# Structural Performance Comparison of Parallel Software Applications

Kurzfassung

**Dissertation**

zur Erlangung des akademischen Grades
Doktor rerum naturalium
(Dr. rer. nat.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

**Matthias Weber, M.Sc.**
geboren am 8. April 1980 in Cottbus

Gutachter:
Prof. Dr. rer. nat. Wolfgang E. Nagel, Technische Universität Dresden
Prof. Dr. techn. habil. Dieter Kranzlmüller, Ludwig-Maximilians-Universität München

Dresden, 30. August 2016

# Abstract

With rising complexity of high performance computing systems and their parallel software, performance analysis and optimization has become essential in the development of efficient applications. The comparison of performance data is a key operation required in performance analysis. An analyst may conduct different types of comparisons in order to understand the performance properties of an application. One use case is comparing performance data from multiple measurements. Typical examples for such comparisons are before/after comparisons when applying optimizations or changing code versions. Besides comparing performance between multiple runs, also comparing performance characteristics across the parallel execution streams of an application is essential to detect performance problems. This is typically useful to detect imbalances, outliers, or changing runtime behavior during the execution of an application. While such comparisons are straightforward for the aggregated data in performance profiles, only limited solutions exist for comparing event traces. Trace-based analysis, i.e., the collection of fine-grained information on individual application events with timestamps and application context, has proven to be a powerful technique. The detailed performance information included in event traces make them very suitable for performance analysis. However, this level of detail also presents a challenge because it implies a large and overwhelming amount of data. Currently, users need to perform manual comparison of event traces, which is extremely challenging and time consuming because of the large volume of detailed data and the need to correctly line up trace events.

To fill the gap of missing solutions for automatic comparison of event traces, this work proposes a set of techniques that automatically align traces. The alignment allows their structural comparison and the highlighting of differences between them. A set of novel metrics provide the user with an objective measure of the differences between traces, both in terms of differences in the event stream and timing differences across events.

An additional important aspect of trace-based analysis is the visualization of performance data in event timelines. This has proven to be a powerful approach for the detection of various types of performance problems. However, visualization of large numbers of event timelines quickly hits the limits of available display resolution. Likewise, identifying performance problems is challenging in the large amount of visualized performance data. To alleviate these problems this work proposes two new approaches for event timeline visualization. First, novel folding strategies for event timelines facilitate visual scalability and provide powerful overviews of performance data at the same time. Second, this work presents an effective approach that automatically identifies and highlights several types of performance critical sections in an application run. This approach identifies time dominant functions of an application and subsequently uses them to analyze runtime imbalances throughout the application run. Intuitive visualizations present the resulting runtime variations and guide the analyst to performance hot spots.

Evaluations with benchmarks and real-world applications assess all introduced techniques. The effectiveness of the comparison approaches is demonstrated by showing automatically detected performance issues and structural differences between different versions of applications and across parallel execution streams. Case studies showcase the capabilities of the event timeline visualization techniques by demonstrating scalable performance data visualizations and detecting performance problems and code inefficiencies in real-world applications.

# 1 Introduction

This work is centered in the field of High Performance Computing (HPC). Performance optimization of parallel applications is critical in this field to efficiently use available HPC resources. The comparison of performance data is an essential part of this optimization process. Especially the detailed structural comparison of performance data is still very cumbersome and involves manual comparison and alignment of related application sections. This work describes alignment-based solutions for automatic structural comparison of performance data and thereby fills a gap of missing comparison functionalities.

Performance optimization is usually an iterative process. Users first start with an initial measurement as a baseline for comparison. Then, they try to find and optimize inefficient parts of an application and conduct a new measurement. A comparison of the two measurements shows the effect of the optimization. Consequently, comparison of performance data is a key operation in performance analysis. Scenarios for comparisons include: before/after comparisons when applying optimizations, comparisons of runs on different platforms, and contrasting the performance of different processes to study load balance.

Performance data is usually stored in form of profiles or traces. In principle, profiles consist of a list of numbers, each describing the performance of a specific part of the application. This renders the comparison of profiles rather straightforward as related numbers can be compared directly. A trace is a sequential record of application behavior. Each application activity, e.g. entering or leaving a function, is stored as time-stamped event. Traces allow in-depth insight into the performance behavior of an application. However, the comparison of traces is a challenging task due to the possibly large amount of complex performance data recorded.

