Analyzing the Score-P Software Profiling Tool on the Summit Supercomputer

Score-P is a software profiling tool that is used specifically on high-performance computing (HPC) systems in order to make large applications run more efficiently. It can collect data on running applications about how parallel processes communicate with each other in addition to time and memory usage of the program. Score-P also works closely with another software tool called Vampir, which generates interactive visualizations of what is going on inside the code. During the summer of 2022, we analyzed Score-P's behavior when profiling code at different scales and programming models. Additionally, we were able to add to the Oak Ridge Leadership Facility's (OLCF) Score-P documentation on how to profile specific regions of a program. We were also able to create a hands-on tutorial that teaches beginners how to profile programs with Score-P. These contributions provide training that can promote Score-P's use in HPC and further educate members of the OLCF userbase.

Analyzing the Score-P Software Profiling Tool on the Summit Supercomputer

Leah Evans (Pellissippi State Community College, Knoxville, TN 37932)
Subil Abraham and Will Castillo (Oak Ridge National Laboratory, Oak Ridge, TN 37830)

Profiling and tracing are two important tasks used within supercomputing that collect data about how a program runs and are important for identifying slow or poorly optimized code. Score-P is a software profiling tool that is used specifically on high-performance computing (HPC) systems. It can collect data on running applications about how parallel processes communicate with each other in addition to time and memory usage of the program. Score-P also works closely with another software tool called Vampir, which generates interactive visualizations of what is going on inside the code. As part of the Oak Ridge Leadership Computing Facility (OLCF), the User Assistance team provides documentation, training, and direct assistance for people who want to use the supercomputer systems, such as Summit or Frontier. During the summer of 2022, we analyzed Score-P's behavior when profiling code at different scales and programming models. We also developed additional Score-P training routines to promote its use in HPC and to further educate members of the OLCF userbase. We made one significant addition to the open-source OLCF Score-P documentation hosted on Github. This addition showed how to profile specific regions of a program, a process called manual instrumentation. We also added a hands-on tutorial to OLCF's Hands-On with Summit training curriculum that teaches beginners how to profile programs with Score-P, using simple examples in addition to a miniaturized HPC application. These two tools in particular are very helpful in teaching users how to more efficiently get the best performance possible from their code on the OLCF HPC systems. Our project contributions will be used for educating new and veteran users at future training events at ORNL, and at universities and conferences across the country and internationally.

Introduction:

High-performance computing (HPC) allows large applications—those that are too large to run on regular computers—to run quickly and effectively by providing computing power on the order of quadrillions of operations a second. These larger applications can include detailed simulations that are used to advance scientific discoveries and technology in various domains, such as astrophysics, meteorology, and nuclear technology. These applications perform dense mathematical operations and generate enormous amounts of data. They need to utilize all of the computing power that is available to them, so profiling and tracing these applications are important tasks within supercomputing. These tasks help to identify slow or poorly optimized code that can be improved upon in order to make the applications more efficient. Score-P is a profiling software tool that is used to perform these tasks and gather data on time and memory usage. Furthermore, it is especially made to analyze parallel-processing—that is, using several processes simultaneously to distribute the workload—as well as how messages are being communicated between these processes. Specifically, Score-P can gather data about different programming paradigms like Message Passing Interface (MPI) and OpenMP threading. Score-P can also create a trace file, which contains all the data from tracing the running application and can be viewed with a software tool called Vampir. Vampir generates interactive visualizations like graphs and timelines of what is going on inside the code.

The User Assistance team in the Oak Ridge Leadership Facility (OLCF) provides training and documentation that help users understand what they need in order to run programs on our supercomputing systems. For Score-P specifically, there is documentation and training videos available that detail how to use the tool. However, there was no hands-on challenge on the subject, and many members of the OLCF did not know much about Score-P.

One objective for the project was to test and analyze how Score-P is used on Summit and how it behaves alongside the use of the Vampir graphical user interface (GUI) tool. Once sufficient knowledge was obtained, the goal was then to write further documentation on Score-P and to create a beginner level, easy to follow tutorial that can train users on how to implement this profiling tool.

