SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY —
INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Extension of MemAxes Sample Visualization to Process AMD IBS Samples

Daniel P. Schenk

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Extension of MemAxes Sample Visualization to Process AMD IBS Samples

Erweiterung der Sample-Visualisierung in MemAxes um AMD IBS Samples

Author:	Daniel P. Schenk
Supervisor:	Prof. Martin Schulz
Advisor:	Stepan Vanecek
Submission Date:	16.8.2023

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 16.8.2023 Daniel P. Schenk

Acknowledgments

I want to thank Ole Hesse for support during the time of writing this thesis. Thanks to Philipp Jurašić's suggestions and feedback the command search feature has its fitting name instead of painfully generic "action search". For providing a very interesting topic that created many opportunities for me to learn, I want to thank my thesis advisor Stepan Vanecek.

Abstract

To optimize a program in the field of High-Performance Computing, it is necessary to have a detailed understanding of the performance characteristics of the code. MemAxes is a visualization tool for gaining insight into the relations between hardware-level events like cache misses, and parts of the code. Originally, MemAxes was only designed to work with Intel processors. In this work, we will present newly developed functionality for analyzing performance data from AMD processors, gathered with the IBS sampling mechanism.

IBS samples provide data, for example of TLB performance which is not available in samples from Intel processors. We designed new visualization features to take advantage of this additional data. In the process we overhauled other functionality of MemAxes, improving performance and providing new, more efficient interaction methods. Among these features is a completely new interaction scheme that allows users to control the visualization using text commands.

Contents

1 Introduction

High-Performance Computing (HPC) drives progress in the natural sciences and engineering. Ever more complex simulations necessitate both more powerful supercomputers, and more efficient use of available resources. More optimized code brings down costs and enables simulations that were not possible before [1]. MemAxes was designed as a tool to help in performance optimization for HPC systems through visualization of performance samples. Visualization enables users to build an understanding of datasets that are too large or too abstract to comprehend without assistance. The sample sets visualized in MemAxes contain hundreds of thousands of numeric values. No conclusions can be drawn from this data without automated processing. Visualization not only processes the data to derive metrics but can present a visual metaphor that enables understanding of the process in which the data was generated [2].

At the core of MemAxes is the goal to not only present hardware usage statistics, but guide users in linking the hardware phenomena to parts of their code. Making the connection between low-level hardware phenomena, and their causes in the code, is key to well-informed optimizations. MemAxes can go one step beyond the hardware and program levels and relate observations on the hardware level to semantics on the application level, like 3D-coordinates in a physics simulation [3][4].

To collect the data across these levels, MemAxes uses hardware performance samples which are enriched with program- and application-level context by the Mitos sampling application. On the lowest level, hardware performance samples are collected using performance monitoring units (PMU) of the processor. These units provide hardware-level information that is hidden from the program level. The implementation of performance monitoring units, what data they can collect, and how they are controlled, is vendor specific. MemAxes was originally designed for use with samples collected by the PEBS sampling mechanism of modern x86 Intel CPUs [3][5]. Processors by Advanced Micro Devices (AMD), the second major manufacturer of modern x86 CPUs, use the Instruction Based Sampling (IBS) mechanism.

This work will describe how we made MemAxes compatible with sample sets collected with the IBS sampling mechanism. We ensured that IBS data is properly processed and all features of MemAxes work with IBS samples. To make IBS-specific data accessible to the user, we implemented new features for visualization of the newly available data.

Apart from visualizing newly available data, we will improve the preexisting visualization in MemAxes. We will show how we improved overall performance of the application for smoother operation. We also modified all visualization modes to improve legibility and make correlations more obvious to the user. For better analysis of correlations in the data, we also introduced a pre-selection system to accompany the MemAxes filtering tools. The new system improves the predictability of the original system and makes the connections between the different view modes clearer.

Our final contribution is a new interaction scheme for MemAxes. With command search it is possible to search for filtering options and explore correlations in the dataset using intuitive text-based search like in an online search engine. This feature enhances analysis of data from all levels by accepting both inputs referencing raw data, and queries based on semantics in the hardware and program context. With this new feature, users can tell the program through high-level text input, what they want to visualize.

2 Background

First, we will give an overview of the aspects of High-Performance Computing that are relevant to this work. Then we will describe the concepts and implementation of MemAxes as of the time we started our development process. Finally, we will show the key differences between IBS and PEBS sampling.

2.1 High-Performance Computing

High-Performance Computing is the field concerned with the use of supercomputers. A supercomputer is a computer of extraordinary compute performance that is built for scientific or engineering calculations. For these calculations, a high throughput of numerical operations is needed [1].

The instruction throughput of a computer can be increased either by executing instructions faster or executing more instructions in parallel. For the last 20 years, increases in performance were mostly gained through increased parallelism [6].

Computers utilize three forms of parallelism.

- Instruction-Level Parallelism (ILP) is the parallelism that can be exploited by concurrent execution of multiple instructions from the same instruction stream. Modern processors utilize ILP automatically and do not expose this functionality to the programmer, so ILP is the least relevant form of parallelism for this work [7].

- Data-Level Parallelism (DLP) is the parallelism that arises from the same instruction being applied to multiple data elements. Specialized hardware exists that can process multiple data elements in parallel, while profiting from the reduced cost of managing only a single control flow. Modern CPUs have vector extensions to exploit DLP, which are exposed to the programmer. Often generating code that utilizes vector extensions is left to the compiler for convenience and to preserve maintainability and portability of the source code [8].

- Thread-Level Parallelism (TLP) is the parallelism of multiple execution threads, independent instances of a program which all have their own control flow. TLP is used by multicore processors which provide multiple cores that can operate in

parallel. Many processors also implement multithreading of cores, so that one core can process multiple instruction streams in parallel. TLP is exposed to the programmer and must be actively managed. For utilizing TLP, many languages provide abstractions of threads, and frameworks like OpenMP exist that allow for a higher-level specification of sections that can be processed in parallel by multiple threads [9].

For the last 20 years, clusters have been the most popular architecture for supercomputers. In clusters, many computer nodes are connected through a network, all executing code to work on the same problem. A node is a device with one or multiple processor sockets, usually accompanied by specialized accelerators. On-node communication is much faster than communication with other nodes. A socket provides a CPU with main memory and I/O. Modern CPUs have multiple cores, independent processing units, which often work on more than one execution stream [10].

This work focuses on performance aspects of processors on nodes. Currently, performance data from more than one node, or anything related to accelerators, are not visualized in MemAxes.

2.2 MemAxes

The original MemAxes was developed by Alfredo Giménez and others around 2014 at Lawrence Livermore National Laboratory (LLNL) [11]. Development continues at the Technical University of Munich (TUM). The main feature added to MemAxes at TUM is support for the sys-sage hardware topology capture library. The development of MemAxes at TUM is part of the EU-funded Software for Exascale Computing (DEEP-SEA) initiative [12]. All work was performed on the version of MemAxes developed at TUM. MemAxes is written in C++ and uses the Qt5 GUI framework.

MemAxes implements three main views.

2.2.1 Hardware Topology View

The first view of MemAxes is the hardware topology visualization. In this view, a hierarchical Icicle or Sunburst plot is used to visualize the memory hierarchy. Compute resources are the leave nodes of the hierarchy, the root is the shared memory. Resources are colored based on the memory access cycles associated with them.

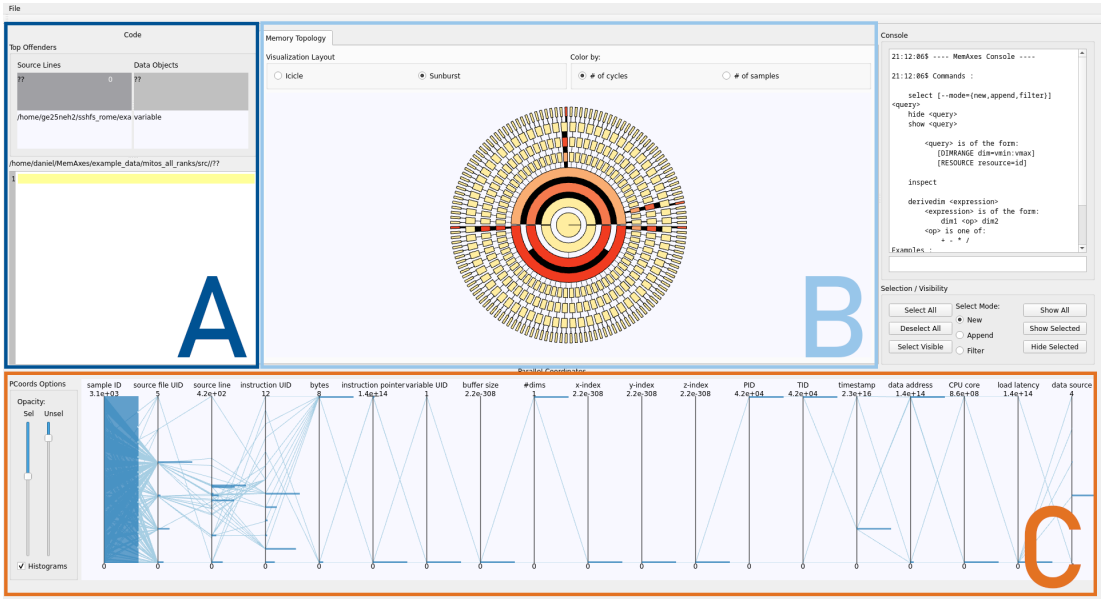
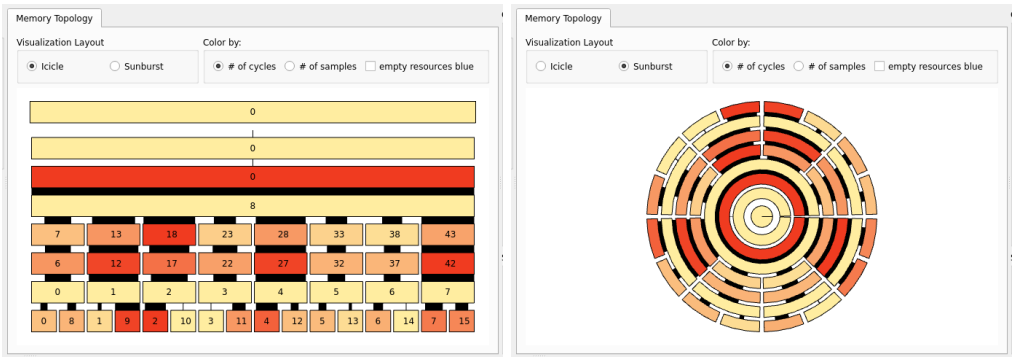


Figure 2.1: The 3 views of MemAxes. View A is the code view. B is the hardware topology view. C is the parallel coordinates view.



