Application Profiling with Score-P and Cube

*A Summary of Essential HPC Profiling and Tracing Methods

Carlos Benavides
Advanced Computing
Universidad de Costa Rica
San José, Costa Rica
carlos.benavidesviquez@ucr.ac.cr

***Abstract*—Profiling and performance analysis are essential for optimizing parallel applications. This summary explores the use of the Score-P and Cube tools in the context of the HPC ecosystem. Score-P provides instrumentation and measurement for various program paradigms (MPI and OpenMP for example) Cube offers a visual interface to analyze collected data. Key topics includes the performance analysis cycle, sampling and instrumentation (measurement techniques) and integration of additional tools for advanced data handling. Insight about effectively using Score-P and Cube for comprehensive performance analysis in parallel applications, empowering developers to optimize code large applications can be found in this summary.**

***Index Terms*—HPC, Score-P, Cube, Optimization, OpenMP, MPI**

I. Introduction

Efficiently analyzing and optimizing code is essential in high-performance computing (HPC) to harness the full potential of parallel programming. Tools like the Score-P and Cube play a very important role in this process, by providing a structured approach for profiling and analyzing performance in applications. This summary delves into the main functionalities of Score-P and Cube, taking into account the core concepts, techniques and principal workflows that facilitate a comprehensive performance analysis for parallel applications. Score-P and Cube empowers developers to fine-tune their code and achieve better execution times and efficiency across large-scale applications.

II. Parallel Performance Tools 101

In this section, we dive into the basics of performance tools used in high-performance computing (HPC) to understand and optimize application behavior. When working with parallel applications, performance issues can be complex, and effective tools are essential to identify and analyze these bottlenecks. Score-P and Cube are examples of such tools, and each provides unique functionality for HPC profiling and tracing. This overview introduces some fundamental terminology and methods that we'll encounter, as different people and tools may refer to these concepts in varying ways. Establishing a consistent understanding of terms—such as instrumentation, measurement, profiling, and tracing—helps set a solid foundation for using performance tools effectively. With these essentials in place, we're ready to explore how these tools fit into the broader HPC performance analysis process, making it easier to pinpoint inefficiencies and improve application speed and scalability.

III. Performance Analysis Cycle

In the *Performance Analysis Cycle* section, we learn about the systematic process used to measure and improve the performance of parallel applications. Essentially, the cycle is broken down into four key phases: **instrumentation**, **measurement**, **analysis**, and **optimization**. Each phase builds on the last, creating a loop that we repeat until we reach satisfactory performance, or run out of time and resources.

The cycle starts with *instrumentation*, which involves adding extra code, or "hooks," to our application. These hooks, often inserted by tools like Score-P, are placed in strategic parts of the code—like function calls or message exchanges—to track important data without changing the original functionality. Next, in the *measurement* phase, we run the application with this added instrumentation, capturing data like execution time or memory usage. This step provides raw data that reflects how the program actually performs.

Once measurement is complete, we move on to *analysis*. This phase is where tools like Cube shine; they help us interpret the collected data, pointing out issues like time-consuming functions or unbalanced loads across processors. Analyzing this data is critical because it guides us to the most impactful areas for improvement. Finally, there's *optimization*. Here, we take the insights from analysis and make targeted changes to the code, like adjusting parallelization strategies or refining algorithms to enhance performance.

The cycle doesn't just end here. Once we make an optimization, we repeat the process—re-measuring and re-analyzing—to see if the changes truly helped or, sometimes, unintentionally made things worse. This iterative approach helps developers fine-tune performance, making the application as efficient as possible.

IV. Measurement Techniques

The techniques introduced in this section presents two key approaches for capturing performance data in high-performance computing applications. **Sampling** and **instrumentation**.

Sampling uses external triggers, like timer interrupts or hardware events, to capture snapshots of the program's state at specific intervals. This method provides a broad statistical view of application performance with minimal overhead, making it useful for identifying general patterns in resource utilization and function timings. However, because sampling relies on intermittent data collection, it may miss finer details of the program's behavior.

