A Distributed Shared Memory Multiprocessor: ASURA — Memory and Cache Architectures —

Shin-ichiro MORI†, Hideki SAITO†, Masahiro GOSHIMA†,
Mamoru YANAGIHARA‡, Takashi TANAKA‡, David FRASER‡,
Kazuki JOE‡, Hiroyuki NITTA‡ and Shinji TOMITA†

† Dep. of Information Science
Faculty of Engineering, Kyoto University
Sakyo-ku, Kyoto, 606-01 Japan

‡ KUBOTA Corporation
2-47, Shikitsuhigashi 1-chome
Naniwa-ku, Osaka, 556-91 Japan

Abstract

ASURA is a large scale, cluster-based, distributed, shared memory, multiprocessor being developed at Kyoto University and Kubota Corporation. Up to 128 clusters are interconnected to form an ASURA system of up to 1024 processors. The basic concept of the ASURA design is to take advantage of the hierarchical structure of the system. Implementing this concept, a large shared cache is placed between each cluster and the inter-cluster network. The shared cache and the shared memories distributed among the clusters form part of ASURA's hierarchical memory architecture, providing various unique features to ASURA.

In this paper, the hierarchical memory architecture of ASURA and its unique cache coherence scheme, including a proposal of a new hierarchical directory scheme, are described with some simulation results.

1 Introduction

Many multiprocessor systems have been developed and have now become widely available as commercial and research machines. However, there is still a strong demand for Massively Parallel Processing since there exist numerous applications (such as QCD, fluid dynamics, and many-body systems) which contain inherent parallelism that is too vast to solve on available machines.

Working towards the goal of massively parallel processing, the cluster based multiprocessor is one of the promising approaches and many recent large scale systems, such as Cedar [21], DASH [22], Paradigm [11], Willow [4], and the Intel Paragon XP/S [31], have followed this approach. Another machine of this class, ASURA, is proposed in this paper.

The ASURA cluster is configured as a bus-based, shared memory multiprocessor with up to 8 processors. The clusters, interconnected through the Inter-Cluster Network (ICN), form a distributed shared memory multiprocessor. Figure 1 and table 1 show, respectively, an overview of ASURA and the specifications of a prototype.

E-mail: moris@kuis.kyoto-u.ac.jp

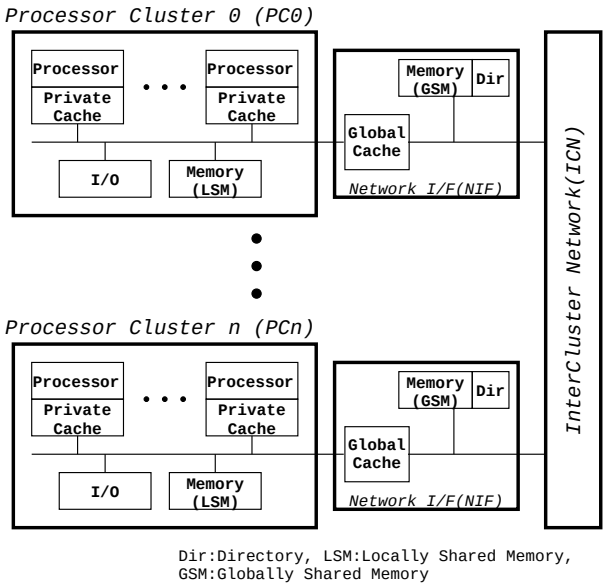


Figure 1: Overview of ASURA

In order for ASURA to scale from 2 to 128 clusters, the ICN must also be scalable over a certain configuration range according to the customer's requirements. Several candidate topologies for the ICN will be provided from which various configurations for the network can be chosen. In all cases, optical fiber is used as the medium of data transfer and, as a result, we can obtain a transfer rate of 400 megabytes per second for each communication link.

For ease of programming and of porting existing programs to the new architecture, ASURA provides a shared memory environment though the memory modules are physically distributed among the clusters. In order to implement a large scale, distributed, shared memory (DSM) machine efficiently, ASURA adopts a hierarchical cache architecture. This allows it to alleviate memory contention and decrease the latency of memory accesses simultaneously.

The key feature of ASURA is its shared cache and the design of its system-wide cache-coherence mechanism which takes into account the line sizes at the various levels of the hierarchy. Although several researchers [4][11][30] have discussed the issue of hierarchical cache design for large scale shared memory machines, little has been discussed about how to deal

Table 1: Specification of ASURA prototype

CPU	R4000MC (50MHz)		15MFLOPS(DP LINPAC)
Number of CPUs per cluster			2, 4, or 8
Number of clusters			2∼128
Private Cache	Associativity		1 (direct-map)
	Size		1MB∼4MB (SRAM)
	Line size		32B
	Write policy		Write-back
LSM	Size	each cluster	256MB
		full system	32GB
Global Cache	Associativity		4
	Size		8MB∼32MB (DRAM)
	Line size		1024B
	Write policy		Write-back
GSM	Size	each cluster	16MB∼4GB
		full system	2GB∼512GB

with the line size difference between different levels of the cache hierarchy. Furthermore most of the previous research has been done on centralized shared memory machines instead of DSM machines.

By adopting this hierarchical cache architecture, the intra-cluster architecture can be designed for fine grain parallel processing while the inter-cluster architecture can be targeted towards coarse grain parallel processing. Consequently, the ASURA architecture provides an effective parallel processing environment over a wide range of shared memory applications.

The next section explains ASURA's memory and cache architectures. Section 3 discusses how cache coherence is enforced in ASURA and a new directory based coherence mechanism is proposed for the hierarchical cache system. Some simulation results are shown in section 4. Section 5 contains a conclusion to this paper.

2 Memory Architecture

2.1 Memory Organization

As shown in figure 1, an ASURA cluster is basically a tightly coupled multiprocessor with centralized shared memory. Considering the system as a whole, ASURA has another level of shared memory which is distributed among clusters. Figure 2 shows the memory architecture of ASURA.

