Visual Representation of Computer System Cache Hierarchy

Submitted by

Joshua Lindner

Computer Science

To

The Honors College

Oakland University

In partial fulfillment of the

requirement to graduate from

The Honors College

Mentor: Dr. Julian Rrushi, Associate Professor

Department of Computer Science and Engineering

Oakland University

April 7, 2023

Abstract

The CPU (Central Processing Unit) is the component in a computer system that carries out all the computations and calculations that are required. Because of this, good CPU performance is key in having a fast computer. To ensure good performance, processor designers often implement an internal cache that works like a hierarchy, with small amounts of very fast storage closest to the processing core, and large amounts of slower storage furthest from the core. Such a hierarchical design takes advantage of spatial locality, which says that if a piece of data is used, the surrounding data is likely to be used, and temporal locality, the likelihood that if a piece of data is used once, it will be used again.

Descriptions of the cache hierarchy can be confusing without models or diagrams. Such visual aids are made all the time, yet they can often be difficult to interpret without prior knowledge, which is a problem for those who have an interest in learning about computer architecture. Our program will serve as a visual representation of the cache hierarchy and will help fill the gap of visual cache representations that display pertinent information with clear visual distinctions between the cache levels.

Introduction

Research topics involving computer memory (and cache management) vary widely in scope and application. Memory is one of the core components that make up a computer, and is heavily relied upon by every other component in the computer system, so there is no wonder researchers are trying to discover the next breakthrough in RAM (Random Access Memory) technology.

Increasing Efficiency

Researchers are frequently theorizing new and varied ways to make the cache hierarchy more efficient. Efficiency of a computer's internal memory (local memory in the CPU) can be measured by multiple metrics, but one of the most useful measurements is memory bandwidth. Bandwidth can be considered the "width" of the connections, and determines the total amount of information that can be transmitted at a time; the memory bandwidth essentially defines the speed at which data can write out/be read from. Having low bandwidth means information has to be stalled in order to wait for data buses to clear, and this wastes precious clock cycles, the rate at which the CPU runs. Fortunately, well-implemented cache hierarchies within the computer's internal memory can help ease the bottleneck that low bandwidth presents. One proposed solution is to make morphable DRAM[1] (Dynamic RAM) cache layers, so that data storage and transferring is optimal. Due to it being dynamic, DRAM can store data and tag information in any arbitrary way. This morphability allows for an integrated DRAM controller to reconfigure the cache's tag and data layout in order to maximize efficiency.

Specialized Engineering

Because memory is so fundamental to a computer's functionality, it can be used in many ways. However memory that can retain information without having power is not very useful for someone whose main use of their computer is to train a machine learning algorithm how to identify human faces. Memory's wide scope of usability is not always ideal for computer systems dedicated to a single purpose, and so researchers have done work to formulate compositions of memory that are more tailored to a specific requirement. One such composition was proposed for IoT (Internet of Things) devices[2], specifically ones that have the ability to learn from their environment. This computer memory composition would utilize approximate computing (computation of approximate values) in order to cut down on computation costs while still providing 90% accuracy while training models.

Cache Simulation

Simulation of the memory hierarchy, which consists of the secondary storage, physical main memory, and the various levels of cache, is usually one of the preliminary steps in developing new internal cache systems. Simulated implementations are often done in place of physical ones due to the expenses that arise in creating new CPU cache layouts. Not only this, but existing tools that measure CPU performance tend to be irregular[3] and leave too much difference in between best- and worst-case estimates for the data to be useful. One published assessment of spin-transfer torque magnetic RAMs (STT-MRAMs)[4] performance at nanometer scales partially used device models in order to reach the conclusion that scaling down the technology resulted in improved performance (faster write operations, energy savings). In this article, Esteban Garzón and his colleagues were able to use cache memory simulations in order

to control for and record the writing and reading performance of their scaled-down implementation. Simulations like these are more complex and realistic, and can really only be understood by readers who are already very familiar with the cache hierarchy and its components. This leaves a gap in research done towards creating educational simulations/visual representations of the cache that less experienced individuals can understand. Visualization of memory content and traffic simulation has a secondary benefit in that it helps predictive analysis models detect side channels, namely leakage of memory data that could be exploited to break the security of a computing machine. On the other side of the scale are the reduced and minimal diagrams presented to readers within textbooks. These textbook representations do not include much extra information for the reader, and instead tend to be very abstract in order to make concepts easier to understand; abstraction, however, results in details being removed, and this causes the applicability of information to become less specific and more general.

