Distributed Shared Memory : A Page-Based Approach

K Pavan Kumar, V Sai Vamsi Krishna

K Pavan Kumar Computer Science & Engineering, Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya

V Sai Vamsi Krishna Computer Science & Engineering, Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya

Abstract - In 1986, Kai Li's PhD thesis, "Shared Virtual Memory on Loosely Coupled Microprocessors," introduced the concept of Distributed Shared Memory (DSM) systems, which has since facilitated further research in this field. One approach to implementing DSM in distributed systems is the page-based method, which utilizes virtual memory techniques to map pages of a process' address space onto the physical memory of multiple nodes. The system automatically moves pages between nodes as needed, ensuring coherence and consistency for shared page updates.

Page-based DSM offers several advantages, including improved performance due to reduced communication overhead and better data locality, simplified programming models that resemble centralized systems, and improved scalability by allowing additional nodes to join the system and expand memory capacity. However, its suitability for certain distributed systems depends on their specific requirements and characteristics. Page-based DSM has been applied in various distributed systems, including high-performance computing, cloud computing, and distributed databases. Nonetheless, remote memory access overheads, cache coherence issues, and the necessity of a well-connected network remain potential limitations of this approach.

Key words: Distributed memory, Shared memory, Paging, Virtual memory, memory coherence, memory consistency

1. INTRODUCTION

Distributed Shared Memory (DSM) is a programming abstraction or to say a form of memory architecture that enables physically separated memories of multiple nodes in a distributed computing system to be addressed and accessed as a single shared memory address space, as if it were a local memory space. The term "shared" does not mean a single centralized memory, but instead indicates that the address space is being shared. Shared memory is a type of memory that multiple programs can access at the same time to enable communication among them or to avoid duplicating data. This means that the same memory location can be indicated by the same physical address on two processors. It's a useful way to transfer data between programs, and it can be used in programs running on either a single processor or multiple separate processors. Shared memory can also refer to using memory for communication within a single program, such as among its various threads. A DSM system is a type of system that implements the shared-memory model on a physically distributed memory system. DSM allows processes running on different nodes to share data and communicate with each other through network interfaces with necessary network bandwidth and speed, depending on the implementation and requirements, even though they are physically separate and potentially located in different geographical locations.

DSM can be realized through either software or hardware, and some systems may use a combination of these approaches. The hardware approach involves extending traditional caching techniques to scalable architectures using cache coherence circuits and network interface controllers. The software approach involves operating system and library implementations that use virtual memory-management mechanisms to achieve sharing and coherence. *Software DSM systems offer the advantage of being able to organize the shared memory region in various ways, providing flexibility in implementation*. Software DSM systems offer different

ways to organize the shared memory region, such as page-based, shared-variable, or object-based approaches. In a page-based approach, virtual memory is used to manage the shared memory region. In a shared-variable approach, routines are used to access shared variables. In an object-based approach, shared data is accessed through an object-oriented approach. Additionally, DSM can also be implemented through compiler implementations, where shared accesses are automatically converted into synchronization and coherence primitives.

With processor and memory speed reaching their physical limits(See figure 2.1, 2.2), using multiprocessors has become a viable option to increase computing power. Two popular types of parallel processors are tightly coupled shared-memory multiprocessors and distributed-memory multiprocessors.