(a) Icicle plot (b) Sunburst plot

Figure 2.2: Visualization modes of the hardware topology view

The speed of memory accesses is the determining factor for the performance of applications on modern computers. Processors are much faster at performing computations than main memory can provide data. For example, the AMD Ryzen 5800X desktop processor that was used to capture the performance data visualized in many of the screenshots in this work, has maximum frequency of 4.7 GHz. Each of the eight cores

can process up to six instructions per cycle [13]. The theoretical maximum instruction throughput are $4.7 \cdot 10^9 s^{-1} \cdot 8 \cdot 6 = 225.6 \cdot 10^9$ instructions per second. The DDR4 main memory of the test system has only two channels capable of a combined $7.2 \cdot 10^9$ transfers of 64 bit per second. Obviously, the main memory cannot feed the processor all the data necessary to achieve maximum compute performance.

Memory hierarchies were designed to counter the gap in performance between main memory and compute units. Faster and higher-bandwidth memories are possible, but there is a trade-off to be considered, as these higher-performance memories are more expensive and consume more power. For this reason, modern CPUs have built-in memories called caches that are much faster than main memory, but also smaller in capacity [14].

Using smaller memories to serve most memory accesses is viable because most programs exhibit spatial and temporal locality. Spatial locality is a program's tendency to make accesses to addresses that are close to previously accessed addresses. An example for good spatial locality is sequential iteration of an array. Programs with good spatial locality exploit that caches are organized in lines of coarse granularity, so two consecutive array elements are likely stored in the same line. The line is loaded into the fastest cache when the first element is accessed, the next elements can then be read from the cache. Modern processors use prefetching, loading lines into the cache before they are used for the first time. For prefetching to give good results, an easily predictable memory access pattern is necessary. A program displays temporal locality if it accesses the same address again shortly after the previous access. This increases the chance of a cache hit because data is removed from faster caches if other data needs the space in the cache. A short reuse distance reduces the probability that the data was evicted from the cache since the last access. It is the programmer's responsibility to write programs that have good locality properties, so that most memory accesses can be served by faster caches [15].

The coloring of resources in the hardware topology view is based on the memory access cycles associated with a resource compared to the access cycles of other resources on the same level of the memory hierarchy. With this design it is easy to spot imbalance between cores, but it is not possible to reason about how well caches were utilized in general [3].

In addition to cache behavior, MemAxes also visualizes Nonuniform Memory Access (NUMA) regions. On systems with NUMA, not all accesses to memory are equally fast, because some must be handled by memory that is slower to reach. An example of this are multisocket systems, where accesses to the memory of the other socket are slower than accesses to the memory of the processor's own socket. While accesses to a socket's own memory can use the memory directly, for accessing the other socket's memory, an access must be processed by two memory controllers and must go through the socket

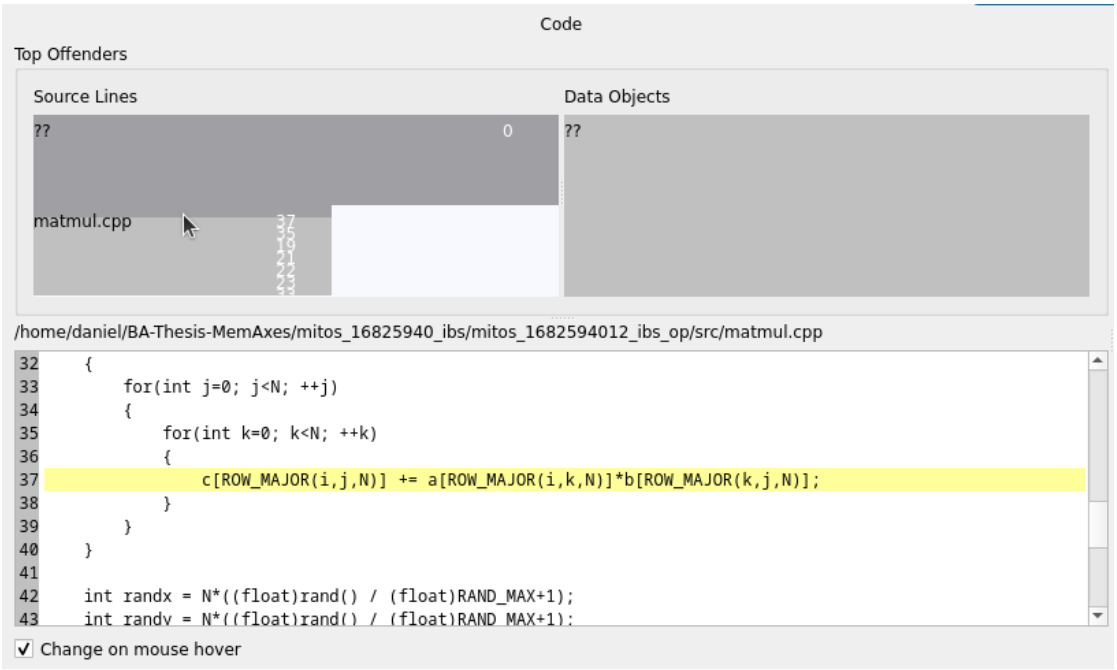


Figure 2.3: Code view in MemAxes.

interconnect [16][17].

Programmers must consider the allocation policy when working with NUMA systems. The most common allocation policy is first-touch, in which memory is allocated in the memory of the processor that first uses, not necessarily the one that allocated it [17].

With the selection tools of the MemAxes hardware topology it is possible to identify if there were accesses across NUMA regions. To do so, one uses the selection tools of MemAxes. In MemAxes the sample set can be filtered. All views share the same selection system, enabling search for correlations across visualization domains. To select all samples associated with a NUMA region, users can click on the representation of the NUMA region in the hardware topology view. Slow accesses from other sockets are clearly visible in the hardware topology views as highlighted compute resource nodes on the opposite side of the sunburst, or not below the selected NUMA region in the Icicle plot. After such a selection, the code view can be used to trace the problem back to a line of code [4].

2.2.2 Code View

The code view serves the important purpose of revealing the relation between the phenomena visualized in the other views, and the source code. Two parts make up the code view, the top offender ranking, and the code editor.

In the top offender ranking, source files and source lines are ordered in descending order by the number of memory access cycles associated with them. the ranking is illustrated by hierarchically stacked horizontal bar graphs. On the top level, the relative share of access cycles caused by the source files can be compared. Inside the source file bars, there are smaller bars representing the code lines.

In the source code view, source lines or source lines can be selected by clicking on the corresponding bar in the top offenders ranking. The read-only code editor widget shows the source code and highlights the top offending line. When a selection is made, the code editor jumps to the selected line and highlights it. To find out about how a phenomenon from the other views is related to the source code, one selects the samples associated with the phenomenon. The top offender ranking automatically changes to a ranking based on only the samples included in the selection [3].

2.2.3 Parallel Coordinates View

In the parallel coordinates view, all sample attributes are plotted. Parallel coordinate plots are general purpose plots for visualizing multidimensional data [2]. Dimensions are visualized as vertical axes in a horizontal layout. For every object in a parallel coordinates plot, a line is drawn between neighboring axes, passing the axis at the value of the object in that dimension. For example, in figure 2.4 one object is marked red. With this representation, correlations between axes form recognizable patterns from the connecting lines in the space between the axes [18].

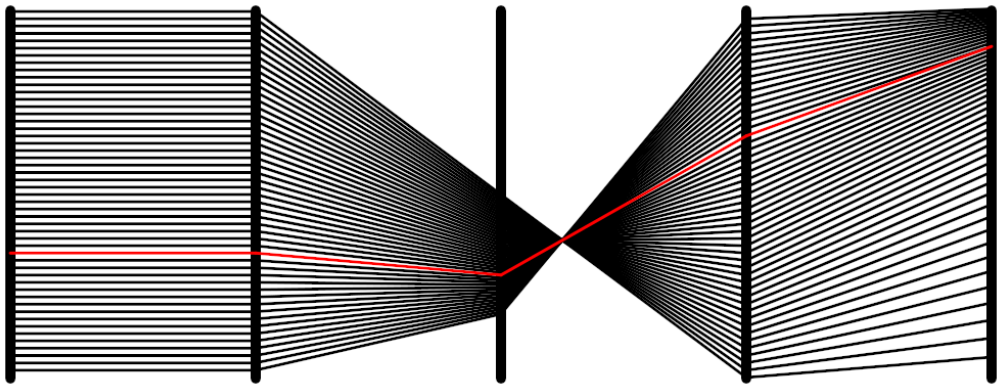


Figure 2.4: Parallel Coordinates example.

Parallel coordinates plots existed before interactive computer visualization was common, but only saw limited popularity. While it is possible to convey information about a multi-dimensional dataset through a parallel coordinates plot in print media, the true strength of this visualization idiom is facilitating data exploration in interactive applications. If users can reorder the axes interactively, they can investigate the correlations between different attributes, not just the ones the visualization designer wanted to show [2]. For this reason, in MemAxes, the parallel coordinates axes can be freely moved without perceptible snapping to a grid [4].

A weakness of parallel coordinates plots is that it is hard to judge the density of the distribution of samples for a point on an axis, because lines can be drawn above other lines [19]. MemAxes addresses this weakness by implementing the parallel histograms variation of parallel coordinates, where Histograms are drawn on every axis. With this modification, the density of samples on the axis can be effortlessly judged. For MemAxes this is especially important, because some sample attributes represent flags, which can take on only two values, so overlap of lines makes estimating their number impossible.

The parallel coordinates view also reacts to selections performed in the other views. Lines representing selected samples are assigned a different color than lines for not selected samples. The opacity of selected and not selected lines can be set with a slider to hide one of the groups. Samples that are not selected are excluded from the calculation of the histograms. Users can select value ranges on axes by dragging a selection box. All selections from all views can be canceled by clicking the "Select All" button [3].

2.3 Sampling and Mitos

For performance analysis of a program, it is necessary to record the performance of a program run. There are multiple methods for collecting performance data, relevant to this work are only sampling and instrumentation.

2.3.1 Sampling-based Performance Analysis

Performance samples contain detailed hardware information about the execution of an instruction, like the cache level that was used to serve the memory access. To collect performance samples, it is necessary to use the Performance Monitoring Units because otherwise, this low-level information is hidden from programs [5].