Aims and Objectives

Introduction

Computer cache can be difficult to understand if one doesn't already have prior knowledge of how it works. The user must understand the benefits that arise from the various designs of computer cache systems, and the differences between them. Visualizations that attempt to illustrate the computer cache are either very complicated and detailed, or made too ambiguous that anything substantial can be learned from the representation. This project is a program that attempts to model a robust visual representation of the computer cache, specifically a hierarchical cache. This program's model is intended to be simple enough that inexperienced analysts can understand, while still containing useful and real-time modeled information.

Aims

1. To research common hierarchical cache implementations, and visualizations of them.

2. To obtain useful information from real computer cache systems.

3. To create a visual representation of a hierarchical cache that can be easily understood and learned from.

Objectives

1. The most common internal cache implementation is a hierarchical one, which takes advantage of spatial and temporal locality. Doing research on common cache implementations and how they are currently modeled will help us form our own visual representation.

2. Obtaining and displaying real cache information is critical to the usefulness of the cache visualization. Implementing real information into the model will help analysts obtain a realistic expectation of what computer cache systems are capable of.

3. Creating a visual representation allows for the making of decisions such as what information should be displayed, and how it can be conveyed to the viewer. This visualization will also fill the currently existing gap of useful and informative visual representations of computer cache systems.

Methodology

First, the focus was on researching common cache implementations cited in recently published papers. This was done by using resources like OU's Library OneSearch, Google

Scholar, and other respected sources available. Doing this allowed us to determine that a hierarchical cache is most commonly used, and therefore is most useful to model in our visualization. This also enables the determination of what information to show, and how not to model our implementation so as to avoid complicating things.

Then, the project's focus was on obtaining real computer cache statistics. Doing this increased the realism of the visualization in order to oppose some of the inevitable abstraction that was done. The purpose of such statistics is to shape the expectations of users who are not already familiar with the potential of computer cache systems. These statistics were obtained through tools that are already available to users, such as PSutil, Windows Task Manager, as well as lower-level tools like the underlying computer operating system.

The statistics measured and recorded in the application are the following: CPU percentage, base clock speed, L1 cache size, L2 cache size, L3 cache size, secondary storage size (RAM), number of logical processors, and number of physical cores. The figure below is a screen capture of the statistics measured and displayed within the application.

Figure 1: Statistics measured in visual application

The above statistics are obtained directly from the computer's operating system, and are implemented to update every second, allowing for accurate resource monitoring. It should be noted that the memory used and memory available values are in units of GB (gigabytes). The

above statistics, namely number of logical processors, number of physical cores, and CPU base frequency, are all consistent with the values that are displayed under the processor's tab in Windows Task Manager, as seen in the figure below.

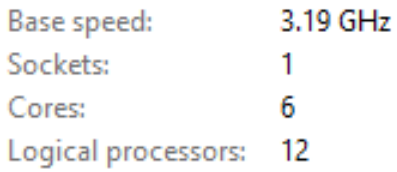

Figure 2: CPU statistics from Windows Task Manager

Finally, the focus was put towards creating a visual model of our decided cache implementation that includes the statistics gathered from our previous gatherings. Visual elements were implemented and modeled in the Python programming language with the inclusion of the Tkinter GUI tool. The application contains two visual representations of the computer cache system: one representation is seen in Figure 3, and contains the capacity of the various levels of cache within the computer system, as well as the main memory (RAM) that can also be considered as a form of cache storage.

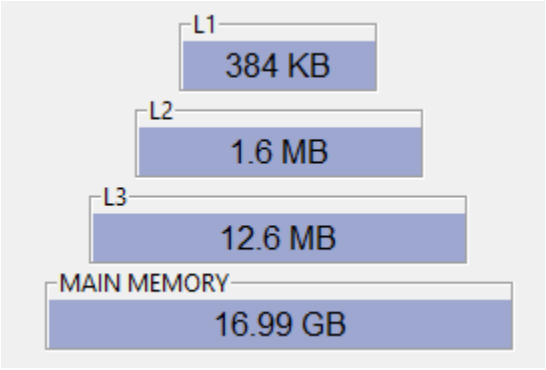

Figure 3: Cache hierarchy on Intel i7-8700 processor

This representation was made more abstract in order to be similar to visualizations that appear in textbooks or other basic forms of representation. In the above figure, the processor used is an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz. The 3.20GHz is the base clock speed of the processor, and represents the minimum number of clock cycles per second the CPU runs at. The values gathered and displayed by the application are specific to the computer system that the application is executed on. This can be seen in Figure 4, where the capacities displayed within the various levels of the hierarchy are different from those shown above in Figure 3.