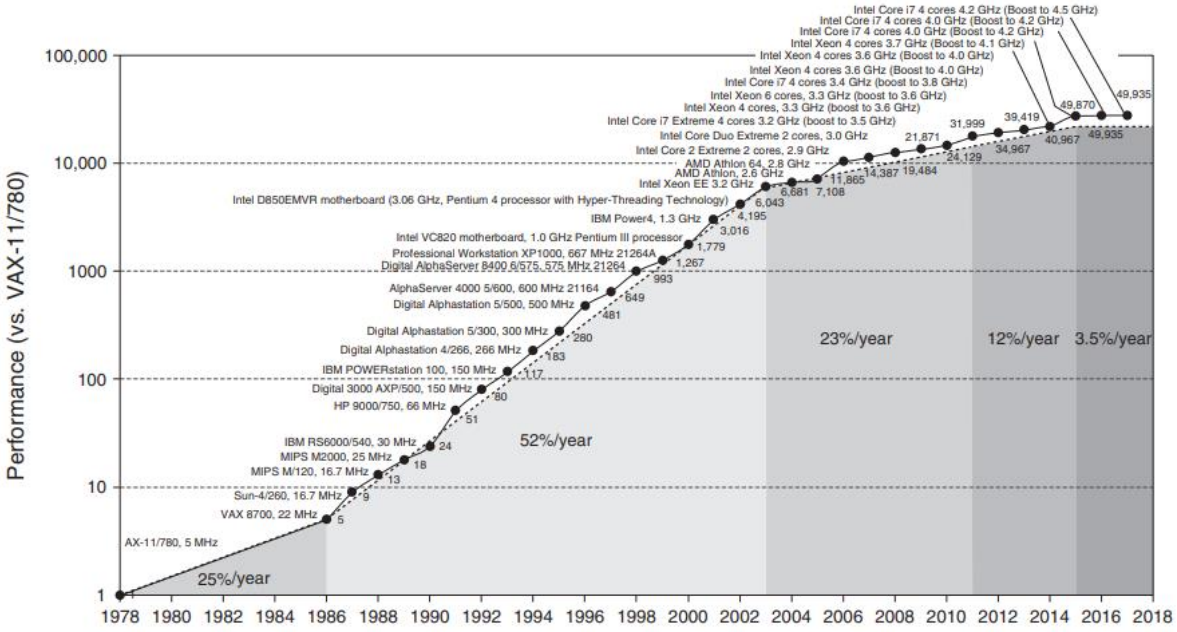
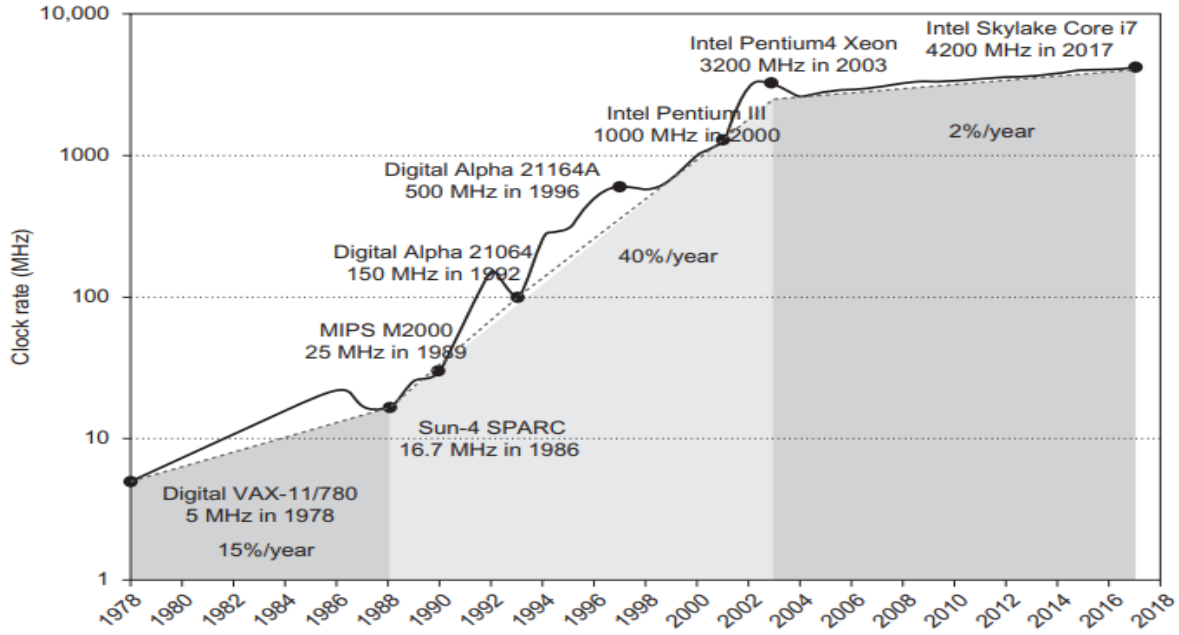