Samples are taken at some fixed or randomized interval, or in Precise Event Sampling (PES), a sample is taken when a specific event, like a cache miss, is triggered. MemAxes was designed for visualizing samples taken at fixed intervals. Samples do not allow to trace the execution instruction by instruction, because not every instruction is sampled. Only a very small fraction of instructions is sampled because otherwise, the amount of data generated would be enormous and writing it to memory would seriously limit performance. Sampling only a fraction of instructions still serves the purposes of performance analysis perfectly well if the sampling is unbiased. In unbiased sampling, how often an instruction is sampled, is proportional to how often it was executed [20]. For performance analysis, the instructions that were executed more often are of higher interest than instructions that were rarely executed. Amdahl's law [21] states, that optimization of a section of the code can reduce the runtime of a program by no more than the runtime of the unoptimized section [22]. For that reason, the instructions that are less likely to be sampled are less needed for performance analysis anyway.

An advantage of sampling is that it barely influences the performance of the execution [23]. As a disadvantage, sampling can only collect hardware-level information, because it is not known to the processor, which program or application-level objects or phenomena addresses relate to [5].

2.3.2 Instrumentation

In instrumentation, the program is extended by functionality to log data. Instrumentation can be performed manually, i.e., by modifying the code to measure the execution time of a section and print the result, or automatically by tools that instrument the program before compilation.

Instrumentation can give insight into program- or application-level behavior that hardware samples cannot provide. The main disadvantage of instrumentation is that it

alters the program, potentially influencing performance. It also does not have access to hardware-level information [23].

2.3.3 Mitos Sampling Interface

MemAxes is designed for use with samples conforming to the Mitos sampling interface. Mitos specifies an interface for collecting hardware samples and enriching them with program-level and application-level context. Users can annotate the buffers they create so that Mitos can reconstruct the program level information of the indices used from the address in instruction samples. More advanced application-level attribution can be defined by the programmer, for example to attribute an instruction to a position in the 3D space of a simulation [5].

A sampling tool that generates IBS samples in compliance with the Mitos sampling interface was developed at TUM. For this work, mainly two pre-existing example data sets were used.

The IBS example dataset was generated on an AMD Ryzen 5800X desktop processor with eight cores and 16 threads. This processor model is based on the AMD Zen 2 microarchitecture and has eight cores, sixteen threads, which all access the same main memory. As a parallel example code, a simple matrix-matrix multiplication was used. No optimizations for better cache uses were implemented. Parallelization was performed using by applying an OpenMP pragma to the outermost loop. 39866 samples are included in this dataset.

A second dataset was captured on a system with two Intel Xeon Platinum 8360Y scalable processors capable of operation up to 2.4GHz with 36 cores, 72 threads each. Both processors have their own memory, so there are two NUMA regions. A Jacobi method iterative system of linear equations solver ran during the performance sampling. Parallelization of the code was performed using the Message Passing Interface (MPI). The dataset consists of 3087 samples, of which four were manually removed because they included nonsensical memory access latencies of about 20 years.

2.4 Differences between Samples from AMD and Intel Processors

In 2007 AMD introduced the IBS sampling mechanism in their 10h family of processors [24]. On Intel processors, Precise Event Based Sampling (PEBS) serves a similar purpose, but these vendor-specific sampling mechanisms are far from interchangeable. PEBS is focused on programmability and precise capture of specific events, IBS collects more data, but lacks the programmability for precise captures of PEBS.

PEBS, which MemAxes currently uses, can be programmed to count four out of 62 subevents of 12 events. When the counter overflows, a sample of the machine state is taken. This precise event sampling allows for sampling only events that are deemed of interest, like misses of some cache level. This advanced technique called Precise Event Sampling (PES) is not utilized by Mitos and MemAxes, Mitos samples are meant to be taken at fixed time intervals.

IBS has two sampling flavors, fetch and op samples. The counters for determining During fetch sampling only support counting fetched microinstructions and clock cycles. This makes it harder to produce samples for gaining insight into a specific event, but for the purposes of this work, this is not a limitation, because MemAxes does not utilize advanced PES. 16 attributes are sampled from microinstruction fetching. Fetch sampling collects data mainly about the instruction caches and instruction TLBs. The 44 attributes of op samples include the cache level used, information about data TLBs and branch prediction. Unlike the PEBS samples MemAxes previously used, IBS samples enable insights into the performance of address translation [20].

Modern CPUs need to perform address translation for every memory access because programs operate on virtual addresses. On modern computers, every program is assigned its own virtual address space by the operating system. Virtual memory is an important security feature of modern computers. The virtual address space is organized in pages, which map to a region in physical memory. To translate an address, the position of the page in physical memory must be looked up in the page table, and an offset must be calculated. CPUs implement dedicated memory management units (MMU) for this purpose, which can walk the page table at high speeds. Even with a highly optimized page table walker, it would not be possible to retrieve the physical address fast enough to not slow down accesses to fast caches. For this reason, page address translations are also cached in dedicated resources called Translation Lookaside Buffers (TLB).

Because the number of pages used is much smaller than the number of addresses, and most programs use only a few of their pages most of the time, caching in TLBs is far more effective than data caches despite the very small size of TLBs [25]. For example, the data TLB of the Zen 4 microarchitecture has only 72 entries in L1, and 512 in L2[26]. Although TLB misses are rare, when they happen, they have a strong negative impact on performance, so support for visualizing these misses should be added to MemAxes.

3 Implementation

We will present our contributions to MemAxes in five areas.

1. Ensuring Compatibility with IBS samples and extending the file parsing.
2. Reimagining the parallel coordinates view.
3. Making the Code view more interactive.
4. Implement interaction with the hardware topology view that is consistent with other views.
5. Introducing Command Search, a new interaction scheme based on the semantics of text input.

3.1 Reading IBS Samples

Both PEBS and IBS sample sets are provided as lists of comma separated values (CSV), where one line holds the attribute values of a sample and a header contains the names of the attribute names. The most important attributes are

- Name of the source file as a string.
- Line of code in the source file that Mitos attributed the sampled instruction to.
- Name of the instruction sampled as a string.
- Sample attributes to identify the buffer and index used in the memory access.
- Identifier of the CPU core used.
- Latency of the memory access in cycles.
- Level of the memory hierarchy used as a string.

Sample data is stored in a list of structs, each holding the attributes of one sample. The sample structs only contain `long long` attributes. String attributes are assigned an identifier that can be used to retrieve the string from a list of the unique values of the string attribute. This does not apply to the memory level accessed, which is decoded into a number between 1 (L1 cache) to 4 (main memory). Treatment of different axes is hard-coded, but the parsing is independent of the order of attributes because the index of the attribute in the line is determined by finding the axis name in the header line [11].

IBS samples contain 44 additional attributes for op samples, or 16 in fetch samples [20]. In this section, we will describe how we decode the cache level used in IBS samples where the `Level` attribute is invalid. We present our new approach to storing the additional data, and how we adapted MemAxes to work with sample sets which use other attribute names than the intended ones.

3.1.1 Decoding Cache Levels

In IBS samples, the cache level used is not stored properly in the `Level` attribute. Information about the cache level used is available through IBS-specific flags. Only the op samples provide the necessary information to decode data cache levels used, fetch samples include only information on instruction caches. This work focuses on op samples because fetch samples cannot provide all data necessary for MemAxes to fulfill its design purpose of visualizing data memory performance. All names of IBS-specific values used in this work are consistent with the AMD Bios and Kernel Developer's Guide (BKDG) which uses a camel case naming convention (first letter of word in upper case), while in the example dataset, the IBS-specific axes have names in snake case (words separated by underscores) [27].

In op samples, the attribute `IbsDcMiss` is set to one, if a L1 cache miss occurred, and to zero otherwise. The attribute `IbsL2Miss` indicates that an access missed in the L2 cache. It is not possible to differentiate between accesses to L3 cache and main memory using the data available in IBS samples. When reading samples for which`Level` could not be decoded, and the IBS op sampling specific attributes are available, our modified MemAxes performs data source decoding based on IBS attributes.

For some op samples, the `IbsL2Miss` flag is set, although no `IbsDcMiss` occurred. This behavior appears illogical because the CPU should not start an access to L2 when the L1 access succeeded. The BKDG explicitly states, that `IbsL2Miss` may be set because of a different operation than the sampled one [27]. This type of wrong attribution is called skid [24]. In MemAxes, we handle these samples as accesses to L1 because the samples instruction was an L1 access, and we do not have further information that we could visualize about the other instruction that missed in L2.

3.1.2 Storing IBS-specific Data

For storing other data than what is available in the base version of MemAxes, we introduced a new data structure. We did not extend the sample structs by new attributes for IBS data for extensibility reasons. Manually creating an IBS sample struct would not make it easier to adapt MemAxes to other additional sample data in the future. Instead, we created a sample attribute matrix. The matrix can be generated for any number of samples and attributes only limited by the memory of the machine MemAxes is running on.

3.1.3 Processing non-conforming Axis Names

Previously, MemAxes could only load datasets where all names of the required attributes exactly match the specification. If an attribute is missing from the data, it was not possible to visualize the remainder of the dataset. To avoid this causing annoyance and to provide broader compatibility, we implemented a feature for remapping attribute names when an attribute name is not found in the header. Users are prompted with a pop-up input dialogue if an attribute name cannot be found, to input an alternative name or axis index. If the window is closed without an input, the highest valid axis index is used as a fallback.

3.2 Parallel Coordinates View

To make useful modifications to the parallel coordinates view in MemAxes, we must first establish its purpose in the context of MemAxes. The key role an interactive parallel coordinates view can serve is as an idiom for supporting the user in searching for correlation in the visualized data. in MemAxes, the parallel coordinates view was implemented to enable search for correlations in all the sample attributes. As this is the only view of MemAxes, where information from the hardware, program and application levels can be visualized, it is critical for the fulfilment of the design goals of MemAxes [3]. In this work, we identify issues with the legibility of the parallel coordinates plot, and present our solutions to these issues, as well as some usability improvements.

3.2.1 Identification of Issues

We identified four areas where improvements would be possible.

- In parallel coordinates visualizations, overplotting is an undesirable effect where so many lines are drawn, that it is not possible to identify the individual con-

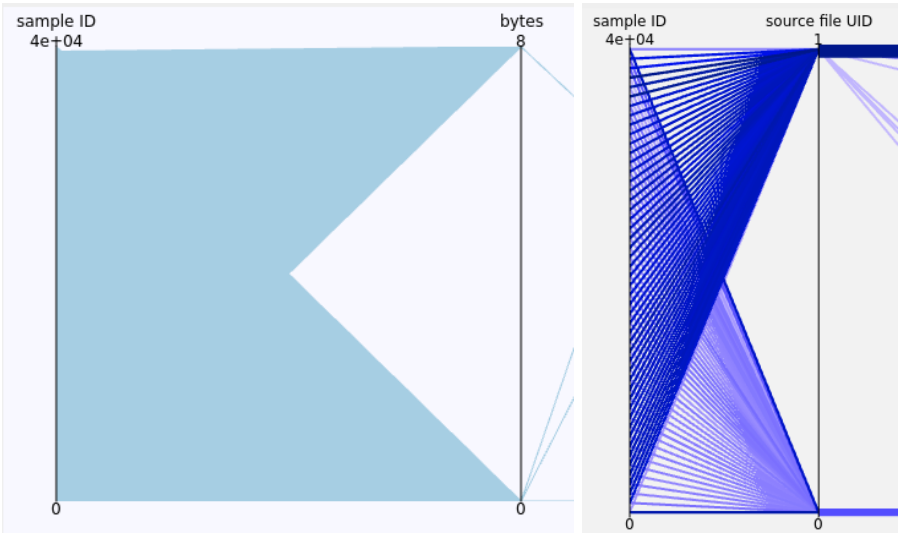
nections. When this effect occurs, correlations are obscured and the parallel coordinates view fails to achieve their purpose [19]. In MemAxes there are thousands or tens of thousands of lines to be drawn inside a box only a few hundred pixels in width and height, overlap of lines will always occur. We demonstrate this effect in figure 3.1.

