Structural Comparison of Parallel Applications

Matthias Weber*
Martin Schulz†
Bronis R. de Supinski†

*Center for Information Services and HPC
Technische Universität Dresden, Germany
{matthias.weber, holger.brunst}@tu-dresden.de

Kathryn Mohror†
Holger Brunst*
Wolfgang E. Nagel*

†Center for Applied Scientific Computing
Lawrence Livermore National Laboratory, USA
{kathryn, schulzm, bronis}@llnl.gov

ABSTRACT

With the rising complexity of both architectures and applications, performance analysis and optimization has become essential in the development of scalable applications. Trace-based analysis has proven to be a powerful approach. However, a manual comparison of traces is extremely challenging and time consuming because of the large volume of detailed data and the need to correctly line up trace events. Our solution is a set of techniques that automatically align traces so they can be compared, along with novel metrics that quantify the differences between traces, both in terms of differences in the event stream and timing differences across events. We apply a hierarchical clustering approach that computes a dendrogram containing groups of structurally identical traces by iteratively increasing the number of included call-levels. We demonstrate the effectiveness of our solution by showing automatically detected performance and code differences across different versions of two real-world applications.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: *Performance attributes, Measurement techniques*

General Terms

Performance, Measurement

Keywords

performance analysis, comparison, alignment, tracing

1. INTRODUCTION

Today's complex architectures and applications make it challenging to fully exploit the performance of HPC machines. A key operation in understanding performance problems is a comparison of performance data from multiple measurements, because there is not sufficient information to understand the performance properties of an application without baseline measurements for comparison. While such comparisons are straightforward for the aggregated data in performance profiles, no good solutions exist for comparing highly-detailed event traces. The level of detail in traces presents a challenge because it implies a large amount of data. Thus, manually aligning event traces for comparison is extremely challenging and error prone, since the event stream may not be equal across application runs or MPI ranks, and events will not occur at exactly the same time across executions. Although several efforts have been made for comparing performance across application runs [5, 7, 8], to the best of our knowledge, no tool supports automatic comparison of event traces. We address this gap with a technique for automatic trace comparison of two arbitrary event traces. Our approach can be applied to a wide range of scenarios such as before/after comparison for optimizations, comparisons between MPI ranks, or to study changes in code versions.

2. ALIGNMENT ALGORITHM

In order to compare two traces, we use a dynamic programming algorithm that finds the optimal alignment for arbitrary sequences [6]. We employ a modification proposed by Hirschberg [4] that computes the optimal alignment with only linear memory complexity. Using the algorithm, we mark portions of traces as equal if they have the same sequence of function calls in both traces. We label portions of traces as different if they contain different function calls at the same sequence position. A gap is a missing section in one trace file. Although the dynamic programming approach is functional, the quadratic time complexity leads to long alignment times. Therefore we evaluate two novel extensions of the dynamic programming algorithm. First, we augment the algorithm with a hierarchical comparison approach based on the call tree structure of the execution that shortens the event sequence length for individual comparisons [9]. Yet, some traces might still require comparisons of long sub-sequences. Thus, we additionally improve the hierarchical approach by utilizing a fast heuristic to identify initial anchor points for the alignment process. The heuristic selects the most frequent function of both sequences as anchors and directly aligns all occurrences. Individual comparisons are only needed between anchor points. To ensure fast alignment times we re-apply the heuristic to sub-sequences between anchor points, if necessary.

3. COMPARISON METRICS

We integrate support for visualizing compared traces into Vampir [2]. To help users understand the differences between traces we introduce new trace comparison metrics.

The Dissimilarity Timeline metric indicates how the similarity between two traces changes over time. This is useful for identifying regions of the trace that exhibit high dissimilarities for further inspection. Also, visualizing the metric along with the traces can help pinpoint periodic differences. To compute the metric, we sample the alignment over time. Then we apply a sliding window technique to sum up all difference and gap pairs under the window to give a metric representing the dissimilarity of the traces over time.

For comparative performance analysis, it is important to understand the behavior of applications over time. Event traces are especially useful for this purpose, because they retain the time-stamp of each event occurrence. However, it is extremely challenging for users to gain this understanding manually. It involves attempting to line up iterations from traces and visually determine the timing differences between them. To aid in this process, we developed the Runtime Skew Timeline metric, so that the user can understand the relative timing behavior across the aligned traces.