In ASURA, the first type of the shared memory mentioned above is shared only among the processors in the cluster and is called Locally Shared Memory or LSM. The other type of shared memory is accessible from any processor in the system and is called Globally Shared Memory or GSM. Thus ASURA's address space is partitioned into two spaces, private space and common space, corresponding to LSM and GSM respectively as indicated in figure 3.

Although this kind of memory organization has been examined also in Paradigm [11], and Willow [4], the distributed style of GSM in ASURA is different from the centralized style of GSM in those machines. According to the notation of Willow, this kind of memory organization is referred to as *layered memory organization*. As long as the hardware cost is acceptable, the layered memory organization in ASURA has advantages over the flat, or non-layered, memory organization like DASH [22]. These advantages are listed below.

1. Hierarchical data sharing: Data which is local to a cluster, such as processor local variables, stack, code, and part of the system data, can be stored in the LSM. Moreover, one may allocate a copy of read only data, shared among many clusters, to the LSM of each cluster. For instance, image data for computer graphics applications, like surface data for ray-tracing and volume data for volume-rendering, can be allocated to the LSM. In addition, such data may be excluded from inclusion in the shared cache to maintain *the Multi Level Inclusion (MLI) property*[1]. In this way, hierarchical data sharing allows the efficient use of GSM and thus increases the hit ratio of the shared cache.
2. Concurrency of memory accesses: By separating GSM and LSM, GSM in different clusters can be accessed without involving the intra-cluster bus on the GSM side. In other words, GSM accesses and LSM accesses can be performed concurrently at the same cluster.
3. Flexibility of LSM management: In ASURA, LSM management in one cluster is independent of LSM management in the other clusters. Thus, a cluster is not restricted in its LSM management policy. However, if the shared memory of each cluster is provided only as GSM, as in DASH, then when a cluster would like to enforce a memory management scheme for local data located in its shared memory, it must first negotiate with the other clusters.
4. Scalability in memory size: The private address space for each cluster is independent of the other clusters as shown in figure 3 so the total LSM size over the whole system scales with the number of clusters, though the size is 32GB currently. However the maximum GSM size is fixed at 512 GB independently of the number of clusters.

Table 2 summarizes the memory architectures of various cluster based systems for comparison.

2.2 Cache Architecture

In order to offer a shared memory environment on a large scale multiprocessor like ASURA, it is necessary to simultaneously alleviate memory contention and decrease memory access latency. To satisfy these requirements, cache memory plays an important role.

For this purpose, ASURA introduces a hierarchical cache architecture reflecting the hierarchy of the sys-

[1] Following the literature [3], "A MultiLevel cache hierarchy has the inclusion property if the contents of a cache at level $i+1$, L^{i+1}, is a superset of the contents of all its children caches, L^i, at level i."

Table 2: The Summary of Memory Architectures of Typical Cluster Based System

	GSM	LSM	GSM Access	Cache Coherence Control	
				w.r.t. GSM	w.r.t. LSM
Paragon XP/S	No	Yes	–	–	?
Cedar	Yes	Yes	UMA	Cacheability control	
DASH	Yes	No	NUMA	Snoop-based(Intra-cluster) Directory-base(Inter-cluster)	–
Paradigm(node)	Yes	Yes	UMA	Directory	Directory?
ASURA	Yes	Yes	NUMA	Snoop-based(Intra-cluster) Directory-base(Inter-cluster)	Snoop-based

GSM: Globally Shared Memory NUMA: Non-Uniform Memory Access
LSM: Locally Shared Memory UMA: Uniform Memory Access

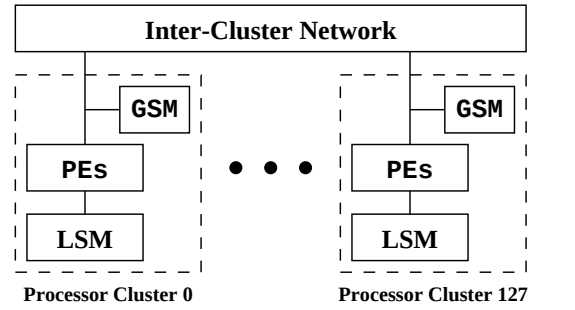


Figure 2: Memory architecture of ASURA

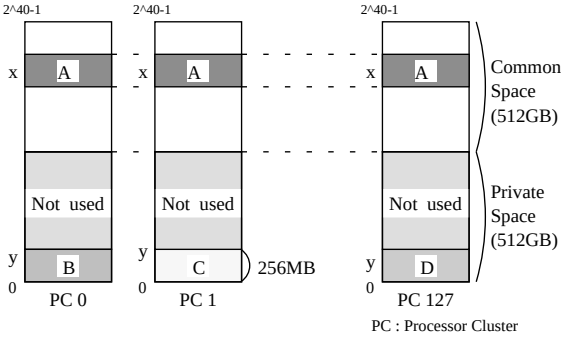


Figure 3: Memory map of ASURA prototype

tem structure, as shown in figure 4. In this figure, and also in the following discussion, the on-chip primary cache together with the secondary cache is treated as a private cache for simplicity. Thus, in ASURA, not only is each processor configured with a private cache but also, each cluster contains a shared cache.

Each processor has a dedicated private cache, or PCache and it allows the caching of data from both the LSM and the GSM. The shared cache placed in the NIF (Network InterFace) of each cluster is shared by at most 8 processors and contains data cached from only the GSM. This shared cache is referred to as the global cache or GCache. In particular, the existence of the GCache distinguishes ASURA from DASH.

The major advantage of the GCache is that it divides the cluster from the network. This enables the GSM to be isolated from the cluster-bus of the cluster. Then the cluster-bus can be optimized to access the LSM for fine grain parallel processing. On the other hand, for GSM accesses, the GCache provides an interface between the intra-cluster bus and Inter-Cluster Network(ICN). This isolates the differences in the communication protocol, transfer rate, cache line size between the two levels. This fact make it possible to optimize the cluster level architecture independent of the system level.