- The parallel coordinates fail to visualize the correlations of flags. Flags are attributes that have values of either 0 or 1. In the IBS performance samples, flags are common. Conventional parallel coordinates like those implemented in MemAxes visualize the distribution of values on an axis as the density of lines. For discrete attributes with very few distinct values, this approach fails because only a few distinct lines are drawn. The histograms drawn on the axes in MemAxes allow judgements of the distribution of values on the axis, but do not properly visualize the strength of correlations between values.
- The parallel coordinates view displays some data from the domains of the code view and hardware topology, like the code line and the CPU Core number. Despite those semantic connections, it is not possible to easily visualize the connections of objects in the other views to value ranges in the parallel coordinates. To find out what value range an object relates to, the user has to select the object. This becomes inconvenient for multiple objects because selections are necessary that are immediately undone afterwards.

3.2.2 Redesigning the Parallel Coordinates to Improve Legibility

We identified the issues of overplotting hiding correlations, and of the correlations between flags not being quantifiable. Our goal is to make the strength of correlations visible, even when lines would be drawn above other lines.

For the purpose of performance analysis, it should become immediately clear what the strong correlations in the data are, and which value ranges are connected. Previously, functionality was implemented to reduce the number of lines to be drawn by skipping a randomized subset of lines [11]. While this approach can resolve the overplotting issue of correlations being obscured because no individual lines are visible, it does not give any information about the number of lines drawn on top of each other.



(a) Original implementation with overplotting (b) Our implementation

Figure 3.1: Comparison of original parallel coordinates implementation with overplotting and our implementation.

Many modifications of the parallel coordinates visualization idiom exist, that visualize the connection density. One approach discretizes the axes, and only draws connections for the discretized value ranges. The color of the lines is used to visualize the correlations strength of two value ranges [28]. Because we already have discretized the axis when we calculated the histogram bins, we could also use this discretization for calculating the correlation strengths between histogram bins.

For visualizing the correlation strength relative to other correlations between the same axes, we use the color luminance channel. Before drawing the lines, we sort them so that the strongest correlations are drawn last, on top of the weaker correlations.

We use the thickness of line to express the absolute number of elements that are part of a correlation. This feature allows users to differentiate between correlations that most samples are part of, and correlations that are strong in the context of the current selection, but only a small minority of samples are part of.

For every pair of neighboring axes, we calculate a correlation matrix for the histogram bins. Previously, there was no information stored about which bin a sample belongs to. When recalculating the number of samples per histogram bin, the categorization of samples into history bins was performed again. A sample cannot change the bin it belongs to, unless the number of bins, or the minimum and maximum values on the axis change. For this reason, we implemented a matrix that stores which bin every

attribute of every sample belongs to. This matrix does only need to be recalculated when the number of bins, or the minimums and maximums of axes change. This matrix makes calculating the histograms and correlation matrices much easier. Only a row of the bin matrix needs to be iterated, and for every sample to bin mapping, the corresponding count of samples per bin, or correlation strength needs to be increased if the sample is part of the set of selected samples.

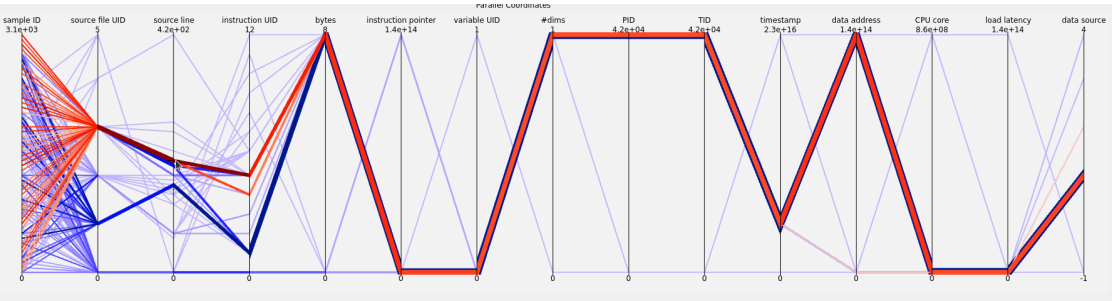


Figure 3.2: Pre-Selection in MemAxes. The cursor hovers over the source line axis. The samples associated with the bin the cursor hovers over are highlighted.

Previously, to visualize the correlations of a value range, it was necessary to perform a selection. In this work, we introduce pre-selection as a more convenient way of exploring the correlations of a single histogram bin. Instead of having to perform a selection, to highlight the samples with a certain value, one can hover the cursor over a value on an axis in the parallel coordinates view. The samples with the pre-selected value are colored red to clearly stand out against the non-pre-selected samples that are colored in blue. For a better pop-out effect of the pre-selected sample set, the opacity of the non-pre-selected samples can be reduced using a slider. Computationally, pre-selection uses a second set of correlation matrices.

3.2.3 Adding and Removing Axes

MemAxes was originally designed to display all available attributes. For the 77 attributes of the IBS example data set, showing all axes all the time would not be a viable solution because of limited screen space. Because the parallel coordinates should be adaptable to new custom sample attributes or other sampling mechanisms than PEBS and IBS, we need to implement functionality that allows users to choose which axes are shown on screen.

To enable adding and removing axes from the view, we needed to change the data structure used to control the parallel coordinates view. Previously, information about the axes was stored in vectors that held histogram data, the minimum and maximum

values, and the position of the axis. The index used to access the values of an axis equals the index of the data visualized on that axis. This solution does not work for when axes can be added or removed. For that reason, we introduced a new vector that maps an axis index to a data index.

We implemented a drop-down menu for choosing axes to add to the parallel coordinates view. After selecting an axis, users can press buttons to add or remove the selected axis from the parallel coordinates view. To avoid showing axes without information to the user, we implemented filtering of axes so that axes where only the first bin contains values, are not displayed by default.

3.2.4 Improved Axis Movement

The original MemAxes implemented movement of axes without a perceivable underlying grid. While this mode of axis movement has its defenders [19], we changed it because we believe it is inferior for the purposes of MemAxes to a system where axes are automatically arranged with equal distances. While a solution that does not snap axes to an underlying grid allows the user to make more space for a correlation they want to inspect, it also does not aid the user in clearing the space from other axes.

We implemented a new axis movement system where axes behave like tabs in a web browser. The axis that is currently moved, can be moved freely without being constrained to a grid, but all other axes automatically move to positions on the grid to make room for the moving axis. This solution still allows users to temporarily increase the space used to display a correlation but makes moving axes more convenient because users do not need to actively move axes out of the way.

3.2.5 Usability Improvements

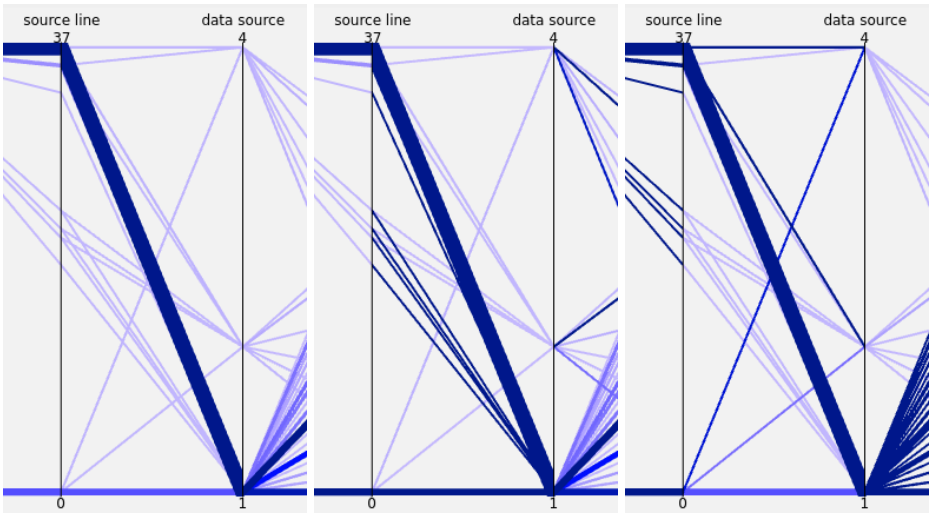
In the parallel coordinates view, it can be hard to retrieve the information, what object a value range on an axis maps to. This is especially a problem with the `Instruction UID` axis, on which the unique identifier of instructions is displayed. Previously, to find out which instruction an instruction identifier maps to, one would have to look up the index in the list of unique instructions in the sample set in order of appearance. What type of instruction was sampled most often is useful information for performance analysis, so this information should be easier to retrieve.

We implemented a cursor tooltip that shows when the cursor hovers over the `Instruction UID` axis. The tooltip displays the instruction type the instruction UID maps to.

For some applications, finer or more coarse histogram bins may be useful. More coarse histogram bins are better for performance, and appear smoothed, while finer

histogram bins might be necessary to avoid aliasing of discrete values into the same bin. We implemented a slider that allows users to control the number of histogram bins in the range of 50 to 1000. With less than 50 bins, significant aliasing, for example, of instruction UIDs would occur. More than 1000 histogram bins are not desirable because of the limited resolution of screens. Assuming a height of the parallel histograms of 500 pixels (about half of the vertical resolution of a common full HD screen), 1000 histogram bins cannot be represented with one line of pixels per histogram bin. Even when maximizing the parallel coordinates view to take all available screen space, more than 1000 histogram bins cannot be displayed on most screens.

To fit different analysis needs, we implemented multiple color modes for the connecting lines in the parallel coordinates view. By default, all lines are assigned a color based on the represented correlation strength relative to all other correlation strengths between the same axis pair. Pre-selected and all other samples share the same minimum and maximums for visualization by default so that users instantly get a sense for the relevance of the pre-selected sample set in the context of all samples. Users can uncheck a box to enable separate coloring ranges for pre-selected and other samples.



