

Distributed Shared Memory: Bridging the Granularity Gap

[position paper]

Ayal Itzkovitz*

Department of Computer Science, CIMS
New York University
ayali@cs.nyu.edu

Assaf Schuster

Computer Science Department
Technion – Israel Institute of Technology
assaf@cs.technion.ac.il

Abstract

In this position paper we explore a very recent technique, called `MULTIVIEW`, its applications, and its implications on the design and usage of distributed shared memory systems (DSMS) [6]. `MULTIVIEW` can be used to bridge the gap between the large, fixed-size memory pages handled by the hardware and operating system, and the relatively-small, varying-size minipages that are used by applications. Using `MULTIVIEW`, the distributed shared memory system can adapt to the native granularity of the application in a natural way. While originally proposed for supporting fine-granularity sharing, `MULTIVIEW` can also be used by all the accompanying DSM services, including sharing across machines, protection and consistency manipulation, detecting racing accesses, collecting garbage, tracing true sharing by application threads, etc. Thus, `MULTIVIEW` simplifies the design and usage of DSM systems in a significant step towards making them a popular technology.

1 Introduction

Consider the services that are supported by a distributed shared memory system (DSM), including sharing across machines, protection and consistency manipulation, detecting racing accesses, garbage collection, tracing true sharing by different threads, etc. Clearly, these services use the application *native granularity*; i.e., all these services manipulate fragments of data that are being used by the application as integral units (e.g., variables, objects, data-structures). Unfortunately, the hardware and operating systems' memory managers are based inherently on large, fixed-size pages, and do not adapt to the application native granularity.

The basic mechanism proposed in Ivy, the first DSM [11], makes use of the protection mechanism which is provided per

*Much of this work was done while the author was with the Computer Science Department at the Technion, Israel.

memory page by a cooperation of the hardware and the operating system. The large (and increasing) page size defines the unit of sharing that can be supported using the protection mechanisms. In a distributed cluster, the penalty for sharing in granularity different from that of the application variables and objects is very high, resulting in false sharing, consistency complications, inflated communication volume, and high overhead on accompanying services.

Thus, since the introduction of Ivy, most of the work in the evolution of DSMS attempted to overcome the obstacles put forth by this problem. Despite the large amount of work that was invested and the resulting progress (see for instance [1, 3, 12, 14]), the problem was never completely solved. Because of this reason many still consider DSMS as doomed, and it was claimed that most work in this field is incremental [4].

We present here a simple approach, called `MULTIVIEW`, that can be used to bridge the gap between the native memory elements of the application and the fixed operating system page size. We show that a non conventional memory layout eliminates false sharing, provides efficient support for strict consistency, reduces communication to minimum, and allows natural integration with satellite services, such as garbage collection and data race detection. Indeed, using `MULTIVIEW` results in a substantial simplification of protocols and a corresponding reduction of overhead, thus making DSMS at the same time both simpler and more efficient.

We skip in this position paper the presentation of `MULTIVIEW`, which may be found in [6], restricting ourself to mentioning basic concepts and conclusions from our experience in implementing `MULTIVIEW` in a DSM system called `MILLIPAGE`. We discuss an avenue of potential developments extending from the basic idea of `MULTIVIEW`. Interestingly, much of the research can be viewed as “Back to the Future”, revisiting previous works that were handicapped by the fixed, large page assumption, repeating these studies under the new model where this is a non-issue.

2 Bridging the Gap: From Pages to Minipages

The basic idea underlying `MULTIVIEW` is as follows. Consider a set of variables whose total size is smaller than that of a page. The page containing those variables may be mapped to different, non-overlapping, virtual address regions, called

views, which can be used to access the variables exclusively. It is now possible to manage a separate access privilege for each variable using its respective view. Moreover, an independent consistency protocol can be implemented for each variable. Because each memory element (memory object, variable) can now be managed independently regardless of its size, we call it a *minipage*. Minipage sizes vary; they can be as large as the virtual page size or as small as the basic memory addressing unit.

An additional, separate view may be constructed, called the *privileged view*, whose protection is fixed and set to `ReadWrite`. The DSM server threads may use it at all times to access the memory, and are thus not constrained by the DSM memory protections, as determined by the consistency guarantees imposed on the application views. The privileged view may also be used by zero-copy communication protocols to directly send/receive minipages from/to the user space.

Figure 1 depicts a sample memory layout of a general DSM system that uses MULTIVIEW.