Several researchers [4][11][30] have discussed the issues of hierarchical cache design, however, little has been reported concerning the line size. In a hierarchical cache architecture, the line sizes at each level of the hierarchy need not be equal and, on the contrary, they should be selected according to the architectural issues present at each level. Hence the GCache line size has been enlarged, taking into account the other cluster level design factors. Note that the effect of enlarging the line size of the GCache is hidden from the PCache and the effect has no influence on the optimization of the intra-cluster architecture. This is because the *MLI (Multi Level Inclusion) directory* , which is provided as a part of the coherence enforcement mechanism in the GCache, absorbs this effect. This is the most important feature of ASURA's hierarchical cache architecture.

The characteristics of the GCache are summarized below.

1. Up to 8 cache misses are serviced concurrently. ASURA adopts a split protocol for the cluster-bus. The memory access request and its reply are treated as separate bus transactions, therefore any processor can issue a memory access request before the previous requests from the other processors are

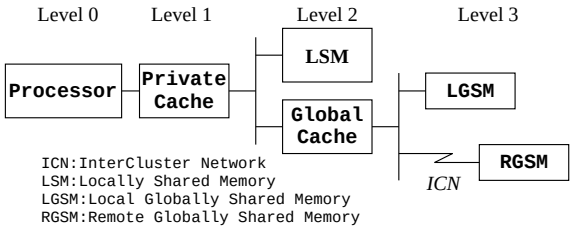


Figure 4: Memory hierarchy of ASURA

completed. These access requests are served in pipelined manner, and in order not to block the pipeline even after a cache miss occurs an eight-way bypass mechanism had been prepared in the GCache. Consequently, the GCache continues to serve consecutive requests while doing line-fetches for the previously missed lines.

2. Large cache size.
   Since the access speed of the global cache need only be as fast as LSM, DRAM can be used to implement the data array. Thus, it becomes cheaper to include a large cache, which may be up to 32 MB in ASURA.
3. Long line size.
   ASURA adopts an unusually long line size of 1024 bytes for the GCache, while the line size of the PCache is 32 bytes. Hereafter, for clarity, the PCache line will be referred to as the *sub-line* and the GCache line as the *line* to distinguish between them.

   The merits of a long line size for the GCache are shown below.

   - Compensation for relatively long ICN latency: ASURA uses optical fiber as an ICN communication medium which provides a maximum transfer rate of 400 MB/s for each link. However, the long latency of the ICN, which is at least 1μs, prevents the effective transfer speed from achieving the maximum transfer speed of the link. The long line size of the GCache compensates for the latency and improves the effective transfer speed.
   - Reduction of directory overhead: As discussed in the next section, to realize directory based cache coherence control for a large scale shared memory system like ASURA, the directory size should be less than a few percent of the total memory size. Since directory size is inversely proportional to the line size, or coherence block size, the long line size for the GCache helps to decrease the directory overhead in hardware cost.
   - Prefetching effect: Under data-parallel, for example `do-all`, type program executions, if a miss occurs in the private cache of one of the processors in a cluster, the other processors are also likely to miss on the same or nearby data. If the GCache line size is several times longer than the PCache line size then these misses can effectively be merged into one GCache line fetch. In this way, the long line size of the GCache provides a prefetching effect for the PCache, while providing the effects of memory access combining for the ICN and GSM.
   - Explicit cache management in terms of software: Explicit software cache management has been proposed in order to improve cache performance [4][11]. However, the overhead of such software schemes makes it hard to apply them to caches with ordinary line sizes. The gain in effective transfer rate with the long line size of the GCache overcomes this overhead. Explicit cache management in ASURA is discussed in the next section.

   Of course, there are some arguments against the long line size, one of which is the problem of false sharing. Some research has been done on this problem using both compile time static analysis approaches [15][29][18] and run time detection approaches [9][14][20]. Hopefully, the results of this research can be applied to ASURA to alleviate the false sharing problems.

3 Cache Coherence

3.1 Coherence Enforcement Strategy

In each of ASURA's clusters because writable, shared data may be cached into both the GCache and each of the processors' PCaches, cache coherence problems will occur and must be solved. There are basically two approaches, the hardware based approach [2][28] and the software based approach [5][10].

Software based approaches become promising as the system size goes up because, in order to preserve cache coherence, they rely on self-invalidation of stale data at each processor and don't require any inter-processor communication for stale data invalidation. They are however still in the research stage, and are not yet sophisticated enough for practical use.

Therefore, a hardware based approach has been adopted in ASURA while leaving it possible for the software to assist with the hardware based cache coherence enforcement. Hence if we have a compiler which has the ability to manage the cache satisfactorily, or if the programmer offers information about data sharing, ASURA can use this information for coherence enforcement as will be discussed later in section 3.2.

In the development of ASURA's cache coherence enforcement strategy, the matters of intra-cluster and inter-cluster cache coherence enforcement are considered first, independent of each other. This is possible because the cluster level and the system level architectures have been separated by the global cache. Then, they are united in a hierarchical fashion to enforce the coherence between PCaches and GSMs. In ASURA, this coherence is enforced by maintaining the Multi Level Inclusion (MLI) property. This is referred to as cache coherence *hierarchical cache coherence*, in contrast with intra- and inter-cluster cache coherence.

Consequently, system wide memory coherence is maintained through these three kinds of coherence control in a hierarchical manner reflecting the system's structure.

The following two sections describe the intra- and inter-cluster cache coherence controls in more detail and section 3.1.3 describes how hierarchical cache coherence is preserved.

3.1.1 Intra-cluster Cache Coherence

Like ordinary bus based systems, ASURA adopts the snooping scheme. To enforce coherence between PCaches and either LSM or the GCache, the write-back write policy has been chosen.

The GCache appears as normal memory from the view point of intra-cluster cache coherence. Therefore, GCache and LSM can be treated as a single unit for implementing the intra-cluster coherence control, except that the GCache has to snoop the cluster bus in order to reflect modifications on PCache lines to the state of corresponding GCache lines.

In order to enforce intra-cluster cache coherence, one of the following states: INVALID, CLEAN (unmodified, possibly shared), and DIRTY (modified, no other copy), is assigned to each PCache line and the state is represented by the V and M bits of the PCache tag array (figure 5(a)). Intra-cluster cache coherence is maintained in the following way (see figure 6).