Progress:

We profiled many programs using Score-P over the course of the project, including two miniaturized applications called MiniWeather [1] and MiniSweep [2]. These mini apps are used for many purposes including algorithm development, performance characterization, hardware benchmarking, etc. We also used more simple programs such as the Monte Carlo Estimating Pi program [3], which uses MPI and the Monte Carlo method to estimate the value of pi.

We tested Score-P on these programs and captured different data flow patterns when running in serial, parallel (MPI), and/or threaded parallelization (OpenMP). After running and profiling a program that just uses MPI, we would run and profile that same code but with additional OpenMP threading. We also applied this to an increasing number of nodes. The data from Score-P showed that, for small amounts of nodes, running with OpenMP combined with MPI was faster. However, as the number of nodes increases, running with OpenMP+MPI actually becomes slower than just MPI. For example, MiniWeather running on 5 nodes with OpenMP+MPI took about 17.1 seconds while just MPI took about 17.9 seconds. On the other

hand, MiniWeather running on 10 nodes with OpenMP+MPI took about 19.2 seconds while just MPI took about 18.2 seconds. Also, with 15 nodes, MiniWeather took about 23.1 seconds with OpenMP+MPI and 18.9 seconds with just MPI. In addition to the overall time of a program, we were able to see how specific function times were affected by the different paradigms. Specifically, computation-heavy functions took significantly less time when OpenMP threading was used to distribute the calculations. Although we already knew that MPI and OpenMP are important to optimize a program on our supercomputers, Score-P was able to give us exact numbers, and Vampir allowed us to see how individual functions behaved differently.

Furthermore, we compared how the different programming paradigms were profiled by Score-P when scaled out to use more nodes. We ran MiniWeather on 0.5, 1, 5, 10, 15, and 100 nodes and MiniSweep on 1, 5, and 10 nodes. As the amount of nodes allocated increased, we saw that the size of the Score-P trace file also increased. Additionally, while going from using half of a node to one node resulted in a faster trace runtime, as expected, scaling above one node actually resulted in longer runtimes. We also noticed that the percent time spent in MPI increased with the number of nodes while the percent time spent in OpenMP decreased. From this as well as the visualization of functions in Vampir, we were able to conclude that the longer runtimes occur because more nodes used on Summit means more time that MPI functions take to communicate to all the nodes.

While profiling and tracing MiniWeather on 100 nodes, we also discovered that one method for launching Vampir on Summit, called VampirServer, was not incredibly efficient in loading large trace files. Therefore, thanks to this project, we were able to identify this problem and work towards fixing it so that large trace files with a lot of data can be visualized more easily.

Additionally, we were able to contribute to the documentation by adding a section on manual instrumentation. Manual instrumentation is a feature of Score-P that allows for a user to profile specific regions of code in addition to automatically profiling functions. Although information on how to manually instrument Score-P is talked over in the OLCF's training archives, the open-soucrce Score-P documentation did not include it. Now, though, there is a section for it which includes additional information on a specific flag needed when compiling.

After learning how Score-P is used on the programs we profiled, we were able to write an additional entry to the challenges section in OLCF's "Hands-On with Summit" training series held on GitHub. This series walks through and gives hands-on exercises for basic concepts used on the Summit supercomputer such as MPI, OpenMP threading, GPU offloading, etc. It is used in our HPC Crash Courses that teach beginners fundamental UNIX and HPC skills. The tutorial we created walks beginners through each step in the process of using Score-P and Vampir with two main example programs: the Monte Carlo Estimating Pi program as well as MiniWeather. The latter was used specifically to familiarize the user on how to instrument Score-P in a CMake system, which is different and slightly harder than a simple Makefile, and how to compare two different trace files in Vampir. Thus, while the Estimating Pi program was used as a simple introduction to Score-P, MiniWeather offered a more challenging example. There were also two other programs (MiniSweep and a Jacobi program) offered as bonus challenges that provide an opportunity to practice the Score-P skills they learned in the rest of the tutorial.

