Toward Performance Visualization for TAU Instrumented Exascale Scientific Workflows

Wei Xu
Brookhaven National Laboratory
Computational Science Initiative
Upton, New York 11973
xuw@bnl.gov

Cong Xie
Stony Brook University
Computer Science Department
Stony Brook, New York 11790
coxie@cs.stonybrook.edu

Kevin Huck
University of Oregon
Performance Research Lab
Eugene, Oregon
khuck@cs.uoregon.edu

Hubertus van Dam
Brookhaven National Laboratory
Computational Science Initiative
Upton, New York 11973
hvandam@bnl.gov

Sameer Shende
University of Oregon
Performance Research Lab
Eugene, Oregon
sameer@cs.uoregon.edu

Kerstin Kleese van Dam
Brookhaven National Laboratory
Computational Science Initiative
Upton, New York 11973
kleese@bnl.gov

Klaus Mueller
Stony Brook University
Computer Science Department
Stony Brook, New York 11790
mueller@cs.stonybrook.edu

Line Pouchard
Brookhaven National Laboratory
Computational Science Initiative
Upton, New York 11973
pouchard@bnl.gov

Abid Malik
Brookhaven National Laboratory
Computational Science Initiative
Upton, New York 11973
amalik@bnl.gov

ABSTRACT

In exascale scientific computing, it is essential to efficiently monitor, evaluate and improve performance. Visualization and especially visual analytics are useful and necessary techniques in the exascale computing era to enable and enhance such a human-centered experience. In this work, we devise a visualization platform for performance evaluation of scientific workflows toward the exascale scenario. We first improve Tau instrumentation toolset to accommodate workflow measurements. Then we design new visualization methods to show trace and profile information of the workflow. In order to support the scalability, a few level-of-detail visual methods are proposed with user interactions. Finally, an NWChem use case is adopted to verify our methods.

CCS CONCEPTS

•Human-centered computing → Visualization; •Computer systems organization → *Parallel computing framework;* Real-time systems;

KEYWORDS

Performance Visualization, TAU, Scientific Workflow, Exascale Computing

ACM Reference format:
Wei Xu, Cong Xie, Kevin Huck, Hubertus van Dam, Sameer Shende, Kerstin Kleese van Dam, Klaus Mueller, Line Pouchard, and Abid Malik. 2017. Toward Performance Visualization for TAU Instrumented Exascale Scientific Workflows. In *Proceedings of AFourth International Workshop on Visual Performance Analysis, Denver, Colorado, USA, November 2017 (VPA'17),* 6 pages.
DOI: 10.475/123_4

VPA'17, Denver, Colorado, USA

DOI: 10.475/123_4

1 INTRODUCTION

Exascale systems allow applications to execute at unprecedented scales. With the increased data volume, the disparity between computation and I/O rates is more intractable, which leads to offline data analysis. Recently, a co-design center focused on online data analysis and reduction at the exascale (CODAR) [9] was founded by the Exascale Computing Project (ECP). We endeavor to provide an infrastructure for online data analysis and reduction to extract and output necessary information and accelerate scientific discovery. Moreover, we will support systematic analysis and improve the accuracy and performance of online data analysis and reduction methods.

On the other hand, scientific workflows are commonly utilized in this scenario to schedule computational processes in parallel and coordinate multiple types of resources for different scientific applications such as quantum chemistry, molecular dynamics and climate modeling. Thus, the capability to capture, monitor, and evaluate the performance of workflows in both offline and online modes is essential to confirm expected behaviors, discover unexpected patterns, find bottlenecks, and eventually improve the performance. Since parallel applications rely on the performance of a number of hardware, software and application-specific aspects, such as multicore clusters, programmable graphics processing units (GPUs), multiple hierarchical memory access, network, and so on, the captured performance evaluation data usually has multiple dimensions and disjoint attributes. This complication makes the exploration and understanding extremely challenging.