1. Read miss : The PCache which causes the miss puts a read request on the cluster bus. If any other PCache in the same cluster has a copy of the PCache line in state DIRTY, this PCache 1) responds the request by writing back the line to LSM or GCache, 2) releases the ownership of the line, and 3) sets its state to CLEAN. Otherwise, LSM or GCache responds to the request by putting corresponding data on the cluster bus. The requesting PCache picks up the corresponding data from the bus, puts it into its data array, and set its state to CLEAN.
2. Write miss : The PCache which caused the miss puts a write request on the cluster bus. If any other PCache in the same cluster has a copy of the PCache line in the DIRTY state then this PCache 1) responds to the request by writing back the line to the LSM or the GCache, 2) transfers the ownership of the line to the requesting PCache, and 3) sets the state of the line to INVALID. Otherwise, the LSM or the GCache responds to the request by putting the corresponding data on the cluster bus. All PCaches other than the requesting one set the state of their copy of the line to INVALID. The requesting PCache reads the corresponding data from the bus, stores it into its data array in the DIRTY state. In any case, if the write miss is caused by a processor's write on a CLEAN line, no data transfer is made on the bus. By observing the ownership transfer on the cluster bus, the GCache knows the state of any of the sub-lines cached in a PCache.
3. Replacement : When the PCache needs to replace a line on a read or write miss, the PCache set the states of the corresponding sub-line to INVALID, and it is written back if the sub-line had been in the DIRTY state.
4. Read hit / Write hit : No coherence enforcement transaction is made.

3.1.2 Inter-cluster Cache Coherence

In this section, the coherence control between the GCaches and the GSM, and between the GCaches is considered. Directory based inter-cluster cache coherence is discussed here.

The directory schemes [8] can be classified as either full-map directory, limited directory, or chained directory.

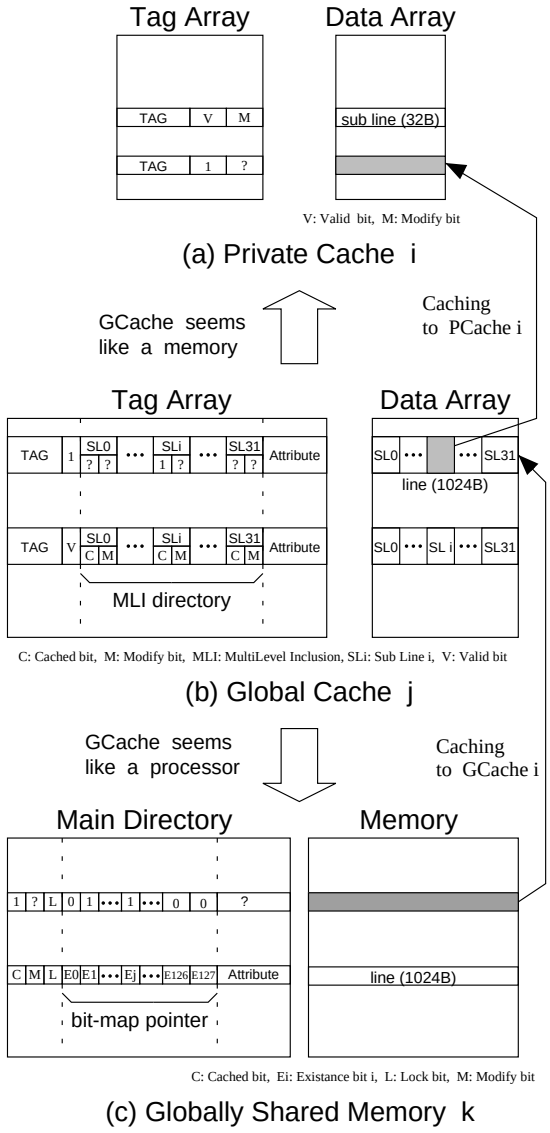
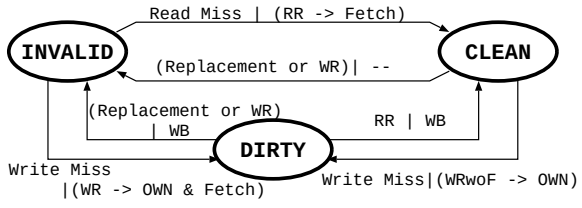


Figure 5: The Hierarchical Cache Coherence Enforcement Mechanism




```
Event | ACTION (Bus Transaction)
        OWN:OWNership acquisition  WBwoF : Write Back without Fetch
        RR : Read Request          WR : Write Request
        WB : Write Back            -- : No transaction
State:  INVALID,  CLEAN(unmodified, possibly shared),
        DIRTY(modified, no other copy)
```

Figure 6: State Transition Diagram

For simplicity of control, ASURA uses a full-map directory. The drawback of using a full-map directory is directory overhead. In our case, this overhead is proportional to the number of GCaches and inversely proportional to the GCache line size.

However, the GCache line size of 1024 bytes, which is 32 times the PCache line size, makes it possible to reduce the directory overhead of ASURA to a fraction of what it would have been if the line size were the same size as the PCache line. At the same time, the width of the directory is kept narrow because the directory is managed in units of clusters, or GCaches, instead of processors, or PCaches.

So far, several interesting techniques have been proposed to reduce the directory overhead without enlarging the line size [6][19][23]. However, most of these techniques aren't adopted because they increase the complexity of the directory itself and this complexity may prevent high speed coherence control and increase time required to develop the machine.

The organization of the full-map directory is shown in figure 5(c). This directory is referred to as the *main directory* in contrast with the *MLI directory* which is provided for the purposes of ensuring the MLI property and absorbing the effect of enlarging the GCache line size. Each *main directory* entry associated with a line is composed of three parts: 1) 3-bit status, 2) a bit map pointer of 128 bits, and 3) a field to specify the line attributes which are mentioned in section 3.2. The 3-bit status indicates 1) whether the line is cached, with the C bit, 2) whether the line is modified, with the M bit, and 3) whether the line is locked, with the L bit. The L bit is used as part of the synchronization mechanism for Locking. Each bit of the bit map pointer indicates if the corresponding GCache has a cached copy of the line or not. The attribute field contains the information not only on the characteristics of the line but also on how the line is manipulated. Such information is stored into this field by software, the operating system perhaps, when the associated page is allocated to a GSM.

