Scientific Computing I

Memory Architecture and Memory Management

Martin Köhler

Computational Methods in Systems and Control Theory (CSC) Max Planck Institute for Dynamics of Complex Technical Systems

Winter Term 2024/2025

This Lecture:

Memory Architecture and Memory Management

Beside algorithmic efficiency, handling memory accesses is a crucial point for obtaining fast algorithms and programs.

Memory Types

Memory Types

- Cloud
- Network Storage
- Local Storage
  - Hard Disk Drive (HDD)
  - Solid State Disk (SSD)

} slow and very slow

- Main Random Access Memory (RAM)

} medium

- L3 Cache
- L2 Cache
- L1 Cache
- Registers

} fast

Memory Types

Hardware sided the relevant memory comes mainly in four types:

- Static Random Access Memory (SRAM),
- Dynamic Random Access Memory (DRAM),
- Flash Electrically Erasable Programmable Read-Only Memory (Flash-EEPROM) Flash-EEPROM
- Magnetic and optical surfaces

Memory Types

Hardware sided the relevant memory comes mainly in four types:

- Static Random Access Memory (SRAM),
- Dynamic Random Access Memory (DRAM),
- Flash Electrically Erasable Programmable Read-Only Memory (Flash-EEPROM) Flash-EEPROM
- Magnetic and optical surfaces

The volatile memory building blocks have the following properties:

Feature	**SRAM**	**DRAM**
Storage Circuit Base	Transistor	Capacitor
Speed	Same as CPU	Slower than CPU
Latency	Low	High
Density	Low	High
Power Consumption	High	Low
Cost	High	Low

Virtual Memory Concept

General

Virtual Memory Concept

General

Virtual memory is an operating system abstraction layer, that allows to access the various memory layers as one large device. It usually consists of *memory pages*, the smallest accessible units of memory (normally 4 or 64 kBytes).

- every program has its own virtual memory space,
- each virtual memory space is structured in the same way,
- memory is divided into "pages" which is the smallest manageable unit on the "physical" side,
- requires a CPU with a memory management unit (MMU),
  - with MMU: x86, x86_64, PowerPC, ARM, RISC-V
  - without MMU: AVR, PIC, WDC 6502, Zilog Z80, Intel MSC-51 (8051)
- supported by almost all modern operating systems.

Virtual Memory Concept

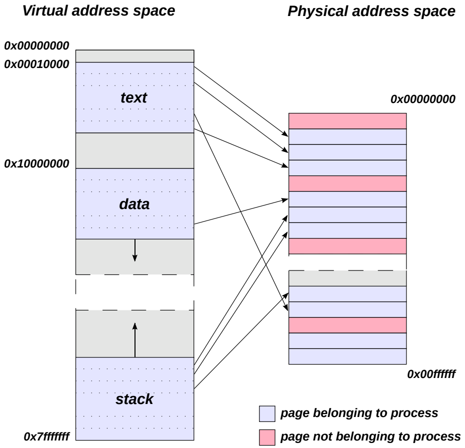
General

Definition (swapping and double buffering)

Relocation of potentially unused data to the local storage by the operating system is called *swapping*. Moving data to the local storage may cause large overhead in waiting time. Any technique that moves that data at strategically better times to avoid swapping is called *double buffering*.

Virtual Memory Concept

General



[0]Image Source: `https://commons.wikimedia.org/wiki/File:Virtual_address_space_and_physical_address_space_relationship.svg`

Paging

Virtual Memory Concept

Paging

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory, thus minimizing issues like fragmentation.

- paged virtual memory is the most common implementation.
- page size 4 kBytes, with huge pages 64 kBytes.
- generally data can be located anywhere in a page.
- some operations expect the data to be located at the start of a memory page.
  $\rightarrow$ *page aligned* memory
  $\rightarrow$ increases memory fragmentation
- *page locked memory* is a special type of memory that is not allowed to get swapped

$\rightarrow$ fundamental concept in modern operating systems, enabling efficient and flexible memory management.

Virtual Memory Concept

Paging

Workflow:

1. Logical and Physical Address Space:
   - The logical address space is divided into fixed-size units called pages.
   - The physical address space is divided into blocks of the same size, called frames.
2. Page Table:
   - Each process has a page table that maps logical pages to physical frames.
   - The page table keeps track of where each page is stored in physical memory.
3. Address Translation:
   - When a process needs to access a memory location, the CPU translates the logical address into a physical address using the page table.
   - The logical address is split into a page number and an offset. The page number is used to find the corresponding frame in the page table, and the offset specifies the exact location within the frame.