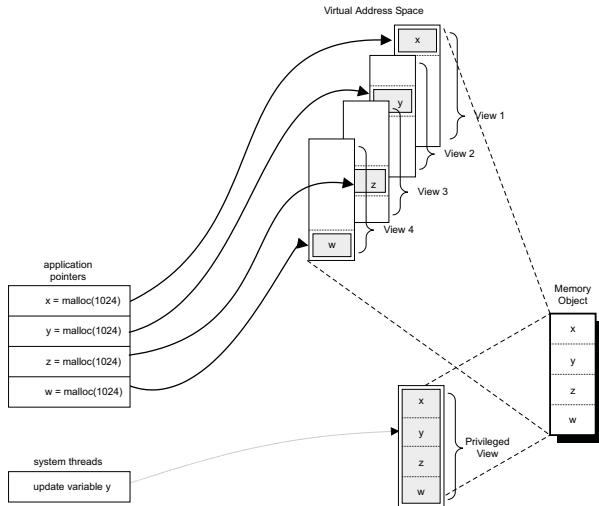


Figure 1: The basic memory layout of an application using MULTIVIEW and four variable x , y , z , and w . Application threads access the variables through four different application views while system threads (DSM server threads) access those variables through the privileged view.

3 Simplicity and Efficiency: The Thin Protocol Layer

Overcoming false sharing and the struggle to provide per-variable coherency guarantees led to techniques which trade additional memory resources and computational overhead with higher performance of the DSM. One of the most popular ideas in this direction is the release consistency (RC) protocol(s).

RC systems provide multi-writers consistency, by allowing concurrent writes to variables residing in the same page. RC is claimed to overcome false sharing thanks to the protocol layer, which, in turn, pays in memory and computing overheads. However, false sharing is not really eliminated,

but appears in a different custom: processors are busy in creating twins and applying diffs to sections of pages that contain variables which they do not access at all!

The major benefit of using the RC protocol appears to be the relative resilience of the memory layout to sharing patterns. We believe, however, that DSM systems should rather focus on the protocol layer, to provide a thin-layer that consume as little memory and computational overhead as possible. Using MULTIVIEW, a strict consistency protocol can be implemented in a straightforward way complying to the single-writer/multiple-reader (SWMR) semantics.

Consider a system which uses MULTIVIEW, where variables are distributed among distinct minipages. When the program contains no data races then no concurrent conflicting accesses (which include writes) are expected. Hence, sequential consistency (e.g., through the SWMR protocol) is perfectly suited to this situation: it guarantees consistent modifications while avoiding protocol-related overheads. Indeed, our implementation, as reported in [6], shows initial performance results that support this claim.

3.1 The DSM System Prototype

We have implemented a prototype DSM system called MILLIPAGE. The main goals in building MILLIPAGE were experimenting with MULTIVIEW and exploring the use of a thin protocol layer.

The programming model in MILLIPAGE is sequential consistency, which is implemented through the SWMR protocol. Aside from the initial setting of the maximal number of views and the size of the shared-memory, the allocation process is transparent from the programmer's point of view. The `malloc`-like API returns a pointer which may point to any of the application views (but never to the privileged one).

The thin protocol layer implements the DSM functionality in a straightforward way. On each host a single MILLIPAGE process is started, running both the application and server threads. One of the processes is the *manager*, whose role is to maintain the minipage-table, containing directory information of minipage copy locations, minipage sizes, and the association of view addresses with their minipages. On an access fault, a request message is sent to the manager, which resolves the minipage boundaries (offset and size), and reroutes the request to the appropriate target host. The design ensures that buffering of requests is only done at the manager, resulting in further simplification of the protocol.

4 Observations on Using MULTIVIEW

Clearly, while there are many advantages in using MULTIVIEW, the limitations of this technique and its behavior should be further understood.

4.1 Performance Issues and Limitations

MULTIVIEW heavily uses the virtual-to-physical memory translation mechanism. Loosely speaking, MULTIVIEW attaches a separate PTE to each individual minipage.

We have studied this issue with MILLIPAGE. For less than 32 views (i.e. less than 32 minipages per page) and for a moderate size of memory objects (up to 32MB) there is a minor slowdown of 1-4%, attributed to the high rate of TLB misses. At certain breaking points, which appear precisely when the L2 cache becomes congested with PTEs (and regardless of the data misses), the slowdowns become significant.

We thus get a tradeoff. For an application with a reasonably local memory access pattern, the advantages of MULTIVIEW dominate the negligible overhead coming from the excessive TLB misses. However, when the application attempts to access too many minipages at the same time (more than 128K for our Pentium II having 1/2MB L2 cache), the translation and protection information exhaust the cache, imposing overhead which is too high.

4.2 Locality and Virtual Locality

The principle of locality implies that in many cases nearly allocated minipages are also accessed together. However, these minipages will be accessed through pages which belong to different views. Thus, the PTEs corresponding to these pages do not reside contiguously in memory, resulting in fetching irrelevant PTEs and failing to fetch the relevant ones when a cache miss occurs. We call the locality of access in the page table *virtual locality*.