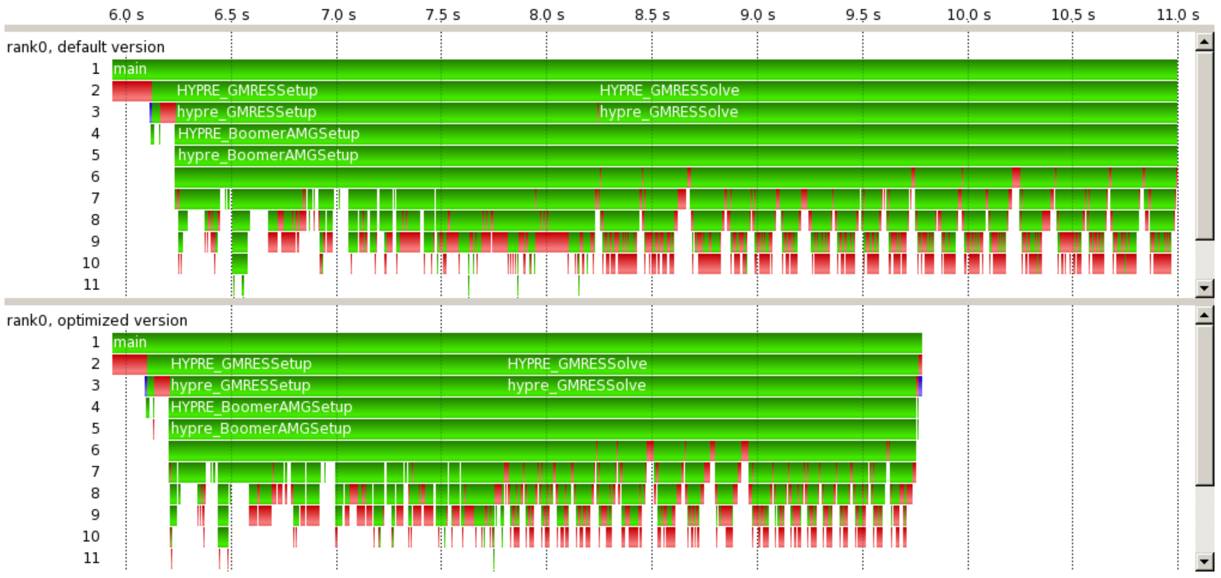


Figure 1.1: Manual visual comparison of two processes of the application AMG. The two timelines contrast a run of the default version (top) with an optimized version (bottom) of the application.

Figure 1.1 gives an example for this situation. When visually comparing both timelines, it is obvious that the optimized version runs faster than the initial version. However, detailed structural differences or performance differences throughout the application run are not immediately visible. To see details, the analyst needs to manually compare both timelines. As the bottom timeline is shorter than the top timeline, related application areas do not appear above each other. For a detailed comparison, the analyst needs to manually align the timelines in order to compare related events. However, manual comparison is extremely challenging due to the large number of events and the need to correctly line up trace events. This renders this task cumbersome and error-prone. This dissertation helps to improve this situation by providing automatic alignment-based comparison methods for trace data.

# 2 Methods for Structural Comparison of Process Pairs

This chapter describes the development of a method for the pairwise comparison of processes. The introduced algorithm [4] compares the function call structure of two processes. This approach lays the foundation for subsequent performance comparison techniques described in the next chapter.

It is challenging to compare the event streams of two processes directly. The reason is the inherent jitter in timing measurements, caused by a range of effects on the target system. Thus, the timings of events are likely to always differ to some extend.

However, it is possible to cope with the timing differences by considering only the function structure of the event streams. For deterministic code, this structure is likely to stay constant throughout multiple application runs. This allows to use application function structures as anchors for a comparison.

In order to compare function structures sequence alignment is a promising method. It arranges two sequences in a way that allows to identify similar and dissimilar parts between the sequences. During the alignment process, characters of two sequences are matched to each other (*equal* or *different*) or may be matched to an empty character "_", called a *gap*. Figure 2.1 shows the alignment of the sequences `m c a c m a m` and `m c a c b c m b m`. The basic sequence alignment algorithms have quadratic worst-case complexity with respect to the sequence lengths, regarding time and memory requirements.

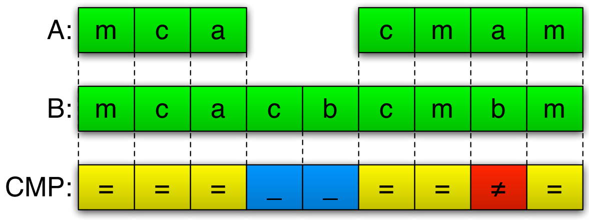


Figure 2.1: Alignment of two sequences. The CMP bar marks equal (=), different (≠), or gap areas (_).

### Accelerating the Alignment by Exploiting Hierarchy in Function Call Structures

Sequences constructed from application event streams may have lengths of millions of elements. Although the sequence alignment approach for the comparison of flat sequences satisfies the functional requirement, the quadratic time complexity means that the alignment of large traces cannot be done in an acceptable amount of time. To align and compare long application runs, their function sequence lengths need to be reduced. In that regard, the function structure of application executions provides an advantage. Most applications do not consist of flat function call sequences. Functions are rather called in a hierarchical fashion and their call structure can be represented in a call tree. Moreover, the consideration of the already available call tree structure allows to design optimized algorithms for program comparison, since it naturally represents the program structure. This work contributes an hierarchical alignment (HA) algorithm [4] that augments basic sequence alignment methods with a hierarchical comparison approach based on the call tree structure of an application.

The proposed HA approach uses the call tree structure to split a flat function call sequence into multiple sub-sequences. Therefore, the HA algorithm exploits the natural call structure by only aligning direct sub-functions of related functions in the call tree. For that purpose the algorithm traverses two call trees in parallel. While traversing the call tree, sub-sequences are built for sub-function calls (child-nodes) of each function (parent node), as depicted in Figure 2.2. For the comparison only sub-alignments of the sub-sequences are necessary. The complete alignment is then constructed from the smaller sub-alignments. For the comparison of processes, the HA heuristic provides a large performance improvement over flat alignment algorithms.

Figure 2.3 shows the HA result for the two example processes shown in Figure 1.1. Detailed structural differences and similarities between the processes are now easy to identify.

