CMPE 511
TERM PAPER

“Distributed Shared Memory Architecture”

by

Seda Demirağ
2005701688

1. INTRODUCTION:

Despite the advances in processor design, users still demand more and more performance. Eventually, single CPU technologies must give way to multiple processor parallel computers: it is less expensive to run 10 inexpensive processors cooperatively than it is to buy a new computer 10 times as fast. This change is inevitable, and has been realized to some extent in the specialization of subsystems like bus mastering drive controllers. However, the need for additional computational power has thus far rested solely on advances in CPU technologies.

In parallel systems, there are two kinds of fundamental models: shared memory and message passing. From a programmer's perspective, shared memory computers, while easy to program, are difficult to build and aren't scalable to beyond a few processors. Message passing computers, while easy to build and scale, are difficult to program. In some sense, shared memory model and message passing model are equivalent.

One of the solutions to parallel systems is Distributed Shared Memory (DSM) whose memory is physically distributed but logically shared. DSM appears as shared memory to the applications programmer, but relies on message passing between independent CPUs to access the global virtual address space. Both hardware and software implementations have been proposed in the literature. The advantages of DSM programming model are well known. Firstly, shared memory programs are usually shorter and easier to understand than equivalent message passing programs. Secondly, shared memory gives transparent process-to-process communication.

COMPUTER SYSTEMS:

Flynn [1966] proposed a simple model of categorizing all computers. He uses the stream concept for describing a machine's structure. A stream simply means a sequence of items (data or instructions). Four main types of computer organizations can be found:

SISD : (**S**inge-**I**nstruction stream, **S**inge-**D**ata stream)

SISD corresponds to the traditional mono-processor (Von Neumann computer). A single data stream is being processed by one instruction stream.

SIMD: (**S**inge-**I**nstruction stream, **M**ultiple-**D**ata streams)

In this organization, multiple processing units of the same type process on multiple-data streams. This group is dedicated to array processing machines. Sometimes, vector processors can also be seen as a part of this group.

MISD: (**M**ultiple-**I**nstruction streams, **S**inge-**D**ata stream)

In case of MISD computers, multiple processing units operate on one single-data stream. In practice, this kind of organization has never been used.

MIMD: (**M**ultiple-**I**nstruction streams, **M**ultiple-**D**ata streams)

This last machine type builds the group for the traditional multi-processors. Several processing units operate on multiple-data streams.