Visualization, as an indispensable technique for big data, has the capability to fuse the multidimensional and heterogeneous evaluation data, provide corresponding visual representations for exploration, and create effective user interaction and steering. Specifically, performance visualization is the technique focusing on performance data of heavy computation applications. The data is acquired through instrumentation of a program, or monitoring system-wide performance information. There are a number of instrumentation and/or measurement toolkits such as TAU [11], Score-P [7] and HPCToolkit [2]. In this paper, we aim to devise a new performance visualization framework dedicated to improving the execution performance of exascale scientific workflows with TAU instrumentation. To the best of our knowledge, there is no performance visualization work focused on workflow execution, especially in preparation for exascale systems. The remainder of this paper is structured as follows: Section 2 summarizes the related works, Section 3 introduces our major method, followed by Section 4 as a conclusion.

2 RELATED WORKS

The general purpose of performance evaluation includes: the global comprehension, problem detection and diagnosis [4]. Performance visualization is therefore designated to fulfill these goals. At a minimum, the design of the visualization must be able to show the big picture of the program execution. When an interesting area is targeted, users must narrow down the region and mine more detailed information. Moreover, comparative study looking for correlation or dependency must be supported. For problem detection, abnormal behaviors can be highlighted in ways that allow users to identify them easily.

Current visualization works can be grouped by their applications in four contexts: hardware, software, tasks and application [4]. In this paper, the acquired data are individual trace and profile files capturing the execution of independent workflow components. Specifically, it includes a few types of data: 1) an event table summarizing the start and end time of all function calls, 2) the message passing among threads, 3) profiling of certain metrics spent in each part of the code on each computing thread, and 4) the call path for each thread. Therefore, we only summarize the existing works that are commonly applied to our data types. Other works such as the visualization for network, system memory usage, or system logs for multicore clusters can be found in [4].

2.1 Trace visualization

Tracing measurement libraries record a sequence of timestamped events such as the entry and exit of function calls or a region of code, the message passing among threads, and job initiation of an entire run. A common practice is to assign the horizontal axis to the time variable, and the vertical axis to the computation processes or threads. Different approaches are usually variations of Gantt charts. Vampir [12] and Jumpshot [5] provide two examples of this kind of visualization. Generally, overview of the whole time period is first plotted. Then users can select interested area to reveal more detailed events happened during the selected period. Different functions or regions of code are colorized, and the black (yellow for Jumpshot) lines indicate message passing such as shown in Fig. 1

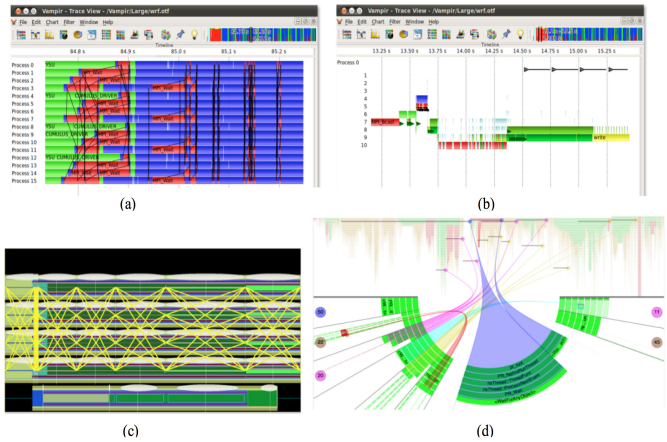

Figure 1: Trace timeline visualization examples: (a) Vampir timeline showing the execution on all processes [12], (b) Vampir timeline for one process with detailed function entry and exit [12], (c) the timeline of Jumpshot, and (d) the advanced visualization for focused thread comparison [6].

In addition, advanced visualization tools such as SyncTrace [6] provide a focus view showing multiple threads as sectors of a circle. The relationships between threads are shown with aggregated edges similar to chord diagram.

2.2 Profile visualization

Profiling libraries measure the percentage of time spent in each part of the code. Profile does not typically include temporal information, but can quickly identify key bottle necks in a program. Stacked bar charts, histogram, and advanced visualization in 3D are commonly used to give a comparative view of the percentage of time or other metric spent for different functions. ParaProf [10] is one example of this kind of visualization as shown in Fig. 2. It also supports the comparison of certain function calls in different execution runs. The functions are color coded and plotted in different stacking modes. Other statistics can also be plotted for a selected function over all threads or for a selected metric correspondingly.