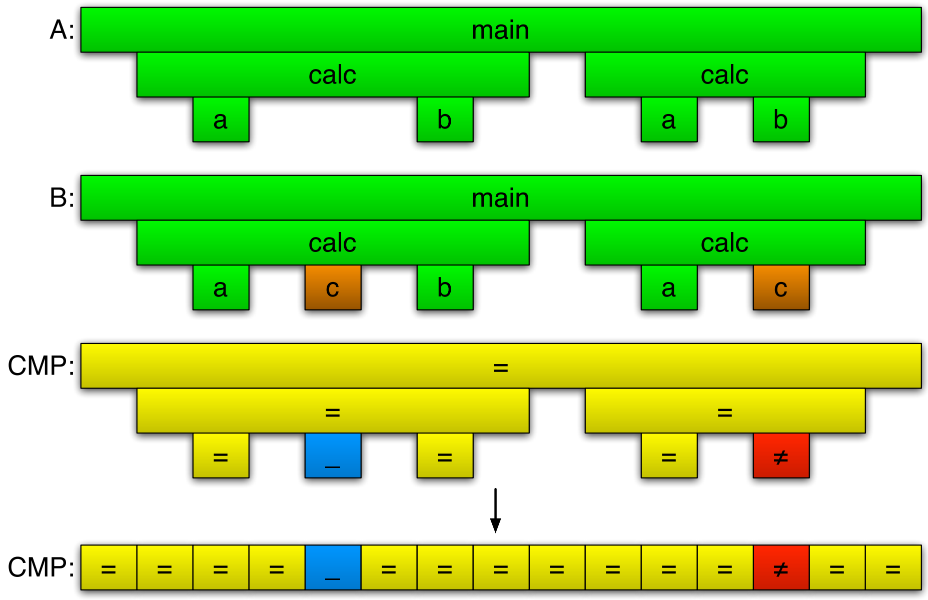


Figure 2.2: Hierarchical alignment scheme. The complete sequence is split up into multiple sub-sequences according to the call tree structure. In multiple sub-alignment steps only related sub-sequence pairs are aligned. Finally, the complete alignment is constructed from the sub-alignment results.

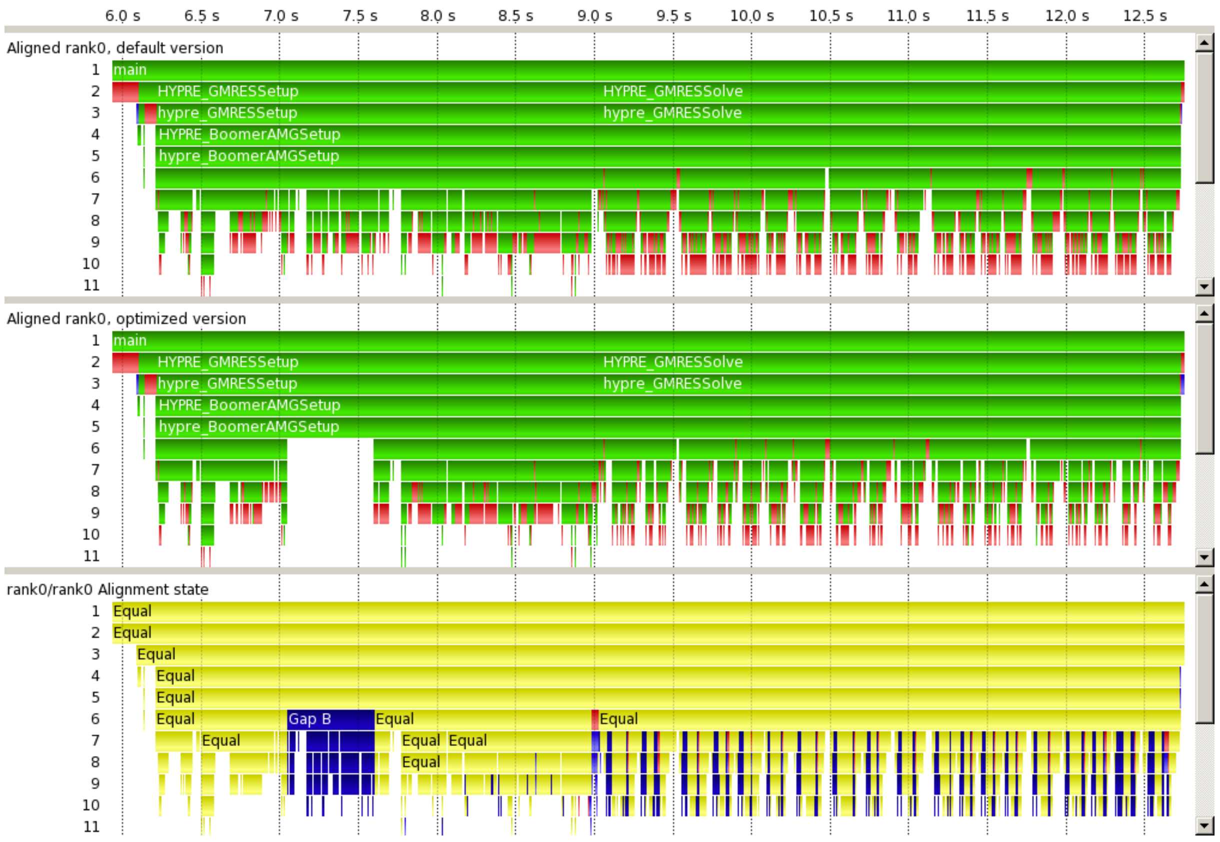


Figure 2.3: Alignment of the two AMG processes shown in Figure 1.1. Structural differences between the default and optimized version are shown in the bottom timeline.