Figure 2.1 The chart provides an overview of the growth in processor performance over the past 40 years, using the VAX 11/780 as a reference point for program performance measured with SPEC integer benchmarks. It shows that while growth in processor performance was mainly driven by technology before the mid-1980s, more advanced architectural and organizational ideas such as RISC architectures led to a significant increase in growth from 1986 to 2003. However, the limits of power due to Dennard scaling and instruction-level parallelism slowed down uniprocessor performance until 2011, followed by further limitations due to Amdahl's Law until 2015. Since then, the end of Moore's Law has resulted in a much slower annual improvement rate in processor performance



.

Figure 2.2 Figure 2.1 illustrates the growth in the clock rate of microprocessors over time. From 1978 to 1986, the clock rate increased by less than 15% annually, while performance improved at a rate of 22% per year. In the "renaissance period" from 1986 to 2003, there was a significant increase in performance, with clock rates increasing by almost 40% per year. Since then, clock rates have remained relatively stable, growing by less than 2% per year, while single processor performance has improved at a rate of only 3.5% per year in recent times.

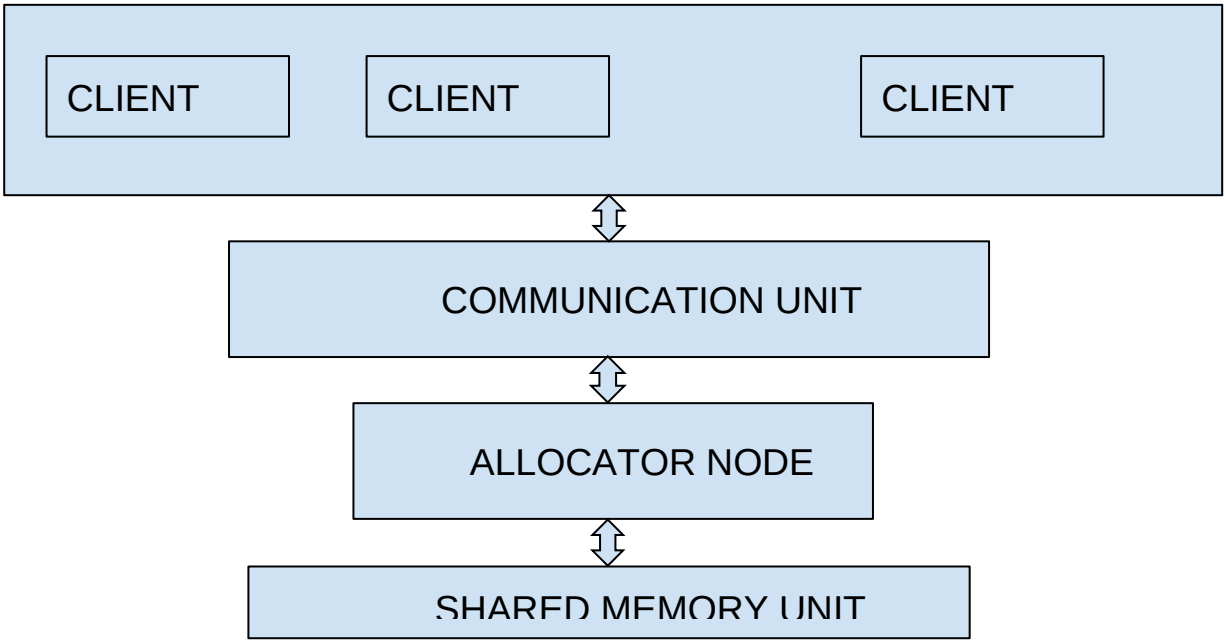
Tightly coupled multiprocessors have multiple CPUs and a single global physical memory, making it easier to program as an extension of a single-CPU system. However, this type of system has a bottleneck due to memory access via a common bus, limiting system size to tens of processors. Distributed-memory multiprocessors, on the other hand, are not limited by this bottleneck as they consist of independent computers connected by a high-speed interconnection network. This allows for many more processors than a tightly coupled system. Communication in distributed-memory systems is typically done through a message-passing paradigm, but recent systems have implemented shared-memory abstractions on top of message-passing systems, allowing programmers to use shared-memory paradigms. Distributed shared memory provides a virtual address space shared among processes on loosely coupled processors, resulting in ease of programming, portability, low cost, and scalability without hardware bottlenecks.