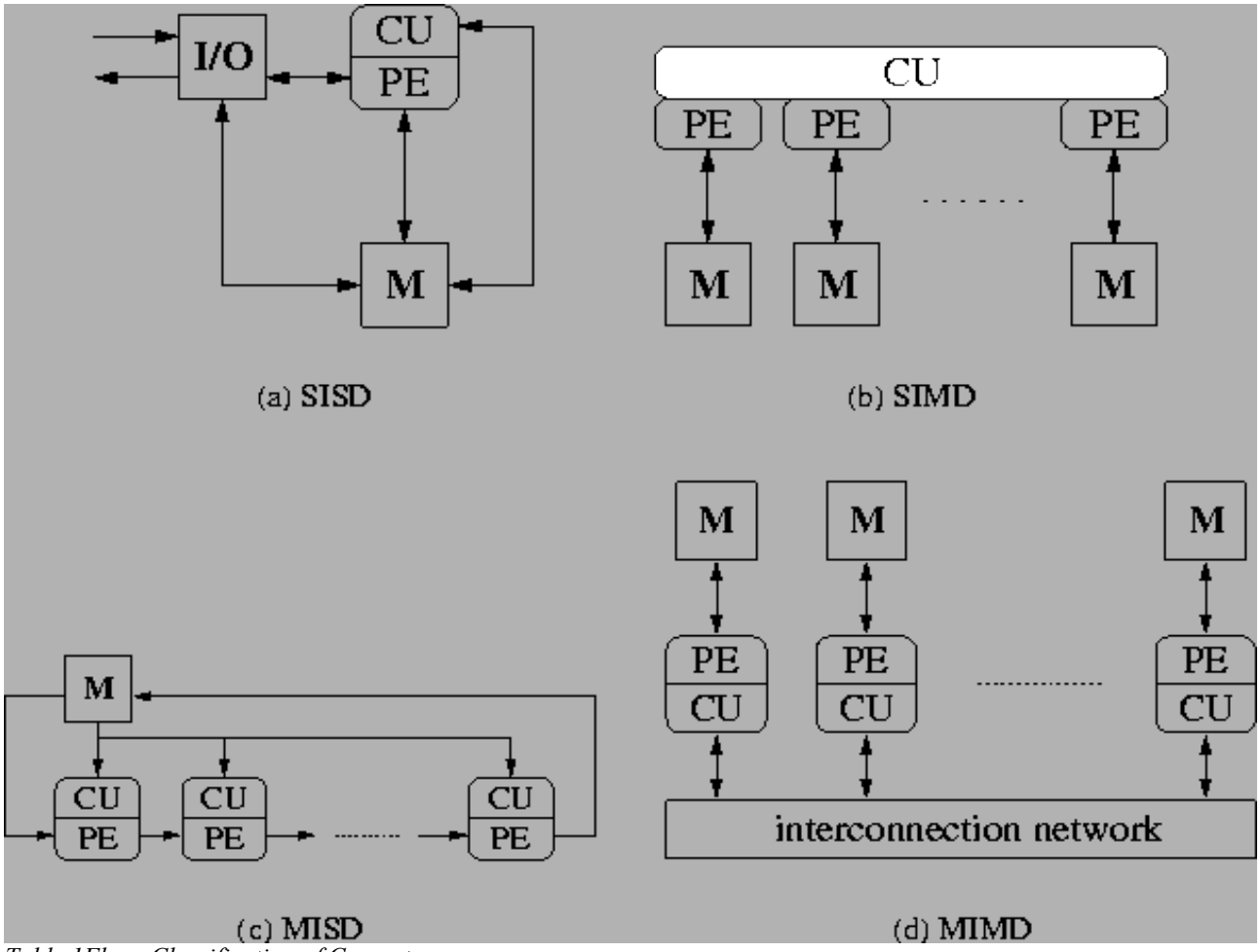

Table 1Flynn Classification of Computer

MIMD (Multiple-Instruction streams, Multiple-Data streams):

MIMDs offer flexibility. With the correct hardware and software support, MIMDs can function as single-user multiprocessors focusing on high performance for one application, as multiprocessors running many tasks simultaneously, or as some combination of these functions.

MIMDs can build on the cost-performance advantages of off-the-shelf microprocessors. In fact, nearly all multiprocessors built today use the same microprocessors found in workstation and single-processor servers.

Existing multiprocessors are divided into two classes, depending on the number of processors involved. The first group, which is called **centralized shared-memory architectures,** has at most a few dozen processors. Multiple processor-cache subsystems share the same physical memory, typically connected by a bus. So the main problem for this type of MIMD systems is the scalability. By replacing a single bus with multiple buses, or even a switch, a centralized shared-memory design can be scaled to a few dozen processors

.