# 3 Alignment-Based Comparison Metrics for Processes

This chapter contributes a set of novel alignment-based metrics [8] that quantify the differences between processes, both in terms of differences in the event stream and timing differences across events.

The *Dissimilarity Timeline* metric indicates how the structural similarity between two traces changes over time. The *Runtime Skew Timeline* metric depicts how the timing behavior between two traces changes over time. By directly comparing time-stamps of related events, this metric identifies to what extent the execution speed between two processes differs.

Figure 2.3 shows a detailed comparison of the default version of AMG [1] with an optimized version. Unaligned and aligned processes of this comparison are already shown in Figures 1.1 and 2.3, respectively.

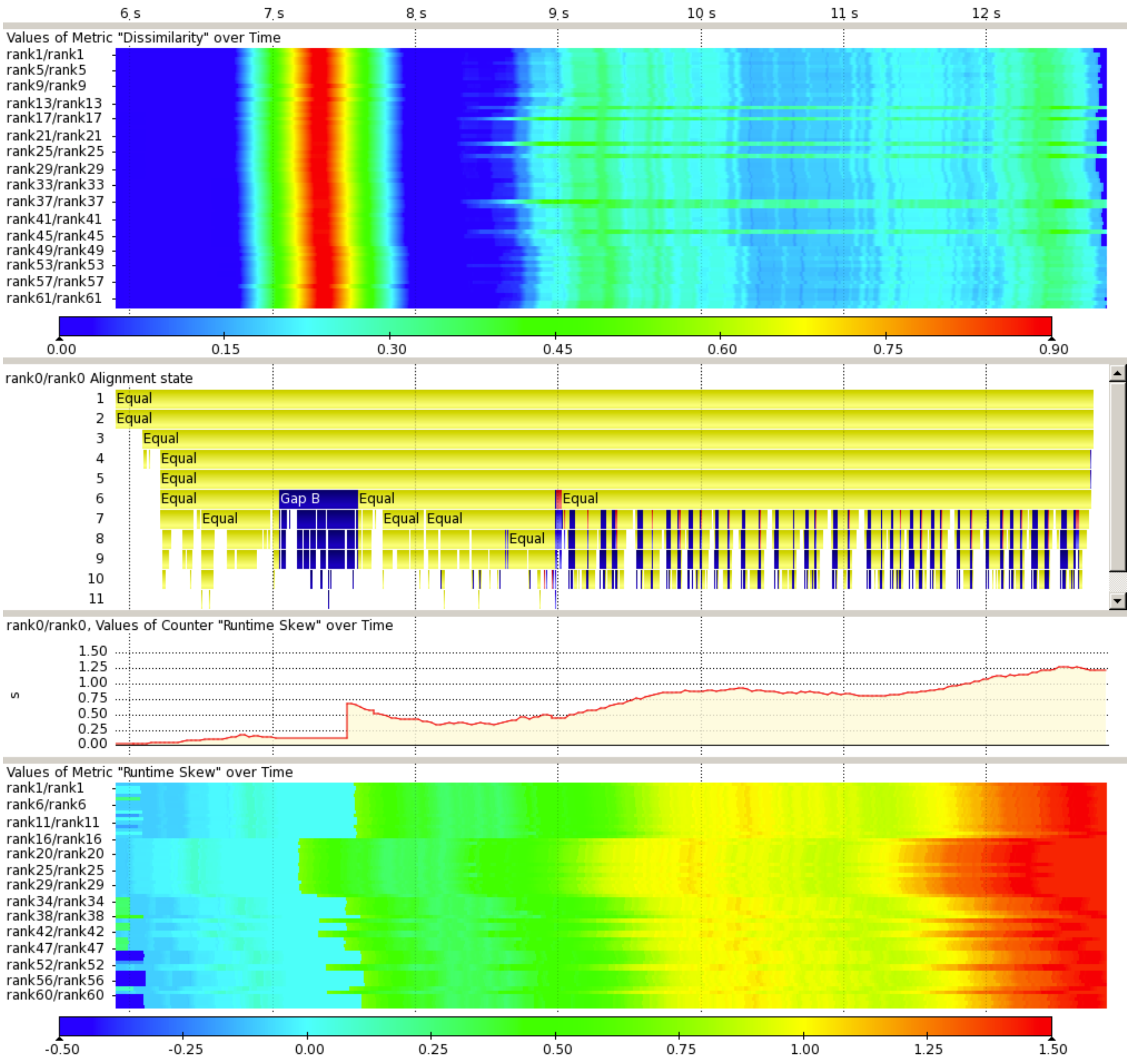


Figure 3.1: Similarity and runtime skew analysis between the default and optimized AMG version.

The optimized version saves considerable work in the initialization as well as in each computation step of the application run. These dissimilarities are the same for all 64 processes as shown by the *Dissimilarity Timeline* visualized as a set of color-coded bars in the top of Figure 3.1. Detailed timing comparisons using the *Runtime Skew Timeline* metric are shown in the bottom two timelines in Figure 3.1. The optimized version achieves a large speed gain in the initialization by avoiding unnecessary work. Also during the iterations of the main body of the code, the optimized version performs faster.

