Question 3:

3. (30) Performance analysis of MPI applications has been an active area of research. There have been many performance tools developed to support performance MPI applications. Please identify two of these frameworks and compare and contrast the capabilities of the toolsets you have selected. Make sure to cite all your resources. Please do not copy text out of user guides when you discuss the frameworks.

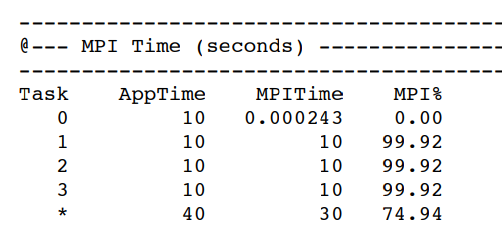
To determine potential frameworks to choose from, I reviewed a paper titled, [Performance Analysis Tools for MPI Application and their use in Programming Education.](https://dl.acm.org/doi/pdf/10.1145/3578245.3584358)

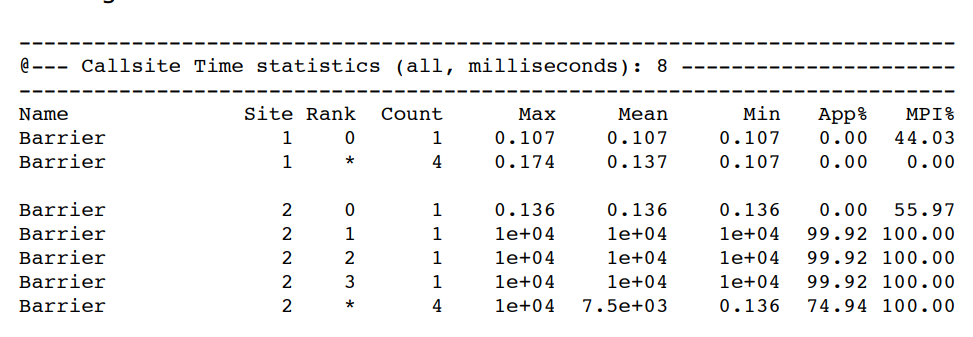
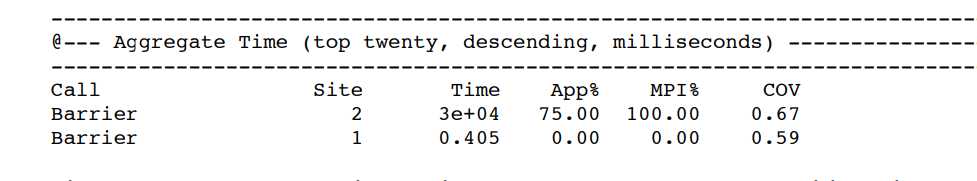
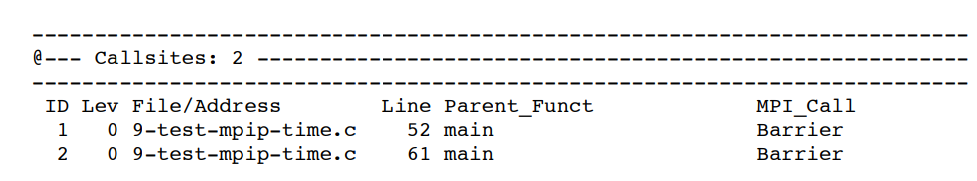
Framework 1) mpiP

MpiP is an open-source tool produced at Lawrence Livermore National Laboratory.

The paper above highlights mpiP, a lightweight profiling library that provides statistical details related to MPI functions and performance. Due to its minimal overhead, it only collects stats and adds them into a report on call sites, run-time, and latency associated with MPI functions. In a 2006 paper, [MPI Performance Analysis Tools on Blue Gene/L,](https://dl.acm.org/doi/pdf/10.1145/1188455.1188583) execution overhead was less than 3%, and provided useful cumulative data.

Using the mpiP, which takes information through the MPI profiling layer, it does not require recompilation. First, compile the program with mpicc, using several commands including –lbfd, -lmpiP, -liberty, and –libunwind, and then run the executable to produce results into a file.



Images Above: Example output of mpiP, providing information related to the application time in MPI from initialization to finalization. The MPI call site locations are also identified and listed, as well as the MPI calls that consume the most overall time in the program, with details as to the variation of normal process time and how much total time the call takes related to the whole application.