3.1.3 Assurance of the Multi Level Inclusion property

To guarantee the MLI property [3] in ASURA, all lines residing in PCaches in a cluster must be contained in either their parent LSM or their parent GCache, where the word "parent" means being closer to the memory in the memory hierarchy. At the same time, all lines residing in the GCaches must be included in the set of all lines in the GSMs.

The inclusion relation 1) between GCaches and GSMs, and 2) between PCaches and their parent LSM with respect to the PCache lines which come from the LSM, are always maintained naturally. Consequently, the inclusion relation that must be considered is that 3) between PCaches and their parent GCache with respect to the PCache lines which come from the GCache. Then only the third inclusion relation will be discussed hereafter.

Assurance of the MLI property requires a slightly stronger restriction than necessary for hierarchical cache coherence enforcement. For instance, from the view point of the hierarchical cache coherence enforcement, a PCache may contain data which does not exist in its parent GCache as long as the hierarchical coherence is maintained, though this situation is not allowed if the MLI property is to be assured. However, the advantages of the MLI property, like the filtering of inter-cluster coherence enforcement traffic, overcomes the disadvantage of this stronger restriction.

As a result, the coherence enforcement to keep the MLI property is initiated when a GCache receives 1) an invalidation or 2) a write-back request from GSM, and when a replacement of 3) clean or 4) dirty line occurs in the GCache.

In the first and third cases, where the data is not modified in this cluster, all of the sub-lines corresponding to the line are invalidated first, and then the GCache line is invalidated.

In the second case, all of the dirty sub-lines corresponding to the line are written back to the GCache first, and then the GCache line is written back to memory. The last case is slightly different from the second case in that it requires invalidations of the GCache line and its corresponding sub-lines after the line is written back.

For simplicity, in the above discussion of coherence enforcement, the difference between the line sizes of the GCaches and the PCaches has not been mentioned. Although the difference is not an essential part of assuring the MLI property, in practice, not only for the efficiency of the coherence enforcement but also for concealing the system level architecture from that of the cluster level, this difference must be taken into account.

As discussed in section 2.2, the line sizes of the GCache and PCache are 1024 bytes and 32 bytes respectively. 32 sub-lines with consecutive addresses belong to the same GCache line. Therefore at most 32 sub-lines with different addresses have to be invalidated or written back when a GCache line is invalidated or written back. The sub-line invalidation requests, which are treated as *Write Request without Fetch*, can be issued in a pipelined manner and they theoretically don't need any acknowledgment. At most 32 invalidation requests need to be issued for an invalidation of a GCache line since all 8 PCaches snoop on the cluster bus. The sub-line write-back requests, which are treated as *Read Request*, can be issued in a pipelined manner as well, but each of them needs an acknowledgment or a write-back corresponding to each sub-line. This causes two more problems to arise.

To explain the first problem, it is assumed that the situation where a sub-line which belongs to a dirty GCache line is cached into PCaches (in state CLEAN) or is not cached into any PCache (in state INVALID). In this situation, there is no PCache which replies to the write-back request for the sub-line because there is no dirty sub-line corresponding to this request, so the GCache which issued this request will wait until a timeout occurs for an acknowledgment.

The second problem is not as serious as the first and is an efficiency problem. If all 32 invalidation requests are issued whenever a GCache line is invalidated or replaced then it may cause unnecessary traffic on the

intra-cluster bus because some of the sub-lines are not cached at all.

These problems have originated due to the the lack of information about each sub-line. It would be possible to solve these problems if some history information for each sub-line were to be kept. For this purpose, a directory, which we refer to as the MLI directory, has been created and placed in the tag array of the GCache as shown in figure 5(b). Each directory entry is created dynamically when the tag entry for the associated line is allocated. In order to track the status of each sub-line, the C bit and the M bit are provided for each sub-line.

The meanings of the C bit and the M bit are the same as the meanings of those in the main directory with the exception that they represent the status of a sub-line instead of a line. These two bits are updated whenever a PCache accesses a corresponding sub-line. When a GCache line is invalidated or written back, whether due to replacement or inter cluster coherence enforcement, by examining the corresponding MLI directory entry only the necessary coherence work need be done.

As well as a C bit and a M bit for each sub-line, each MLI directory entry has some bits to store the attributes associated with the line. These attributes are used for coherence enforcement in the GCache.

Here, we should mention that the GCache is not required to be as fast as the PCache, and hence the overhead of the MLI directory itself and that of the directory management are not significant.

The organization of the MLI directory itself is quite similar to the sectored directory [6] except that the MLI directory does not need either a bit vector to track the PCaches which have a copy of the sub-line nor the pointers to the PCaches which have ownership (write privileges) of the sub-line. The reason why we don't need them is simply that the cluster is a bus-based architecture with a snooping based coherence scheme. However there are great differences between them. 1) In the MLI directory, the unit of data caching to the PCache is a sub-line, while it would be a line in the sectored directory. This characteristic makes it possible to offer transparency of the system level architecture, such as the line size of the GCache, to the cluster level architecture. 2) Each entry of the MLI directory is dynamically allocated in the tag array of the GCache when the corresponding GCache line is allocated, so that the directory overhead is proportional to the total cache size instead of the total memory size which may be a few tens, or rather hundreds, times larger than the total cache size.

3.2 Software assist for cache coherence

As has been shown in the previous two sections, ASURA implements a hierarchical cache coherence scheme in hardware. However, with this scheme, inter-cluster communication is required to enforce cache coherence on the fly. In order to reduce this run time overhead, software can assist with the hardware coherence enforcement.

The advantages of software based coherence enforcement are derived primarily from the use of 1) knowledge about the characteristics of the data, and 2) knowledge of how the data is manipulated during the program execution, or the semantics of the data.