# 4 Methods for Structural Comparison of Multiple Processes

This chapter introduces a two step approach for the structural comparison of multiple processes. First, a scalable method clusters structurally similar processes [5]. Second, subsequent multiple sequence alignment methods perform a detailed comparison of grouped processes.

## Pre-Clustering Structurally Similar Processes

This section introduces a scalable clustering algorithm that reliably identifies groups of structurally similar processes from a large number of processes.

A good criterion for structural comparison of processes is information on the invocation of functions, disregarding timing. This work uses a simplification of the information contained in a call tree, the so-called *function pairs*, made up out of the caller/callee relation in the call tree. The function pairs are a set of pairs, where for each pair the first function calls the second one. The number of function pairs is independent of the application runtime or the process count and is limited to the squared number of existing functions.

The clustering algorithm works with a cryptographic hash function to scalably compute natural clusters of structurally similar processes. The goal of the hash-based clustering method is, that the hash function produces equal hashes for structurally equal processes. The output hash of the hash function is then used as key in a map structure that holds all processes. As equal processes share the same key, the map automatically groups similar processes.

Advantages of this approach are the very fast computation of the hash function and that processes can be added sequentially. The algorithm requires no direct comparison of structural information between processes. This prevents the computation of a similarity matrix and drastically reduces memory requirements. The approach groups equal processes, according to their function pair invocations. The grouping result is suitable as starting point for successive analysis methods like alignment-based comparisons.

## Hierarchical Alignment of Multiple Processes

Processes grouped in a cluster can still exhibit structural differences. The reason is that the underlying data used to compute the similarity—function pairs extracted from the call tree—only contains an aggregated representation of the structural information. In order to compare the actually executed function call structures between processes, this work develops a hierarchical multiple sequence alignment (MSA) method.

The employed MSA method works by progressively adding process sequences to an alignment. Each addition of a process's sequence requires one pairwise alignment step. As shown in this work, the times required for aligning complete process sequences might render the MSA approach infeasible. Therefore, the progressive MSA approach is placed on top of a hierarchical scheme, similar to the hierarchical approach described for pairwise alignment. By augmenting the MSA approach with a hierarchical scheme, individual pairwise alignments are executed with significantly shorter sequences.

To compare the processes of a parallel application, the hierarchical MSA computes a merged call tree of all input processes. The resulting merged call tree reveals detailed structural differences and similarities between the input processes. Figure 4.2 shows the resulting merged call tree for the example processes given in Figure 4.1. Figure 4.3 depicts a portion of the merged call tree generated from a 64 process run of the application AMG [1]. The center of Figure 4.3 shows the recursive execution of the sort function `hypre_qsort2i`. Starting at call-level 10, more and more processes have finished their sorting and stop calling the sort function.

Concluding, the merged call tree combines the structural information of multiple processes in one graph, while highlighting similarities and differences at the same time.

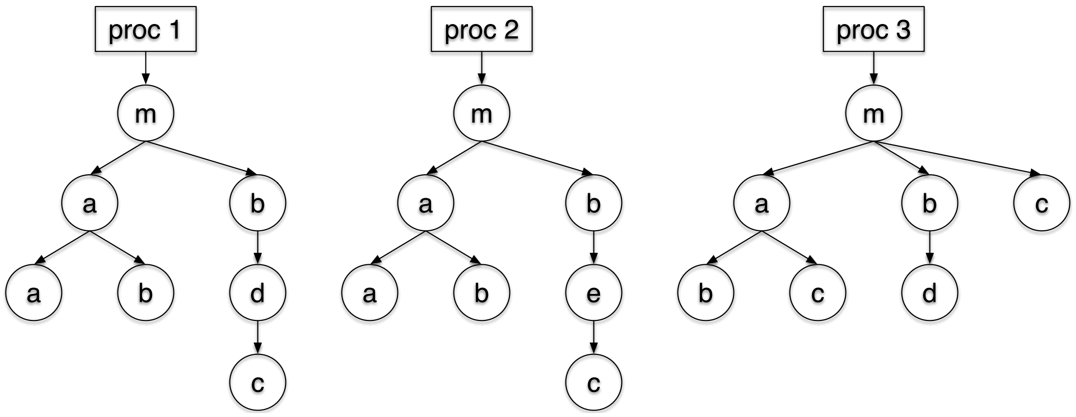


Figure 4.1: Three input processes for the hierarchical MSA algorithm.

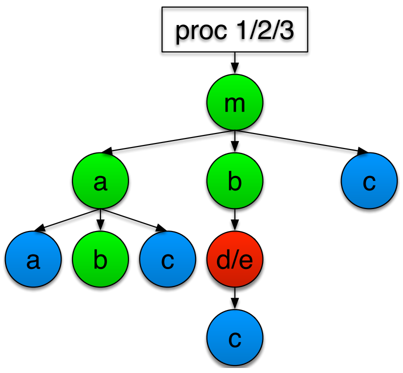


Figure 4.2: Merged call tree for the three input processes of Figure 4.1, computed by the hierarchical MSA algorithm. Green nodes depict similarity between all processes, red nodes depict differences between processes, and blue nodes indicate that not all processes executed the related function.