Over the course of this project, not only was I able to learn Score-P and how to use it with Vampir, but I also gained significant knowledge in both HPC and the Summit supercomputer. With HPC, I learned how to use programming paradigms like MPI and OpenMP threading, and with Summit, I learned its structure as well as the process of running programs on it. I was also able to learn basic GPU coding, which, though not a major part of the project, was beneficial to my overall experience.

Future Work:

The Score-P hands-on challenge, in addition to the rest of the challenges in OLCF's "Hands-On with Summit" training, will be part of many future HPC Crash Courses in conferences and universities. Specifically, in September, this challenge will be used in the Tapia conference in D.C. and the SIAM conference in San Diego. It will also most likely be used in the SC'22 conference in November.

This challenge can also set the groundwork for more Summit challenges to be created about other profiling tools such as TAU. Furthermore, with the recent development of the Frontier supercomputer, we will need new training. The documentation and challenge on Score-P we created can be implemented into the Frontier training, either for using Score-P on Frontier or as a blueprint for other profiling tools.

Impact:

The added section of the Score-P documentation as well as the hands-on tutorial will be used to expand the use of Score-P throughout the OLCF user base. Not only will new users and beginners to HPC be able to learn Score-P, but veteran users will be able to as well. This is important because profiling applications can lead to improved efficiency and better optimization. Therefore, users can get the best performance possible from our HPC systems.

Conclusions:

Score-P is an incredibly useful tool on the Summit Supercomputer that was tested and used to analyze many programs. Not only did we add to the OLCF's documentation on Score-P, but we were also able to write a hands-on challenge for users to learn the basics of the profiling tool as well as the Vampir GUI. This provides an easy way to increase the use of Score-P so that large applications can be optimized on our supercomputing systems such as Summit and Frontier.

References:

1. Norman, M (2018) The MiniWeather Mini App [Source code]. https://github.com/mrnorman/miniWeather#running-the-code.

2. Joubert, W (2014) MiniSweep [Source code]. https://github.com/wdj/minisweep.

3. "Monte Carlo Estimate Pi," [Online]. Available: http://www.selkie.macalester.edu/csinparallel/modules/MPIProgramming/build/html/calculatePi/Pi.html

Appendix

Participants:

Name	Institution	Role
Subil Abraham	Oak Ridge National Laboratory	Mentor: provided guidance and resources throughout the project and helped explain key concepts like the HPC systems, programming paradigms, git and Github software, etc.
Will Castillo	Oak Ridge National Laboratory	Provided knowledge on Score-P and helped work through using the profiling tool.

Scientific Facilities:

Oak Ridge Leadership Facility

Notable Outcomes:

1. “Score-P and Vampir Basics – OLCF Hands-On with Summit” https://github.com/olcf/hands-on-with-summit/tree/master/challenges/Score-P_and_Vampir_Basics

2. "Manual Instrumentation in Score-P – OLCF Documentation" https://docs.olcf.ornl.gov/software/profiling/Scorep.html#manual-instrumentation.

Analyzing the Score-P Profiling Tool on Summit Supercomputer

Leah Evans[1], Subil Abraham[2], Will Castillo[2]

[1] Pellissippi State Community College, Knoxville TN
[2] National Center for Computational Sciences Division, Oak Ridge National Laboratory, Oak Ridge, TN

leah28evans@gmail.com

Introduction

Background:
Profiling is an important part of high-performance computing (HPC) in which data like time and memory usage can be gathered to analyze the efficiency of a program. Score-P is a profiling tool that collects this data on large applications. It is especially good at assessing how parallel processes communicate with each other while the program runs with Message Passing Interface (MPI) functions and OpenMp threading functions. This data can then be visualized with another software tool called Vampir.

