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

Amir Bahmani
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
abahman@ncsu.edu

Frank Mueller
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
mueller@ncsu.edu

ABSTRACT

Exascale computing pose a number of challenges to application performance. Developers need to study application behavior by collecting detailed information with the help of tracing toolsets. But not only applications are scalability challenged, current tracing toolsets also fall short of exascale requirements for low background overheads since trace collection for each execution entity is becoming infeasible. One effective solution is to cluster processes with the same behavior into groups. Instead of collecting performance information from all individuals, this information can be collected from just a set of representatives. This work proposes a fast, scalable, signature-based clustering algorithm that clusters processes that exhibit the same execution behavior. Instead of prior work for statistical clustering metrics, it produces precise results without loss of events or accuracy. The proposed algorithm combines $log(P)$ time complexity, low overhead at the clustering level, and it splits the merge process to make tracing suitable for exascale computing.

Categories and Subject Descriptors

I.5.3 [**Clustering**]: Algorithms; D.1.3 [**Programming Techniques**]: Concurrent Programming; D.4.8 [**Performance**]: Measurement

Keywords

High-Performance Computing, Message Passing, Tracing, Clustering Algorithms

1. SUMMARY

As the size of problems to which HPC can be applied continues to increase, the processing demand also is increasing. Countries are competing fiercely to build the world's fastest supercomputer, similar to the *SpaceRace* in the mid-to-late 20th century. Undoubtedly, creating such a powerful supercomputer is not inexpensive. Hardware and software development, maintenance and, these days, power, constitute the main costs.

To efficiently employ such large systems, developers need to study application behaviors by collecting detailed performance information with the help of performance tracing toolsets. Most of these tools either obtain lossless trace information at the price of poor scalability, such as Vamipir, [1] or preserve only aggregated statistical trace information to limit the size of trace files, as in mpiP [2].

At exa-scale problem sizes, tracing toolsets could severely affect the efficiency and scalability of the system. The tracing processes may compete with the application for resources, which can perturb the application's behavior. Moreover, 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. Therefore, developers need to propose new strategies to solve these problems.

This poster proposes a fast, scalable, signature-based clustering algorithm that clusters processes that exhibit the same execution behavior. 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 [3].

ScalaTrace has two-stage trace compression, i.e., intra-node and inter-node compression. It utilizes Extended Regular Section Descriptors (RSDs) to capture the loop structures of one or multiple communication events. Power-RSDs (PRSDs) are utilized to recursively specify RSDs in nested loops [3]. 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. 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 applied our clustering algorithm to reduce the effect of the bottleneck.

1.1 Call-Path Clustering

The proposed clustering algorithm has two levels, the first of which is the call-path signature created based on the stack signature of MPI events. A stack signature may consist of a number of backtrace addresses of the program counters (return addresses), one for each stack frame. Our call-path

Table 1: Components of Parameter Signature

Component Descriptions	Bit Positions
Average **COUNT** sent or received for MPI events	0-15
DEST: XOR of the relative address of destinations of MPI events	16-31
SOURCE: XOR of the relative address of sources of MPI events	32-47
MPI Data Types: such as 48:MPI_CHAR, 49:MPI_INTEGER, etc.	48-54
MPI Operation Types: such as 55:MPI_MAX, 56:MPI_MIN, etc.	55-61
MPI Communicator Type: such as 55:MPI_COMM_SELF, etc.	62-63

signature is a 64-bit signature. To represent large stack signatures as 64 bits, we computed the exclusive or (XOR) of each part with the current 64-bit signature value. After each computation, we shift the 64-bit value of the signature by eight to the left to create sufficient entropy. We used the stack signature to distinguish events from different locations. The call-path signature is the aggregated version of stack signatures of different events. The first level of clustering distinguishes processes with different execution structures.

1.2 Parameter Clustering

Parameter clustering is the second level of clustering. At this level, we used a different signature called the parameter signature. This signature was created based on details of the MPI event, such as its counts, source, destination, etc., see Table 1. After the algorithm clustered processes with different execution structures, with the help of parameter clustering, we were able to distinguish processes with the same execution structure but different parameters.

1.3 Reference Signature

To evaluate the accuracy and scalability of our algorithm, we created reference clustering that uses a reference signature. The reference signature covers call-path signatures by adding a sequence number to each MPI event, as well as parameter clustering by keeping each MPI event's parameters uncompressed.

1.4 Results and Analysis

Figure 1 presents topologies of different benchmarks. In this figure, main clusters are separated by solid lines, and subclusters are separated by dotted lines (e.g., BT has one main cluster and three subclusters). From this figure, we make the following observations:

(1) IS has three main clusters and no sub-clusters. These three groups of processes display very similar execution behavior, except when each process sends its largest key value to the next process. (2) BT and SP experience only one main cluster, meaning that all processes have the same sequence of MPI events. However, there are three sub-clusters with different communication patterns that are captured by parameter clustering. (3) EP has only one main cluster and no sub-clusters, so all processes display the same execution behavior. (4) $Sweep3D$ and LU have nine main clusters and no sub-clusters, meaning that processes within the same main clusters display the same communication patterns. Sweep3D and LU are stencil codes. (5) The number of main clusters for MG is not constant; as shown in Figure 1, for 16 processes, this value is two, and it increases sublinearly (e.g., (P=32, MC=4), (P=64, MC=8), (P=256, MC=16), etc.). (6) CG, FT and POP only have one main cluster, meaning that there is only one execution structure. However, many sub-clusters exist within the main cluster. For CG and FT , we reduced the number of sub-clusters to one by utilizing user-provided plug-in functions. POP has one main cluster and several sub-clusters for the problem size of 320×384 blocks and the individual block size of 10×12.

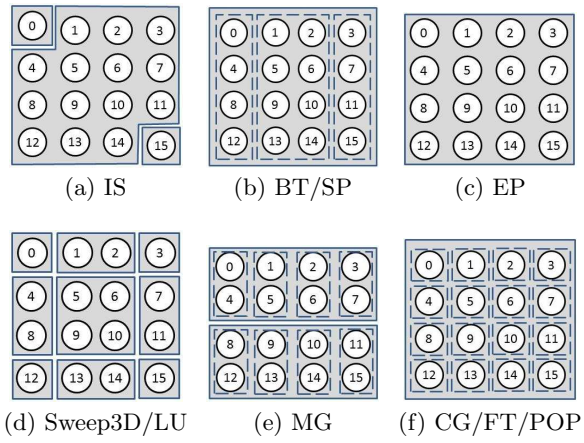

Figure 1: Topology of Different Benchmarks for 16 Processes through Call-path+Parameter Clustering

2. CONCLUSION

This poster proposed a novel clustering algorithm with $logP$ time complexity and low overhead at the clustering level. The results of our experiments indicate that the proposed clustering algorithm is appropriate for ExaScale computing. The proposed algorithm, unlike other clustering algorithms designed for large-scale problems, is based on exact matching rather than on random processes for sampling that may cause low accuracy.

3. REFERENCES

[1] H. Brunst, M. Winkler, W. E. Nagel, and H.-C. Hoppe. Performance optimization for large scale computing: The scalable vampir approach. In *Computational Science-ICCS 2001*, pages 751–760. Springer, 2001.

[2] J. S. Vetter and M. O. McCracken. Statistical scalability analysis of communication operations in distributed applications. In *ACM SIGPLAN Notices*, volume 36, pages 123–132. ACM, 2001.

[3] X. Wu and F. Mueller. Elastic and scalable tracing and accurate replay of non-deterministic events. In *ICS*, pages 59–68, 2013.