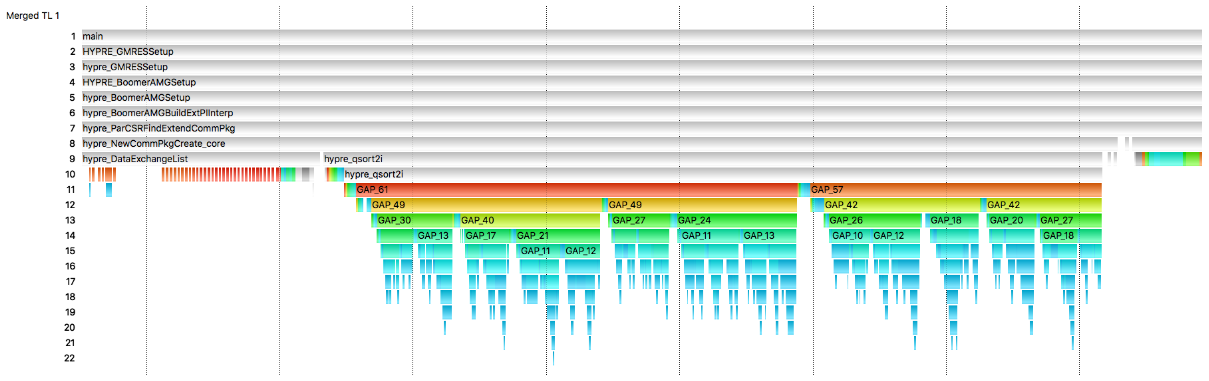


Figure 4.3: Portion of the merged call tree of 64 AMG processes. The colored parts in the center show the recursive execution of a sort function. Starting from call-level 11, more and more processes stop execution of their sorting (indicated by colors changing from red to blue).

# 5 Visualization Techniques for Event Timelines

Timeline visualization of event streams is suitable for the detection of various types of performance problems. However, visualization of large numbers of streams quickly hits the limits of available screen resolution. This chapter makes two contributions to advance visualization techniques for event streams.

## Timeline Folding Methods

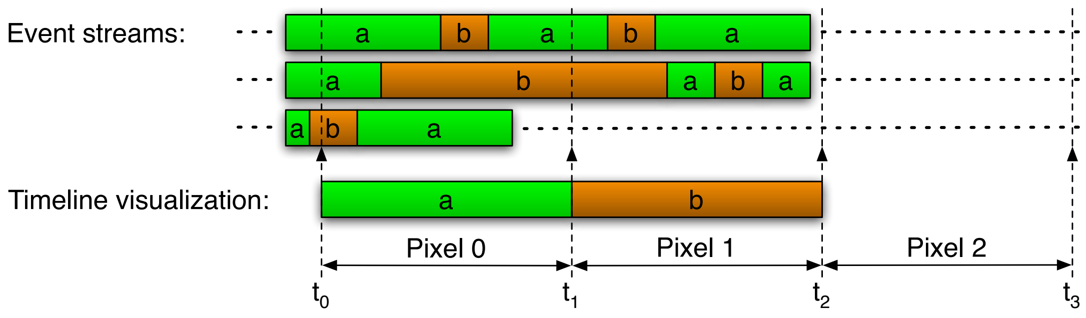


Figure 5.1: Illustration of the folding operation *Dominating Behavior* for three event streams.

By folding multiple event timelines into one result timeline, scalability and usability of timeline visualizations can be greatly improved. Figure 5.1 illustrates a folding of three streams (top) into a single aggregated stream (bottom). Consequently, the folded timeline visualization must both provide a state for each stream of event data, as for a regular timeline visualization, and must then use a selection operation to combine the states for each stream into a single result state.

This work proposes a list of selection operations that particularly target specific use cases [6]. *Dominating Behavior:* This operation represents dominant behavior across all folded streams and provides a general overview of an application's execution. *Outlier Detection:* This operation leaves pixels empty in which all streams exhibit the same state and selects the outlier behavior for differing regions. *Accelerator Utilization:* This operation highlights whether any computation kernel executes on an accelerator device and correctly considers the idle state for accelerator devices.

Figure 5.2 demonstrates the *Outlier Detection* folding operation applied to a COSMO-SPECS [2] code run with 100 processes. Colored areas visualize dissimilar executions, indicating imbalances, across the processes. As visible in Figure 5.2, the load imbalances increase throughout the application execution. The folding techniques intuitively visualize the load imbalance of the application in only one timeline.

## Detection and Visualization of Runtime Imbalances

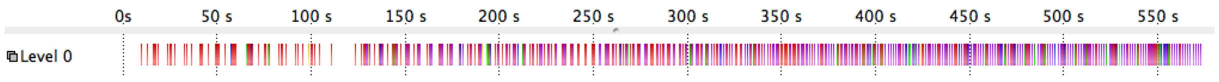
Identification of performance bottlenecks in parallel applications is a challenging task, as data sets from parallel performance measurements are often large and overwhelming. This section presents an effective and lightweight approach to facilitate visual analysis of performance data [7]. The approach automatically identifies and highlights runtime imbalances in an application run. For parallel applications that must strive to achieve good load balance, this metric highlights a wide range of load balancing problems.

To analyze runtime imbalances the following three steps are performed:

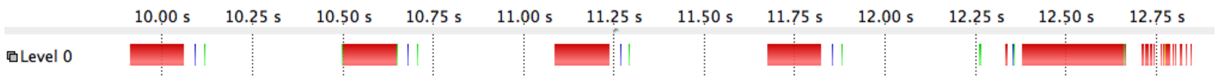
- Identification of time dominant functions to partition the complete run into small segments,
- Computation of runtime imbalances between the segments, and
- An intuitive visualization to present the overall result.