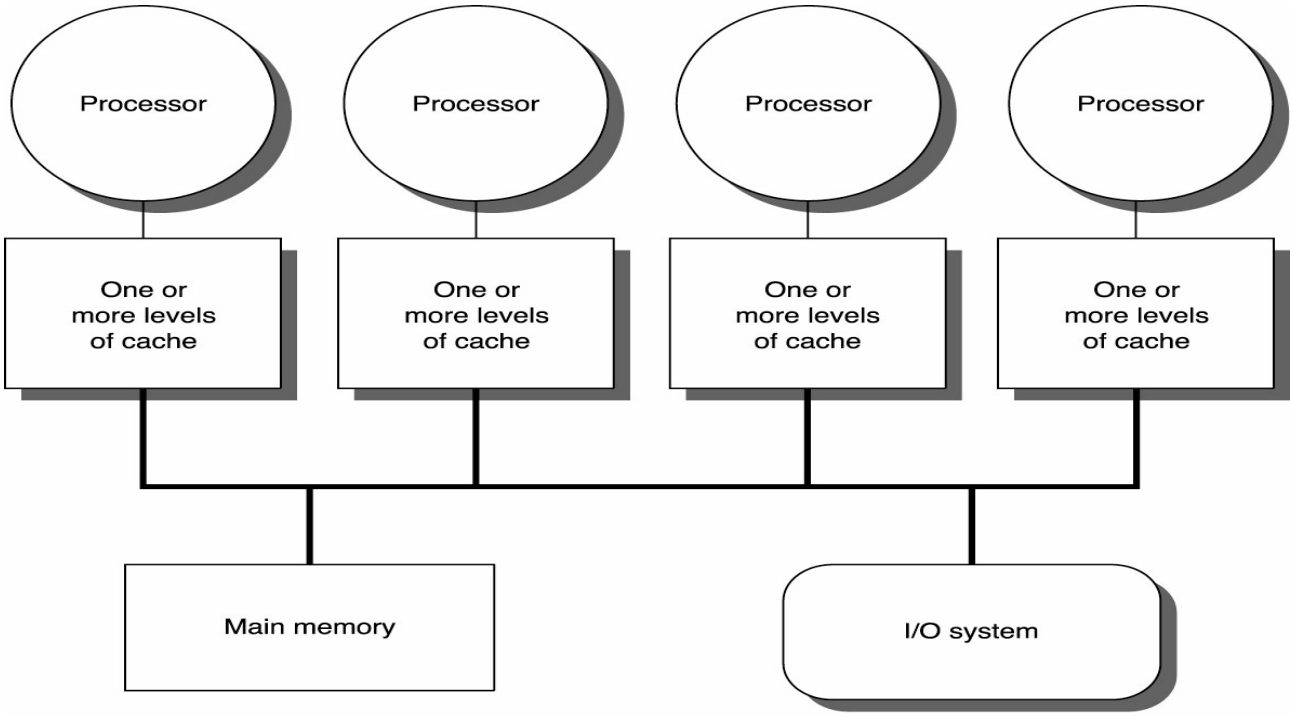

Table 2Centralized(Symmetric) Shared-Memory Architecture

Because there is a single main memory that has a symmetric relationship to all processors and a uniform access time from any processor, these processors can be called symmetric multiprocessors.

The second group consists of multiprocessors with physically distributed memory. To support larger processor counts, memory must be distributed among the processors. The basic architecture of a distributed-memory multiprocessor consists of individual nodes containing a processor, some memory, typically some I/O, and an interface to an interconnection network that connects all the nodes.