Firstly the typical case where software assistance can greatly reduce the run time overhead [27] is shown. Counter based barrier synchronization is examined, where the counter is decreased by one each time the process participating in the barrier reaches the synchronization point in its program. If hardware based cache coherence is enforced without software assistance, then network traffic to invalidate or update the line arises at each counter modification. For the processes which are awaiting the completion of the synchronization, however, the intermediate value of the counter toward the completion is irrelevant. They only need to know when the value of the barrier counter reaches zero. Therefore all this network traffic except the last message, which signifies the completion of the synchronization, is unnecessary. If the hardware were informed of the semantics of the data in some way by software, this unnecessary traffic could be avoided. This is what the software is asked to do.

Hereafter, the manner in which software assist is implemented in ASURA is described. The basis of our software assist is the attribute marking scheme [4][5] where the attributes corresponding to the line guide the hardware coherence enforcement.

Other than the basic attributes like *cacheable*:allows caching of data, *non-coherent*: disables hardware coherence control, and *exclusive*: allows only a single PCache (or GCache) to hold a cached copy of data, the following attributes are presently being considered and implemented in ASURA by the addition of some additional hardware. A line may be assigned several of these attributes simultaneously.

1. *Update on write-back* : When a line marked with this attribute is written back from a GCache, the main directory updates the other cached copies of the line, and thus the corresponding PCache sub-lines too. For data with one writer and many readers this avoids extra cache misses. A broadcast option is also available for this attribute. Note that this is not like a *reflective memory system* [24] where the update is enforced whenever a processor modifies data; ASURA enforces the update only when a GCache line is written back to memory.
2. *Resident* : is used to force a specific line to reside in the GCache. Once a line marked with this attribute is fetched into the GCache, this line will never be replaced unless it is forcibly invalidated with the GCache I/O command mentioned later.
3. *Store Allocate No Fetch* : When a write access to a line marked with this attribute causes a miss in a GCache, the GCache allocates an entry for this line and requests the GSM to acquire the ownership of the line, but does not require the GSM to supply the data. In order to reduce unnecessary data transfer through the IPN, this attribute can be added to a line which a cluster completely overwrites without any previous reference to the line.
4. *Out-of-MLI* : allows a PCache to hold copies of GSM data regardless of the existence of corre-

sponding GCache lines in its parent GCache. All coherence enforcement requests related to lines with this attribute have to be forwarded to cluster buses even if the GCache does not have an entry for the corresponding line.

5. *Barrier Synchronization variable* : can be thought of as a variety of the *update on write-back* attribute. The difference is that, instead of updating the GCache lines and the corresponding PCache sublines at every occurrence of write-back, it updates them when the GSM storing the line has received the specified number of writes to the line.

These attributes are stored into corresponding attribute fields of the main directory (figure 5(c)) first and are copied into attribute fields of the GCache tag array (figure 5(b)) when the line is fetched. Then these attributes are used as a guideline both by the GCaches and by the main directory whenever coherence enforcement actions are required.

In addition to the attribute marking scheme mentioned above, ASURA provides the software (or the programmer) the means to manage both GCaches and GSMs explicitly in the form of I/O commands to the GCache. Currently, the following I/O commands for data transfer [7], cache coherence [10], and synchronization [22] are provided.

- *MOVE* command: forces a pre-fetch or a write-back on the specified line depending on the argument of the command. This command can be used to hide memory access latency.
- *Line-transfer* command: performs data transfer from a specified source address to a specified destination address through the GCache where this command is initiated. The length of data to be transferred is fixed as the GCache line size. The data transfer is performed as follows: 1) Invalidate the GCache line associated with the destination address if it exists in the GCache. Note that the write back is not needed even if the line is in the DIRTY state. 2) Fetch the source line into the GCache if the GCache does not contain the line. 3) Change the tag address of the line associated with the source address with the destination address, and inform the main directory of the change if needed. If the source line was in the DIRTY state it must be written back before changing the tag address. 4) Write back the destination line if needed. It is recommended that the *store allocate no fetch* attribute is assigned to the destination line. Since the line-transfer uses a tricky technique of operating on the GCache tag array, data transfer can be performed without moving the data at all if the GCache contains the source line in CLEAN state. This command is well suited for the copy-on-write operation.
- *Invalidation* command: invalidates the specified line by invalidating or if needed, forcing a write-back before the invalidation, and makes the associated GCache entry empty.
- *Lock/Unlock* command: provides GCache line based synchronization primitives. ASURA supports a *queue based lock* mechanism, as well as DASH, where the existence bits of the main directory are used to trace clusters waiting the acquirement of the lock. *Lock/Unlock* commands combined with *MOVE* command could support a rather complex command which forces a pre-fetch of a line when the Lock associated with the line is acquired or forces a write-back of a line when the Lock associated with the line is released.

4 Performance Simulation

4.1 Simulation Environment

ASURA's performance has been investigated by the use of trace-driven simulation. As the trace data, transaction sequences on each cluster bus were used, which are composed of the time, address, and the access type for each transaction, instead of the address traces for each processor.

Although various simulations have been done [16][26], here the scalability of ASURA and the effect of the long GCache line size is focused upon.

Matrix multiplication ($C = A \times B$) and LU decomposition ($A = L \times U$), where A,B,C,L and U are matrices of size 1024×1024, have been chosen as sample programs.

All matrices are initially distributed among the GSMs so that their sub-matrices, of form $MATRIX[nI+i,*]$ {n | 0,1,...,1024/I} where I: Number of Clusters, are allocated to the GSM of cluster i.

In order to extract cluster level parallelism, the *data owner compute rule* is applied to the code. To extract fine grain parallelism inside each cluster, the code was further optimized so that it could fit within the subline size of a PCache.

Among the various available ASURA configurations, the configuration shown in figure 7 was used throughout these simulations, where the number of processors in a cluster is fixed at 8. A cache size of 8 KB for the primary on chip cache, 2 MB for the secondary cache, and 16 MB for the GCache and the memory access latencies shown in table 3 were assumed. The memory access latencies were derived from a hardware logic simulator.