As discussed earlier, reducing the virtual locality has a major performance implication when many views are used. One way in which we could improve the virtual locality is to interleave the virtual pages which belong to different views. Alternatively, the PTEs in the page table itself could be interleaved.

4.3 Adapting to Application Granularity via Composed Views

A natural extension of MULTIVIEW is to overlap views and compositions of views. A minipage can be associated with a set of views, each representing a different consistency or a different sharing granularity [5]. Accessing the same variable through different views invokes different protocols, which in turn, may imply a different behavior.

This can also improve the performance of fine-grain DSM systems which suffer from poor performance during coarse program phases [2]. For example, consider a matrix whose rows are associated with minipages. At certain points in the program, it might be desirable to read the entire matrix, bringing in all the minipages (matrix rows) at once. In order to do so, an additional, large minipage could be defined, which overlaps with all the minipages associated with the matrix rows. When the large minipage is accessed, the DSM system ensures it is brought in, which implicitly means aggregating and prefetching all the sub-minipages.

Figure 2 demonstrated the use of composed views to provide multiple sharing granularities that can dynamically be employed for a running application.

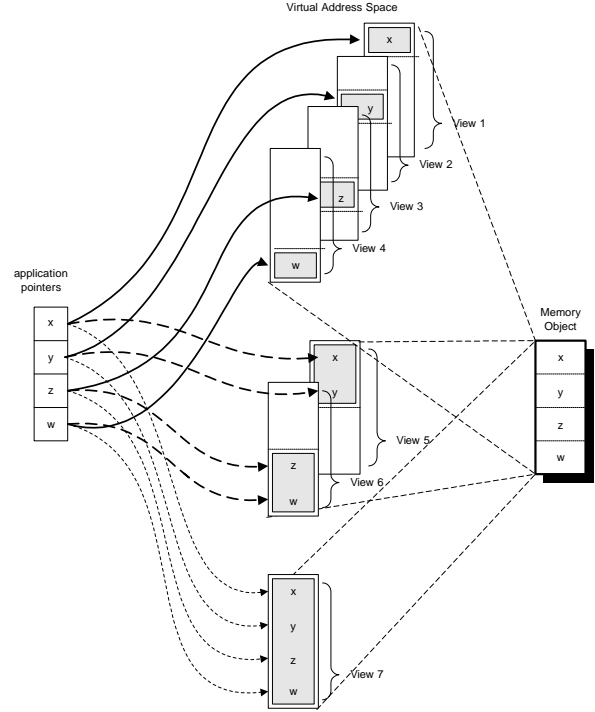


Figure 2: Using composed views for providing adaptive sharing granularity. Application pointers are manipulated to use different sets of views to arbitrate between different sharing granularity levels.

4.4 The Importance of Relaxed Consistencies

As we stated before, relaxed consistencies are considered very useful in overcoming problems like false sharing and unnecessary page shuttling. However, relaxed consistencies involve non trivial computation and memory overheads. Along the good performance we got in our sequential consistency implementation, the benefits of using relaxed consistency should further be explored.

An earlier study [15] showed that sequential consistency with a relatively fine grain access control (256 bytes) performs as well as relaxed consistency protocols with coarse grain memory access control (more specifically, home-base lazy release consistency), however, this study was done for fixed-size fine-grain access control. We predict that since access control can now be customized and adjusted to the application native needs, the performance advantages might further lean towards the strict protocols.

5 Relations to Other DSM Services

The discussion above concentrated on the DSM major task, namely, to share memory among hosts. As it turned out, using MULTIVIEW may improve other applications and DSM services as well. We mention some of these below.

5.1 Data Race Detection

Programming errors in parallel programs commonly show themselves in unsynchronized accesses to data, called *data-races*. Unfortunately, the complexity of discovering such bugs is very high, which gave rise to run-time detection techniques. In run-time methods the detection overhead may become very large when applied to memory fragments that are irrelevant to the checked program variables, as is the case in page-based DSMs.

In contrast, using MULTIVIEW we can ensure that only the relevant minipage is checked, by manipulating the protection of the associated page. Since the minipages access pattern represents the variables access pattern by the application, data race detection is efficiently integrated in the DSM by embedding a detection protocol in the minipage consistency management.

We have recently used MULTIVIEW for this purpose and found a potential for dramatic performance improvement [7].

5.2 Distributed Garbage Collection

MULTIVIEW can help collecting garbage distributively in native application granularity, at the level of a standard DSM service. The main contribution of MULTIVIEW here is that marshaling and tracing pointer assignments are easy when different objects are pointed via different pages. Thus, implementing counting methods, which are best suited for distributed garbage collection, become a lot easier and employ less penalty.

We integrated a distributed garbage collector in our system MILLIPAGE using the method developed in [10].