2.3 Call paths and Call graphs

Through profiling, call path information can also be included. The percentage of time spent in each call path can be illustrated. The caller and callee relationship can also be identified. The common approach to visualize this kind of data is tree structure that utilizes node-link metaphor, or indented tree to preserve collapsible hierarchies. In Vampir, ParaProf and Cube [3], we can see the examples of such approach, shown in Fig. 3. Note that a circular layout similar to a sunburst is also useful to illustrate the caller-callee relationship [1] as shown in Fig. 3. The edge is bundled to avoid clutter. The structural dependency is thus easier to capture. The color is used to encode call direction as well as call time.

2.4 Message communication

As mentioned in the timeline visualization, message passing is also important. A straightforward approach is to draw a line between

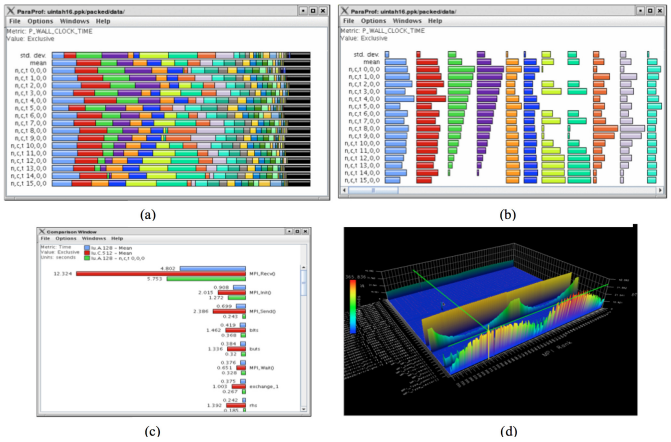

Figure 2: Profile visualization examples: (a) ParaProf showing the profile of all functions in stacked view [10], (b) separated view [10], (c) the comparative view of different execution runs [10], and (d) the 3D visualization comparing different metrics [10].

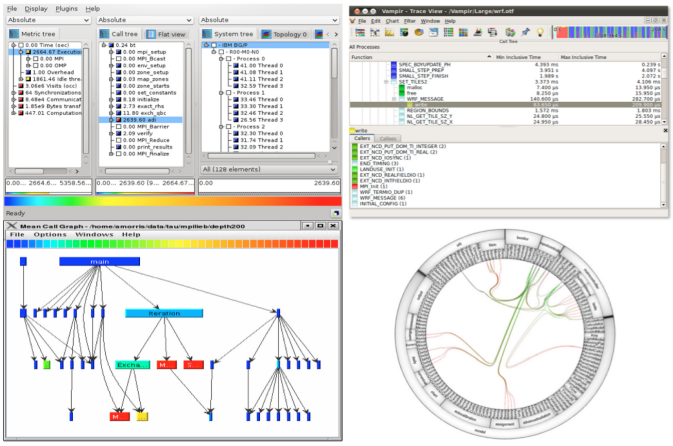

Figure 3: Call path and call graph visualization examples: the call tree structure in CUBE (top left) [3] and Vampir [12] (top right), the call graph in ParaProf [10] (bottom left), and the circular layout for caller-callee relationship [1] (bottom right).

two functions for each message, as how Vampir and Jumpshot implement. On the other hand, the message communication between threads or processes can also be summarized in terms of a matrix, with proper colorization indicating additional information as shown in Fig. 4.

2.5 Limitations

As part of the CODAR project, we aim to visualize the measurements for workflows of various applications. However, existing tools are mostly for single application execution, which is not in the form of workflow composing a number of serial or parallel executions. Thus they lack the capability to illustrate the workflow

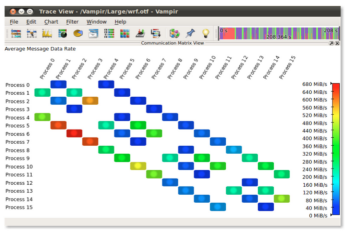

Figure 4: Message communication: the message matrix used in Vampir [12].

structure and the I/O and other metadata between workflow components. Another major issue is the challenge for online performance evaluation. This not only requires the streaming visualization updates, but also relies on the online data reduction and sampling mechanism. Most existing works are designed for offline analysis. Although the visual representations are still effective for online evaluation, they must be adjusted to accommodate the streaming fashion so as to incrementally update and visualize data.

