Shared memory vs Distributed memory

Cengiz Önkal
215045

Abstract

This paper discusses and compares Shared Memory structure to Distributed Memory structure.

1 Introduction

Shared memory systems form a major category of multiprocessors. In this category, all processors share a global memory. Communication between tasks running on different processors is performed through writing to and reading from the global memory. All interprocessor coordination and synchronization is also accomplished via the global memory On the other hand **distributed memory** refers to a multiprocessor computer system in which each processor (CPU) has its own private memory.

2 Shared Memory

The simplest shared memory system consists of one memory module (M) that can be accessed from two processors. Requests arrive at the memory module through its two ports. An arbitration unit within the memory module passes requests through to a memory controller. If the memory module is not busy and a single request arrives, then the arbitration unit passes that request to the memory controller and the request is satisfied. The module is placed in the busy state while a request is being serviced. If a new request arrives while the memory is busy servicing a previous request, the memory module sends a wait signal, through the memory controller, to the processor making the new request. In response, the requesting processor may hold its request on the line until the memory becomes free or it may repeat its request some time later. [1]

Two main problems need to be addressed when designing a shared memory system: performance degradation due to conflict, and coherence problems. Performance degradation might happen when multiple processors are trying to access the shared memory simultaneously. A typical design might use caches to solve the contention problem. However, having multiple copies of data, spread throughout the caches, might lead to a coherence problem.

Based on the interconnection network used, shared memory systems can be categorized in the following categories.

2.1 Uniform Memory Access (UMA)

In the UMA system a shared memory is accessible by all processors through an interconnection network in the same way a single processor accesses its memory. All processors have equal access time to any memory location. The interconnection network used in the UMA can be a single bus, multiple buses, or a crossbar switch. Because access to shared memory is balanced, these systems are also called SMP (symmetric multiprocessor) systems. Each processor has equal opportunity to read/write to memory, including equal access speed. Commercial examples of SMPs are **Sun Microsystems multiprocessor servers** and **Silicon Graphics Inc. multiprocessor servers**.

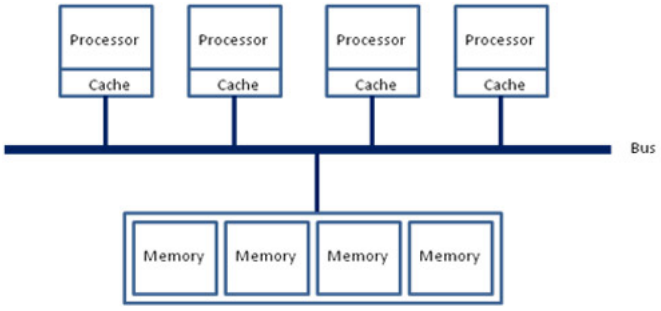

Figure 1: Uniform Memory Access

2.2 Nonuniform Memory Access (NUMA)

In the NUMA system a, each processor has a part of shared memory. The shared memory has a single adress space. This means each processor can access any memory location directly using its real address. However, the access time to modules depends on the distance to the processor.Beacuse of

this memory access time is not uniform. **SGI Origin 3000** can be given as example for this architecture.

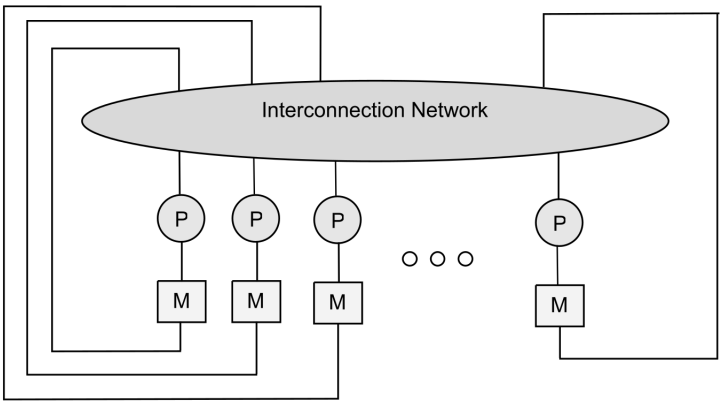

Figure 2: Nonuniform Memory Access

2.3 Cache-Only Memory Architecture (COMA)

In this system each processor has part of the shared memory like in the NUMA. However, in this case the shared memory consists of cache memory. A COMA system requires that data be migrated to the processor requesting it. There is no memory hierarchy and the address space is made of all the caches. There is a cache directory (D) that helps in remote cache access. The Kendall Square Research's **KSR-1** machine is an example of such architecture.

