Mathematical Problems of Computer Science 23, 2004, 119–126.

On System with Distributed Shared Memory

Hovhannes Z. Nalbandyan

State Engineering University of Armenia
e-mail hovign@web.am

Abstract

Distributed shared memory systems combine the scalability of loosely coupled multiple computer systems with the ease of usability of tightly coupled multiprocessors, providing with transparent replication and caching of data. This paper introduces distributed system for parallel computing – DSPC, that provides distributed shared memory on top of network of workstations. Programming model, memory organization, cache-coherence protocol and adaptive techniques are discussed in the paper. An evaluation with some well-known DSM benchmarks was done to present the overall performance of the DSPC system.

1 Introduction

Networks of single or multi processor workstations have become an alternative to large bus-based distributed multiprocessor systems, due their inexpensiveness. Nevertheless, application development for such distributed systems is difficult, as the application should use message passing mechanisms, to explicitly send and receive data between computers. Enabling global shared memory on top of the physically distributed local memories of networked workstations makes it possible to offer developers with programming advantages of shared memory.

Software distributed shared memory (DSM) runtime systems use operating system memory management facilities to transparently intercept user application accesses to local memory and perform communications appropriate to the underlying shared memory organization. Thus, distributed application developer is given with a large global address space, which eliminates the task of transferring data between processes located on different computers. The memory organization in DSM systems is usually page based. The large size of the unit of sharing (a page) and the high latency associated with communications between different computers challenge the performance potential of software DSM systems.

This paper discusses DSPC system distributed shared memory organization and programming interface. The basic architecture of DSPC systems can be referred in previous publications [2], [5].

2 DSPC

The goal of DSPC system is to provide parallel computing environment, with programming model similar to Windows programming. The system includes dynamic workload balancing, automatic on-demand executable code transfer, computing environment monitoring and dynamic reconfiguration.

2.1 Memory organization

DSPC has page based software distributed shared memory organization, and it is different from other systems, in that it provides not only distributed shared memory abstraction, but a complex programming environment, which allows programmers to write applications, that may use no shared memory at all. It provides single address space Win32 like programming environment, with multithreading, shared memory and synchronization mechanisms, but over multiple workstation and server interconnect. The concurrent execution unit in DSPC system is the *call*, which can be thought of as thread in Win32 terms.

Distributed shared memory is organized as separate shared memories. An application can allocate any number of shared memories, each having size of up to 4GB. The amount of distributed memory is limited only by overall systems resources. This enables the DSPC system to provide larger than 4GB memory space.

The memory organization principles in DSPC system is illustrated in figure 1.

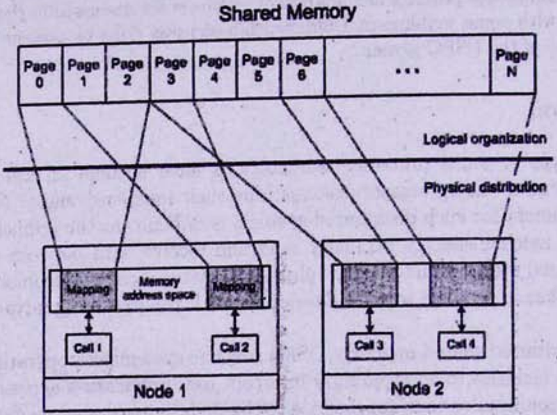

Fig. 1. Shared memory organization

Shared memory is divided into pages, where page size is equal to Windows 2000/XP operating system page size - 4 kilobytes. Each page can reside on one or more nodes, depending on data distribution and application access pattern. Each call, that accesses a shared memory, should make a mapping by calling appropriate DSM function. In figure 1, it is illustrated that pages 0, 1, 2 and 3 are in use by Call 1, and are mapped into Node 1 memory address space. Same page 1 is also mapped by Call 2. Call3 and Call4 on Node2 have mappings of other pages in shared region. In this current illustration, page 3 resides both on Node1 and Node2.

Generally page based DSM systems use the virtual memory management mechanism, provided by the operating system, to trap accesses to shared memory. DSPC uses *structured exception handing* and page protecting mechanisms to catch and process memory access violations. Using these mechanisms, DSPC system is able to protect invalid and read-only pages and fetch latest updates from other nodes on page faults, occurring on read or write instructions in user application.