On the other hand, instrumentation adds additional code—known as hooks or probes—within the application to track specific events, such as function calls or data exchanges, as they occur. This technique offers more detailed insights into program execution but can slow down performance due to the overhead it introduces. As such, it is particularly useful when a highly detailed view of program behavior is needed, and when this added cost is acceptable for the analysis.

In summary, sampling is ideal for capturing high-level trends with minimal performance impact, while instrumentation provides a more in-depth look at specific events, albeit with a potential slowdown. The choice between these methods depends on the level of detail required and the trade-off in performance overhead that can be accommodated.

V. Details on Profiling and Tracing

A. Details on Profiling

Profiling in high-performance computing focuses on collecting aggregated data to identify which parts of an application consume the most resources or time. Profiling captures metrics like how often functions are called, the total time spent in each function, and overall system performance statistics. This method is particularly effective for long-running programs since it sums up the data over time without storing every individual occurrence. By the end of a profiling run, the data provides an overview of the application's performance, highlighting major hotspots or functions that contribute most significantly to resource usage.

The main advantage of profiling is that it requires relatively little storage, which is especially valuable for applications running for hours or days. Instead of tracking every action continuously, profiling gives a summarized view, which can reveal broader patterns in application behavior. However, the trade-off is that profiling doesn't capture fine-grained temporal details. This means profiling might miss specific moments when the workload varies significantly or when performance bottlenecks develop only at certain times. For applications needing precise, real-time data on performance fluctuations, another method—tracing—may be more appropriate.

B. Details on Tracing

Tracing goes a step beyond profiling by recording each relevant event in an application's execution as it happens, creating a comprehensive, time-ordered log. This level of detail is valuable for understanding the dynamic behavior of parallel applications, especially those with performance issues that occur inconsistently over time. By capturing every event with a timestamp and related context (e.g., which function was executed or which message was sent), tracing allows developers to observe exactly when and where delays, synchronization problems, or load imbalances occur within an application.

However, tracing can generate vast amounts of data, especially in parallel applications running across multiple processes and threads. The high volume of data often limits tracing to shorter program runs or smaller scales to keep storage manageable. While this can restrict its applicability for very long-running applications, tracing remains an essential technique for gaining insight into time-sensitive bottlenecks. For many applications, using profiling for a high-level overview, followed by selective tracing for deeper analysis, is a balanced approach that combines the benefits of both methods.

The Score-P Ecosystem

The Score-P ecosystem offers a suite of tools designed to support profiling and tracing in high-performance computing (HPC) applications. Score-P itself provides a flexible, scalable framework for measuring performance, supporting various programming paradigms such as MPI, OpenMP, CUDA, and OpenACC. By allowing developers to collect both profiles and traces, Score-P captures a range of performance data, from general usage statistics to detailed, time-ordered events. This versatility makes it an invaluable tool in the HPC community, where applications often require insights into both high-level performance and fine-grained behavior.

Beyond Score-P's core functionality, its integration with other tools like Cube, Vampir, and Scalaska strengthens its analytical capabilities. Cube, for example, serves as a powerful visualization tool to interpret Score-P profiles, while Vampir is used for in-depth tracing analysis. Scalaska adds a layer of automatic analysis for specific performance bottlenecks, providing an enhanced performance report. Together, these tools enable a comprehensive ecosystem where users can analyze and optimize their applications more effectively across all stages of the performance analysis cycle.

Performance Analysis Workflow

The performance analysis workflow using Score-P involves several stages, beginning with **instrumentation**, where the application is prepared to collect performance data. During this step, Score-P inserts code hooks, or "probes," into the program to record relevant events or resource metrics. After instrumentation, the program is run to gather performance measurements, resulting in either a profile or a trace file. Profiles give summarized data, while traces capture a sequence of events over time, allowing developers to choose the best option based on their analysis needs.