The purpose of this paper is to provide an introductory overview of distributed shared memory systems through a page-based approach using virtual memory. *Run multithreaded shared memory parallel programs on a cluster of machines, using paging(virtual memory) to give the illusion of real shared memory.* The page-based DSM approach provides many benefits for distributed systems. First, it can provide a significant performance boost compared to message-passing approaches due to the reduced communication overhead and improved data locality. Second, it simplifies the programming model, allowing developers to write code that is more similar to that of a centralized system. Third, it can improve the scalability of the system by allowing additional nodes to join the system and expand memory capacity as needed.

Page-based DSM has been used in a variety of distributed systems, including high-performance computing, cloud computing, and distributed databases. However, it also has some limitations, such as high overheads for remote memory accesses, potential cache coherence issues, and reliance on a well-connected network. Therefore, the suitability of page-based DSM depends on the specific requirements and characteristics of the distributed system.

Threads are a widely used programming concept that enables multiple threads of execution within a program to share a common memory space and resources like open files. They offer concurrent programming and true parallelism on multiprocessor systems. Linux has a unique implementation of threads where all threads are implemented as standard processes, and the kernel does not have a specific concept of a thread. In Linux, threads are just a way of sharing resources between processes, unlike Windows or Solaris which have explicit kernel support for threads. Linux considers threads to be lightweight processes that share resources with other processes. Each thread has its own unique *task_struct* and appears to the kernel as a normal process. In contrast, other operating systems have a process descriptor that points to different threads, and the threads describe the resources they possess. Linux is different in that it simply has multiple processes that share specific resources. Overall, this approach to threads in Linux is quite elegant.

3.SYSTEM ARCHITECTURE:



4.IMPLEMENTATION

The Distributed Shared Memory is mainly build around three components:

- ➔ The allocator component processes arguments, creates a child process, and allocates a shared region for synchronization between child processes. Once the client program starts, the allocator manages and records the memory allocation and usage of the shared memory region.
- ➔ The child processes are forked from the allocator before the creation of shared memory. They manage the connection and data transfer with their corresponding client program and communicate with the allocator through shared data.
- ➔ The client program interface includes functions that can be called by the client programs.

The structure of the DSM system is as follows.

- Node processes run on separate hosts. All node processes run the same client program.
- Node processes cooperate by reading and writing memory pages in a shared virtual address space.
- Each node process can access the whole shared virtual address space, and each page is accessed at the same virtual address in all node processes (with different pages being accessed at different addresses).
- Client programs that are run by the node processes are linked to an SM library, which implements the functionality that allows access to the shared memory.
- The library handles page faults when the client attempts to read pages that are not locally available, or when it tries to write to pages that are not both available and write-enabled.
- The library also implements synchronization functions and allows clients to allocate shared memory from the virtual address space.

When a process tries to reference or access a page which is not present locally, the processor treats it as invalid memory reference which then triggers a page fault, and transfers control from program to the operating system. The OS then must :

1. Determine the location of data.
2. Obtain an empty page frame in memory to use as a container for the data.
3. Load requested data into the page frame.
4. Update page table to refer to the new page frame.
5. Return control to the program, transparently retrying to execute the instruction that caused the page fault.