(a) All bins (b) First axis (c) Second axis

Figure 3.3: Parallel coordinates coloring modes.

We implemented additional coloring modes that color correlation lines in other ways than the default, based on the correlation strength relative to the minimum and maximum strengths of correlations between the axis pair. With first axis coloring, every line is colored based on the correlation strength relative to the bin on the left axis.

Second axis coloring does the same for the right axis. In figure 3.3 we can see the effects of the different modes. In the default all bins coloring, lines are colored based on their correlation strength relative to the minimum and maximum strength of all correlations between the source line and data source axes. Only the line between source line 37 and data source 1 is dark blue. We cannot make a statement about the correlations strengths of other lines from their light blue color.

If for example, we wanted to find out, if there is a source line, that has its strongest correlation with a different data source than 1 (the L1 cache), without the different coloring modes, we would have to preselect all source line bins to see the correlation strengths relative to only that bin. With first axis coloring, we can make the strongest correlation of every bin on the left side stand out. We can easily tell from the second image, that all dark blue lines are connected to 1 on the data source axis. With this tool we can quickly answer our initial question, all lines have their strongest correlation with data source 1.

3.2.6 Performance Analysis of Parallel Coordinates View

Our new implementation, despite the additional cost of calculating correlation matrices, performs equally good or better than the original implementation. It also scales better for high numbers of samples. The additional computations are offset by the reduced number of lines that must be drawn.

We performed all performance measurements on a desktop computer running Ubuntu 23.04 on an AMD R9 3900X 12 core CPU with a maximum operating frequency of around 4.3GHz. Graphics were rendered using a NVIDIA RTX 2070 Super graphics card. All tests were performed in full-screen mode at 1080p resolution. By default, we used 100 histogram bins.

We instrumented MemAxes to report the time needed to render a frame. Our test consisted of moving the CPU core axis to the left edge of the parallel coordinates view, then to the right edge. We recorded fewer frames from the original implementation because MemAxes rendered fewer frames in the time we performed our inputs. We did not filter out frames where the parallel coordinates connecting lines were not recalculated. For this reason, both sets of frame times contain times below one millisecond.

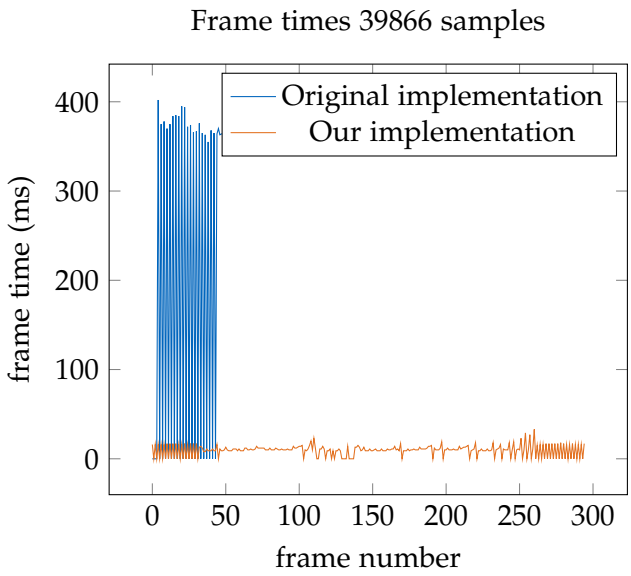


Figure 3.4: Frame times when moving an axis in the parallel coordinates view.

Axis movement in the original implementation feels noticeably detached from the user input because of the high frame times up to 402ms. Still, the original implementation is perfectly usable, at least on the powerful desktop test system. Our implementation achieves much lower frame times consistently.

We also used MemAxes on a less powerful laptop without a dedicated graphics card in a Windows Subsystem for Linux (WSL) virtual machine. In that scenario, the frame times of the original implementation were about two seconds, mainly because rendering of 677722 lines per frame is beyond the processor's capabilities. Our implementation, which draws only about 2000 lines per frame when used with the default number of histogram bins, consistently achieved frame times below 100ms.

We performed a similar test with a sample set consisting of nearly ten times as many samples. In the second test, our implementation still achieved interactive frame rates, while the original implementation displayed new frames only once every two seconds.

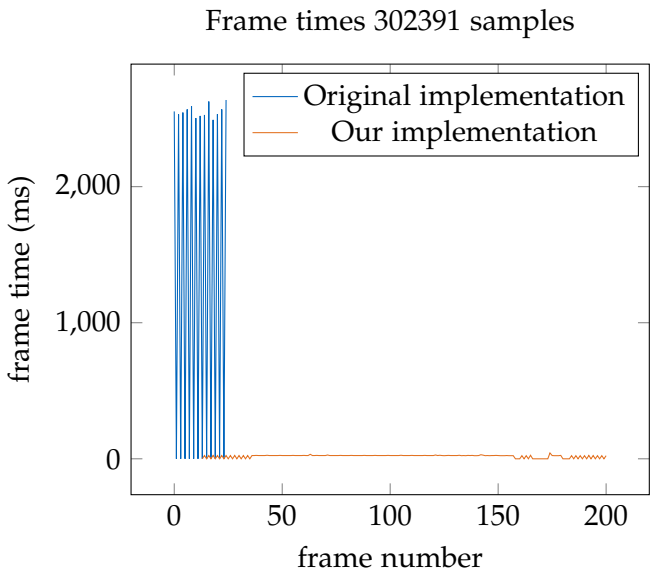


Figure 3.5: Frame times when moving an axis in the parallel coordinates view.

We will further investigate the performance of MemAxes for different numbers of samples.

In the stacked bar graph visualizing the parts of frame time spent on different calculations, we can clearly see that creating the line geometry causes the biggest fraction of frame times in the original implementation. This is mainly because of comparatively slow memory accesses in a vector of structs.

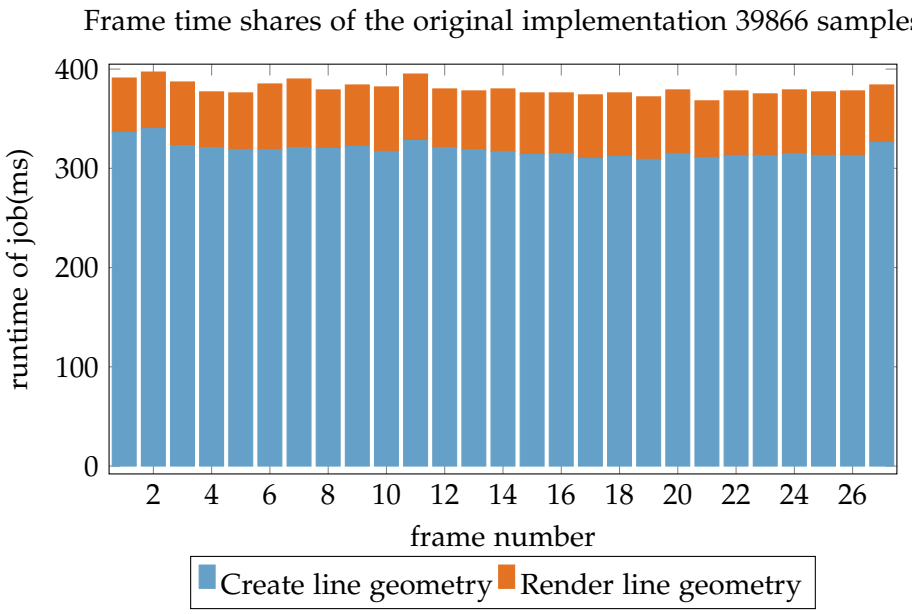


Figure 3.6: Shares of time to calculate a frame of parallel coordinates visualization in the original implementation.

The computational cost of the original implementation scales in $O(n)$ with the number of samples. Because our iterations must also iterate all samples, it has the same scaling behavior for high numbers of samples, but our code runs much faster. The reason for this is that the code does not access values from the vector of structs, but instead iterates lines of the bin matrix. While creating the geometry in the original implementation took about 300ms, our implementation achieves times below 10ms.

We do not achieve the steady frame times of the original implementation because the cost of calculating the correlation matrices depends on the axis order. Denser correlation matrices take longer to compute because of caching. Pre-selection introduces additional varying cost of logic determining if the currently processed sample is part of the pre-selection. We clearly see in the frame time shares plot 3.7 that the time spent on creating the correlation matrix displays the strongest variance.

Creating a list of vertices from the correlation matrix requires iterating the elements of the correlation matrix twice. One time for finding the minimum and the maximum value, and another time to generate a list of lines that have to be drawn. The size of the correlation matrix is the square of the number of histogram bins. For this reason, frame times of our implementation depend not only on the number of samples, but also on the number of histogram bins. In practice, it is often possible to skip most rows of the correlation matrix because the bin the row represents contains no elements.

We can see in the plot of frame time shares of our implementation, that the cost of our newly introduced operations is comparatively low.

We increased the number of points that need to be pushed to the vertices vector to draw a line threefold because we no longer use line primitives, but pairs of triangles. Unlike lines, for which a thickness can only be set for all lines, triangles allow for lines of varying thickness. Because our approach to accumulated parallel coordinates uses much fewer lines, the total number of vertices is still much lower than in the previous implementation. We can see in the frame time shares plot, that generating the geometry vector takes a comparatively small fraction of frame time.

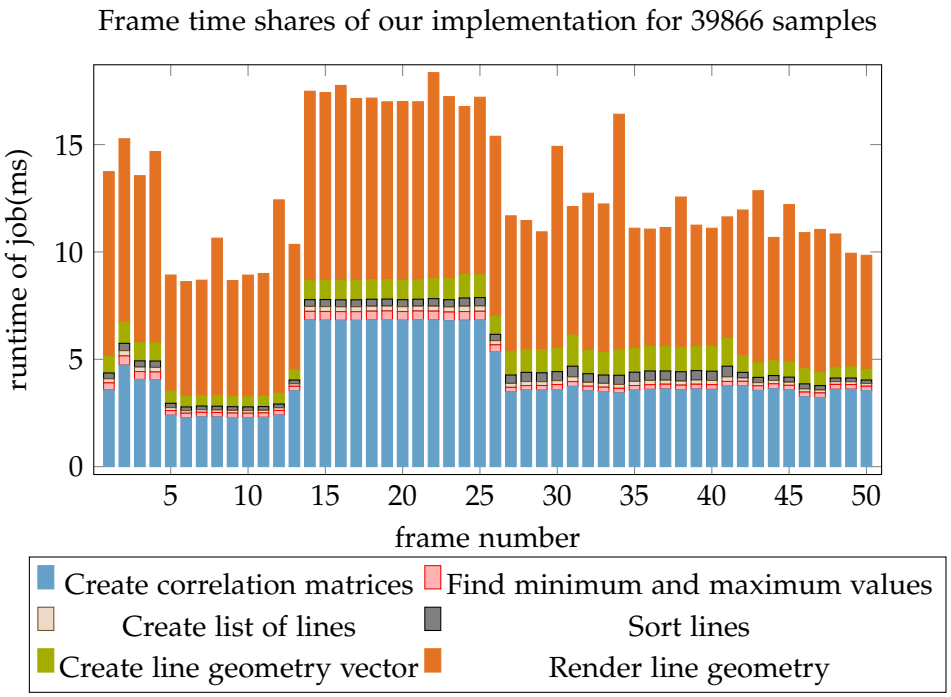


Figure 3.7: Steps of our implementation of the parallel coordinates visualization and their shares of time in displaying a frame. IBS example dataset.