3 METHODOLOGY

In this work, since our major focus is for offline workflows, we chose an NWChem use case to demonstrate our contributions. NWChemEx [8], as the next generation of NWChem, is a scientific toolkit for simulating the dynamics of large scale molecular structures and materials systems on large atomistic complexes. For this specific use case, it includes two modules: molecular dynamics (MD) module and the analysis module. Its workflow has a structure where the MD simulation is run, emitting snapshots of the protein structure along the trajectory, and concurrently the data analysis is triggered whenever the expected data is produced.

3.1 TAU Output

Our acquired data with TAU instrumentation are individual trace and profile files capturing the execution of independent workflow components. TAU was used to instrument the application code, as well as collect MPI inter-process communication using the standard PMPI interface. The post-processed TAU measurements include several types of data: 1) event table listing start and end time of all function calls, 2) the messages passed between processes, 3) profiling of certain metrics spent in each code region on each computing node/thread, and 4) the call path for each node/thread. We aggregated the profiles and/or traces collected by each component with purpose-built post-processing scripts that generate structured JSON output and merged traces. Aggregating the profile data is somewhat straightforward, as the time dimension is collapsed within each component measurement. However, the trace data includes detailed communication information between processes within the component (i.e. MPI messages from rank m to rank n). Additional post-processing must be done to remap component ranks within a global, workflow instance rank structure.

Finally, a trace file and profile file both in JSON format were generated. The size of this trace file is 16GB, and represents a classical MD run of 2200 timesteps on 4 nodes as well as the corresponding analysis of 1000 timesteps on 1 node. The MD execution took 309.3

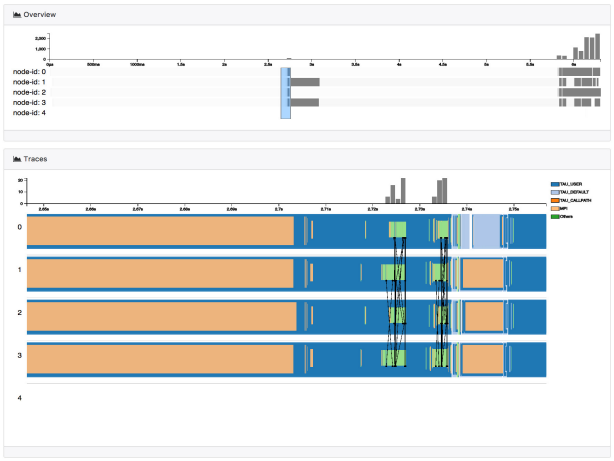

Figure 5: The overview panel of our framework that summarizes the whole workflow execution in timeline format, where the color density indicates the depth of call path; the top axis shows message counts among nodes/threads.

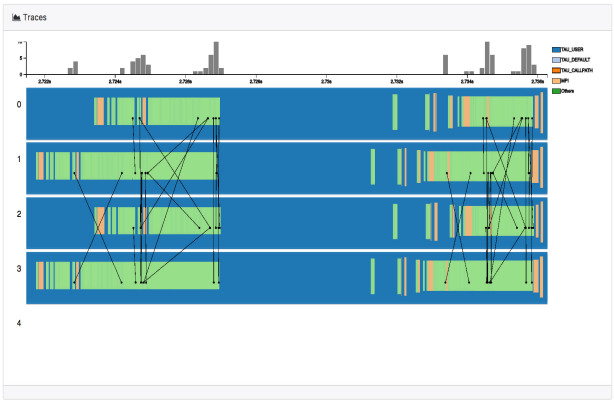

Figure 6: The detailed view panel: a coarse level of selected timeline is shown.

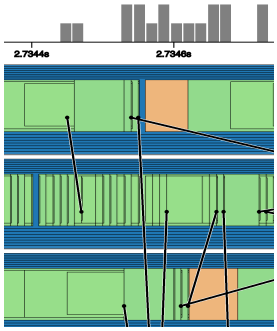

Figure 7: The detailed view effect: when the depth of the call is straightforward to observe.

seconds wall clock time in total. The size of the profile file is 21MB. We selected the first 6.5s to illustrate how our visualization method works.

3.2 Our Framework

In order to explore all the above information, we devised and developed a web-based level-of-detail and multiple-channel visualization framework with a front-end plotting the data and a back-end performing necessary analysis and computation.

