Scalable Performance Analysis of ExaScale MPI Programs through Signature-Based Clustering Algorithms

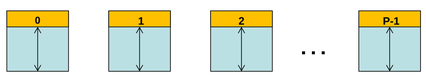
Student: Amir Bahmani, PhD student, North Carolina State University

Faculty Advisor: Dr. Frank Mueller, Professor, North Carolina State University

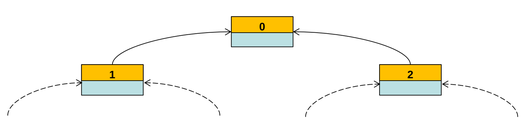
INTRODUCTION

- As the size of problems to which HPC can be applied continues to increase, so does the processing demand. Countries are competing fiercely to build the world's fastest supercomputer, similar to the Space Race in the *mid*-to-late 20th century.
- To efficiently employ such large and expensive systems, developers must study application behaviors by collecting detailed performance information with the help of performance tracing toolsets.
- Tracing toolsets could severely affect the performance and scalability of the running application:
  - Scalability: Due to the large I/O ratio that must be computed for modern large-scale machines, collecting all detailed performance information may not be feasible from a scalability perspective
  - Performance: The tracing processes may compete with the application for resources, which can perturb the application's behavior
- The overarching objective of this project is to develop a scalable tracing toolset that can cluster processes with the same behaviors into groups quickly and with low overhead; then, instead of collecting performance information from all individuals, it can collect the information from just a set of representatives.
- We applied our clustering algorithm on trace files created by ScalaTrace, an MPI tracing toolset that provides orders of magnitude smaller, if not near-constant size, communication traces regardless of the number of nodes, while preserving structural information.
- ScalaTrace has two-stage trace compression:

1. Intra-Compression:
   - Extended Regular Section Descriptors (RSDs) capture the loop structures of one or multiple communication events. Power-RSDs (PRSDs) recursively specify RSDs in nested loops.

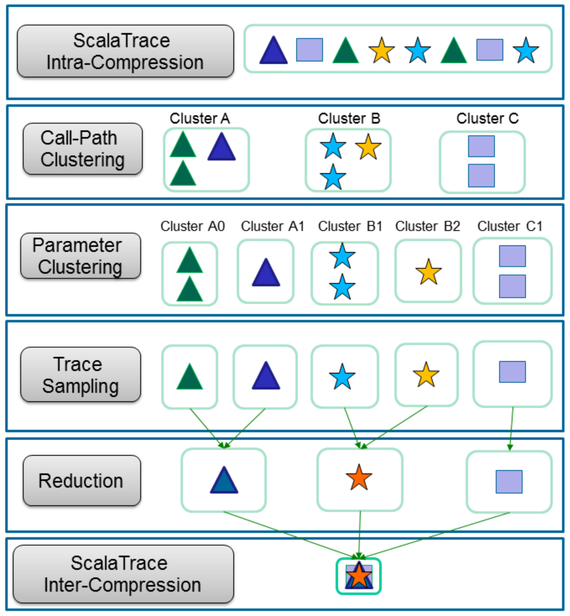


2. Inter-Compression
   - After each node has created its own compressed trace file and the program is completing, ScalaTrace performs an inter-node compression over a radix tree rooted in rank 0. During this reduction, internal nodes combine their traces with other task-level traces received from child nodes.



A NOVEL CLUSTERING ALGORITHM

- While intra-compression is fast and efficient, inter-compression is a costly operation that is $O(n^2 \log P)$, where n is the number of MPI events and P is the number of processes. We proposed and applied a signature-based clustering algorithm to reduce the bottleneck effect.



Overview of Proposed Clustering Algorithm

- The proposed clustering algorithm called *Call-Path, Parameter Clustering*, has two levels:
  - Call-path Clustering (Main Clusters): A call-path signature was created based on the stack signature of MPI events. We used the stack signature to distinguish between events from different locations. The 64-bit call-path signature is the aggregated version of the stack signatures of different events. The first level of clustering distinguishes between processes with different execution structures.
  - Parameter Clustering (Sub-Clusters): A 64-bit parameter signature was created based on details of the MPI event, such as its counts, source, destination, etc. After the algorithm clustered processes with different execution structures, with the help of parameter clustering, we were able to distinguish between processes with the same execution structure but different parameters.
- Both signatures were created based on the result of intra-compression at the node level.
- The computational cost of the clustering algorithm is O(log P).
- The algorithm focuses on creating the full trace file by the end of the second level of clustering. All similar processes are grouped together after call-path parameter clustering, so at the Trace Sampling stage, we can select *any* member of the sub-cluster as a representative.