Figure 5.3 visualizes the runtime imbalances of an application run of the *Weather Research and Forecasting model* (WRF) [3]. The segments located in the lower right part highlight increased durations. Particularly *Process 39* has performance issues and exhibits higher durations than the other processes.

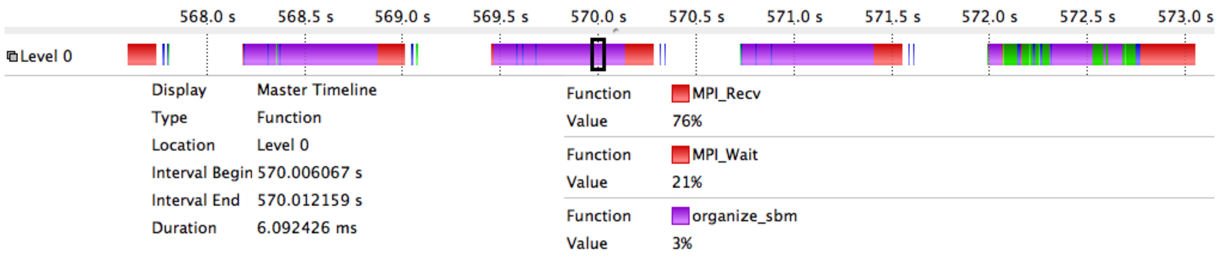
Since the runtime imbalance analysis directly identifies the location of performance issues, it enables focused subsequent analysis to find the underlying root-cause of the problem. Effectively, the introduced method supports the performance analyst in helping him to focus on performance problems faster.



(a) Visualization of the full application execution. The colored portions indicate areas in that processes execute different functions. The frequency of such areas increases over time.



(b) Visualization of the first iteration. Red areas indicate the exchange of simulated values between processes. The workload is still balanced and computations are executed in the white areas.



| | |
|---|---|
| Display | Master Timeline |
| Type | Function |
| Location | Level 0 |
| Interval Begin | 570.006067 s |
| Interval End | 570.012159 s |
| Duration | 6.092426 ms |

| | |
|---|---|
| Function | 🟥 MPI_Recv |
| Value | 76% |
| Function | 🟥 MPI_Wait |
| Value | 21% |
| Function | 🟪 organize_sbm |
| Value | 3% |

(c) Visualization of the last iteration. Large purple areas are visible. In that areas most processes are waiting and only a few processes are computing.

Figure 5.2: Visualization of the folded timelines of a COSMO-SPECS weather forecast code run.

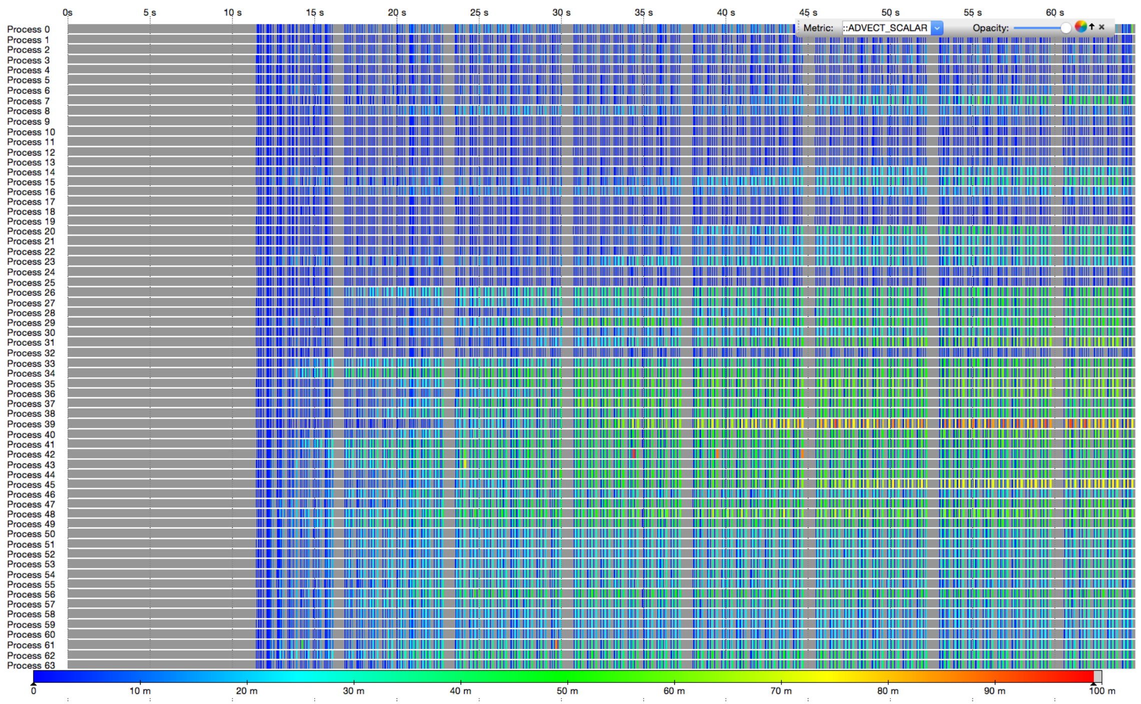


Figure 5.3: Runtime variation analysis of a WRF application run. Especially *Process 39* causes performance problems. The analysis shows high durations in its segments (red areas).