We also performed performance analysis of our implementation for a larger sample set. In plot 3.8 we can clearly observe that only the time spent on computing the correlation matrix increases significantly, about four times for a sample set nearly ten times as large. The time spent processing the lines did not increase significantly because the number of histogram bins did not increase. The number of lines that have to be rendered also did not increase.

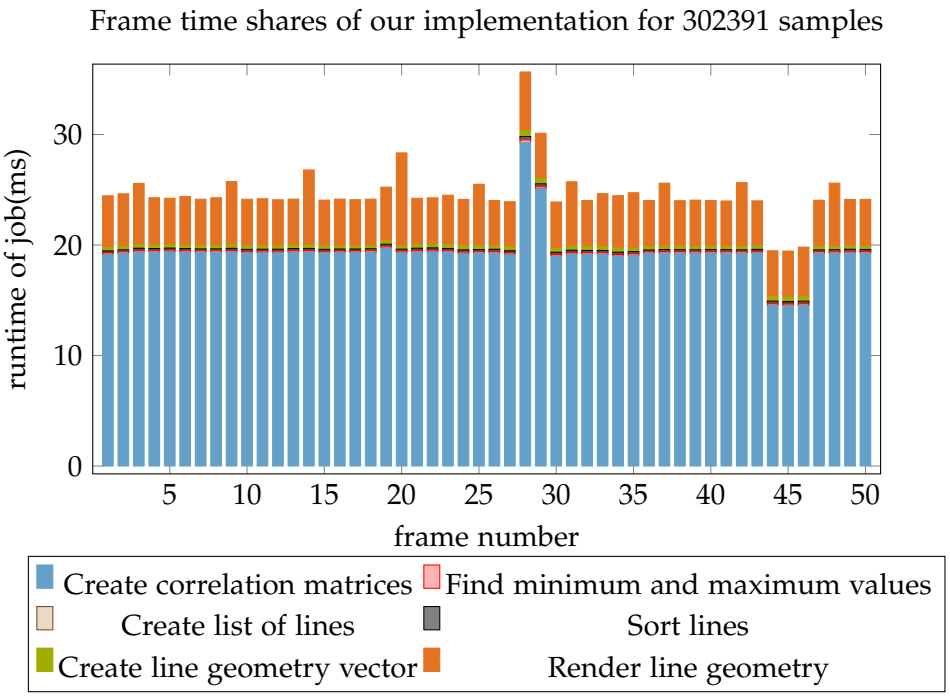


Figure 3.8: Steps of our implementation of the parallel coordinates visualization and their shares of time in displaying a frame. High number of samples.

We know from our analysis of contributors to computational cost, that the size of the correlation matrix is the square of the number of histogram bins. The size of that matrix is important because the matrix must be iterated twice. To test the behavior of our implementation for larger numbers of bins, we recorded frame times for a large number of samples and the maximum number of histogram bins, 1000. In the plot 3.9 we see increases in the areas of calculating the minimum and maximum value, creating the geometry vector and rendering the lines. Unlike the part creating the list of lines, the minimum and maximum calculation is not optimized to skip lines in the correlation matrix that only contain zero values. For this reason we see a strong increase in the frame time fraction of the minimum and maximum calculation, but no such increase in the part that creates the list of lines. The increase in time spent on rendering the geometry is due to the higher number of lines that have to be drawn.

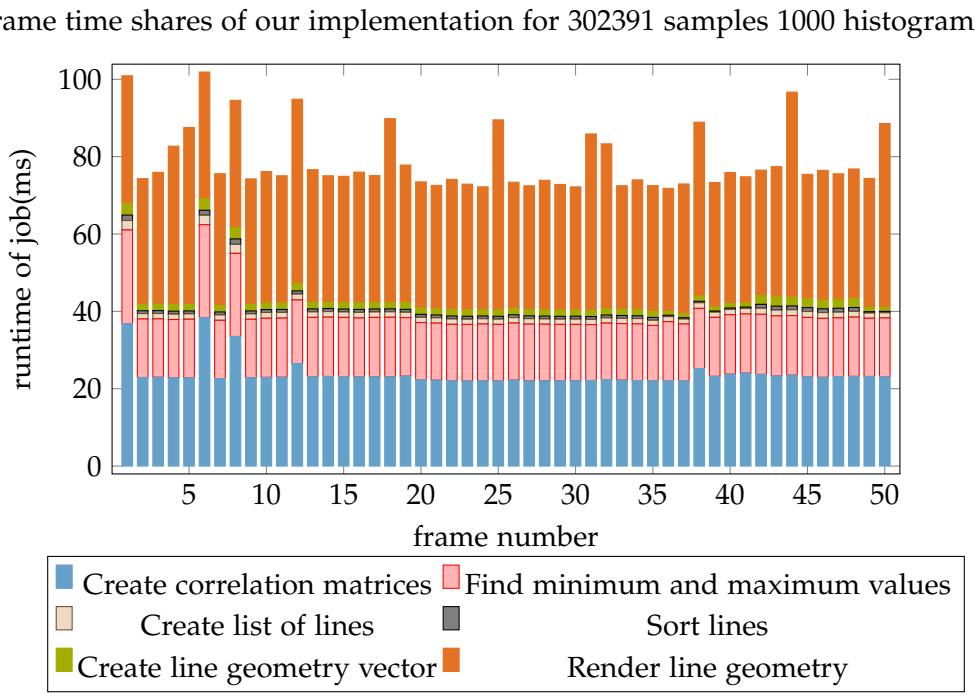


Figure 3.9: Steps of our implementation of the parallel coordinates visualization and their shares of time in displaying a frame. High number of samples and high number of histogram bins.

3.3 Code View

For this work, we adapted the code view to properly display IBS samples in the top offenders plot. We identify usability issues with the original implementation and describe changes made to the code view to address those issues and integrate the code view with the pre-selection system.

3.3.1 Adapting the Top Offender Plot to IBS Samples

MemAxes assumes that the `Latency` attribute of samples contains the total memory access latency for the sampled operation. In AMD IBS samples, the attribute `IbsDcMissLat` contains information about the latency of memory accesses, but only of accesses that missed in the L1 cache. The latency is measured in cycles starting at the cycle the cache miss was detected. Operations that hit in the L1 cache have `IbsDcMissLat` of 0. This warps the top offender ranking, for example in the matrix

multiplication IBS sample set, the line that performs most computations is not first in the top offenders ranking because the L1 hits are assumed to be truly instantaneous, although in reality they also have a latency. This version of the top offenders ranking can mislead users to believe that code that is executed fewer times but misses the L1 cache, is more expensive overall than code that is executed many more times, but does not miss the L1 cache. This misconception can lead to underperforming optimizations because optimization of a section cannot reduce total runtime by more than the runtime of the unoptimized code section.

To not make this unrealistic assumption of free L1 cache accesses, we implemented an L1 latency estimate that is added to all `IbsDcMissLat` values to get the final `Latency` values. Because the latency of L1 caches varies between processors of different generations, and for example for AMD Zen 4 processors, varies between integer and floating-point units [26], we let the user set the L1 latency estimate. On loading an IBS data set, the user is prompted to input an estimate for the latency of L1 cache access.

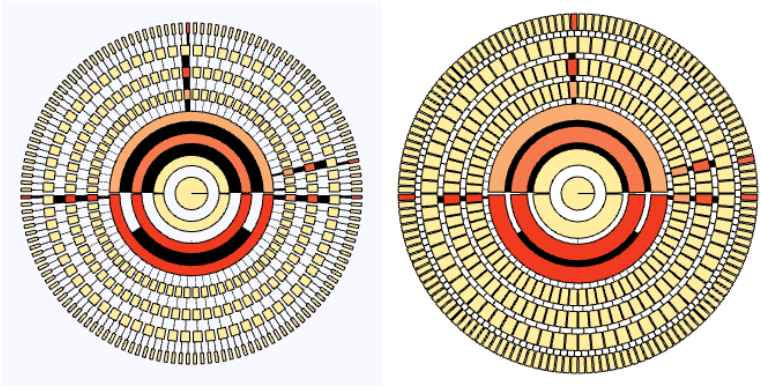
3.3.2 Integration into the Pre-Selection System

To inspect the contents of a line of code in the code editor, it was necessary to select the code block in the top offenders view. Forcing users to select a code block to inspect the code is inconvenient because users will have to undo such a selection afterwards. We implemented a new feature to jump to the code line in the code editor when the cursor hovers over a code block in the top offender view to alleviate the need to make a selection for looking up a single value.

We also integrated cursor hover in the code view into the pre-selection system of the parallel coordinates view. This allows users to preview the effects of selecting a code block before committing to the selection. Previously, to find out what part of the code a value rang in the parallel coordinates has the strongest correlation with, a selection was necessary. We extended pre-selection to make looking up the code with the strongest correlation to a value in the parallel coordinates faster. The code view jumps to the line that is most strongly correlated with the histogram bin the cursor hovers over. Because the code view jumping to a different line can be distracting when working with the parallel coordinates, this behavior can be deactivated by clicking a checkbox.

3.4 Hardware Topology View

The hardware topology view received two main updates, support for the pre-selection system and highlighting of resources without samples. We also fixed the scaling behavior of objects in hardware topology plots with many resources. Instead of enforcing a minimum distance between objects on the same level that might result in



(a) Original scaling (b) Our scaling rules

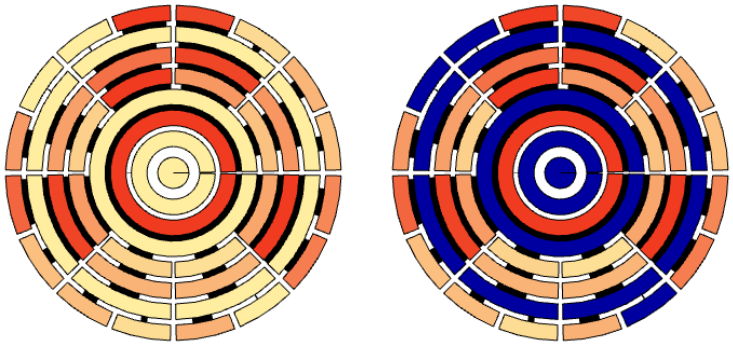
Figure 3.10: Hardware topology visualization scaling for high number of compute resources.

objects being reduced to thin lines, we allow for less distance between objects if the objects would be too small. The new scaling is preferable because the old scaling failed at providing space that can be colored to communicate the utilization of the resource. Our solution ensures that every object has as much area as possible at a given scale.