4. CLUSTERING

We cluster traces by their structure using a hierarchical approach that iteratively adds call-levels to the computation. Result of the algorithm is a graph showing how traces fork at call-levels as structural differences increase. The computed clusters, groups of structurally identical traces up to a certain call-level, are highly suitable for further analysis or alternative clustering methods.

5. CASE STUDIES

We demonstrate the effectiveness of our approach with two real-world applications, AMG and ParaDiS.

AMG2006 [3] is a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids. We compared the default version of AMG with an optimized version that performs less coarsening. This new version does more overall work, but avoids a lot of expensive communication. We show the aligned traces for rank 0 for both versions along with the alignment state. The optimized version spends less time in initialization, identified by a blue gap area at the beginning of the run. Also, the optimized version saves work in each computation step, indicated by the repeating blue gap areas starting at the middle of the run. The Dissimilarity Timeline metric shows the comparison of all processes. The differences agree with the alignment of the rank 0 comparison and are nearly the same throughout all processes, except for a few outliers. The Runtime Skew Timeline depicts the runtime skew for all process comparisons. The optimized version achieves a large speed gain in initialization. However, in the later stages of initialization, it performs slower than the original version. Yet, the speed gain in the beginning overshadows this slow down. During the iterations of the main body of the code, the optimized version performs faster again. In this case the speed gains are not consistent from iteration to iteration. The gains level off in the middle of the execution and rise again at the end.

ParaDiS [1] models the dynamics of dislocation lines as they interact and move in response to forces imposed by external stress. We compared two versions of ParaDiS, v2.2.3 and v2.3.5.1, with release dates about two years apart. Version 2.3.5.1 includes bug fixes, improvements and corrections, as well as advanced load balancing. We show a comparison of two representative iterations of both versions. In the beginning of each iteration, the function structure is the same. However, at the end of each iteration, blue gap areas indicate new functions. These additional functions come from added capability in v2.3.5.1. The Runtime Skew Timeline metric shows that this change comes at the cost of higher runtime. ParaDiS v2.3.5.1 runs consistently slower than v2.2.3. Added functionality is not the only cause of differences. The Dissimilarity Timeline Metric shows that dissimilarity varies across the processes. This variation is caused by the changes to the load balancer in v2.3.5.1.

6. CONCLUSIONS

In this work we present a set of metrics and visualizations based on a novel alignment algorithm for traces. We applied our techniques to two applications and showed how they facilitate the identification of differences between code versions. Our metrics exposed differences that otherwise would have been hard or even impossible to find. Our techniques provide detailed insight into the performance of applications and will be of substantial help in optimizing them.

7. REFERENCES

[1] A. Arsenlis, W. Cai, M. Tang, M. Rhee, T. Oppelstrup, G. Hommes, T. G. Pierce, and V. V. Bulatov. Enabling Strain Hardening Simulations with Dislocation Dynamics. *Modelling and Simulation in Materials Sci. and Eng.*, 15(6):553, 2007.

[2] H. Brunst, M. Winkler, W. Nagel, and H.-C. Hoppe. Performance Optimization for Large Scale Computing: The Scalable VAMPIR Approach. In *Computational Science - ICCS 2001*, volume 2074. 2001.

[3] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. 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.

[4] D. S. Hirschberg. A Linear Space Algorithm for Computing Maximal Common Subsequences. *Commun. ACM*, 18(6):341–343, June 1975.

[5] K. L. Karavanic and B. P. Miller. Experiment Management Support for Performance Tuning. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing*, pages 1–10, 1997.

[6] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[7] M. Schulz and B. R. de Supinski. Practical Differential Profiling. In *Euro-Par 2007 Parallel Processing*, volume 4641, pages 97–106. 2007.

[8] F. Song, F. Wolf, N. Bhatia, J. Dongarra, and S. Moore. An Algebra for Cross-Experiment Performance Analysis. In *Proceedings of the 2004 International Conference on Parallel Processing*, 2004.

[9] M. Weber, R. Brendel, and H. Brunst. Trace File Comparison with a Hierarchical Sequence Alignment Algorithm. In *Parallel and Distributed Processing with Applications, IEEE 10th Int. Symposium on*, 2012.