Memory Related Error Signals

Virtual Memory Concept

Memory Related Error Signals

Memory accesses can cause two memory related error signal:

- SIGSEGV[1]
  - segmentation violation or segmentation fault signal
  - usually leads to immediate abortions of the process
  - caused by accessing memory segments in foreign address spaces.
- SIGBUS
  - Bus error signal
  - abortion also immediate
  - one common cause: using a processor instruction with an address that does not satisfy its alignment requirements

[1]mostly in combination with **`malloc`** and **`free`** and/or wrong array indices

Volatile Memory

Registers

Volatile Memory

Registers

- very small number
- very fast, access within a **single CPU cycle** possible
- generic registers typically 8, 16, 32, or 64 bytes
- vector registers for vectors of length 2, 4, 8, 16
  - **MMX**: integer operations (deprecated, ancient)
  - **SSE2/3/4**: integers, floating point numbers (default)
  - **AVX/AVX2/AVX512**: integers, floating point numbers
  - **AMX**: half precision floating point numbers
- managed by the compiler and one mostly relies on the compiler's capabilities.
- vector registers can be programmed with ASM-intrinsics by hand

Cache

Volatile Memory

Cache

- **L1**: typically 32 or 64 kBytes, split into a data and an instruction part, **installed per core**, direct access to the registers, transfer-rate: 1TB/s.
- **L2**: $\approx 256 - -2048$ kBytes, **installed per core**, keeps frequently used data and instructions of the current core, transfer-rate: 1TB/s
- **L3**: $\approx$ few MBytes per core, same as L2 **for a group of cores** making a processor, connects to RAM, transfer-rate: >400 GB/s
- **L4**: only on few CPU architectures, cache of the memory controller, transfer-rate: 400 GB/s

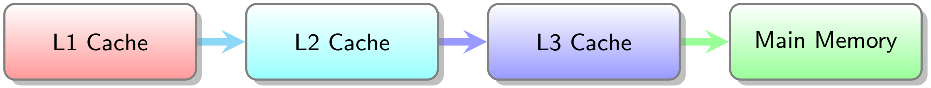
$\rightarrow$ Cache is small, high speed memory made out of SRAM.

$\rightarrow$ Arranged in **Cache-Lines** of 4 to 128 bytes $\nearrow$ page.

Volatile Memory

Cache

Data-Lookup:



Successful lookup in the cache is called **Cache Hit**, otherwise it is a **Cache Miss**.

Cache Hit:

- data transfer at maximum speed
- no interaction with main memory

Cache Miss:

- data not available in cache
- needs to be loaded from main memory
- results in a **miss penalty** (Cache Latency)

Hit ratio: percentage of memory accesses satisfied by the cache ($\approx 80 - 90\%$).

Cache-Lines are replaced either **randomly** or by an **LRU**(last recently used) principle.

Main Memory

Volatile Memory

Main Memory

- mostly built of DRAM cells
- three main types available:
  - asynschronous
    (FPRAM, EDORAM) (outdated)
  - synchronous
    (SDRAM, DDRSDRAM, DDR2SDRAM, DDR3SDRAM, DDR4SDRAM, DDR5SDRAM, HBM-DDRx)
  - Rambus
    (RDRAM, XDRDRAM, XDR2DRAM)

Today:

- **Desktop PCs, Laptops, Tablets, Smartphones**: DDR4SDRAM, DDR5SDRAM
- **Servers**: DDR4SDRAM, DDR5SDRAM with error correction
- **GPUs**: DDR5SDRAM
- **Data Center GPUs**: DDR5SDRAM or HBM DDR5SDRAM with error correction

Volatile Memory

Main Memory

Definition

*Columns Address Stroke Latency (*CAS Latency*)*: time for waiting between a request of data and their availability at the memory pins.