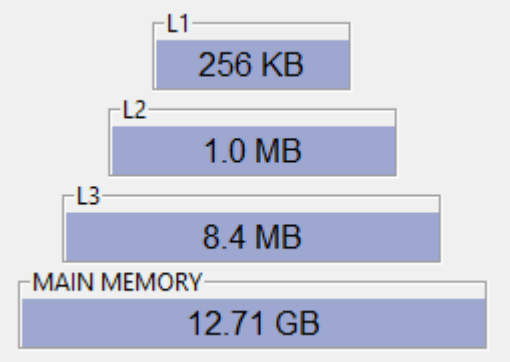

Figure 4: Cache hierarchy on Intel i7-8550U processor

The processor in Figure 4 is an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz. It can be seen from the visualizations that the capacity of the processor in Figure 3 is greater than that of the one in Figure 4. It should also be noted that the capacities of the different cache levels that are displayed in the hierarchy are consistent with the capacities shown when observing the computer's processor information in Windows Task Manager, as seen below.

Figure 5: Cache capacities shown in Windows Task Manager

The second visualization implemented in the application contains more details than the first, and is more similar to the physical hardware implementations of computer caches. Figure 6 shows an example of this visualization.

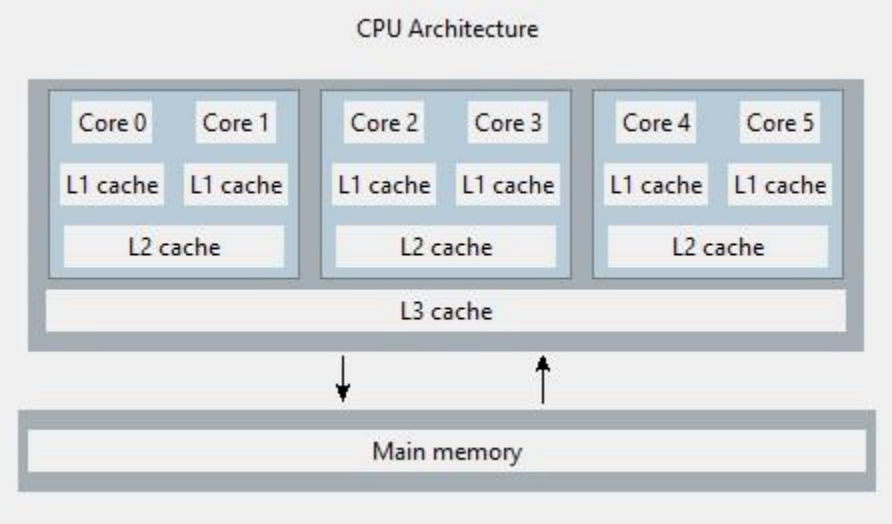

Figure 6: Multi-core cache representation

The visualization in the above figure contains more information than the first, and highlights the number of cores contained in the processor as well as the separation of cache levels between components. Each core in a processor is responsible for executing instructions issued to it by the scheduler, and having multiple cores allows for multiple operations to be executed in parallel, or at the same time. In typical CPU architectures, each core has its own separate level 1 (L1) cache, followed by a larger level 2 (L2) cache that is shared between two

cores. Separation of cache in this way prevents the various cores from writing over each other's data, and increases the total throughput of data. Shared by all of the cores in the processor is the embedded level 3 (L3) cache, which acts as an intermediary between the small but quick L1 and L2 cache levels and the large but slow retrievals of data from main memory. The arrows in the figure represent the movement of data between the processor and RAM.

The number of cores displayed is algorithmically implemented, allowing for a dynamic representation that is specific to the computer's CPU. Figure 7 shows the resulting visualization when the program is run on a different computer.