Figure 2 illustrates mapping principle. Mapping same memory data to different addresses is done using file-mapping system mechanism. This allows having different access modes to the same data in shared memory for different calls on a single node. For example, one call may have read access to some data in shared memory, as the corresponding pages in its mapping has read-only protection, but meanwhile another call may have read-write access.

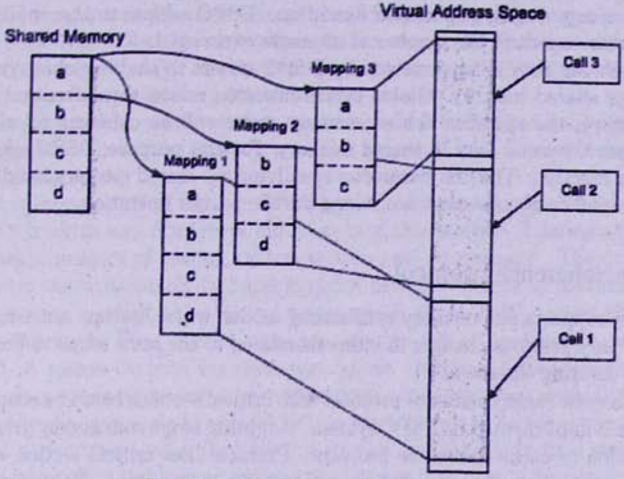

Fig. 2. Memory mapping schema

2.2 Application programming interface

Application developed to run in DSPC environment consists of two modules: the main executable application, and the dynamic link library (DLL) with user distributable functions. Main parallel application calls distributed functions from user library using DSPC system routines. Each call to a user function, which in terms of DSPC is a *call*, is placed and run concurrently on remote computer taking part in computations.

Calls are able to suspend their execution waiting for some event to occur. This event may be another call completion or release of some synchronization object. Calls are able to spawn new calls. But besides these possibilities calls can share data using shared memory mechanism.

In figure ??, in terms of DSPC architecture, the node, where the main application is running, is called *central node*. Main application or any other executing call may create a *named shared memory*. This is done by calling *DSM_MemoryAllocate* function, which returns a *handle* to shared memory. The storage for shared memory is not allocated initially. Other calls can query for shared memory handle by name. Before call can access the shared memory, it has to map a portion or the whole shared memory into its address space. This is done by calling *DSM_MapRegion* function, specifying the offset and the length of region to be mapped. The result of function will be the virtual address of this mapping region in process memory address space. After this point, it can access this block for read and write as usual. System automatically handles page faults and performs coherence tasks. Once call has finished accessing mapped region, it should unmap region using *DSM_UnmapRegion* function.

A *memory synchronization object* is provided by DSPC system for supporting scope consistency model, which is named *critical section.* *DSM_Lock* and *DSM_Unlock* take the critical section unique name as parameter, and perform enter and leave critical section operations correspondingly. For each critical section there is an assigned manager node, which can dynamically change during application execution. DSPC adapts to the application access pattern, in order to reduce the number of messages exchanged.

A *barrier mechanism* is implemented by DSPC system to enable global synchronization over particular shared memory. Global synchronization means that after end of particular barrier operation, the specified shared memory views will be coherent on all nodes, and all calls will see the same data in shared memory. For this purpose, DSPC system provides *DSM_Barrier* function. The first parameter specifying the shared memory handle, the second specifies the number of calls, that are taking part in barrier operation.

2.3 Cache coherence protocol

DSPC system supports one memory consistency model, which is *scope consistency* (ScC)[3]. ScC requires only previous changes in intervals related to the same scope to be visible to the processor on entering the scope.

We propose new cache coherence protocol with critical sections based on scope consistency model, which is implemented in DSPC system. Adopting scope consistency greatly simplifies the organization of cache coherence protocol. Protocol uses critical section mechanism to enable mutual exclusion of concurrent execution in the same scope. Coherence information is maintained using write-notices associated with critical section.