Our visualization includes five major components: overview, detailed view, node detailed view, statistical view, and profile view that together establish an interactive analysis platform to visually explore and analyze the performance of a workflow.

3.3 Overview

Overview shows the summary of the whole workflow execution as in Fig. 5(top). There are two parts: the trace events, and message counts. The trace events indicating the start and end time of each function call are shown as timeline. We use intensity to indicate the depth of the call path. A darker color represents a more nested function call. For each node/thread, the trace events are plotted separately. Above the timelines, we also visualized the message counts (sent or received) in a separate histogram view along the timeline. For the interaction, it allows the user to select a time range of interest and see more details in the detailed view panel.

3.4 Detailed View

Detailed view shows the function calls and the messages in the selected time range as in Fig. 5(bottom). Each function call is visualized with a rectangle and the its color representing corresponding its call group. The functions are visualized with nested rectangles that indicate their depths in the call path. This fashion is similar to what Jumpshot [5] utilized, but we chose a more compact layout. In this view, users can still zoom in to explore more details by selecting a smaller time range. This is shown in Fig. 6. When there are many short function calls in a nested call structure, it can be difficult to observe small events and differentiate each call. Therefore, we designed two features for that issue. First, we use different transparency to enhance the visibility of overlapped functions. Second, when zooming in, we add the stroke of the rectangle to enhance the separation of different functions, as shown in Fig. 8(b). They are colored according to different call groups. Furthermore, in order to enhance the call path structure, we reduce the height of the rectangle along a call path. Therefore, a callee function must have smaller height than a caller function. As in Fig. 7, where by observing the number of horizontal lines around a function, users can easily detect the "depth" of the call. For the message visualization, Additionally, we visualized the message passing (send and receive) between functions as straight black lines (see Fig. 8). As being organized in a timeline, the line direction is ignored since the message is always passing from left to right. Finally, when hovering over each rectangle, detailed function name can be seen in text.

3.5 Node Detail View

In the detailed view, all functions are plotted as nested rectangles. This design is compact and reflects caller-callee relationship well. However, when there are too many short function calls one after

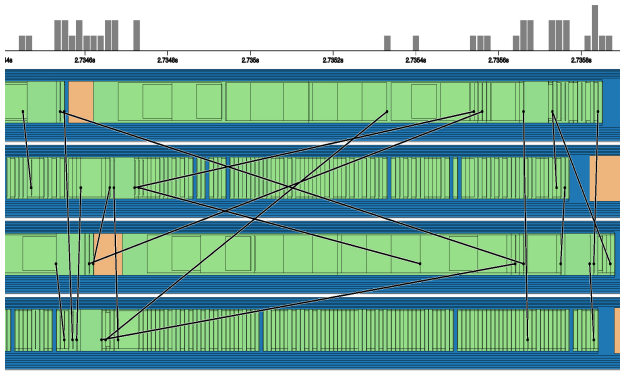

Figure 8: The detailed view panel: a fine level of selected timeline is shown when further zooming in.

Figure 9: The node detail panel: a coarse level of selected timeline of node 3 is shown. The trace panel and the node detail panel are aligned along the time axis.

another, it can be difficult to track how often and how long each function is executed. Therefore, we design a node level detail exploration tool as shown in Fig. 9. This view is aligned with detailed view panel when user selects one specific node to explore. Then the nested rectangles are replaced with stacked bar graphs, where overlapping is avoided. In this new view, the vertical order of each bar graph reflects its call path depth. In Fig. 10, a fine level design with highlighted strokes around bar graphs further enhances the separation of functions.

3.6 Statistical and Profiles Views

Statistical view summarizes the time spent for each function call in the selected region, as shown in Fig. 11. In this aggregated visualization, it is easier to compare total execution time difference side by side. We also adjust the text display in order to avoid clutter issue.

Figure 10: The node detail panel: a fine level of selected timeline is shown when further zooming in; the node 3 is selected, and the strokes are plotted.