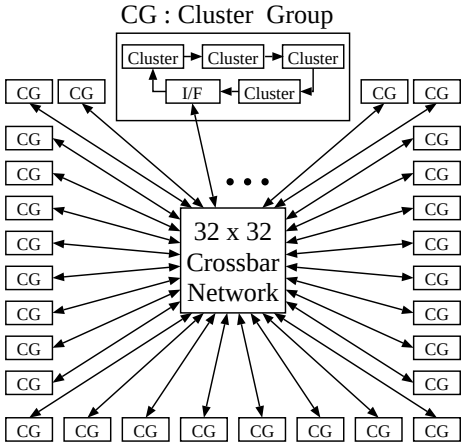
4.2 Simulation Results

Figure 8 shows the simulation results both for matrix multiplication(**MATMUL**) and for LU decomposition(**LU**), where (a) and (c) show performance versus the number of clusters, and (b) and (d) show performance versus the GCache line size.

Table 3: Memory Access Latencies

Memory		Latency [ns]
Private Cache	Primary	10
	Secondary	200
Global Cache		1500
GSM	Local Cluster	2000 (+ 2.5 × n)
	Local Ring	2500 (+ 2.5 × n)
	Remote Ring	3000 (+ 2.5 × n)

n: GCache line size [Byte]



Note 1 : Number of Clusters in a Cluster Group is in the range from 1 to 4. Clustes inside a CG are interconnected by a ring network.
Note 2 : In case of 16 clusters organization, a 16x16 crossbar network which is a quarter of the 32x32 crossbar network is used.

Figure 7: A ring-crossbar configuration of ASURA

In the case of **MATMUL**, the effect of the software prefetching(**PF**) and of the GCache bypass buffer(**BP**) were examined. When the GCache bypass buffer is used the requested data passes through the GCache without waiting the completion of GCache line refill. As shown in figure 8(a), the basic performance scales well up to 64 clusters. Using **BP**, scalability was confirmed up to 128 clusters. The beneficial effect of **PF** was also confirmed. By using **BP** together with **PF**, ASURA could achieve a performance of 9.6 GFLOPS with 128 clusters. This is about 100 times the performance of 93 MFLOPS which is achieved by a single cluster.

For **LU**, the effect of broadcasting(**BC**) was examined. This was simulated by setting the *Update on Write-Back* attribute and explicitly executing the *MOVE* command. The effect of **BP** was also examined. In these simulations (see figure 8(c)), performance increases with the number of cluster until about 32 or 64 clusters. It then starts decreasing with any further increases in the number of clusters. It is assumed that this is because the amount of computation becomes too small compared to the amount of communication required.

For both applications, as shown in figure 8(b) and (c), the performance scales up with the increase of GCache line size. We also confirm that, the larger the line size becomes, the more the effect of **BP** becomes significant because the time spent to transfer a line becomes dominant factor of memory access latency.

From these results, it is concluded that the ASURA architecture and its uniquely long GCache line size are justified as long as the applications have enough parallelism to exploit both system-level coarse grain parallelism and cluster-level fine grain parallelism.

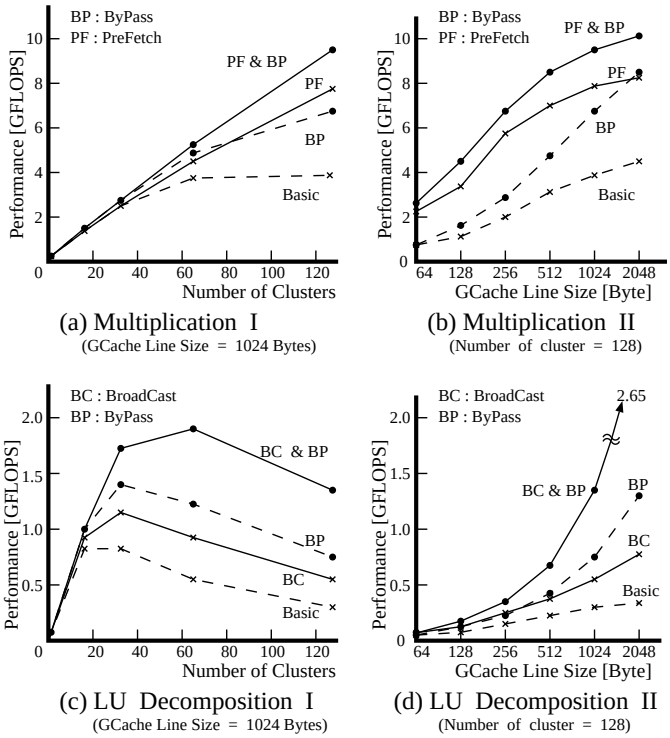


Figure 8: Simulation Results

5 Conclusion

An overview of ASURA, a cluster based multiprocessor system has been given. ASURA offers a shared memory environment though Globally Shared Memories (GSMs) are distributed among the clusters. For the efficient use of distributed shared memory, ASURA is configured with a shared cache, the Global Cache (GCache), on each cluster as an interface between the cluster and the InterCluster Network (ICN). Along with the Locally Shared Memory (LSM), provided independently of the GSM, the GSM and the GCache form a layered memory architecture.

As a result of providing a GCache on each cluster, it becomes possible to consider the inter-cluster and intra-cluster architectures independent of each other. This is also true for cache coherence enforcement. In order to enforce intra-cluster cache coherence, a snooping mechanism is provided for each processor. To enforce inter-cluster cache coherence, assuring the MLI property, a new hierarchical directory scheme has been developed where the directory is composed of two separate directories corresponding to the memory hierarchy: the main directory for the GSM and the MLI directory for the GCache.

In addition, ASURA provides software the means to improve the efficiency of cache coherence enforcement. The property labeling scheme allows the software to set cache line attribute which are used for coherence enforcement, while several I/O commands are available for software to manage the GCache directly.

Finally, simulation results have been presented that

confirm that the ASURA architecture could offer a wide range of scalability and it could achieve a performance level of 9.6 GFLOPS in a 128 cluster configuration. It was also confirmed that the long GCache line size of 1024 bytes is justified for some large scale numerical computation.

Acknowledgment

We would like to acknowledge the considerable contributions from M. Yamaguchi and K.Yamada both with KUBOTA Corp. We would like to thank Prof. S. Agusa of Nagoya Univ., Prof. H.Nakashima of Kyoto Univ. and the referees of this paper for the useful comments. We would also like to thank students in the Tomita Laboratory of Kyoto University and also the staff of the office of computer business in KUBOTA Corporation for their collaboration.