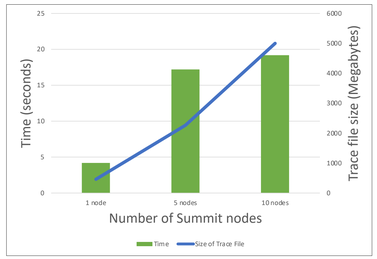
Objectives:
To test and analyze how Score-P profiles different programming paradigms i.e. serial, MPI, OpenMP and how Score-P profiles when programs are scaled up, and lastly to create further training and documentation on Score-P for users of the Summit supercomputer.

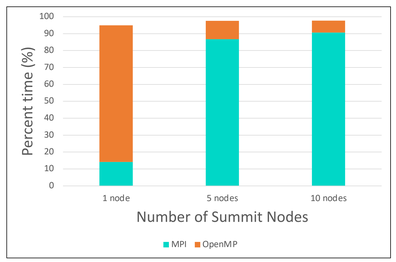
Method

- Tested Score-P on specific programs that captured different data flow patterns when running in serial, parallel, and/or threaded parallelization
- Profiled two miniApp programs (miniWeather & miniSweep). MiniApps are reduced versions of full-scale applications that are typically used for performance characterization, algorithm development, hardware benchmarking, and other studies. MiniApps are also often used for training users of HPC systems because of their relative simplicity.
- Compared how the different programming paradigms were profiled by Score-P when scaled out to use more nodes, specifically 1, 5, & 10 nodes
- Used version control tools git and Github for documenting my findings

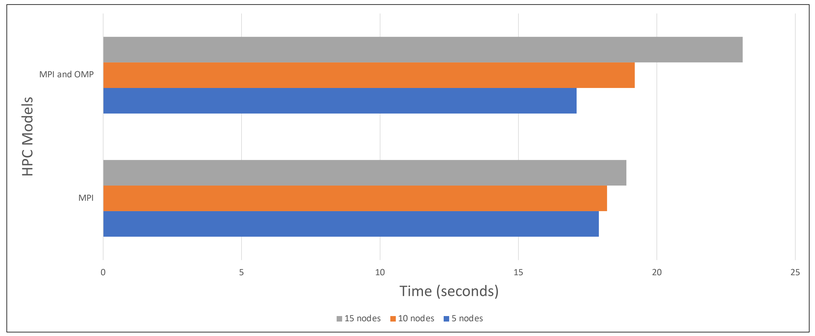
Results

As the number of nodes allocated increases, three things also increase: the size of the Score-P trace file, the accumulated time across all processes, and the amount of time spent in MPI functions. Additionally, while the percent time of MPI functions increase, the percent time of OpenMP functions decrease. This is because a larger number of nodes requires more time for MPI functions like “MPI_Bcast” and “MPI_Reduce” to communicate to all the processes. Below is data collected from the MiniWeather app, but the trends are consistent with the other programs we tested on.





When we used Score-P to compare just MPI vs MPI and OpenMP, we found that, for small amounts of nodes (like 5), using OpenMP resulted in a faster runtime. However, using more nodes showed that using OpenMP resulted in a slower runtime. Below is the results from MiniWeather for 5, 10, and 15 nodes. The number of ranks and threads are, respectively: 30 ranks and 90 threads, 60 ranks and 120 threads, and 90 ranks and 270 threads.



Exploring the usage of Score-P on multiple applications this way fed into developing essential beginner friendly training material to supplement the existing documentation.

Conclusions

Outcomes:
- Contributed to the Oak Ridge Leadership Computing Facility (OLCF) documentation on Score-P, specifically on how to profile specific regions of a program, a process called manual instrumentation. (https://docs.olcf.ornl.gov/software/profiling/Scorep.html#manual-instrumentation)
- Created a hands-on Score-P and Vampir tutorial on OLCF’s ”Hands-On With Summit” challenges hosted on Github. (https://github.com/olcf/hands-on-with-summit/tree/master/challenges/Score-P_and_Vampir_Basics)

Future Benefits:
- Knowing how to use Score-P and Vampir on our systems will help identify bottlenecks and optimize applications, maximizing their HPC resource usage.
- Our contributions and knowledge will be used to train all the users of Summit, including members of the OLCF team.

Acknowledgements

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Community College Internships program.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.