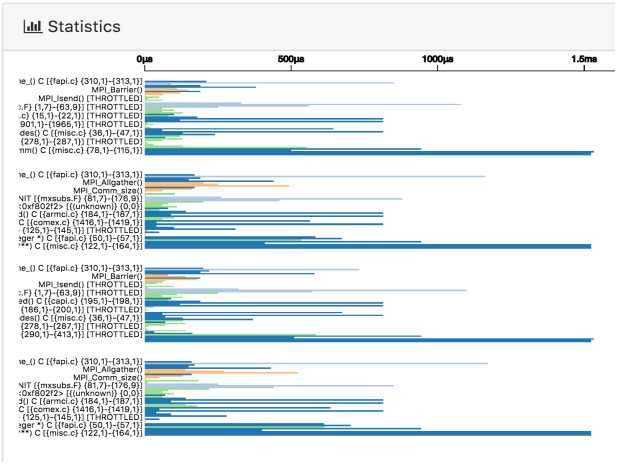

Figure 11: The Statistical view: the time spent for each function is summarized in an aggregated display.

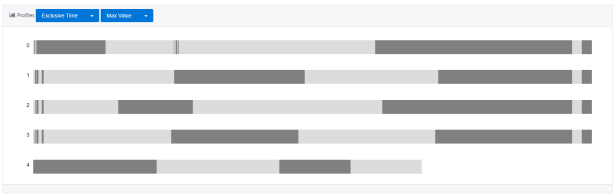

Figure 12: The Profiles view: the profile information of the workflow is visualized as stacked bar graphs; exclusive time metric with maximum values is chosen.

In the Profiles view, with the selected metric (time or counter), we visualize the percentage spent on each function for each node/thread in stacked bar graphs. For example, in Fig. 12, we plotted the profile for "exclusive time" metric.

4 CONCLUSION

In summary, we have devised a visualization platform for workflow performance evaluation. We utilized nested bar graphs and stacked graphs to represent events in terms of timeline. We also connect

that to a profile representation in the form of stacked bar graphs to reveal the corresponding statistics of the chosen metric. In order to support the scalability, we implemented a few types of visualization for different levels of details: overview, zoom-in with transparency, and zoom-in with separation enhancement. The message communication is also visualized by line connections and message count histograms. As future work, we will show data I/O and message communications among nodes. Also, we will enhance the data query performance.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration. The Project Number for the Co-design center for Online Data Analysis and Reduction (CODAR) that supported this research is 17-SC-20-SC.

REFERENCES

[1] 2007. *15th International Conference on Program Comprehension (ICPC 2007), June 26-29, 2007, Banff, Alberta, Canada*. IEEE Computer Society. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4268224

[2] Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R Tallent. 2010. HPCToolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 685–701.

[3] CUBE. 2017. (July 2017). http://apps.fzjuelich.de/scalasca/releases/cube/4.3/docs/manual/userguide.html

[4] Katherine E. Isaacs, Alfredo Giménez, Ilir Jusufi, Todd Gamblin, Abhinav Bhatele, Martin Schulz, Bernd Hamann, and Peer-Timo Bremer. 2014. State of the Art of Performance Visualization. In *Eurographics/IEEE Conference on Visualization State-of-the-Art Reports (EuroVis)*.

[5] Jumpshot. 2017. (July 2017). https://www.cs.uoregon.edu/research/tau/docs/newguide/bk01ch04s03.html

[6] Benjamin Karran, Jonas Trumper, and Jurgen Dollner. 2013. SYNCTRACE: Visual thread-interplay analysis. *2013 First IEEE Working Conference on Software Visualization (VISSOFT)* 00 (2013), 1–10. https://doi.org/doi.ieeecomputersociety.org/10.1109/VISSOFT.2013.6650534

[7] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. 2012. *Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir*. Springer Berlin Heidelberg, Berlin, Heidelberg, 79–91. https://doi.org/10.1007/978-3-642-31476-6_7

[8] NWChemEx. 2017. (May 2017). https://www.pnnl.gov/science/highlights/highlight.asp?id=4411

[9] US Department of Energy Office of Science / NNSA. 2016. (November 2016). https://exascaleproject.org/ecp_co-design_centers/

[10] ParaProf. 2017. (July 2017). https://www.cs.uoregon.edu/research/tau/docs/newguide/bk01pt02.html

[11] Sameer S Shende and Allen D Malony. 2006. The TAU parallel performance system. *The International Journal of High Performance Computing Applications* 20, 2 (2006), 287–311.

[12] Vampir. 2017. (July 2017). https://www.vampir.eu/tutorial/manual