Once measurement data is collected, tools like Cube (for profiling) and Vampir (for tracing) enable in-depth analysis. These tools help visualize and explore performance metrics, such as function execution times and system resource use. With insights from the analyzed data, developers identify

optimization targets within the code, make adjustments, and rerun the workflow to confirm improvements. This iterative process of measuring, analyzing, and optimizing enables refined performance and scalability in complex HPC applications.

VI. Profile Analysis with Cube

Cube is a versatile tool within the Score-P ecosystem designed to visualize and analyze profile data. It provides a three-dimensional data structure representing **metrics**, **call paths**, and **system locations** in a hierarchical format, making it easier for users to navigate the details of large-scale performance data. Each dimension represents a unique aspect of the application's execution, such as time spent in specific functions (metrics), the structure of function calls (call paths), and the distribution of tasks across processes and threads (system locations). Cube's hierarchical views allow developers to "drill down" from high-level metrics to finer details, enabling targeted analysis.

Using Cube, developers can identify time-intensive functions, resource imbalances across nodes, and areas with high overhead. Color-coded metrics highlight critical performance issues, making bottlenecks easy to spot. Cube's visual representation helps developers quickly assess where time and resources are concentrated within an application, allowing for more focused optimization efforts. By leveraging Cube's insights, developers can iteratively improve application performance, achieving better efficiency and balance across computing resources.

VII. Cube Command-line Tools

The Cube Command-line Tools offer a range of utilities designed to extend and enhance the analysis capabilities of the Cube profiler. One of the most useful tools is cube-diff, which compares two Cube files by calculating the differences between them. This is especially helpful for evaluating the impact of optimizations, as users can quickly see whether changes improved or worsened specific metrics. Another tool, cube-merge, combines multiple measurements into a single Cube file. This functionality is ideal for situations where profiling needs to be done in stages, such as collecting different sets of hardware counters in multiple runs and then merging them into one file for comprehensive analysis.

Other command-line tools in the Cube suite include cube-derive for calculating custom metrics and cube-cut, which enables users to isolate specific portions of a Cube file for focused analysis.

These tools help tailor Cube's data output to specific research needs, making it easier to obtain relevant metrics or ignore unimportant data. For more detailed examination, cube-stat and cube-dump provide statistical summaries and data exports, respectively. Together, the Cube command-line tools support a more flexible and efficient performance analysis workflow, enabling users to adapt Cube's functionality to fit a variety of HPC performance optimization tasks.

VIII. Conclusion

In summary, the combination of Score-P and Cube provides a powerful, flexible solution for performance analysis in high-performance computing (HPC) environments. Score-P offers the foundational capabilities of profiling and tracing, which allow users to capture essential performance data, from general trends to specific events. With its ability to support various parallel programming models, including MPI, OpenMP, and CUDA, Score-P adapts well to the diverse needs of HPC applications, enabling detailed and reliable data collection for large-scale systems.

Cube complements Score-P by offering an intuitive interface to explore and interpret the collected profile data. With its hierarchical visualization of metrics, call paths, and system locations, Cube helps developers quickly identify performance bottlenecks and resource imbalances. The tool's color-coded and hierarchical structure makes complex data manageable, guiding developers to the most critical areas for optimization. Together, Score-P and Cube form a comprehensive ecosystem, allowing users to effectively move through the stages of the performance analysis cycle—measurement, analysis, and optimization.

Overall, using these tools enables a systematic approach to improving HPC applications. By following an iterative workflow that combines broad profiling with in-depth tracing, developers can achieve significant performance gains and resource efficiency. This structured analysis not only enhances application performance but also supports scalability, making it possible to meet the increasing demands of modern HPC workloads.

References

[1] HPC NRW, "Application Profiling with Score-P and Cube," YouTube, 26 Oct. 2023. [Online Video]. Available: https://www.youtube.com/watch?v=3HtR89L2u9Q. [Accessed: 26-Oct-2023].