# 6 Conclusions

This dissertation presents novel analysis techniques for structural and temporal comparison of parallel processes. In the context of this work, the term *process* relates to any type of processing element of a parallel application, such as an MPI process, a thread, or a CUDA stream. Measurement systems that record parallel application runs, at very high detail, save their measurement data in the form of application event streams. The detailed event-wise comparison of such streams currently requires manual inspection by the analyst, which is cumbersome and error-prone. This work presents methods that facilitate this comparison and provide powerful and intuitive analysis capabilities for the purposes of parallel performance analysis.

The first contribution of this work consists of methods for the pairwise comparison of processes. The methods are based on sequence alignment algorithms used in bioinformatics and allow structural comparison of event streams on the level of individual events. The key contribution is a hierarchical scheme for sequence alignment algorithms which enables a comparison of complete application event streams. Without the hierarchical scheme, the computational demand of such comparisons easily exceeds the capabilities of available computing hardware.

The second contribution of this work introduces analysis metrics that effectively exploit the advantages of the new structural comparison approach. These alignment-based metrics present intuitive overviews as well as detailed comparisons of performance characteristics between application processes. This type of analysis is suitable for comparing two application runs. As such comparisons are common tasks when optimizing and analyzing parallel applications performance, both contributions fill a crucial gap in available performance analysis tools by adding efficient comparative analysis capabilities.

The third contribution is an approach for structural comparison of multiple event streams. First, a fast and scalable hash-based pre-clustering step identifies structurally similar processes. Then, multiple sequence alignment methods analyze the event streams in a cluster for detailed structural similarities and differences. This work introduces several adaptions that enable multiple sequence alignment methods for the comparison of large numbers of event streams. Most notably, this work contributes a heuristic scheme accelerating the computation of the required guide-tree and a hierarchical scheme that augments the multiple sequence alignment algorithm. The presented comparison approach computes a so-called merged call tree, that combines the structural information of all compared event streams. As structural similarities and differences of large numbers of processes are combined in one call tree, this structure enables fast comparisons of differences between large numbers of processes.

The fourth contribution of this work addresses visual scalability of performance data displays. It introduces folding strategies for event timeline visualizations, which add powerful methods for visual aggregation of event streams that help achieving visual scalability and support easy detection of performance issues, such as imbalances or accelerator usage inefficiencies. An additional technique automatically identifies and highlights several types of performance critical sections in an application run. This technique analyzes runtime imbalances throughout the application run and presents the resulting runtime variations in an intuitive visualization that guides the analyst to performance hot spots.

In summary, structural comparison of process event streams is a viable approach for analysis and comparison of parallel applications. The presented contributions add analysis techniques for large-scale application runs and advance scalable event visualization. The techniques have been applied to real-world applications and this work showed how they facilitate the identification of differences between code versions. Novel alignment-based metrics exposed differences that otherwise would have been hard or even impossible to find. Moreover, the developed techniques of this work help achieving scalable and useful trace visualizations for parallel applications. Therefore, this work contributes methods that provide users with a new level of detailed insight into the performance of their codes and will be of substantial help in optimizing them.

# Bibliography

[1] Hormozd Gahvari, Allison H. Baker, Martin Schulz, Ulrike Meier Yang, Kirk E. Jordan, and William Gropp. Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 172–181, 2011.

[2] V. Grützun, O. Knoth, and M. Simmel. Simulation of the influence of aerosol particle characteristics on clouds and precipitation with LM-SPECS: Model description and first results. *Atmospheric Research*, 90(2–4):233–242, 2008.

[3] Gilad Shainer, Tong Liu, John Michalakes, Jacob Liberman, Jeff Layton, Onur Celebioglu, Scot A Schultz, Joshua Mora, and David Cownie. Weather Research and Forecast (WRF) Model Performance and Profiling Analysis on Advanced Multi-core HPC Clusters. In *10th LCI International Conference on High-Performance Clustered Computing*, 2009.

[4] Matthias Weber, Ronny Brendel, and Holger Brunst. Trace File Comparison with a Hierarchical Sequence Alignment Algorithm. In *Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, ISPA '12, pages 247–254, Washington, DC, USA, July 2012. IEEE Computer Society.

[5] Matthias Weber, Ronny Brendel, Tobias Hilbrich, Kathryn Mohror, Martin Schulz, and Holger Brunst. Structural Clustering: A New Approach to Support Performance Analysis at Scale. In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 484–493. IEEE Computer Society, May 2016.

[6] Matthias Weber, Ronald Geisler, Holger Brunst, and Wolfgang E. Nagel. Folding Methods for Event Timelines in Performance Analysis. In *Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 205–214. IEEE Computer Society, May 2015.

[7] Matthias Weber, Ronald Geisler, Tobias Hilbrich, Matthias Lieber, Ronny Brendel, Ronny Tschüter, Holger Brunst, and Wolfgang E. Nagel. Detection and Visualization of Performance Variations to Guide Identification of Application Bottlenecks. In *Proceedings of the 45th International Conference on Parallel Processing Workshops (ICPPW)*. IEEE Computer Society, 2016.

[8] Matthias Weber, Kathryn Mohror, Martin Schulz, Bronis R. de Supinski, Holger Brunst, and Wolfgang E. Nagel. Alignment-Based Metrics for Trace Comparison. In *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, pages 29–40. Springer-Verlag, Berlin, Heidelberg, 2013.