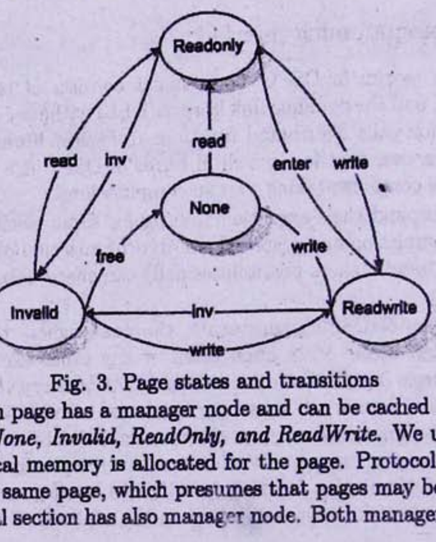

Fig. 3. Page states and transitions

In the protocol each page has a manager node and can be cached by other nodes. There are four page states: *None, Invalid, ReadOnly, and ReadWrite.* We use None page state to indicate that no physical memory is allocated for the page. Protocol allows multiple nodes simultaneously modify same page, which presumes that pages may be in different states, at different nodes. Critical section has also manager node. Both managers of page and lock can dynamically migrate during program execution, depending on decisions made by adaptive mechanisms. Figure 3 illustrates page states and transitions.

On a read page fault, the faulting page is fetched from the page manager node in read-only state into the local memory. On a write page fault, if the page is in state None or in state Invalid in the local memory, it is fetched from the page manager in read-write state. If the faulting page is in read-only state in the local memory, the state is turned into read-write, and a twin of this page is created before allowing write.

When call is trying to enter a named critical section, corresponding request is sent to the critical section manager from the node where the call is executing. The requesting call is then stalled until it receives reply. The critical section manager also includes associated write-notices in reply to request. After the requesting processor receives this granting message, it invalidates all cached pages by the associated write-notices.

During execution inside critical section, when page fault is detected on a page, which has associated write-notice in critical section, a request is sent to critical section manager to get the diffs for faulting page.

When call is leaving critical section, DSPC system compares pages, modified in critical section with their twins and constructs diff records of this session. These changes are then piggybacked with message of leaving critical section sent to manager. The critical section manager updates the write-notice list, and appends new diff records to previous ones.

Barrier operation performs the central node, which is the node, where the main part of program is executed. First barrier manager gets the list of modified pages from every node. Second, it makes decision for each page about changing page manager. Third, it acknowledges page manager nodes, to get diffs from others for each owned page. During this step, also the changes made in critical section are transferred to the corresponding page manager's nodes, and after that critical section changes are cleared. In final step, the central node sends to every node the corresponding list of pages that should be invalidated.

3 Reducing system overhead

We have implemented several techniques to reduce overhead in various parts of the system. They are intended to minimize the number of messages, the amount of data used for cache coherence protocol related information transfers.

3.1 Dynamic page and critical section manager

Using dynamically migrating page manager node enables the system to maintain page coherence where the page is frequently changed, which eliminate unnecessary diffs transfers. For example, in application with low sharing pattern, after first barrier, the system will detect single writer page usage, and automatically reassign page managers.

Using dynamically changing manager of critical section may reduce as much as twice the amount of data transferred on entering and leaving the critical section. Suppose that one node enters a critical section and receives list of invalid pages. Upon leaving the critical section, node detects, that all previously invalid pages were accessed, and thus all the previous changes in critical section are here. As this data may be huge, and there is no assumption that current critical section manager may need them, it is worth to reassign section manager to this node, and do not perform data transfers. Once node has decided to become critical section manager, notifying also the central node will help other nodes to find easily the critical section manager later.

3.2 Page data compression

On page fault, the faulting node fetches full page from a page manager node. This approach simplifies system internal architecture, and alleviates need for complex tasks such as page versioning, diff storage and garbage collection. On the other hand, transferring full pages over the network may greatly impact on overall system performance. We introduced page data compression module, which compresses page data, and significantly reduces the size of data for transfer.

The compression algorithm is not fixed, and the page compression module can be extended by the user. This makes DSPC system highly flexible to user application needs.

The basic compression algorithm used in DSPC system, tries to compress the page, by encoding most frequent byte with 1 bit. A 512 byte bit vector is constructed, where each bits tells whether this is frequent byte or not. The bit vector itself is also compressed two times with the same schema. In case of page that is filled with same byte, the compressed output size is 9 bytes. On the other hand, it is obvious, that using this compression schema, in some cases output size may be greater than page size. If we encounter such situation, no compression is performed.