- Reduction: Representatives within each main cluster are merged; sub-clusters, such as A1 and A2, with different parameters are merged linearly on a local radix tree.
- Inter-Compression: In ScalaTrace without clustering, all processes must follow this operation over a radix tree. In the clustering algorithm, however, instead of P, only a set of representatives from main clusters must follow the operation.

THE EXPERIMENTAL RESULTS

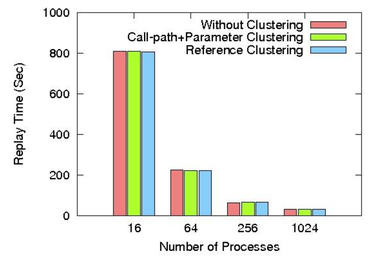
- To evaluate the accuracy and scalability of our algorithm, we created Reference Clustering which is the un-aggregated version of Call-path, Parameter Clustering.
- We utilized ARC cluster computer. All machines were 2-way SMPs with AMD Opteron 6128 processors, 8 cores per socket.

Benchmarks	BT	CG*	EP	FT	IS	LU	SP	MG*	Sweep3D‡
# of main clusters	1	1	1	1	3	9	1	2	9
# of sub clusters	3	8	1	1	1	1	3	8	1

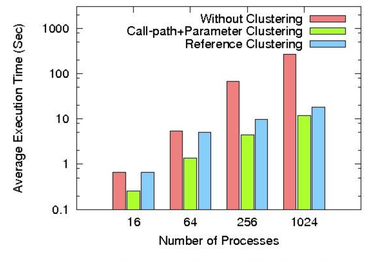
All NAS benchmarks except MG/CG: Class: **any**, # of processes: **any valid number**

‡ Prob. size: **100×100×1000**, # of processes: **any valid number**

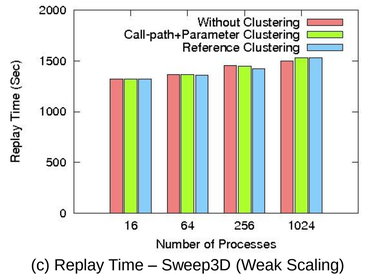
* For MG/CG: Class: **any**, # of processes: **16**



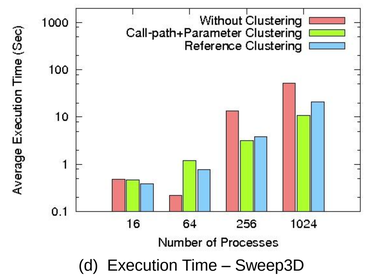
(a) Replay Time – SP Class C (Strong Scaling)



(b) Execution Time – SP Class C



(c) Replay Time – Sweep3D (Weak Scaling)



(d) Execution Time – Sweep3D

- (a) and (c): Over 96% accuracy on replaying trace files
- (b) and (d): Low overhead for large number of processes (logarithmic scale)

CONCLUSIONS

- For ExaScale computing, it is appropriate to split the merge process and to have *log P* time complexity and *low overhead* at the clustering level.
- Unlike CLARA [1] and CAPEK [2], our algorithm is based on exact matching, not on a random sampling process that may reduce accuracy

[1] Kaufman, L., and Peter J. Rousseeuw. "Finding groups in data: an introduction to cluster analysis. 2005."

[2] Gamblin, T., De Supinski, B. R., Schulz, M., Fowler, R., & Reed, D. A. "Clustering performance data efficiently at massive scales." *Proceedings of the 24th ACM International Conference on Supercomputing*. ACM, 2010.

ACKNOWLEDGEMENTS

- This research was supported by the **National Science Foundation**, award number 1217748.