5.3 Tracing True Sharing - Collecting Memory Access Information

As was mentioned for data race detection mechanisms, tracing and collecting information regarding accesses to *application-level* shared variables becomes a natural service of the DSM. Once again, the basic property is that the minipages access pattern also represents the variables access pattern. Thus, it is relatively simple to implement a light-weight (remote) access logging mechanism, maintaining access histories per variable. This gives rise to all kinds of optimization techniques, which, for instance, may relocate threads and balance the load according to the true-sharing pattern [13] or deal with automatic prefetching and recording/replaying of memory accesses for improving various relaxed consistency implementations [9].

5.4 Global Memory Systems

Recently, there has been a lot of interest in harnessing remote memories for storing memory pages, thus establishing an additional level in the memory hierarchy. It was shown that using subpages as the basic transfer unit leads to substantial performance boost [8]. Clearly, MULTIVIEW can be used for implementing subpages (minipages in MULTIVIEW terminology) in global memory systems. Furthermore, prefetching can be implemented, by referencing large,

composed minipages. The sub-minipages may be pipelined through the communication layer, so that when the first arrives it can be instantly used while the others are still flowing in.

6 Conclusions

In this paper we presented a newly introduced technique called MULTIVIEW and surveyed its use both for building efficient fine-grain DSM systems and for other satellite services. We briefly described MULTIVIEW and the ways it can be used to bridge the gap between the memory units, managed by the operating system, and the natural sharing units of the applications in a DSM system. In addition, we mentioned areas of research in which we suggested the employment of MULTIVIEW to enhance capabilities and improve the performance.

References

- [1] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer*, 29(2):18–28, Feb. 1996.
- [2] C. Amza, A. L. Cox, K. Ramajamni, and W. Zwaenepoel. Trade-offs between false sharing and aggregation in software distributed shared memory. In *Proc. of the Sixth ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'97)*, pages 90–99, June 1997.
- [3] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. Munin: Distributed Shared Memory Using Multi-Protocol Release Consistency. In A. I. Karshmer and J. Nehmer, editors, *Operating Systems of the 90s and Beyond*, pages 56–60. Springer-Verlag Lecture Notes in Computer Science #563, July 1991.
- [4] J. B. Carter, D. Khandekar, and L. Kamb. Distributed shared memory: Where we are and where we should be headed? In *Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, pages 119–122, May 1995.
- [5] A. Itzkovitz, N. Niv, and A. Schuster. Dynamic Adaptation of Sharing Granularity in DSM Systems. In *Proc. of the 1999 Int'l Conf. on Parallel Processing (ICPP'99)*, Sept. 1999. To appear.
- [6] A. Itzkovitz and A. Schuster. MultiView and Millipage — Fine-Grain Sharing in Page-Based DSMs. In *Proc. of the 3rd Symp. on Operating Systems Design and Implementation (OSDI'99)*, pages 215–228, New Orleans, Feb. 1999.
- [7] A. Itzkovitz, A. Schuster, and O. Zeev-Ben-Mordechai. Towards Integration of On-the-fly Data Race Detection in DSM Systems. Technical Report CS-0948, Technion, October 1998.
- [8] H. A. Jamrozik, M. J. Feeley, G. M. Voelker, J. Evans II, A. R. Karlin, H. M. Levy, and M. K. Vernon. Reducing Network Latency Using Subpages in a Global Memory Environment. In *Proc. of the 7th Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, pages 258–267, Oct. 1996.
- [9] P. Keleher. Tapeworm: High-level abstractions of shared accesses. In *Proc. of the 3rd Symp. on Operating Systems Design and Implementation (OSDI'99)*, Feb. 1999.
- [10] D. Kogan and A. Schuster. Collecting Garbage Pages with Reduced Memory and Communication Overhead. In *Proc. 5th European Symposium on Algorithms*, pages 308–325, Graz, September 1997.
- [11] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. In *Proc. of the 5th Annual ACM Symp. on Principles of Distributed Computing (PODC'86)*, pages 229–239, Aug. 1986.

- [12] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proc. of the 21th Annual Int'l Symp. on Computer Architecture (ISCA '94)*, pages 325–336, Apr. 1994.
- [13] A. Schuster and L. Shalev. Using Remote Access Histories for Thread Scheduling in Distributed Shared Memory Systems. In *12th Intl. Symp. on Distributed Computing*, pages 347–362, Andros, September 1998.
- [14] E. Speight and J. Bennett. Brazos: A Third Generation DSM System. In *First Usenix-NT Workshop*, pages 95–106, Aug. 1997.
- [15] Y. Zhou, L. Iftode, K. Li, J. P. Singh, B. R. Toonen, I. Schoinas, M. D. Hill, and D. A. Wood. Relaxed consistency and coherence granularity in dsm systems: A performance evaluation. In *Proc. of the Sixth ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP '97)*, pages 193–205, June 1997.