3.4.1 Pre-Selection

We implemented a selection preview for selecting samples associated with a resource in the hardware topology view. When the cursor hovers over a resource, the samples associated with that resource are pre-selected in the parallel coordinates view. Implementing this feature was harder than pre-selection for the code view because attribution to a hardware resource is not as easy to check for a given sample as source line and source file ID. The hardware topology view itself uses a list of pointers to samples to store which samples are associated with a hardware resource.

Because a list of sample struct pointers does not allow for efficient categorization whether a sample is part of the set of samples, we implemented a second data structure for every hardware resource. A list stores the sample UIDs in ascending order. The parallel coordinates correlation matrix computation iterates the samples in the order of ascending unique identifiers. For every UID between 0 and the number of samples, a sample exists. This allows MemAxes to instead of searching for every UID in the list, to keep a pointer on the next element in the list of sample UIDs related to the resource that could match the sample UID in the iteration. When the two UIDs match, the sample with that UID is part of the pre-selected set and the pointer in the smaller



(a) No highlighting (b) Highlighting

Figure 3.11: Highlighting of resources without associated samples.

set is increased.

3.4.2 Highlighting Resources without Samples

It might be useful to easily distinguish between resources with few samples and without samples. For this reason, we implemented functionality to display all resources without associated samples in dark blue. The dark blue color clearly sets empty resources apart from those with samples, which have colors from a yellow to red palette.

3.5 Command Search

As a tool that allows exploring the correlations between tens of attributes of thousands of samples, MemAxes confronts the user with an enormous search space. Especially the parallel coordinates give little context to the data. Currently, IBS-specific data like TLB misses or mispredicted branches can only be visualized as axes in the parallel coordinates. This approach requires that users know which axes they have to add to the parallel coordinates view, and what the semantics of the values and the axes are.

We want to introduce a new interaction idiom that can assist users with controlling the visualization in MemAxes. Users should be able to perform management of axes in the parallel coordinates view and selections based on higher level input referencing not names and value ranges, but semantics. Such a tool would also be a good starting point for new or infrequent users, reducing the burden of having to learn the semantics of the axes in the parallel coordinates view.

We first evaluated a design where users are presented recommended actions, which can be performed with a single press of a button. The results were not satisfying. Users could be walked through a basic analysis cycle, but with this system, only the developer would be dictating to the user, what might interest them. Of course, such a system is not easily extensible and cannot handle data it was not designed to handle. The system also did not allow the user to express what they are actually looking for. To allow for more expressive user input than choosing a recommended command from a menu, we started to implement a text-based search function for commands.

MemAxes already has a console that can be used for selecting samples through queries. We do not aim to extend this query functionality, which is not based on semantics, but purely on axis names and value ranges. The new search functionality should also be able to handle input based on semantics. The console can only process input conforming to its query language. The new search feature should behave more like a search engine that recommends results that might be of interest to the user, and not require input to strictly adhere to some format. Unlike the console, our new interaction idiom should also be useful for rearranging the parallel coordinates view to investigate the correlations of two axes. We want to put the command search box front and center in the UI design, so we placed it on top of the page in the middle of the screen. This placement for a search bar should be familiar to many users from web browsers.

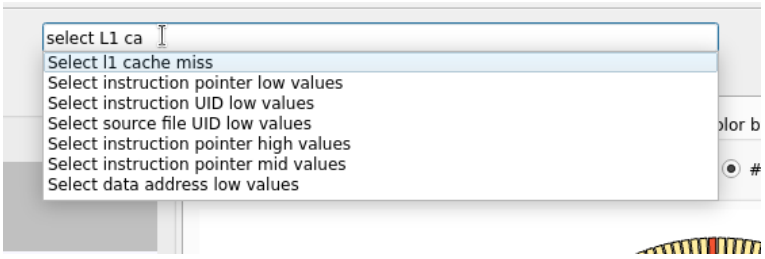


Figure 3.12: User interface of command search.

The main challenge of designing the new semantics-based search was defining a specification that allows for free form input while providing the outputs that users expect. To keep the scope of this work reasonable, we do not want to go into the topic of Natural Language Processing (NLP) but interpret user input based on simple rules.

3.5.1 Specification Design

We want to enable three types of interactions through command search.

- **Show and hide axes**. This is the most important usability feature of command search because it removes the need to navigate the axis names drop-down menu. The speedup in this scenario is enormous because the dropdown menu contains up to 72 axes, while command search recommends fewer axes that match the user input. Because it is not required that user input matches the axis name perfectly, we also alleviated the need for users to know exactly the name of an axis.
- **Select** is the action that profits the most from an interface that utilizes semantics of user input. With command search, users should not need to know which axis they have to bring to the parallel coordinates view, and what value range they have to select to observe some phenomenon. With command search, they should be able to search for the name of the phenomenon. Alternatively, users should be able to select values on axes through command search by typing minimum and maximum values.
- To **Correlate** two phenomena without command search requires the user to know the names of the axes they want to investigate the relations of, potentially add them to the parallel coordinates view, and move the axes next to each other by drag and drop. Command search should allow to correlate two axes or phenomena with a single command. Optionally, users should also be able to specify value ranges to be selected.

The parsing of user input should give a result for every input, but to keep the complexity of parsing manageable, we make some assumptions about the user input.

- Only one command at a time. Users are not supposed to issue multiple commands through one input.
- Verb first. If user input features the name of a command, it is either the first word, or will be ignored.
- Words can contain numbers, but not only numbers.
- Names of Commands, axes or phenomena only consist of words.
- Clear separation. The two axis or phenomena names used in the correlation actions are separated by numbers for value range selection, or a "with" keyword.

3.5.2 Software Design

Our goal was to design the command search software to be as extensible as possible. It should support manipulating any axis of the parallel coordinates plot, even those

for custom attributes. Defining a new group for semantic search should require only minimal changes to the code of MemAxes.

We created two abstractions for the command search system. `Action` and `Group`.

Actions are abstractions for commands that can be executed. We created subclasses of `Action` for selection, show, hide and correlate actions.

Groups define a function to select the members of the group. Groups can be defined by an axis and a value range, or a custom type of `Group` can be created with custom functionality, like the `AllGroup` undoing all selections.

To manage the search box, we implemented an `ActionManager` Qt widget. When the user inputs text into the search box, actions that fit the user input are created. A dropdown recommendations list, like in web search engines, contains the text descriptions of the actions.

We split parsing the user input into two subtasks

- Determining the best match for the meaning of every block. For this we need a string distance function that accounts for how the user is most likely to complete their input.
- Splitting the input into blocks of meaning. For example, the name of an axis and a command word, should be in separate blocks. We perform splitting based on our previously defined assumptions about the user input.

3.5.3 Interpreting Incomplete Blocks of User Input

As an abstraction for strings that should be matched against user input, we introduce `Phrases`. We implemented two algorithms for determining the distance between user input and a `Phase`, Longest Common Subsequence (LCS) [29] and Levenshtein Distance [30].

Both algorithms calculate so-called edit distances, the number of edits a user would have to make to transform the input into the phrase. LCS only accounts for insertions and deletions, Levenshtein distance also allows substituting characters [31]. We normalize the edit distance by division by the length of the phrase, similar to [32]. Without normalization, shorter phrases would have lower edit distances than longer ones, even if no character of the user input matches the shorter phrase. Because it is unlikely, if no character matches the short phrase, that the user intends to type the phrase, we normalize to assign short phrases comparatively higher distances in such scenarios.

We chose Levenshtein distance over LCS because it handles typos better. Levenshtein distance has a higher tolerance for typos because substituting characters is treated as a single operation of cost one, not two operations as in LCS. One can also define custom distances of character substitutions in Levenshtein distance, for example, a

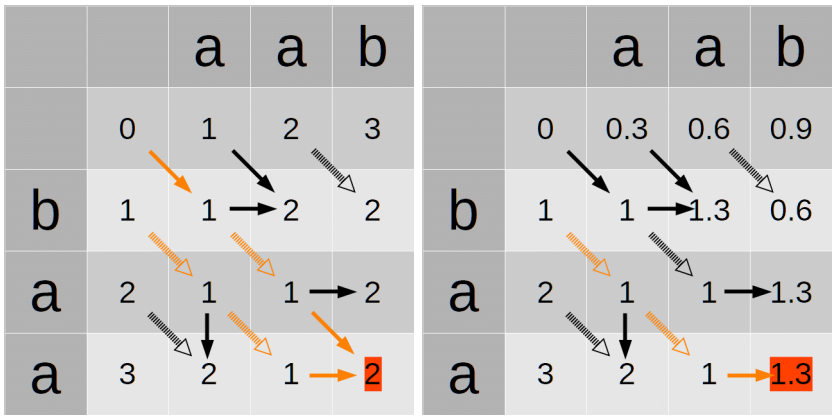
lower distance for letters located next to each other on a standard keyboard [31]. Our implementation implicitly defines a zero distance between upper- and lower-case letters, as it is not case sensitive. We also defined an equality of blanks and underscores because users are unlikely to use underscores to separate words, but the IBS example data set uses underscores to separate words in the names of IBS-specific attributes.

The recursive definition of Levenshtein distance lev of strings a and b is

$$lev(a,b) = \begin{cases} |a| & |b| = 0 \\ |b| & |a| = 0 \\ lev(tail(a), tail(b)) & a[0] = b[0] \\ 1 + min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} \end{cases} \tag{3.1}$$

where *tail* is the function returning all of a string except the first character [30]. To better predict user intent, we implemented different costs of actions. Insertions have a low cost (0.3) because users are assumed to be more likely to input additional characters than to delete their input. For example, if the input was "abcd", and recognized phrases are "abc" and "abcdefg", the text that is a possible completion of the input should have a lower distance than the text that shares a prefix with the input, but also has characters that do not match the input.

We implemented the Wagner-Fischer matrix algorithm for Levenshtein distance. In the Wagner-Fischer matrix, every element corresponds to the distance of prefixes of the two strings. An element at W_{ij} can be calculated from its left, upper left, and upper neighbor. If the characters at the indices i and j are equal, the value of W_{ij} is $W_{i-1,j-1}$. Otherwise, it is the minimum of $W_{i-1,j-1} + substitution\ cost$, $W_{i-1,j} + deletion\ cost$ and $W_{i,j-1} + insertion\ cost$[33].