5.CONCLUSION:

In our assessment of current distributed shared memory research, we acknowledge significant progress made over the past ten years, with ongoing work in the field. The breadth of research suggests that no single technique is capable of providing an efficient implementation on its own. Hardware-based approaches face limitations due to insufficient technology but can be complemented by advanced software methods. Conversely, relying solely on software approaches results in communication latencies, even when techniques are employed to reduce them. The use of language-level primitives can reduce such overhead, but language extensions add programming complexity to the system.

Our goal is to assess the effectiveness of different compression algorithms by testing them against a range of representative applications and analyzing their respective performance benefits. To this end, we will be comparing the outcomes of each compression technique and examining how they operate under varying conditions, such as varying memory access patterns and high-speed network configurations. We are currently working on creating specialized compression algorithms that can optimize memory access patterns and high-speed networks, which would improve system performance. These innovative algorithms would work alongside existing memory access patterns and networks, enabling better overall performance for the system.

Additionally, it is possible to decrease network overhead by eliminating the requirement for handshake between the nodes and implementing them instead at the interrupt level. This can result in boosted performance levels for the system.

REFERENCES:

1. Kai Li and Paul Hudak, 'Memory coherence in shared virtual memory systems', Proceedings 5th ACM SIGACT-SIGOPS Symposium of Principles of Distributed Computing, Canada, ACM Press.
2. Kai Li and Paul Hudak, 'Memory coherence in shared virtual memory systems', ACM Trans. Computer Systems
3. [Tanenbaum1992] - Bal, H., E., Tanenbaum, A, S., "Distributed programming with shared data," International Conference on Computer Languages '88, October 1988.
4. Brett D. Fleisch, 'Distributed shared memory in a loosely coupled environment', Ph.D. dissertation, Computer Science Department, University of California, Los Angeles, CA, USA, September 1989.
5. R. M. Metcalfe and D. R. Boggs, 'Ethernet: distributed packet switching for local computer networks', Communications of the ACM, 19, (7), 395–403 (1976).
6. [AGARW90] - Agarwal, A., Lim, B., Kranq D., Kubiatowin, J., "APRIL: A Processor Architecture for Multiprocessing," Proceedings of the 17th Annual InternationalSymposium on Com$e;. Architecture, 1990
7. Message Passing Interface (MPI) standard. http://www-unix.mcs.anl.gov/mpi/
8. Jelica Protic, Milo Tomasevic, Veljko Milutinovic, "A Survey of Distributed Shared Memory Systems" Proceedings of the 28th Annual Hawaii International Conference on System Sciences, 1995.
9. V. Lo, "Operating Systems Enhancements for Distributed Shared Memory", Advances in Computers, Vol. 39, 1994.
10. B. Chapman, T. Curtis, S. Pophale, S. Poole, J. Kuehn, C. Koelbel, L. Smith "Introducing OpenSHMEM, SHMEM for the PGAS Community", Partitioned Global Address Space Conference 2010.
11. B. Nitzberg and V. Lo, 'Distributed shared memory: a survey of issues and algorithms', IEEE Computer, 24, (8), 52–60 (1991).
12. Pete Keleher, Alan L. Cox, Sandhya Dwarkadas, and Willy Zwaenepoel. An evaluation of software-based release consistent protocols. Journal of Parallel and Distributed Computing, 29(2):126--141, September 1995.
13. Pete Keleher, Lazy Release Consistency for Distributed Shared Memory, PhD thesis of Rice University, Huston, Texas, January, 1995.
14. Patterson, David A.; Hennessy, John L. (2019). *Computer Architecture: A Quantitative Approach* (6th ed.). Burlington, Massachusetts: Morgan Kaufmann. ISBN: 978-0-12-811905-1.

Author's Profile

1. V. Sai Vamsi Krishna, Student, B.E. Computer Science and Engineering, Sri Chandrasekharendra SaraswathiViswa Mahavidyalaya deemed to be university, Enathur, Kanchipuram, India.

2. K. Pavan Kumar, Student, B.E. Computer Science and Engineering, Sri Chandrasekharendra SaraswathiViswa Mahavidyalaya deemed to be university, Enathur, Kanchipuram, India.