Top Challenges in Profiling Parallel Programs

Mirko Adari
University of Tartu
mirko.adari@live.com

Abstract—Supercomputers have become enormously complex, consisting of tens of thousands of nodes. The ability to tackle large problems of interest relies on writing parallel programs that utilize these systems efficiently. Towards that goal effective performance analysis tools are needed. This paper gives an overview of the top challenges and some common solutions for state of the art performance analysis tools.

I. INTRODUCTION

High-Performance Computing (HPC) involves thousands of processor cores working in parallel to analyze petabytes of data in real time and perform calculations thousands of times faster than a traditional computer. It plays an important role in the field of computational science and is applied for a wide range of computationally intensive tasks in various fields.

Today’s HPC systems (supercomputers) have become enormously complex, consisting of tens of thousands of nodes. Each node itself could have many multi-core microprocessors, often supporting additional levels of parallelism like pipelined execution of multiple instructions. Multi-level memory hierarchies, inter-processor communication and parallel I/O only add to the overall complexity. The trend is expected to continue with the growing data sets, and specialized hardware like graphical processing units and co-processors.

The ability to tackle large problems of interest relies on writing parallel programs that utilize these systems efficiently. Towards that goal effective performance analysis tools are needed to pinpoint and resolve both scalability and performance bottlenecks. This paper gives an overview of the top challenges and some common solutions for state of the art performance analysis tools.

II. CODE INSTRUMENTATION

A. Problem

In order to measure performance, additional instructions are typically inserted into the program. As these instructions execute, their execution time and context are captured—performance measurements performed.

The process of course distorts performance characteristics of the program. The most obvious problem is overhead. As illustrated by HPCToolkit[1], vendor-provided tools could add anywhere from 100% to 400% in program execution time. Another problem is accuracy. Instrumentation can interfere with inlining and template optimization, thus reducing the accuracy of measurements.

B. Solutions and Challenges

The challenge in overhead is that it is a fundamental part of instrumentation. Scaslasca[3] and TAU[2] don’t address this challenge directly, instead providing controls to blacklist parts of the program to reduce overhead, which reduces precision and creates blind spots. Distortion in accuracy is more difficult to tackle. TAU builds an estimated model of added overhead at runtime and adjusts the profile for analysis.

Another approach by Caliper[4] and HPCToolkit[1] is to skip instrumentation completely. Instead a sample is recorded at a recurring interval or on a hardware-based event. As most parallel programs generate significant amount of performance data, sample set approximates true distribution. This allows to capture both accurate and precise samples.

III. LIBRARY SUPPORT

A. Problem

Programs use other libraries for common tasks—like math and communication libraries. Frequently the source code for these libraries is not available and program has to link against fully optimized or even partially stripped libraries. This can create critical blind spots in the application performance.

B. Solutions and Challenges

Caliper[4] proposes a modular approach with a dedicated interface for library developers to integrate with. However, TAU[2] with a most diverse options for instrumentation at all levels has significant community adoption, which Caliper lacks yet. This is understandable as it was only announced in 2016.

HPCToolkit[1] intercepts process control routines either through preloading supported by modern dynamic loaders or by a script at link time for statically linked libraries. Supported by hpcstruct tool to create program structure from fully optimized or even stripped binaries users get full precision.

Fig. 1. HPCToolkit workflow.
IV. CALLING CONTEXT

A. Problem

Context is important to pinpoint issues in modern, modular programs. Costs incurred by each procedure have to be attributed to the different contexts in which the procedure is called. As an example, let’s consider MPI.Wait. It’s not helpful to know that the program spends significant time on waiting for MPI requests, we’re interested in which requests specifically. Often these procedures are also wrapped in other procedures, so we need to capture the entire call path to have the necessary context for resolving performance issues.

B. Solutions and Challenges

The challenge is in recording call paths at runtime in an efficient manner. Sample-based are less sensitive to this, but still have to account for data compactness. Tree-based structures, where aggregated metrics are calculated at each node, are used by all except TAU[2]. TAU speeds up the node checks by using a map with indexes of call depth and an array of performance events in a call path. Although by definition a map-based structure, the underlying logic for traversal is the same.

V. PERFORMANCE METRICS

A. Problem

Measurement of time helps to identify hot spots, but seldom a performance bottleneck. It is more useful to look at derived metrics—where are resources wasted instead of consumed. Moreover a different sets of metrics are necessary for discovering and troubleshooting the issue. Application-specific semantic informational is often valuable for the latter.

B. Solutions and Challenges

The standard set of metrics—e.g. operation counts, times, hardware counters—have no differences between the tools. Differences are present in how additional metrics are defined. Caliper[4] generalizes all metrics, whether standard, library or application-provided, into attributes (key-value pairs) on a snapshot level. These can be added as simple annotations in the source code in the aspect oriented programming fashion. TAU[2] allows for performance data source modules with a wide range available both by TAU itself and the community. HPCToolkit[1] is most limited due to its design choices. As instrumentation is avoided completely, there is no interaction point to introduce additional measures. It does allow for defining derived metrics in the hpcuser tool based on captured data. Same functionality is available though TAU’s profiling query interface.

VI. DATA VOLUME

A. Problem

Parallel and distributed systems or applications frequently generates significant amount of performance data. For performance tools to be useful for these systems, measurement and analysis techniques have to able to scale with the system.

B. Solutions and Challenges

In general, all tools record data locally on the nodes in compact data structures and then parse it into an aggregated database for user analysis. The novel approach by Scalasca[3] is in trace analysis in the system itself. This has two advantages: more granular data is available then would be in an aggregated database, and all the resources of the system are available to run the analysis in parallel. Eventually a report is generated for further exploration and analysis.

Fig. 2. Scalasca performance data flow.

VII. CONCLUSION

Many of the top challenges in profiling parallel programs—code instrumentation, calling context and actionable performance metrics—are indifferent to challenges in application profiling, and also approached similarly. However, the sample size of parallel programs lends itself to novel approaches such as statistical sampling used by HPCToolkit and Caliper.

The large sample size also creates a unique problem of managing performance data. The most common approach is to capture data on a node level, which is then streamed into a database of aggregated metrics for analysis. Scalasca instead leverages parallel analysis to produce a report from granular local traces on nodes.

REFERENCES