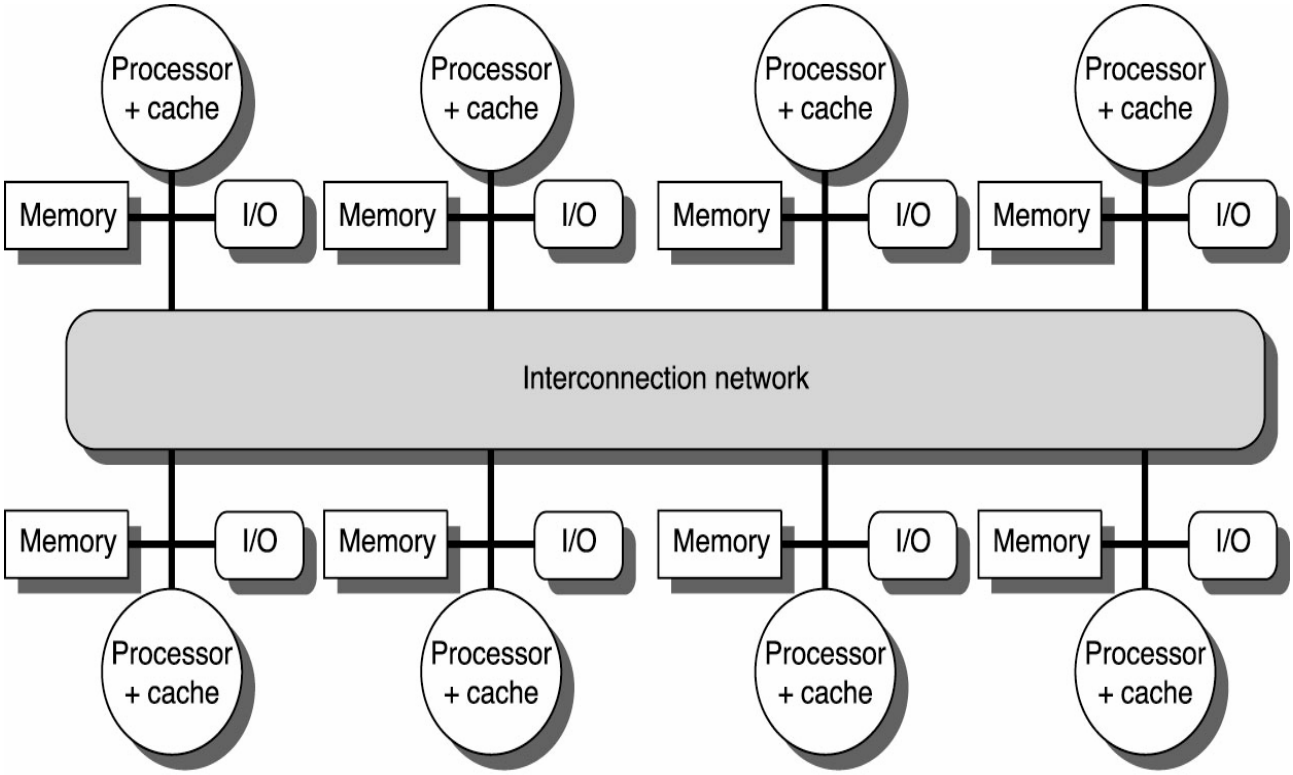

Table 3Distributed Memory Architecture

Distributing the memory among the nodes has two major benefits: it is a cost-effective way to scale the memory bandwidth and it reduces the latency for accesses to the local memory. The key disadvantages for a distributed memory architecture is that communicating data between processors becomes somewhat more complex and has higher latency, at least when there is no contention, because the processors no longer share a single,centralized memory.

DISTRIBUTED SHARED-MEMORY ARCHITECTURES (DSM):

The physically separate memories can be addressed as one logically shared address space, meaning that a memory reference can be made by any processor to any memory location. These systems are called *Distributed shared-Memory (DSM) Architectures.* Shared memory means that the address space is shared. The same physical address on two processors refers to the same location in memory.

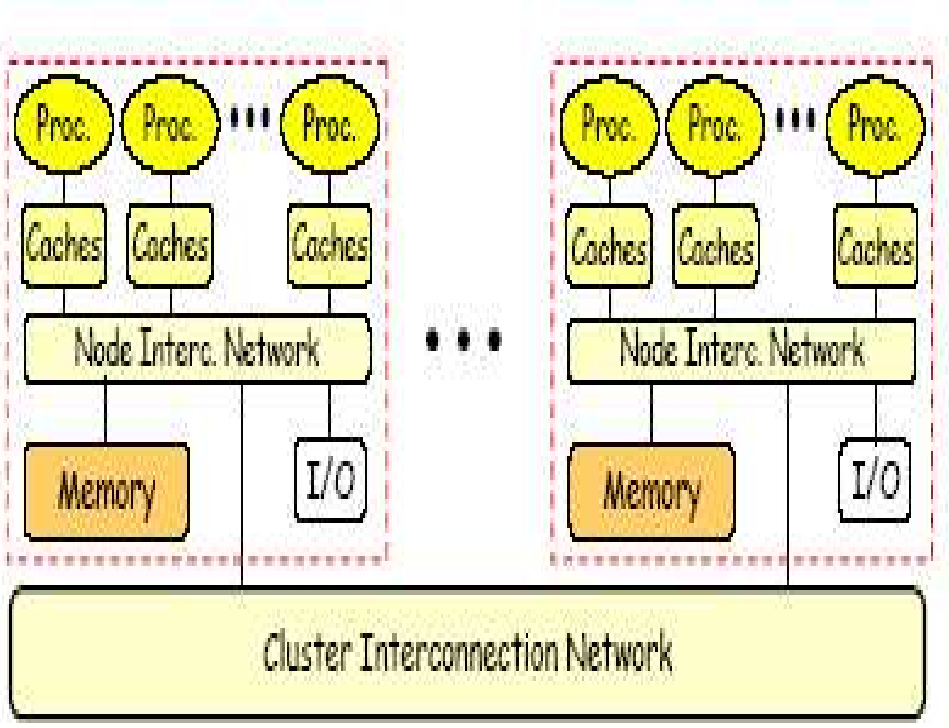

Table 4Distributed Shared-Memory Architecture

A memory reference can be made by any processor to any memory location and also this is called NUMA (Nonuniform memory access).

The C*ache Coherence* problem is the most important problem in a multiprocessor system. A memory system is coherence, when it satisfies these conditions:

- To the same location, a write immediately followed by a read by the same processor will always return the written value.
- To the same location, a read from P2 immediately follows a write by P1 will returns the value written by P1
- Two writes to the same location by any two processors are seen in the same order by all processors

Distributed shared-memory architectures can exclude cache coherence and can easily focus a scalable memory system. The Cray T3D/E is the best known example. These systems have caches, but to prevent coherence problems, shared data are marked as uncacheable and only private data are kept in the caches. In this case, cache coherence can be controlled by software of course.

There are same disadvantages of this type of distributed shared memory architecture. First of all, compiler-based software cache coherence is currently impractical. The basic difficulty is that software-based coherence algorithms must be conservatives. Every block that might be shared must be considered as shared block. So the programmers do not want t o deal with the coherence problem due to the complexity of the possible interactions.

The multiprocessor loses the advantages of being able to fetch and use multiple words in a single cache coherence block for close to the cost of fetching one word without cache coherence.

For these reasons, in small multiprocessors, cache coherence is an accepted requirement. Also for larger architecture, there are some new methods to be able to deal with the cache coherence problem. Although the bus can certainly replaced with a more scalable interconnection network, the lack of scalability of the snooping coherence scheme needs to be addressed.

There are two protocols to be able to deal with the cache coherence problems shared memory architectures.

- Snooping protocol: This is for Symmetric shared-memory architectures. Every cache that has a copy of data from a block of physical memory also has a copy of the sharing status of the block, and centralized state is kept. The caches are usually on a shared memory bus, an all cache controllers monitor or snoop on the bus to determine whether or not they have a copy of a block that is requested on the bus.

    - Send all request for the data to all processors.
    - Processors snoop to see if they have a copy and respond accordingly.
    - Requires broadcast, since caching is at processors.
    - Works well with bus (natural broadcast medium)
    - Dominates for small scale machines.

- Directory -based protocols: This is for distributed shared memory architectures (larger-scale). The sharing status of a block of physical memory is kept in just one location, called the directory. Like snooping protocol, handling a read miss and handling a write to a shared,clean cache block are the main operationsthat a directory protocol must implement. The directory must track the state of each caches and the states are as follows:

    - Shared: one or ore processors have the block cached
    - Uncached: no processor has a copy of a cache block
    - Exclusive : exactly one processor has a copy of a cache block, the processor is called the owner of the block.

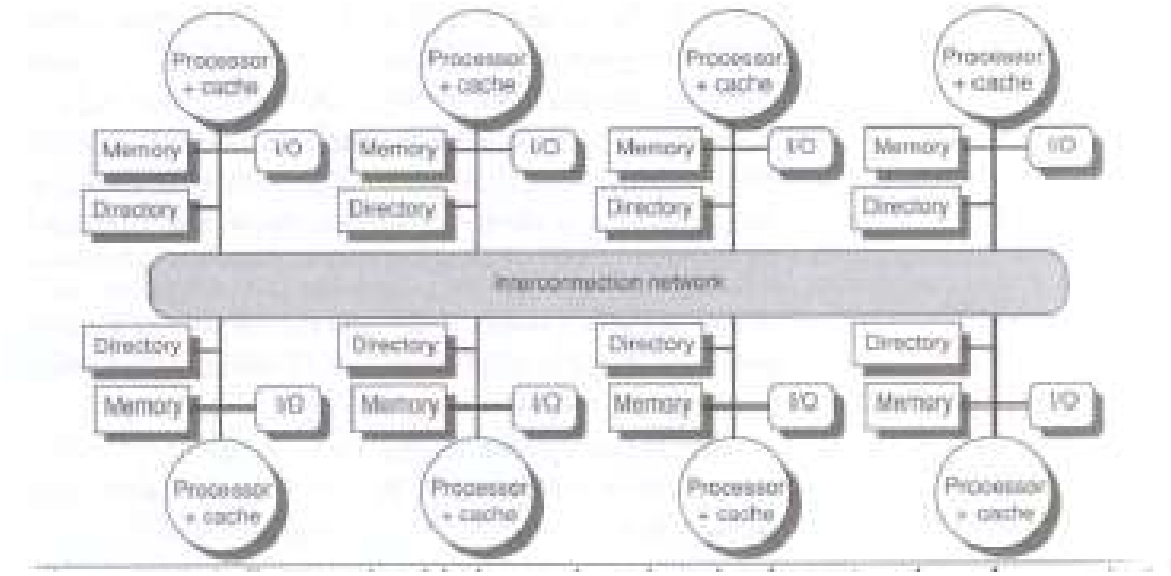

Table 5Directory added to distributed shared-memory architecture

To prevent directory becoming the bottleneck, we distribute directory entries with memory, each keeping track of which processors have copies of their memory blocks. It will be good to be consider the message types that may be sent between the processors and the directories.

