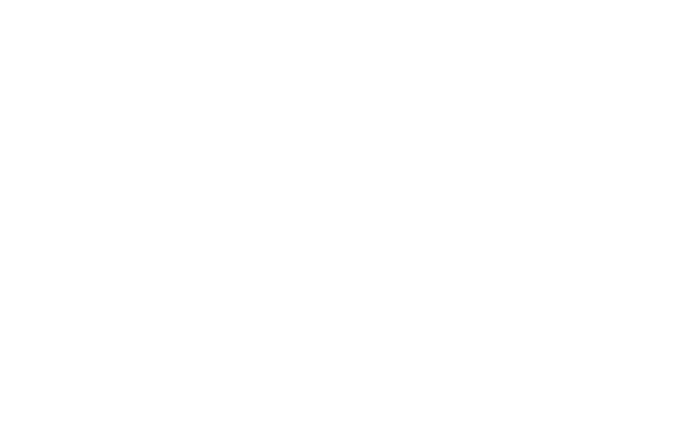
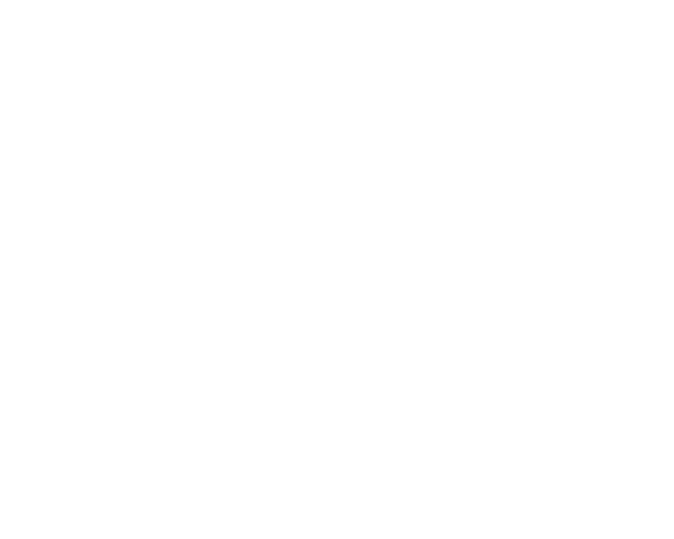
A circuit board

Description automatically generated



2020



Technical University of Cluj Napoca

Transfer Speed with Cache Memories

01.11.2020

Florin-Daniel Petrean

**Transfer Speed with Cache Memory**

Contents

[**1.Introduction** 1](#_Toc16890)

[1.1 Context 1](#_Toc16891)

[1.2 Specifications 1](#_Toc16892)

[1.3 Objectives 2](#_Toc16893)

[**2.Bibliographic study** 2](#_Toc16894)

[**3.Analisys** 3](#_Toc16895)

[3.1 Multiplication 3](#_Toc16896)

[3.2 Division 4](#_Toc16897)

[**4.Design** 5](#_Toc16898)

[**5.Implementation** 7](#_Toc16899)

[**6.Testing and validation** 8](#_Toc16900)

[**7.Conclusions** 9](#_Toc16901)

[**8.Bibliography** 10](#_Toc16902)

# 1.Introduction

## 1.1 Context

The goal of this project is to design and implement a cache memory and compare its data transfer speed with other memories such as internal and virtual memory based on different parameters that speed can depend on: transfer block dimension, cache size etc. The different performances of each memory parameter choice will be studied using graph plots.

This project can be used by people that need to see what sort of modifications to different parameters have to be done in their use or implementation of cache memory as to improve performance of the overall project they have to finish. Also, it will encourage people to make use of the cache memory as much as possible as it is good practice in any program that needs to perform access to memory faster.

## 

## 1.2 Specifications

This project will be implemented in Visual Studio Code provided by Microsoft in C++ programming language in order to have higher flexibility over memory management and other aspects needed to improve the accuracy of the measurements. The code wil be run on a virtual machine running the latest Ubuntu version on an HDD with 4 GB RAM and dual core processor.

## 1.3 Objectives

Design and implement a cache memory (and possibly a memory manager), then compare its transfer speed with internal and virtual memory using different varying parameters such as transfer block dimension and cache size. The tests will be displayed using a 3rd party program or using a custom library for graph plotting and tracing.

# 2.Bibliographic study

Cache memory is a small-sized type of volatile computer memory that provides high-speed data access to a processor and stores frequently used computer programs, applications and data. A temporary storage of memory, cache makes data retrieving easier and more efficient. It is the fastest memory in a computer, and is typically integrated onto the motherboard and directly embedded in the processor or main random access memory (RAM).

Diagram

Description automatically generated

The above diagram can give you a better picture of how the cache memory works.

A cache is made up of a pool of entries. Each entry has associated *data*, which is a copy of the same data in some *backing store*. Each entry also has a *tag*, which specifies the identity of the data in the backing store of which the entry is a copy. Tagging allows simultaneous cache-oriented algorithms to function in multilayered fashion without differential relay interference.

When the cache needs to access data presumed to exist in the backing store, it first checks the cache. If an entry can be found with a tag matching that of the desired data, the data in the entry is used instead. This situation is known as a cache hit. The percentage of accesses that result in cache hits is known as the hit rate or hit ratio of the cache.

When the cache is checked and found not to contain any entry with the desired tag, is known as a cache miss. This requires a more expensive access of data from the backing store. Once the requested data is retrieved, it is typically copied into the cache, ready for the next access.