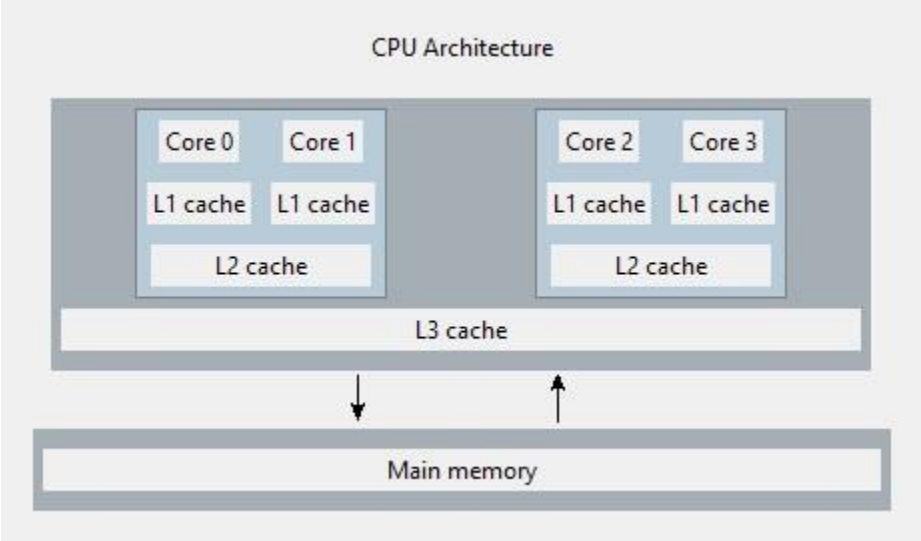

Figure 7: Multi-core representation on 4 core processor

It is important to note that the layout depicted in the visual representation is not completely accurate to the architecture of the computer system's CPU, but rather is a general representation.

One additional visual implementation of the application is a graph of the CPU utility percentage over time. The measurement of CPU utility refers to the amount of the CPU's total resources being used at any given time, and is another important metric in determining CPU performance. It's important to note that in general the lower the CPU utilization, the better. Consistently high CPU utilization can be the result of overloaded processor cores, and is a notable sign that the processor is being overwhelmed by requests. Typical CPU utilization values fluctuate often, and therefore graphs of the performance over time can give the user valuable information into the performance of their processor. Figure 8 shows a graph of the CPU utilization (y-axis) over time (x-axis).

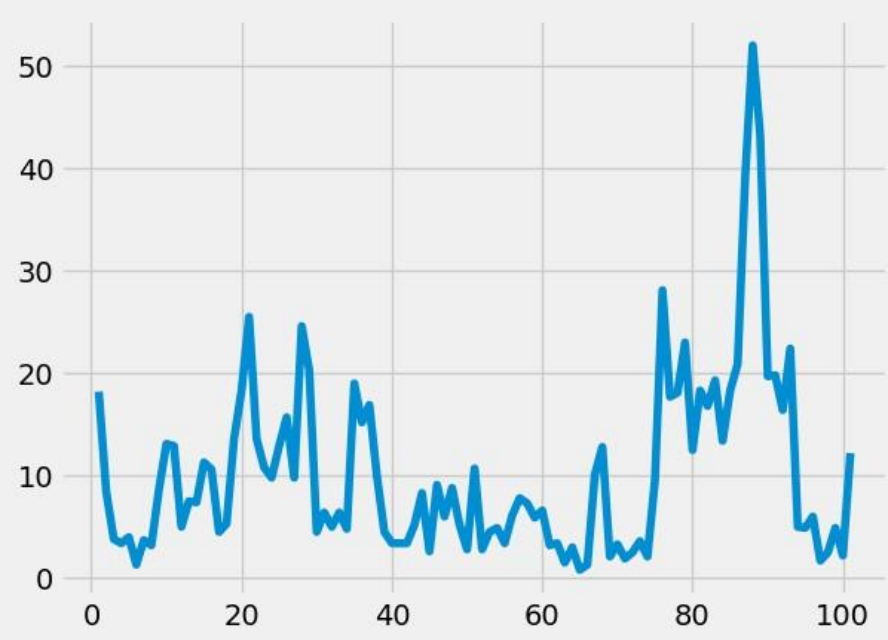

Figure 8: CPU utilization over a period of 100 seconds

The scale of both axes varies with both time and the utilization percentage, and updates once per second in order to provide real time performance statistics to the user. Figure 9 below shows an instance of the CPU utilization graph over a longer period of 350 seconds.