Technical Specifications:

Memory clock	2000–4000 MHz
Data rate	4000–6000 MT/s
Peak transfer rate	32–200 GB/s
CAS Latency	30–40 cycles (at best) $\approx$ 7.5–20ns

Typical memory size:

- **Smartphone/Tablet**: 2 - 6 GB
- **Laptop**: 4 - 16 GB
- **Desktop**: 8 - 64 GB
- **Server**: 192 GB - 2 TB

Non-Volatile Storage

Local Storage

Non-Volatile Storage

Local Storage

Maximum possible transfer rates are bounded by the capabilities of the bus interface

Type	theoretic peak transfer	release / introduction
ATA 33/66/100	33/66/100 MB/s	
SATA I	150 MB/s $\hat{=}$ 0.15 GB/s	
SATA II	300 MB/s $\hat{=}$ 0.30 GB/s	$\approx$ 2005
SATA 3.0	600MB/s $\hat{=}$ 0.60 GB/s	05.2009
SATA 3.2	up to 1969MB/s $\hat{=}$ 1.97 GB/s	08.2013
SAS	300 MB/s – 22.5 GB/s	current developments
NVMe	up to 7.5 GB/s	2011 to current

Non-Volatile Storage

Local Storage

Either Solid-State-Drives (SSD) or Harddisk-Drives are used:

Feature/Property	SSD	HDD
Noise	$++$	$-$
Reliabilty, Lifetime	$-$	$+$
Price	$-$	$+$
Capacity	$-$	$+$
Fragmentation	$+$	$-$
mechanical delay	$+$	$-$
environmental influence	$+$	$-$
practical transfer rates	100–900 MB/s	$\leq$ 140 MB/s
random access time	0.1 ms	2.9–12 ms

Non-Volatile Storage

Local Storage

Either Solid-State-Drives (SSD) or Harddisk-Drives are used:

Feature/Property	SSD	HDD
Noise	$++$	$-$
Reliabilty, Lifetime	$-$	$+$
Price	$-$	$+$
Capacity	$-$	$+$
Fragmentation	$+$	$-$
mechanical delay	$+$	$-$
environmental influence	$+$	$-$
practical transfer rates	100–900 MB/s	$\leq$ 140 MB/s
random access time	0.1 ms	2.9–12 ms

RAID (Redundant Array of Independent Disks):

- can increase total storage capacity by grouping disks to larger logical volumes
- can increase the performance and data safety by multiply/redundantly storing the same data.

Network Storage

Non-Volatile Storage

Network Storage

Local Network

- **Ethernet**: 100 Mbit/s to 40 Gbit/s
- **Infiniband**: 40/56/80/100 Gbit/s with low latency
- **Omnipath**: 25/100 Gbit/s with low latency

Filesystems:

- NFS (Network File System, Linux, *BSD, MacOS)
- SMB/CIFS (Common Internet Filesystem, Windows)
- LustreFS (HPC Filesystem)
- BeeGFS (HPC Filesystem)
- OrangeFS (HPC Filesystem)

Cloud

- speed depends on the internet connection of the client and the server
- high latency
- only for final/backup storage, not feasible for computations
- additional synchronization required

Non Uniform Memory Access

Non Uniform Memory Access

Non-Uniform Memory Access (NUMA) is a computer memory design used in multiprocessor systems, where the memory access time varies depending on the memory location relative to the processor. In NUMA, a processor can access its local memory faster than non-local memory (memory local to another processor or shared between processors).

- → typical design of systems with two or more CPU sockets
- → also on one CPU possible, e.g. AMD Epyc CPUs with multiple Chiplets
- → CPU + GPU systems can also follow the NUMA design principle

Non Uniform Memory Access

Non-Uniform Memory Access (NUMA) is a computer memory design used in multiprocessor systems, where the memory access time varies depending on the memory location relative to the processor. In NUMA, a processor can access its local memory faster than non-local memory (memory local to another processor or shared between processors).

→ typical design of systems with two or more CPU sockets

→ also on one CPU possible, e.g. AMD Epyc CPUs with multiple Chiplets

→ CPU + GPU systems can also follow the NUMA design principle

Example

A system is equipped with 2 processors an 16 GB of main memory, which is separated into two blocks of 16 GB, one for each processor.
The MMUs each organize 16GB locally and need to access the other 16GB via the other MMU.

Cache Coherence and Memory Consitency

Non Uniform Memory Access

Cache Coherence and Memory Consitency

Example

Consider a dual Core system with L1/L2 caches for each processor core. The situation that a memory block is present in both caches and one of the copies invalidates the other copy due to a write access, can appear.

$\rightarrow$ results in the **cache coherence problem**

Non Uniform Memory Access

Cache Coherence and Memory Consitency

Example

Consider a dual Core system with L1/L2 caches for each processor core. The situation that a memory block is present in both caches and one of the copies invalidates the other copy due to a write access, can appear.

→ results in the **cache coherence problem**

- CPU designs must keep track of the different copies of the data in the cache
- consistent view required with respect to read operations
- write operations invalidate the consistency.

→ A system that is investing this extra work is called **ccNUMA** (for cache coherent NUMA) machines.