, Y. J. Chen, H. D. Wang, et al. System architecture of Godson-3 multi-core processors. *Journal of Computer Science and Technology*, 2010, 25(2): 181–191.

[8] Y. H. Liu, M. F. Zhu, L. M. Xiao, et al. Design of high productivity computing node based on Godson 3A CPU. *High Performance Computing Technology*, 2010, 207(6): 46–53. (in Chinese)

[9] T. Li, L. K. John. ADirpNB: a cost-effective way to implement full map directory-based cache coherence protocols. *IEEE Trans. on Computers*, 2001, 50(9): 921–934.

[10] J. Kong, P. C. Yew, G. Lee. Minimizing the directory size for large-scale shared-memory multiprocessors. *IEICE Trans. on Information and Systems*, 2005, E88-D(11): 2533–2543.

[11] M. E. Acacio, J. Gonzalez, J. M. Garcia, et al. A two-level directory architecture for highly scalable CC-NUMA multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 2005, 16(1): 67–79.

[12] G. T. Pan. Research on key technologies of CC-NUMA based memory architecture. Changsha: Graduate School of National University of Defense Technology, 2007. (in Chinese)

[13] G. T. Pan, Q. Dou, L. G. Xie. A two-level directory organization solution for CC-NUMA systems. *Algorithms and Architectures for Parallel Processing*, 2007, 4494: 142–152.

[14] Y. H. Liu, M. F. Zhu, J. S. Cui, et al. Multi-objective design methodology for CMP based high density computers. *Systems Engineering and Electronics*, 2012, 34(4): 806–812. (in Chinese)

[15] A. Ros, M. E. Acacio, J. M. García. A novel light weight directory architecture for scalable shared-memory multiprocessors. *Proc. of the 11th International Euro-Par Conference*, 2005: 582–591.

[16] C. J. Hughes, V. S. Pai, P. Ranganathan, et al. RSIM: simulating shared-Memory multiprocessors with ILP processors. *IEEE Computer*, 2002, 35(2): 40–49.

[17] Z. B. Yu, H. Jin, N. H. Zou. Computer architecture software based simulation. *Journal of Software*, 2008, 19(4): 1051–1068. (in Chinese)

[18] I. E. Venetis. The modified SPLASH-2 benchmarks suite. http://www.capsl.udel.edu/splash/

[19] V. K. Pingali, S. A. McKee, W. C. Hsieh, et al. Computation regrouping: restructuring programs for temporal data cache locality. *International Conference on Supercomputing*, 2002: 252–261.

[20] J. L. Hennessy, D. A. Patterson. *Computer architecture: a quantitative approach*. Burlington: Morgan Kaufmann, 2011.

Biographies

Yuhang Liu was born in 1985. He received his combined M.S. and Ph.D. degrees in computer architecture from Beihang University in 2013. He is currently a post-doctor of the Department of Computer Science in the Illinois Institute of Technology. He is a member of China Computer Federation (CCF). His research interests include scalable parallel and distributed processing, memory system, performance evaluation and optimization.
E-mail: liuyuhang@cse.buaa.edu.cn

Mingfa Zhu was born in 1945. He received his Ph.D. degree in computer science from Michigan State University, in 1985. He is currently the president of the Institute of Computer Architecture (ICA) of Beihang University and he is a professor in Beihang University. He is a fellow of Lenovo Group. He is a senior member of CCF and a member of IEEE. His research interests are computer architecture, parallel computing, high performance computer system and the foundation of cloud computing.
E-mail: zhumf@buaa.edu.cn

Limin Xiao was born in 1970. He received his M.S. and Ph.D. degrees in computer science from the Institute of Computing Technology of the Chinese Academy of Sciences, in 1996 and 1998, respectively. He is the vice-president of ICA of Beihang University and a professor in Beihang University. He is also a senior member of CCF. He is a member of IEEE and ACM. His research interests are computer architecture, computer system software, high performance computing, virtualization and cloud computing.
E-mail: xiaolm@buaa.edu.cn