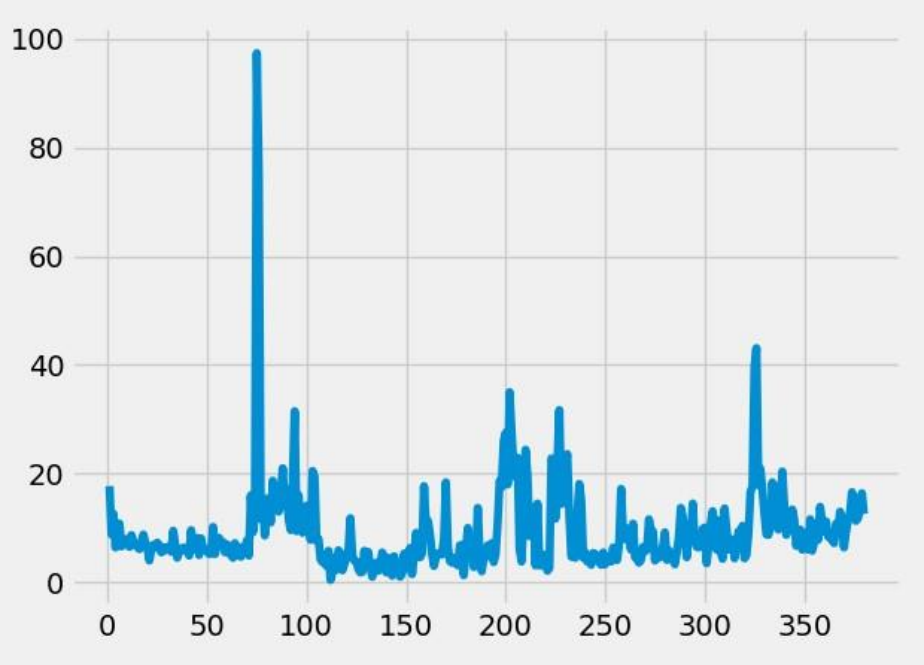

Figure 9: CPU utilization over a period of 350 seconds

Because of the tools used, this application and its visualizations are viewable on both Windows and Linux operating systems, with Linux systems providing slightly more information due to the way the OS collects data about the CPU. In all, this visualization tool acts as a midpoint between complex cache representations and abstract cache diagrams, and serves as an educational tool to deepen the user's interest and understanding of computer cache systems.

IRB

This project does not involve human subject research and does not require IRB approval.

Results

Initially, the focus was on creating an application that would record the real time paths of data as they travel through the multiple levels of internal CPU cache. This capturing of data would be simple yet robust, allowing for the complex processes of CPUs to be easily encapsulated and displayed in our application. In doing research on possible methods of recording the CPU's operations, our team discovered that CPU manufacturers have built-in transparency of their CPU's tasks, meaning analysis of the CPU's direct actions is very limited and difficult to do. This transparency exists as monitoring of the CPU causes unnecessary overhead and negatively affects performance, both of which are counter to the desired nature of computer processors. Faced with this realization, our team decided to shift our focus onto making an application that measures CPU statistics rather than mapping out and capturing the paths taken by data. This change still allowed for the cache visualization application to serve as an educational and informative resource for those who are interested in gaining more knowledge on computer cache systems.

Discussion

It was expected that this project is more applicable than visualizations that appear in textbooks, yet less complex than representations that appear in research articles. The inspiration for the topic of this project came from the personal difficulties our team faced when gaining understanding of modern cache implementations. Acquiring knowledge on such a topic is not straightforward, as a lot of information mentioned in articles mentioning cache implementations is either abbreviated, or ambiguous. Comprehension of literature can also be difficult due to the assumptions of prior knowledge made by authors, which results in a very steep learning curve of information.

There were also considerations of the possibility that this project could inspire the creation of other cache models, and through the making of those models, a more efficient cache layout could be developed. There is strong reason to conclude that the usefulness of the application is enough that it can serve as a "launch pad" to project the observer's interests further into the topic of computer cache systems.

Another theorized outcome of the application is the use of it within learning/educational settings, as it could be a good tool to help students better understand real cache implementations. In order for this to be a possibility, widespread implementation must be established and the application distributed across hundreds of devices. Such a scenario would prompt more attention and refinement in our implementation of recording the computer system's activity.