3 Distributed Memory

Distributed Memory systems provide alternative methods for communication and movement of data among multiprocessors (compared to shared memory multipro- cessor systems). A message passing system typically combines local memory and the processor at each node of the interconnection network. There is no global memory so it is necessary to move data from one local memory to another by means of message passing. This is typically done by send/receive pairs of commands, which must be written into the application software by a programmer.

The basis for the scheme is that each processor has its own local memory and communicates with other processors using messages. The elimination

of the need for a large global memory together with its synchronization requirement, gives message passing schemes an edge over shared memory schemes.

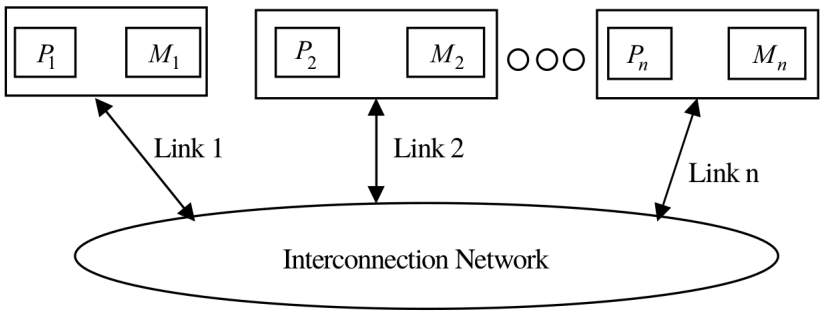

Figure 3: Distributed Memory

Process Granularity The size of a process in a distributed memory system can be described by a parameter called process granularity. This is defined as follows.

$$\textbf{Process Granularity} = \frac{\text{computation time}}{\text{communication time}} \tag{1}$$

4 Distributed Memory vs Shared Memory

As indicated before shared memory enjoys the desirable feature that all communications are done using implicit loads and stores to a global address space. Another fundamental feature of shared memory is that synchronization and communication are distinct. Special synchronization operations (mechanisms), in addition to the loads and stores operations, need to be employed in order to detect when data have been produced and/or consumed. On the other hand, message pas- sing employs an explicit communication model. Explicit messages are exchanged among processors. Synchronization and communication are unified in message pas- sing. The generation of remote, asynchronous events is an integral part of the mess- age passing communication model. It is important, however, to indicate that shared memory and message passing communication models are universal; that is, it is possible to employ one to simulate the other. However, it is observed that it is easier to simulate shared memory using message passing than the converse. This is basically because of the asynchronous event semantics of message

passing as compared to the polling semantics of the shared memory. A number of desirable features characterize shared memory architectures (see Chapter 4). The shared memory communication model allows the programmer to concentrate on the issues related to parallelism by relieving him/her of the details of the interprocessor communication. In that sense, the shared memory communication model represents a straightforward extension of the uniprocessor programming para- digm. In addition, shared memory semantics are independent of the physical location and therefore they are open to the dynamic optimization offered by the underlying operating system. On the other hand, the shared memory communication model is in essence a polling interface. This is a drawback as far as synchronization is concerned. This fact has been recognized by a number of multiprocessor architects and their response has always been to augment the basic shared memory communication model with additional synchronization mechanisms. An additional drawback of shared memory is that in order for data to cross the network, a complete round trip has to be made. One-way communication of data is not possible. Message passing can be characterized as employing an interrupt-driven com- munication model. In message passing, messages include both data and synchroni- zation in a single unit. As such, the message passing communication model lends itself to those operating system activities in which communication patterns are explicitly known in advance, for example, I/O, interprocessor interrupts, and task and data migration. The message passing communication model lends itself also to applications that have large synchronization components, for example, solution of systems of sparse matrices and event-driven simulation. In addition, message passing communication models are natural client – server style decomposition. On the other hand, message passing suffers from the need for marshaling cost, that is, the cost of assembling and disassembling of the message. One natural conclusion arising from the above discussion is that shared memory and message passing communication models each lend themselves naturally to certain application domains. Shared memory manifests itself to application writers while message passing manifests itself to operating systems designers. It is therefore natural to consider combining both shared memory and message passing in general- purpose multiprocessor systems. This has been the main driving force behind systems such as the Stanford FLexible Architecture for SHared memory (FLASH) system. It is a multiprocessor system that efficiently integrates sup- port for shared memory and message passing while minimizing both hardware and software overhead.

References

[1] Hesham El-Rewini and Mostafa Abd-El-Barr. *Advanced computer architecture and parallel processing*, volume 42. John Wiley & Sons, 2005.