3.3 Early updates

There are two places, where early data fetching and updating invalid pages may be applied.

The first place is at the final step of barrier operation, when each node invalidates its locally cached pages. Here we introduced adaptive technique – *early page update*, which based on observations on page usage, may decide to fetch a number of pages in one request from appropriate page managers, before actually the page will be accessed. Doing this minimizes overall system overheads, and eliminates page faults that may occur when updated pages are accessed later.

The second place is when the call enters critical section. By protocol, node receives only the list of pages to be invalidated, related to the critical section. We introduce another adaptive mechanism - *early critical section changes fetching*, where system can decide to early fetch diffs of several pages from critical section manager, based on frequency of usage.

For Ethernet networks, sending one byte or one kilobyte message, considerably takes almost the same time, which comes from the fact that latency for accessing the media is very high. Using early update mechanisms practically reduces overhead time, when getting several pages with one message.

3.4 Barrier with asynchronous invalidation

During final step in barrier operation, the central node sends to each node a corresponding list of pages that should be invalidated. The task of each node is to invalidate its local copy of the page, and then proceed with execution on the program. We see that at the start of this step, all page managers have finished obtaining their appropriate changes, and have up-to-date page data. Thus, executing invalidating operations can be done concurrently. Also at this step, each node considers that barrier operation has finished, and continues execution of the program.

All the implemented adapting mechanisms are configurable at runtime, and can be enabled, disabled and adjusted depending on application needs.

4 Performance Evaluation

Evaluation was done with well-know benchmark tests taken from SPLASH2 [6] (Barnes, LU, Radix, Water), NAS Parallel Benchmark [1] (IS, EP), and two well-known applications (SOR, TSP) [4] from Rice University. The computing environment consisted of eight workstations with Pentium 1 GHz processors, running Windows 2000/XP, and connected with 100 Mbps switched Ethernet.

Table 1 shows characteristics, sequential time, and speedups of tested application and speedup results.

Table 1: Problem Characteristics and Speedups

	Data Set	Sequential Time (sec.)	Speedup
Barnes	16K bodies	46.80	6.24
LU	4096 x 4096	706.60	4.32
EP	2^{28}	130.25	8.00
IS	2^{26}	285.10	6.48
Radix	2^{24}	47.00	6.21
SOR	8192 x 8192	395.40	7.14
TSP	19 cities, r12	56.90	6.89
Water	1792 moles	159.60	6.25

5 Conclusion

In this paper we discussed newly developed DSM system implemented in Windows environment. We proposed new cache coherence protocol and adapting techniques, that balance between simplicity and performance. Performance evaluation shows that most tested benchmarks achieve high speedups.

References

[1] D. Bailey, L. Dagum, E. Barszcz and H. Simon. NAS parallel benchmark results. In *Supercomputing*, pages 386–393, 1992.

[2] A. Ghazaryan. On system for distributed parallel computations - dspc. In *Proc. of the Int'l Conference on Computer Science and Information Technologies (CSIT 2001)*, August 2001.

[3] L. Iftode, J. P. Singh, and K. Li. Scope consistency: A bridge between release consistency and entry consistency. In *Proc. of the 8th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'96)*, pages 277–287, June 1996.

[4] H. Lu, S. Dwarkadas, A. Cox, and W. Zwaenepoel. Quantifying the performance differences between pvm and treadmarks. *Journal of Parallel and Distributed Computing*, 43, No. 2:65–78, June 1997.

[5] H. Nalbandyan. On parallel processing with distributed shared memory. In *Proc. of the Int'l Conference on Computer Science and Information Technologies (CSIT 2003)*, pages 317–320, September 2003.

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

Բաշխված ընդհանուր հիշողությունով համակարգի մասին

Հ. Զ. Նալբանդյան

Ամփոփում

Աշխատանքում դիտարկվում է զուգահեռ հաշվարկների համար ստեղծված նոր բաշխված ընդհանուր հիշողությունով համակարգի նկարագրությունը: Քննարկված են համակարգի կառուցվածքը, հիշողության մոդելը, տվյալների համապարփակության արձանագրությունը: Իրագործվել են մի շարք խնդիրներ համակարգի արտադրողականությունը գնահատելու նպատակով: Ստացված արդյունքները վկայում են համակարգի բարձր արագագործության մասին: