Part 3 : The Memory Hierarchy

3.1 Principle Of Locality

locality of reference: The principle reflects the observation that during the course of execution of a program, memory references by the processor, for both instructions and data, tend to cluster. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references.

A distinction is made in the literature between two forms of locality:

1. Temporal locality: Refers to the tendency of a program to reference in the near future those units of memory referenced in the recent past. *For example*: when an iteration loop is executed, the processor executes the same set of instructions repeatedly.
2. Spatial locality: Refers to the tendency of a program to reference units of memory whose addresses are near one another. That is, if a unit of memory x is referenced at time t, it is likely that units in the range through will be referenced in the near future

EXAMPLE: For cache memory:

- temporal locality is traditionally exploited by keeping recently used instruction and data values in cache memory and by exploiting a cache hierarchy.
- Spatial locality is generally exploited by using larger cache blocks and by incorporating prefetching mechanisms (fetching items of anticipated use) into the cache control logic.

Typically, each instruction execution involves fetching the instruction from memory and, during execution, accessing one or more data operands from one or more regions of memory. Thus, there is a *spatial dual locality* :

- ✓ data spatial locality
- ✓ and instruction spatial locality.

And, of course, temporal locality exhibits this same *temporal dual behavior*:

- ✓ data temporal locality
- ✓ and instruction temporal locality.

Figure(3.1),show that, when an instruction is fetched from a unit of memory, it is likely that in the near future, additional instructions will be fetched from that same memory unit;

and when a data location is accessed, it is likely that in the near future, additional instructions will be fetched from that same memory unit

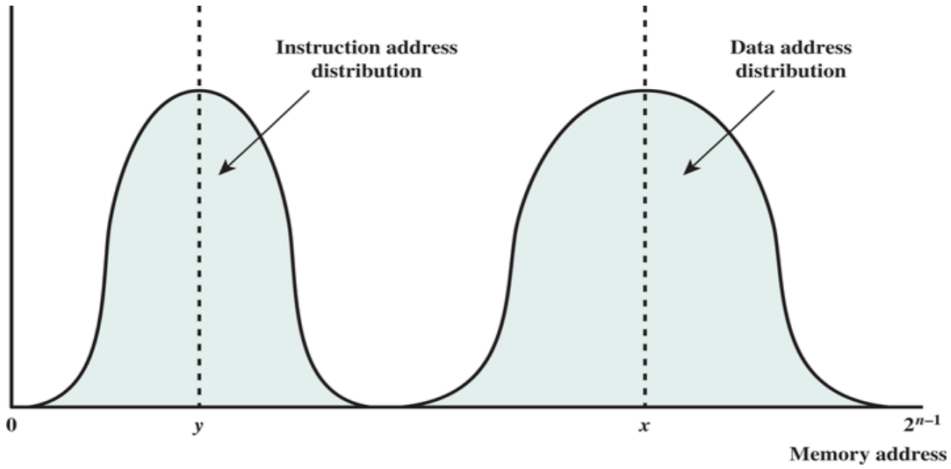

Figure 3.1 Idealized Spatial Locality Behavior: Probability Distribution for Next Memory Access (most recent data memory access at location x; most recent instruction fetch at location y)

3.2 Characteristics Of Memory Systems

The complex subject of computer memory is made more manageable if we classify memory systems according to their key characteristics. The most important of these are :

Location

- Internal: internal memory (e.g., processor registers, cache, main memory)
- External: External memory consists of peripheral storage devices, such as disk and tape, that are accessible to the processor via I/O controllers

Capacity

- Number of words : Common word lengths are 8, 16, and 32 bits.
- Number of bytes : External memory capacity is typically expressed in terms of bytes

Unit of Transfer(Word ,Block): For internal memory, the unit of transfer is equal to the number of electrical lines into and out of the memory module. This may be equal to the word length, (1byte = 8bits)but is often larger, such as 64, 128, or 256 bits.

- Word: For example, the CRAY C90 has a 64-bit word length .The Intel x86 architecture has a wide variety of instruction lengths, expressed as multiples of bytes, and a word size of 32 bits.
- Addressable units: many systems allow addressing at the byte level. In any case, the relationship between the length in bits A of an address and the number N of addressable units is : $2^A=N$

Access Method

- Sequential : Memory is organized into units of data, called records. Access must be made in a specific linear sequence. (shared read–write mechanism)

- Direct : Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting, or waiting to reach the final location(shared read–write mechanism)
- Random : Any location can be selected at random and directly addressed and accessed.
- Associative : This is a random access type of memory that enables one to make a comparison of desired bit locations within a word for a specified match, and to do this for all words simultaneously

Performance

- Access time : : For random-access memory, this is the time it takes to perform a read or write operation. For non-random-access memory, access time is the time it takes to position the read–write mechanism at the desired location.
- Memory cycle time: This concept is primarily applied to random-access memory and consists of the access time plus any additional time required before a second access can commence
- Transfer rate : This is the rate at which data can be transferred into or out of a memory unit , it is equal to 1/(cycle time)

Physical Type (Semiconductor , Magnetic ,Optical ,Magneto-optical)

Physical Characteristics (Volatile/nonvolatile , Erasable/nonerasable ,Organization ,Memory module)

3.3 The Memory Hierarchy

A variety of technologies are used to implement memory systems, and across this spectrum of technologies, the following relationships hold:

- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access time

The way out of this dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy. A typical hierarchy is illustrated in Figure 3.2. As one goes down the hierarchy, the following occur:

a. Decreasing cost per bit
b. Increasing capacity

c. Increasing access time

d. Decreasing frequency of access of the memory by the processor

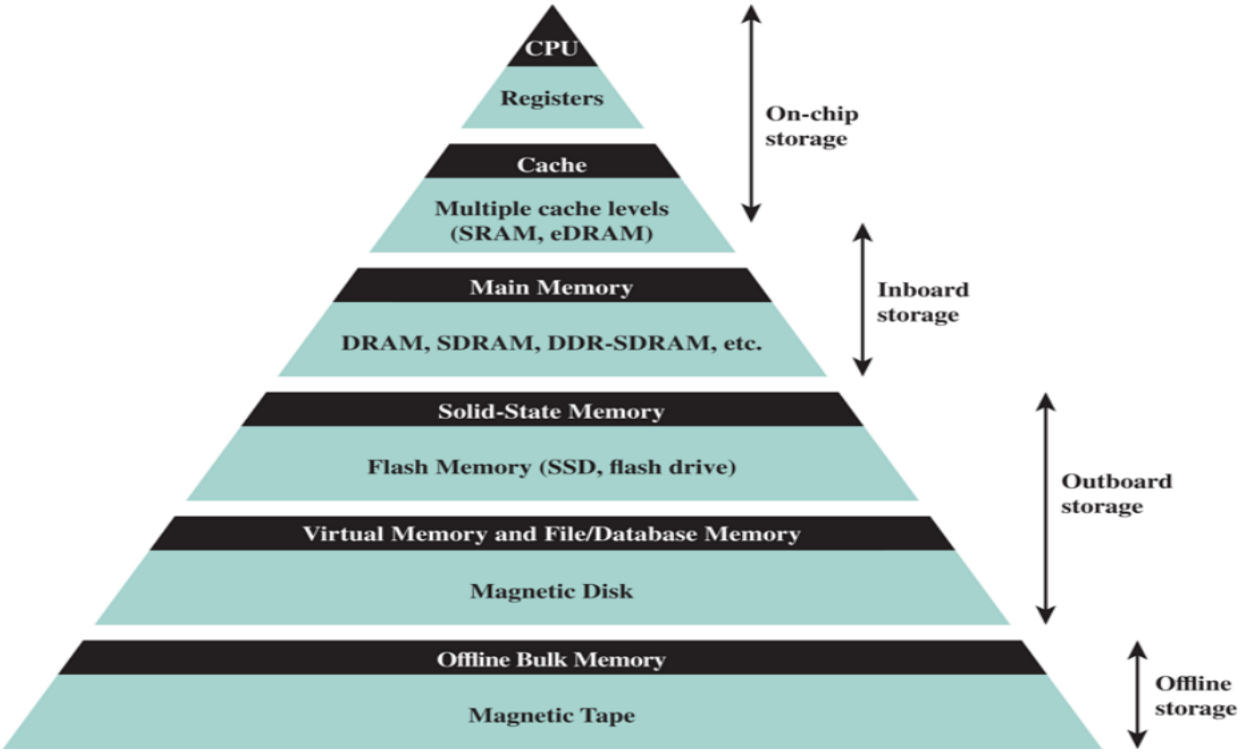

Figure 3.2 The Memory Hierarchy

Let us label the memory at level i of the memory hierarchy such that is M_i closer to the processor than M_{i+1} If C_i , T_i and R_i and S_i are respectively the cost per byte, average access time, average data transfer rate, and total memory size at level i, then the following relationships typically hold between level I and i+1:

$$\begin{aligned} C_i &> C_{i+1} \\ T_i &< T_{i+1} \\ R_i &> R_{i+1} \\ S_i &< S_{i+1} \end{aligned}$$

EXAMPLE

Suppose that the processor has access to two levels of memory. Level 1 contains X words and has an access time of $0.01\mu s$; level 2 contains $1000 \times X$ words and has an access time of $0.1\mu s$.

- Assume that if a word to be accessed is in level 1, then the processor accesses it directly.
- If it is in level 2, then the word is first transferred to level 1 and then accessed by the processor.

In our example, suppose that 95% of the memory accesses are found in Level 1. Then the average time to access a word can be expressed as

$$(0.95)(0.01\mu s) + (0.05)(0.01\mu s + 0.1\mu s) = 0.0095 + 0.0055 = 0.015\mu s$$

The average access time is much closer to $0.01\mu s$ than to $0.1\mu s$, as desired.

Let level 2 memory contain all program instructions and data. Currently used clusters can be temporarily placed in level 1. From time to time, one of the clusters in level 1 will have to be swapped back to level 2 to make room for a new cluster coming in to level 1. On average, however, most references will be to instructions and data contained in level 1.

In practice, the dynamic movement of chunks of data between levels during program , shown in Figure 3.3, with an indication of the size of the chunks of data exchanged between levels.

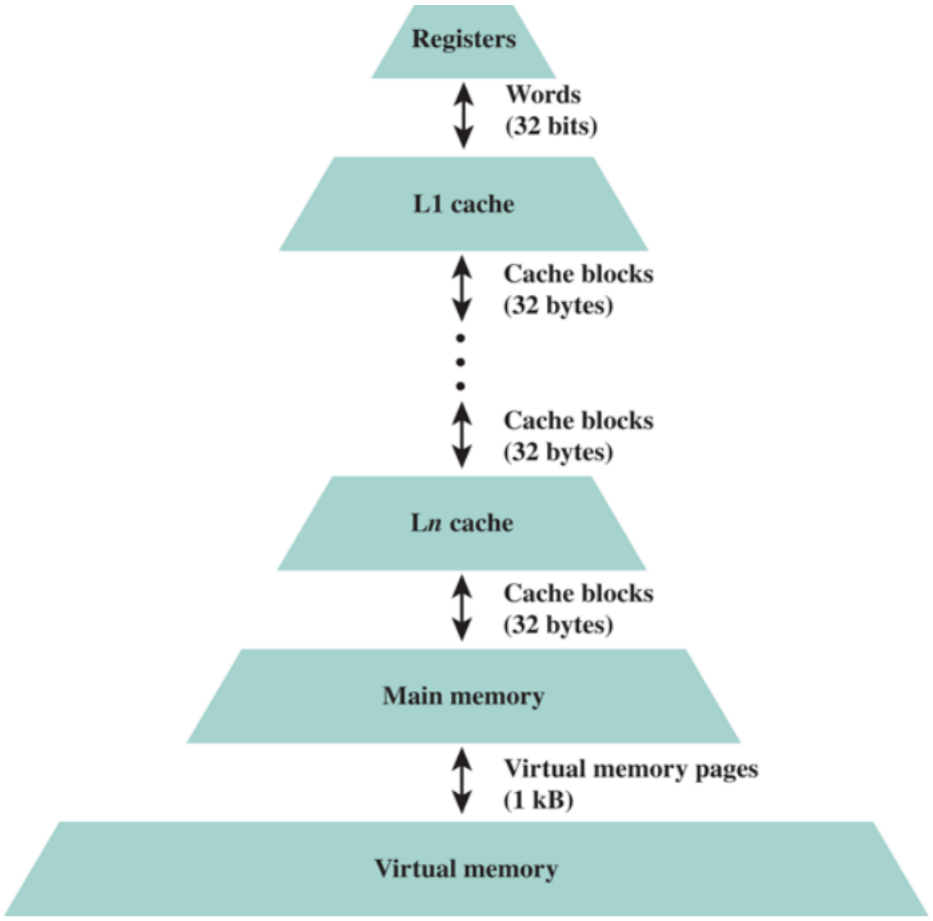

Figure 3.3 Exploiting Locality in the Memory Hierarchy (with typical transfer size)

3.3.1 Typical Members of the Memory Hierarchy

Table 3.1 lists some characteristics of key elements of the memory hierarchy:

- The fastest, smallest, and most expensive type of memory consists of the *registers internal to the processo*r.
- Next will be typically multiple layers of cache. Level 1 cache (L1 cache), closest to the processor registers, is almost always divided into an *instruction cache* and

a *data cache*. This split is also common for L2 caches. L3 cache and some have an L4 cache.

- *Main memory* is the principal internal memory system of the computer. Each location in main memory has a unique address. Main memory is visible to the programmer, whereas cache memory is not.
- External, nonvolatile memory is also referred to as secondary memory or auxiliary memory. These are used to store program and data files and are usually visible to the programmer only in terms of files and records, as opposed to
- Table 3.1

Memory level	Typical technology	Unit of transfer with next larger level (typical size)	Managed by
Registers	CMOS	Word (32 bits)	Compiler
Cache	Static RAM (SRAM); Embedded dynamic RAM (eDRAM)	Cache block (32 bytes)	Processor hardware
Main memory	DRAM	Virtual memory page (1 kB)	Operating system (OS)
Secondary memory	Magnetic disk	Disk sector (512 bytes)	OS/user
Offline bulk memory	Magnetic tape		OS/User

3.3.2 The IBM z13 Memory Hierarchy

Figure 3.4 illustrates the memory hierarchy for the IBM z13 mainframe computer ,It consists of the following levels:

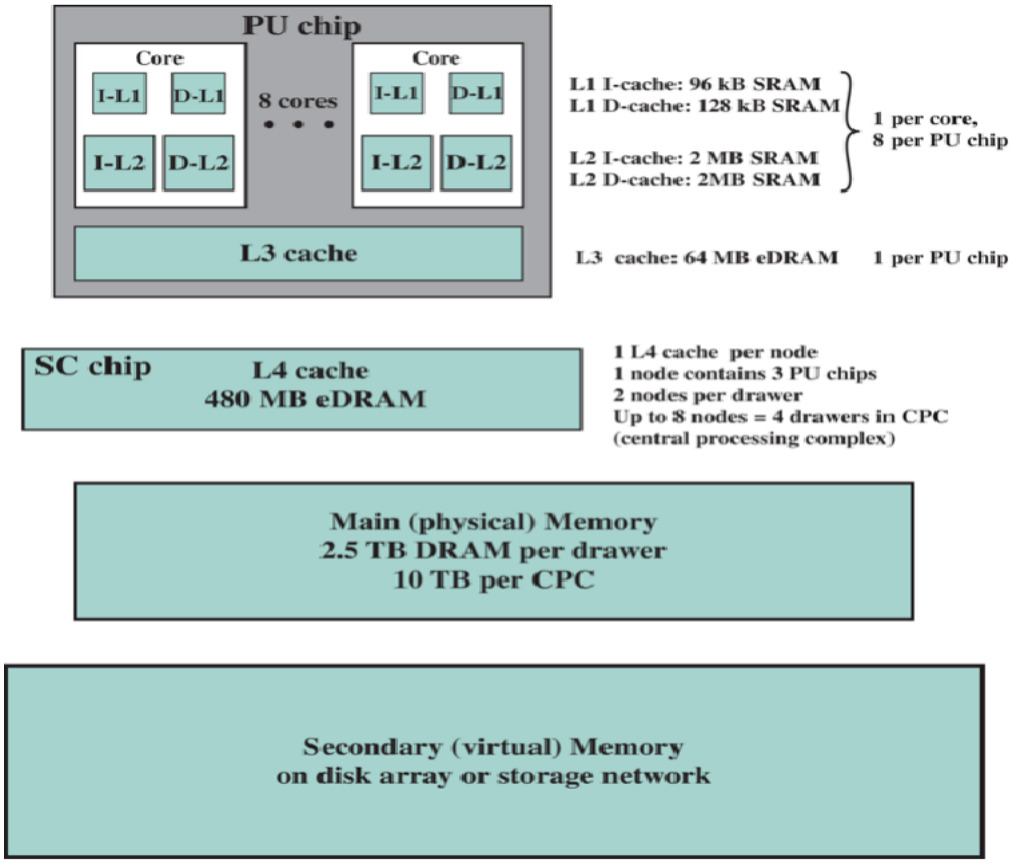

Figure 3.4 IBM z13 Memory Hierarchy

3.3.3 Design Principles for a Memory Hierarchy

Three principles guide the design of a memory hierarchy :

1. **Locality:** Locality is the principle that makes effective use of a memory hierarchy possible.
2. **Inclusion:** This principle dictates that all information items are originally stored in level M_n, where *n* is the level most remote from the processor. During the processing, subsets of M_n are copied into M_{n-1}. similarity, subsets of M_{n-1} are copied into M_{n-2}, and so on. This is expressed concisely as $M_i \subseteq M_{i+1}$. Thus, this is in contrast to our simple example of **Figure 4.1**, where Bob moved a folder from the file cabinet to his desk. With the memory hierarchy, units of data are copied rather than moved, so that the data unit that is moved to M_i remains in M_{i+1}. Thus, if a word is found in M_i, then copies of the same word also exist in all subsequent layers $M_{i+1}, M_{i+2}, \ldots, M_n$.
3. **Coherence:** Copies of the same data unit in adjacent memory levels must be consistent. If a word is modified in the cache, copies of that word must be updated immediately or eventually at all higher levels.

This leads to two requirements:

Vertical coherence: If one core makes a change to a cache block of data at L2, that update must be returned to L3 before another L2 retrieves that block.

Horizontal coherence: If two L2 caches that share the same L3 cache have copies of the same block of data, then if the block in one L2 cache is updated, the other L2 cache must be alerted that its copy is obsolete.