MpiP profiling can also be limited to certain regions of an application with MPI\_Pcontrol(), which takes in either a 0 or 1 to renable or disable profiling statistics collection.

Sources:  
[MPI Tool Libraryl](https://github.com/LLNL/mpiP)

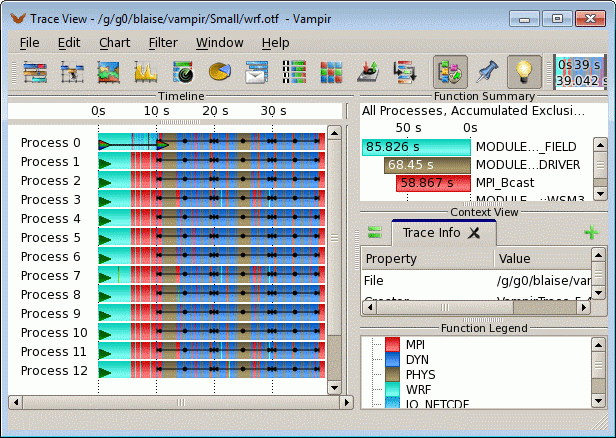
[MPI Performance Analysis Tools on Blue Gene/L,](https://dl.acm.org/doi/pdf/10.1145/1188455.1188583)

[MpiP: Lightweight, Scalable MPI Profiling](http://gec.di.uminho.pt/Discip/MInf/cpd1415/PCP/MPI/mpiP_%20Lightweight,%20Scalable%20MPI%20Profiling.pdf)

[Research on MpiP Performance based on Parallel Programs](https://ieeexplore.ieee.org/abstract/document/8663759)

Framework 2) Vampir

Vampir is a trace-based visualization tool for performance analyzation of MPI programs in a chronological manner. To understand how it visualizes messaging between processes, below is an image of the tool’s GUI:



The tracing technique instruments a program before execution to collect a sequence of events, as shown by this GUI. This helps programmers identify and address inefficiencies in MPI programs across a timeline. Since MPI standards include a profiling interface, Vampir can use a routine wrapper to generate traces. However, more commonly, Vampir needs to be installed alongside a tracing tool software like Score-P. These can be paused and resumed with control directives. The primary value of the tool is its visualization, which needs to be understood by users in order to maximize potential bottlenecks while programming.

The program must be instrumented at compile time, using a trace software or Vampir’s trace, and then run as instrumented. As shown by the image, a timeline view appears, illustrating the locations of message-passing and summarizing execution time.

More specifically, the Global\_Display/Node\_Style displays processors as boxes. Boxes are arranged and partitioned based on upper and lower, where the upper section indicates the time for each activity and the lower part mentions current activity for each process.

To best pivot after viewing this, it is important to determine potential gaps that are elongated in the timeline, potentially resulting from delays in synchronization. Additionally, frequently transferring inconsequential information can increase message overhead, which can be identified through Vampir.

Sources:

[Vampir Tool](https://vampir.eu/)

[VAMPIR: Visualization and analysis of mpi resources](https://juser.fz-juelich.de/record/189233/files/ib-9528.pdf?subformat=pdfa)

[Parallel Performance Engineering using Score-P and Vampir](https://research.spec.org/icpe_proceedings/2023/companion/p121.pdf)

[Interactive Trace Analysis with Vampir](https://www.olcf.ornl.gov/wp-content/uploads/2021/11/03-vampir.pdf)

Compare & Contrast)

MpiP and Vampir are both useful tools for analyzing inefficiencies in programs utilizing a message passing interface. MpiP is a much more valuable tool for low-overhead, high-level performance reporting about MPI function calls. Due to its lightweight capabilities, it does not greatly affect the performance of applications. However, it lacks an event timeline and doesn’t analyze full computational performance.

On the other hand, Vampir is a great visual tracing tool for timeline perspective on a program using MPI. This tool is more adequate for detailed temporal analysis, helping to optimize parallel execution with visuals to show message passing. It can also show CPU utilization and memory usage, which enables effective analyzation of HPC workloads. Large traces, however, can introduce significant overhead. Additionally, the GUI can be somewhat complex and is not the most beginner friendly, whereas mpiP’s statistics are easily readable and understandable.