Conclusions

On a more personal note, this project has allowed me to expand my knowledge on a topic that I find to be very interesting. For my other group partners, they no doubt benefitted from the knowledge gained in not only researching this topic, but also through the process of developing our application. I would also like to thank my faculty mentor Dr. Julian Rrushi for giving our team the initial idea of developing a cache visualization application, as well as the opportunity to work with him and communicate on the progress of our capstone project and the formulation and composition of this thesis.

Despite the changes that were made to the initial project plans, we believe the application is still useful for not only informative purposes but also for foundational ones that can be worked off of and potentially used for later advancements in technology. The project still meets the proposed objectives of bridging the gap between abstract computer cache visualizations and complex, detail-oriented cache architecture representations.

Biographical Note

Joshua Lindner is a Senior at Oakland University pursuing a Bachelor's of Science in Computer Science and is on track to graduate in April 2023. Joshua is in the Honors College where he has worked hard to earn and maintain a GPA suitable for the Dean's List thanks to his desire to constantly learn more and challenge himself. Joshua is very passionate about everything computer-related, and has been since playing video games as a young child. Today, he is most passionate about cybersecurity and all the components that make up a secure computer network.

Joshua is pursuing his interests in cybersecurity and advancing his career by studying and preparing for the CompTIA Network+ certification exam, and hopes to obtain the CompTIA Security+ certification by the end of his final semester.

Bibliography

1. Cha S, Kim B, Park CH, Huh J. Morphable DRAM Cache Design for Hybrid Memory Systems. ACM Trans. Archit. Code Optim. 2019 Jul [cited 2023 Jan 15];16(3):1-24. doi:https://doi-org.huaryu.kl.oakland.edu/10.1145/3338505

2. Tastan I, Karaca M, Yurdakul A. Approximate CPU Design for IoT End-Devices with Learning Capabilities. Electronics (Basel). 2020 [cited 2023 Jan 15];9(1):125-. doi:https://doi.org/10.3390/electronics9010125

3. Trotter JD, Langguth J, Cai X. Cache simulation for irregular memory traffic on multi-core CPUs: Case study on performance models for sparse matrix-vector multiplication. J Parallel Dist Comp. 2020 [cited 2023 Jan 15];144():189-205. doi:https://doi.org/10.1016/j.jpdc.2020.05.020

4. Garzón E, De Rose R, Crupi F, Trojman L, Lanuzza M. Assessment of SST-MRAM performance at nanoscaled technology nodes using a device-to-memory simulation framework. Microelectric engineering. 2019 [cited 2023 Jan 15];215():111009. doi:https://doi.org/10.1016/j.mee.2019.111009

5. Thompson C, Gould M, Topham N. High Speed Cycle-Approximate Simulation of Embedded Cache-Incoherent and Coherent Chip-Multiprocessors. Int J Parallel Prog. 2018 [cited 2023 Mar 1];46(6):1247–1282. doi:https://doi.org/10.1007/s10766-018-0566-x.

6. Siddique NA, Grubel PA, Badawy A-HA, Cook J. A performance study of the time-varying cache behavior: a study on APEX, Mantevo, NAS, and PARSEC. J Supercomput. 2018 [cited 2023 Mar 1];74(2):665–695. doi:https://doi.org/10.1007/s11227-017-2144-1.

7. Tarapore D, Roozkhosh S, Brzozowski S, Mancuso R. Observing the Invisible: Live Cache Inspection for High-Performance Embedded Systems. IEEE Transact on Comput. 2021 [cited 2023 Mar 1];71(3):559–572. doi:https://doi.org/10.1109/tc.2021.3060650.

8. Le A, Hoang T, Tsukamoto A, Suzaki K, Pham C. A Real-Time Cache Side-Channel Attack Detection System on RISC-V Out-of-Order Processor. IEEE Access. 2021 [cited 2023 Mar 27];9(1):164597-164612. doi:https://doi.org/10.1109/ACCESS.2021.3134256

9. Lv M, Guan N, Reineke J, Wilhelm R, Yi W. A Survey on Static Cache Analysis for Real-Time Systems. Leibniz Trans. Embedded Syst. 2016 [cited 2023 Mar 27];7(3):05:1-05:48. doi:https://doi.org/10.4230/LITES-v003-i001-a005

10. Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, and others. The worst-case execution-time problem–overview of methods and survey of tools. 2008 [cited 2023 Mar 27];7(3):1-53. doi:https://doi-org.huaryu.kl.oakland.edu/10.1145/1347375.1347389