References

[1] Adve, S.V. and Hill, M.D., "WEAK ORDERING - A NEW DEFINITION AND SOME IMPLICATIONS," *Comuter Sciences Technical Report*, #902, Computer Sciences Department, University of Wisconsin-Madison, December 1989.

[2] Archibald, J. and Bear, J.-L., "An Economical Solution to the Cache Coherency Problem," *Proc. 11th Ann. Int'l. Symp. Comput. Architect.*, pp.355–362, June 1984.

[3] Baer, J.-L. and Wang, W.-H., "Architectural Choices for Multilevel Cache Hierachies," *Proc. 1987 Int'l. Conf. on Parallel Processing*, pp.258–261, 1987.

[4] Bennet, J.K., et al., "Toword Large-Scale Shared Memory Multiprocessing," *SCALABLE SHARED MEMORY MULTIPROCESSORS*, ed. by Michel Dubois and Shreekant Thakkar, pp.281–300, Kluwer Academic Publisheers, 1991.

[5] Brantley, W.C., et al., "RP3 Processor-Memory Element," *Proc. 1985 Int'l. Conf. on Parallel Processing*, pp.782–789, Aug. 1985.

[6] O'Krafka, B.W. and Newton, A.R., "An Empirical Evaluation of Two Memory-Efficient Directory Methods," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.138–140, May 1990.

[7] Callahan, D., et al., "Software Prefetching," *ASPLOS IV*, pp.40 – 52, 1991.

[8] Censier, M. L. and Feutrier, P., "A New Solution to Cache Coherence Problems in Multicache System," *IEEE Trans. on Computers*, Vol. C-27, No. 12, pp.1112 – 1118, Dec. 1978.

[9] Chen, Y.-C. and Veidenbaum, A.V., "A Software Coherence Scheme with the Assistance of Directories," *CSRD Report No. 1106*, pp.1–22, June 1991.

[10] Cheong, H. and Veidenbaum, A.V., "Compiler-Directed Cache Management in Multiprocessors," *IEEE Computer*, vol.23, no.6, pp. 39-47, June 1990.

[11] Cheriton, D.R., et al., "Pradigm:A Highly Scalable Shared-Memory Multicomputer Architecture," *IEEE computer*, pp.33–46, February 1991.

[12] Dubois, M., et al., "Memory Access Buffering in Multiprocessors," *Proc. 13th Ann. Int'l. Symp. Comput. Architect.*, pp.434–442, 1986.

[13] Dubois, M., et al., "Synchronization, Coherence, and Event Ordering in Multiprocessors," *IEEE computer*, pp.8–21, February 1988.

[14] Dubois, M., "Delayed Consistency," *SCALABLE SHARED MEMORY MULTIPROCESSORS*, ed. by Michel Dubois and Shreekant Thakkar, pp.207–218, Kluwer Academic Publisheers, 1991.

[15] Eggers, S.J. and Jeremiassen, T.E., "Eliminating False Sharing," *Proc. 1991 Int'l. Conf. on Parallel Processing*, pp.I-377 – I-381, Aug. 1991.

[16] Fraser, D. and Tanaka, T., "An overview of ASURA's Network with Simulation Results," *IPSJ technical report of SIG. Arc.*, Vol.93, No.20, pp.133–140, March 1993.

[17] Gharachorloo, K., et al., "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.15–26, May 1990.

[18] Guarna, Jr., V.A., "A Technique for Analyzing Pointer and Structure References in Parallel Restructuring Compilers," *Proc. 1988 Int'l. Conf. on Parallel Processing*, pp.212–220, 1988.

[19] Gupta A., et al., "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," *Proc. 1991 Int'l. Conf. on Parallel Processing*, pp.I-312 – I-321, 1990.

[20] Hosomi, T., et al. "A Hardware Cache Coherence Scheme with the assistance of software," *IPSJ technical report of SIG. Arc.*, Vol.93, No.20, pp.117–124, Mar. 1993 (in Japanease).

[21] Konicek, J., et al., "The Organization of the Cedar System," *Proc. 1991 Int'l. Conf. on Parallel Processing*, pp.149–156, Aug. 1991.

[22] Lenoski, D., et al. "The Directory-based Cache Coherece Protocol for the DASH Multiprocessor," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.148–159, May 1990.

[23] Maa, Y.-C., et al., "Two Economical Directory Schemes for Large-Scale Cache Coherent Multiprocessors," *ACM SIGARCH Comput. Architect. News*, vol.19, no.5, pp.10–18, Sept. 1991.

[24] Maples, Creve., "A High-Performance Memory-Based Interconnection System for Multicomputer Environments," *Proc. of Supercomputing'90*, pp.295-304, 1990.

[25] Mori, S., et al., "The Kyushu University Reconfigurable Parallel Processor – Cache Architecture and Cache Coherence Schemes –," *Proc. Int'l Symp. on Shared Memory Multiprocessing*, pp.218–229, April 1991.

[26] Naito, J., et al., "Performance Evaluation of the ASURA Cluster," *IPSJ technical report of SIG. Arc.*, Vol.92, No.97, pp.73–80, December 1992 (in Japanese).

[27] Saito, H., et al, "The Event Correspondent Cache Coherency Scheme and Its Application to Barrier Synchronization," *IPSJ technical report of SIG. Arc.*, Vol.92, No.95, pp.9-16, Aug. 1992 (in Japanese).

[28] Stenström, P., "A Survey of Cache Coherence Schemes for Multiprocessor," *IEEE Computer*, vol.23, no.6, pp. 12-24, June 1990.

[29] Torrellas, J., et al., "Shared Date Placement Optimization to Reduce Muliprocessor Cache Miss Rates," *Proc. 1991 Int'l. Conf. on Parallel Processing*, pp.II-266 – II-270, 1990.

[30] Wilson Jr., A.W., "Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors," *Proc. 14th Ann. Int'l. Symp. Comput. Architect.*, pp.244–252, 1987.

[31] "Paragon XP/S Product Overview," *Intel Corporation*, 1991.