- Local node: node where a request is originates.
- Home node: node where the memory location and the directory entry of an address reside.
- Remote node: node that has a copy of a cache block, whether exclusive or shared.

We can take a look at the messages in the directory-based protocol in a table:

Message type	Source	Destination	Message contents	Function of this message
Read miss	local cache	home directory	P, A	Processor P has a read miss at address A; request data and make P a read sharer.
Write miss	local cache	home directory	P, A	Processor P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	home directory	remote cache	A	Invalidate a stored copy of data at address A.
Fetch	home directory	remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch/invalidate	home directory	remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
Data value reply	home directory	local cache	D	Return a data value from the home memory.
Data write back	remote cache	home directory	A, D	Write back a data value for address A.

Table 6Messages for Directory Protocols

In table 6, P is requesting processor number, A is requested address and D is data contents. The first two messages are miss requests sent by local cache to the home. The third through fifth messages are messages sent to a remote cache by the home when the needs the data to satisfy a read or write miss request.

PERFORMANCE of DISTRIBUTED SHARED-MEMORY MULTIPROCESSORS:

In DSM architectures, the memory requests between local and remote is key to performance. It affects the bandwidth and the latency seen by requests. In the performance example we will separate the cache misses into local and remote requests.We will also compare the performance changings of the computational kernels FFT, LU; the applications Barnes and Ocean.

In the first performance table, we will see that the miss rates are not affected by the increasing the number of processors. Only the application Ocean is affected by this changing. This is mainly because of the two factors: an increase in mapping conflits in the cache that occur when the grid becomes small, and an increase in the number of the coherence misses, which are all remote.

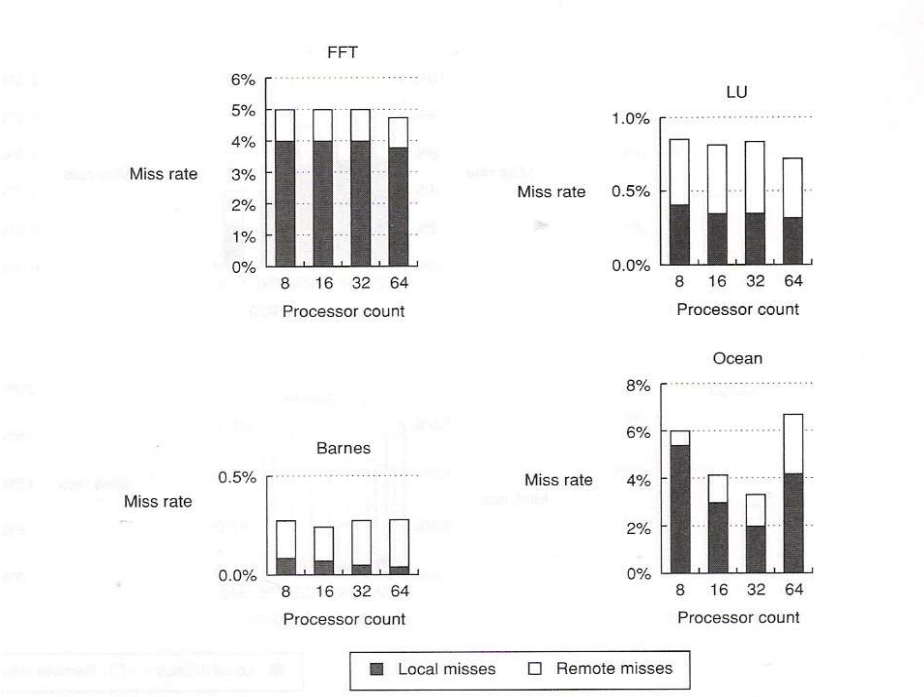

Table 7The data miss rates

In the second performance table, we will see how the miss rate change as the cache size is increased. These miss rates decrease as we might expect. By the time we reach the larger cache size, the remote miss rate is equal to or greater than the local miss rate. Larger caches would amplify this trend.