During a cache miss, some other previously existing cache entry is removed in order to make room for the newly retrieved data. The heuristic used to select the entry to replace is known as the replacement policy. One popular replacement policy, "least recently used" (LRU), replaces the oldest entry, the entry that was accessed less recently than any other entry (see cache algorithm). More efficient caching algorithms compute the use-hit frequency against the size of the stored contents, as well as the latencies and throughputs for both the cache and the backing store. This works well for larger amounts of data, longer latencies, and slower throughputs, such as that experienced with hard drives and networks, but is not efficient for use within a CPU cache.

The implementation used in this project will be based on LRU cache and is composed of 2 data structures:

* a doubly linked list used for the cache frame (The maximum size of the queue will be equal to the total number of frames available where the most recently used pages will be near front end and least recently pages will be near the rear end)
* a hash table in order to speed up the access on all elements of the cache (with page number as key and address of the corresponding queue node as value)

3.Analisys

3.1 Cache memory

By analysing the nature of the cache memory, it can be observed that it is not accessible to a user-space program, as the operating system basically makes it invisible to processes and manages the allocations in the cache according to its architecture.

In order to access some basic control over the CPU cache, a kernel driver must be used. This driver will disable/enable the CPU cache in order to not disrupt the measurements on the Virtual and Physical memory. It will be enabled when running the performance test on the Cache memory.

3.2 Virtual memory

Regarding the virtual memory, each process has its share of it. Each program has the same amount of virtual memory, which makes this part a bit easier to measure. Any allocation within the testbench program will use virtual memory, resulting in a fast way to acquire the data needed for the measuring.

3.3 Physical memory

On the physical memory part, which is basically the disk memory, the measuring will be done on a newly created file which will be read and written data. The disk cache will also need to be disabled in order to have better accuracy on the measurements.

3.4 Testbench

The top module of the program will need to access all results acquired through measuring and modify the data depending on various parameters which could affect the transfer of data. Also, it will have to job to plot all results in order to better see the difference some parameters could make on the transfer speed. The destination of all transfers of each memory will be a newly created file on the unbuffered disk.

4.Design

For the creation of this benchmark program the following design decisions were made:

1. **Data source and destination**

The source of the data will be virtual, physical, respectively cache memory and the destination will be a file on the system drive.

1. **Virtual memory data**

A volatile memory mapped buffer with 4KB size that will be continuously copied into the output file according to a counter in order to better simulate the normal behaviour of transferring data.

1. **Cache memory data**

A dynamically allocated buffer with 4KB size will also be copied the in the manner as the virtual memory. In addition, this buffer will be prefetched in the cache memory and the output file will be artificially accessed a lot of times in order to ensure it stays in the cache memory.

1. **Physical memory data**

A previously created output file will be used as input file for the physical data. A new output file will be created in which the data will be transferred. For all the operations above the disk cache will be disabled.

**5.Timer**

A high accuracy clock will be needed to test the performance of transfers which will ideally count the processor clock cycles that happen only during our program. This could not be the case so we will need to perform multiple runs of the program and take the average.

1. **Kernel module**

The kernel module will run on the ring 0 of the OS and will disable the CPU cache when loaded. Using assembly instructions the CR0 register will be modified in order to achieve this goal.

1. **The testbench results**

The transfer speed of all computations will be written in a .csv file which will automatically position them in a more readable table for analysis.

. Diagram

Description automatically generated

5.Implementation

The program starts off with a for loop which will increase the data size at each iteration and test the transfer speed of each memory type while also putting the results in dataTable.csv file.

**1.The cache memory benchmark**:

Text

Description automatically generated

As we can see, the first step is to create the destination file descriptor with O\_DIRECT flag in order to deactivate the disk cache, then prefetch the newly generated data and access the output file multiple times to force it into the cache memory.

The next part simply tests the time it takes to transfer all data into the output file and computes the transfer speed using the std::chrono library and std::ios\_base. This part is very similar to the rest of the memories.

The final part is loading the kernel module and disabling the cache in order to not interfere when the virtual and physical memory are tested.

**2.The virtual memory benchmark:**

**Text

Description automatically generated**

Here the volatile memory mapped buffer is created and tested the same way the cache memory was but with no cache to speed up the computations.

**3.The physical memory benchmark:**

**Text

Description automatically generated**

For the physical memory test, a new output file is being created and the contents of a previously created file is being used as input. The transferred data speed is then calculated as in the other memory types and printed in the dataTable.csv file.

**4.Kernel module:**

**Text

Description automatically generated**

**Text

Description automatically generated**

Above is the implementation of the kernel module in which we set the 30th bit of CR0 register in order to disable the cache. For each function (cache on / cache off) we call the smp\_call\_function in oder to disable caches of all processors.

6.Test and validation

Graphical user interface, application, table, Excel

Description automatically generated

Here are the results of the benchmark program. We can see that the cache memory is indeed the fastest when increasing the data size and the other 2 have a slight increase when this happens, but nowhere near the speed of the cache. Please note that this results could be more accurate but due to the high time this program needs to be ran, the tests were run only 3 times each and the averages were noted.

# Conclusions

In conclusion, we can see how useful cache memories are at speeding up computations on our PC.

This project was hard to implement due to the way the OS manages the cache memory and hides it from the user space, resulting in the need to create kernel modules which can control it in some way. But this way, a lot of new programming concepts and new knowledge about how operating systems handle all types of data were taught to me.

# Bibliography

<https://stackoverflow.com/questions/29446895/hdd-benchmark-in-c-measured-transfer-speed-is-too-fast>

<https://lwn.net/Articles/252125/>

<https://unix.stackexchange.com/questions/6593/force-directory-to-always-be-in-cache>

<https://github.com/IT302/cache-lkm/blob/master/cache.c>

<https://stackoverflow.com/questions/48360238/how-can-the-l1-l2-l3-cpu-caches-be-turned-off-on-modern-x86-amd64-chips>

<https://stackoverflow.com/questions/29446895/hdd-benchmark-in-c-measured-transfer-speed-is-too-fast>