(a) Standard cost (b) Custom cost of insertion

Figure 3.13: Wagner-Fischer matrix example. The string "baa" is transformed into "aab". The result is in the lower right corner. Arrows mark which data was used to calculate the value of a cell. Highlighted arrows mark the path that leads to the result.

3.5.4 Splitting the Input into Blocks

We first split the user input into words. We combine words into blocks based on our previous assumptions about the user input. Because we require that the first word be a verb, or otherwise, there is no verb, we check whether the first word strongly matches an action type phrase. A new block starts after numbers, which are assumed to be minimum and maximum value range quantifiers. The word "with" cannot be part of a block because it indicates the start of a new block.

When the user defines the type of action, MemAxes constructs the actions of that type for the groups with the names that match the remaining user input best. If no block is part of the user input, MemAxes uses the index of the axis in the dataset to order recommendations. For correlate actions, if the user input only contains one block, MemAxes recommends correlating the axis with the name that matches the block best, with the source line axis, the CPU Core axis and the Data Source axis. Our assumption is, that users are more likely interested in how a phenomenon can be attributed to the code, if its distribution across processors was balanced, and which levels of the memory hierarchy were used in the memory accesses related to the phenomenon.

If no action type was defined by the user, but a block is given, we interpret a single block as the name of a group. For the groups that match the input best, we perform context-aware recommendation of actions. If the group refers only to an axis without

selection limits, we either recommend showing or hiding the axis depending on whether it is visible in the parallel coordinates view. In case the group refers to a selectable set of samples, we recommend a select action. If the input is split into more than one block, we recommend actions correlating the groups the block most likely refer to.

3.5.5 Evaluation

We evaluate the performance of our input processing through some example scenarios. In general, we want to minimize the number of characters a user should have to input. Ideally, the command the user intends to execute should be the first recommended command. Our system should be able to handle some errors in the input.

The implemented variation of Levenshtein string distance generally performs well when confronted with abbreviated names. Because of the low cost of insertions and the normalization, when typing a common prefix, followed by a distinct letter, the intended action is suggested.

There is one main class of problems where our implementation does not perform well. Reordered names of groups, like "branch mispredicted" instead of "mispredicted branches", are not recognized as similar to the group name. This is because Levenshtein distance does not give a lower cost to strings that are reordered versions of one another.

This is mainly a problem for the automatically generated group names, making it harder to perform an action based on parts of a name that might be in the wrong order. For custom groups, like L1 cache misses, this is less of a problem because aliases can easily be introduced in our system. Still, this is not an ideal solution, as it increases the amount of work necessary to implement a custom group.

Intended command	**Input**	**First Appearance**	**Highest position**
select L1 cache miss	L1 misses	2	2
	L1 cache miss	2	1
	miss L1	-	-
hide timestamp	timestamp	1	1
	hide timestamp	6	1
correlate L1 TLB miss with source line	correlate L1 TLB miss with source line	12	1
	L1 TLB with source line	15	1
correlate instruction UID with data source	correlate instruction UID with data source	14	1
	instruction UID with data source	23	1
correlate L2 TLB miss with load latency	correlate L2 TLB miss with load latency	28	1
	L2 TLB with latency	14	1
select mispredicted branches	brn misp	-	-
	select brn misp	-	-
	misp brn	4	1

Table 3.1: Performance of command search in example scenarios. We tested, if command search would present the intended command for different user inputs. Performance metrics are the number of characters the user has to type before the intended action first appears in the recommendations (lower is better), and the highest position of the intended command in the recommendations ranking (lower is better).

4 Conclusion and future work

In this work, we presented new features we implemented in the performance visualization tool MemAxes. Our main contribution was implementing compatibility with samples from AMD processors, but in adapting the visualization to work with the newly available data, we also overhauled the user interface of MemAxes. Lastly, we introduced a new interaction scheme that makes low-level data accessible through high-level requests.

To make the core functionality of MemAxes compatible with AMD IBS samples, we had to implement decoding of the cache level used from IBS-specific sample attributes. We also introduced user-defined remapping of attribute names to ease the strict requirements on the input format.

For visualization of the data specific to IBS, we focused mainly on the parallel coordinates view. We improved the legibility of the view by communicating the strength of correlations through color. Because this new approach drastically reduces the number of lines that must be drawn, we were also able to improve performance. The pre-selection feature makes it easier for users to quickly browse the dataset.

Pre-selection was implemented as a global system for exploring the correlations between objects from different views. While previously many selections might have been necessary to answer the question "which objects have samples correlated with this attribute value?", with pre-selection users can easily browse the correlations of different objects without performing a single selection.

We introduced command search as a tool for users to interact with MemAxes on higher-level terms. Command search allows users to select sets of samples through text input describing phenomena, not just value ranges. Users can also rearrange the parallel coordinates view using command search to inspect the correlations of phenomena.

In the areas where we improved MemAxes, there is potential for future work.

- MemAxes could be made compatible with performance data from more types of hardware. Most of the compute performance of a modern HPC node is provided by accelerators, often General Purpose Graphics Processing Units (GPGPU). For analysis of on-node performance, integrating accelerator performance into the visualization could be useful. GPGPUs do not provide performance samples like

processors, so integrating accelerator performance in the MemAxes visualization poses additional challenges.

- Command search could be extended to give context-aware recommendations. Currently command search processes only the user input and state of the parallel coordinates. A more advanced command search tool could give recommendations, which characteristics might be worth investigating based on automatic analysis of the sample set. A step in this direction, although without the text-based interaction, was also proposed in a 2016 paper on MemAxes [3].

We hope that the new features of MemAxes presented in this work make the tool more useful and provide a better basis for future improvements.

List of Figures

List of Tables

Bibliography

[1] M. B. Thomas Sterling Matthew Anderson, "High performance computing: Modern systems and practices," in Morgan Kaufmann, 2018, ch. 1.1 High Performance Computing Disciplines.

[2] R. Spence, *Information Viszualization*, Third. Springer, 2014.

[3] A. Giménez, T. Gamblin, I. Jusufi, A. Bhatele, M. Schulz, P.-T. Bremer, and B. Hamann, "Memaxes: Visualization and analytics for characterizing complex memory performance behaviors," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 7, pp. 2180–2193, 2017.

[4] A. Giménez, T. Gamblin, B. Rountree, A. Bhatele, I. Jusufi, P.-T. Bremer, and B. Hamann, "Dissecting on-node memory access performance: A semantic approach," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 2014, pp. 166–176.

[5] A. Giménez, B. Husain, D. Böhme, T. Gamblin, and M. Schulz, *Mitos: A simple interface for complex hardware sampling and attribution*, 2015.

[6] J. L. Hennessy and D. A. Patterson, "Computer architecture: A quantitative approach," in Elsevier, 2011, ch. 1.1 Introduction.

[7] J. L. Hennessy and D. A. Patterson, "Computer architecture: A quantitative approach," in Elsevier, 2011, ch. 3. Instruction-Level Parallelism and Its Exploitation.

[8] J. L. Hennessy and D. A. Patterson, "Computer architecture: A quantitative approach," in Elsevier, 2011, ch. 4. Data-Level Parallelism in Vector, SIMD, and GPU Architectures.

[9] J. L. Hennessy and D. A. Patterson, "Computer architecture: A quantitative approach," in Elsevier, 2011, ch. 5. Thread-Level Parallelism.

[10] M. B. Thomas Sterling Matthew Anderson, "High performance computing: Modern systems and practices," in Morgan Kaufmann, 2018, ch. 2. HPC Architecture 1: Systems and Technologies.

[11] T. G. Alfredo Giménez, *Memaxes github repository*, 2014. [Online]. Available: `https://github.com/LLNL/MemAxes`.

[12] N. E. Simon Pickartz Manolis Marazakis, *Deep-sea*, 2021. [Online]. Available: `https://www.deep-projects.eu/images/DEEP-SEA-deliverables/DEEP-SEA_D31_Software_specification_Published.pdf`.

[13] A. Fog, *The microarchitecture of intel, amd, and via cpus*, 2023. [Online]. Available: `https://www.agner.org/optimize/microarchitecture.pdf`.

[14] J. L. Hennessy and D. A. Patterson, "Computer architecture: A quantitative approach," in Elsevier, 2011, ch. 2. Memory Hierarchy Design.

[15] T. Rauber and G. Rünger, "Parallel programming: For multicore and cluster systems," in Third. Springer science & business media, 2022, ch. 2.9 Caches and Memory Hierarchy.

[16] D. A. Patterson and J. L. Hennessy, "Computer organization and design arm edition: The hardware software interface," in Morgan kaufmann, 2016, ch. 7.3 Shared Memory Multiprocessors.

[17] C. Lameter, "Numa (non-uniform memory access): An overview: Numa becomes more common because memory controllers get close to execution units on microprocessors.," *Queue*, vol. 11, no. 7, pp. 40–51, 2013.

[18] A. Inselberg and B. Dimsdale, "Parallel coordinates: A tool for visualizing multidimensional geometry," in *Proceedings of the first IEEE conference on visualization: visualization90*, IEEE, 1990, pp. 361–378.

[19] J. Heinrich and D. Weiskopf, "State of the art of parallel coordinates.," *Eurographics (state of the art reports)*, pp. 95–116, 2013.

[20] M. A. Sasongko, M. Chabbi, P. H. Kelly, and D. Unat, "Precise event sampling on amd versus intel: Quantitative and qualitative comparison," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1594–1608, 2023.

[21] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[22] J. L. Hennessy and D. A. Patterson, "Computer architecture: A quantitative approach," in Elsevier, 2011, ch. 1.9 Quantitative Principles of Computer Design.

[23] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "State of the art of performance visualization.," *EuroVis (STARs)*, 2014.

[24] P. J. Drongowski, *Instruction-based sampling: A new performance analysis technique for amd family 10h processors*, 2007.

[25] D. A. Patterson and J. L. Hennessy, "Computer organization and design arm edition: The hardware software interface," in Morgan kaufmann, 2016, ch. 5.4 Virtual Memory.

[26] *Software optimization guide for the amd zen4 microarchitecture*, Advanced Micro Devices, 2023.

[27] *Bios and kernel developer's guide (bkdg) for amd family 16h models 30h-3fh processors*, Advanced Micro Devices, 2016.

[28] M. Novotny and H. Hauser, "Outlier-preserving focus+ context visualization in parallel coordinates," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 893–900, 2006.

[29] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM (JACM)*, vol. 24, no. 4, pp. 664–675, 1977.

[30] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, Soviet Union, vol. 10, 1966, pp. 707–710.

[31] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, no. 1, pp. 31–88, 2001.

[32] F. Petroni and M. Serva, "Language distance and tree reconstruction," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 08, P08012, 2008.

[33] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.