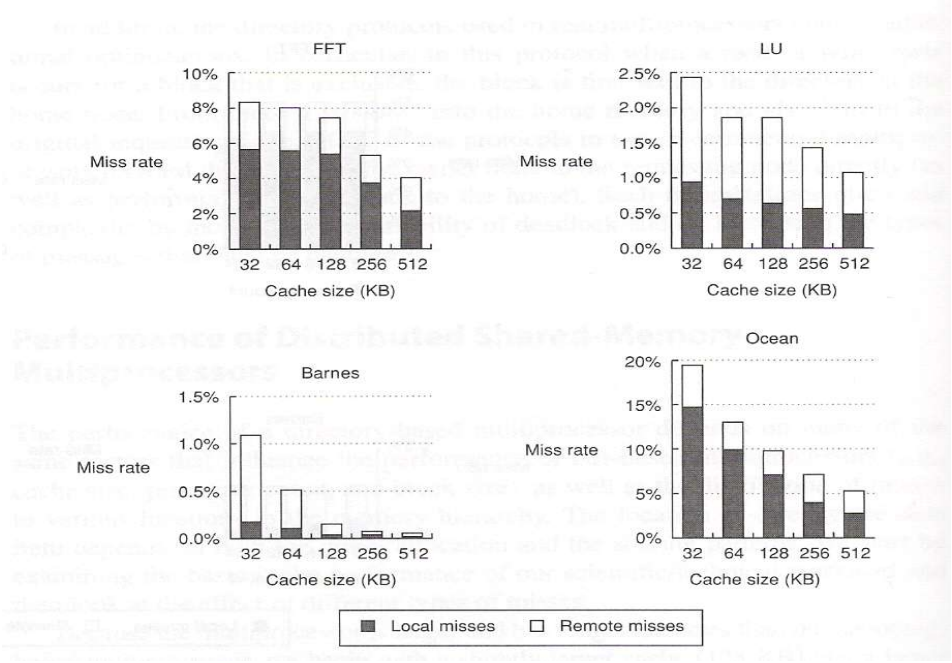

Table 8Miss rates versus increased cache size

In the third performance table, we examine the effect of changing the block size. Increasing the block size is reducing the miss rate.

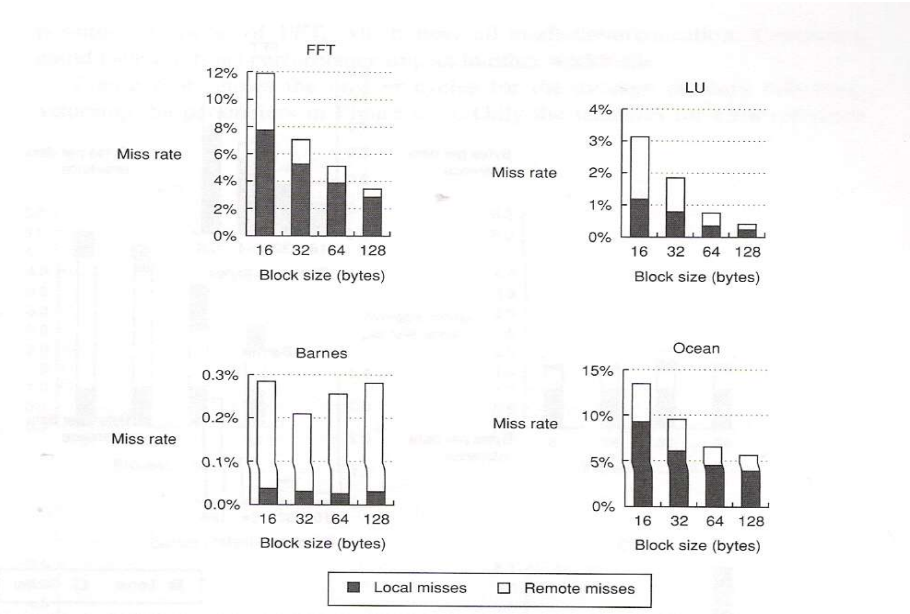

Table 9Data miss rate versus block size

REFERENCES:

- Andrew S. T., Maarten V. S., *Distributed Systems,* 2002
- John L. H., David A. P. , *Computer Architecture: A quantitive Approach,* 2003
- Abraham S., Peter B. G., Greg G., *Operating Systems Concepts, 2003*
- Jinseok K., Gyungho L., *Binding Time in Distributed Shared Memory Architectures,* 1998 International Conference on Parallel Processing.
- Bill N., Virginia L., *Distributed Shared Memory: A Survey of Issues and Algorithms,* Volume 24, Issue 8, August 1991, IEEE Computer Society Press
- S. Zhou, M. Stumm, D. Wortman, K. Li, *Heterogeneous Distributed Shared Memory,* IEEE Transactions on Parallel and Distributed Systems, v.3 n.5, p.540-554, September 1992.