

Capturing Petascale Application Characteristics with the Sequoia Toolkit

J. S. Vetter^{a *}, N. Bhatia^a, E. M. Grobelny^b, P. C. Roth^a

^aFuture Technologies Group, Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831 USA

^bHigh-Performance Computing and Simulation Research Laboratory, Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA

Characterization of the computation, communication, memory, and I/O demands of current scientific applications is crucial for identifying which technologies will enable petascale scientific computing. In this paper, we present the Sequoia Toolkit for characterizing HPC applications. The Sequoia Toolkit consists of the Sequoia trace capture library and the Sequoia Event Analysis Library, or SEAL, that facilitates the development of tools for analyzing Sequoia event traces. Using the Sequoia Toolkit, we have characterized the behavior of application runs with up to 2048 application processes. To illustrate the use of the Sequoia Toolkit, we present a preliminary characterization of LAMMPS, a molecular dynamics application of great interest to the computational biology community.

1. Introduction

The Future Technologies Group [1] at Oak Ridge National Laboratory performs basic research in core technologies for future generations of high-end computing systems. An important aspect of our work involves the characterization of existing scientific applications to understand their demands for computation, communication, memory, and I/O. By understanding the demands of current applications, we hope to gain insight regarding which technologies will best satisfy the needs of those applications in the future.

Building on experience with the mpiP profiling library [2], we have developed the Sequoia Toolkit to support characterization of applications that use the Message Passing Interface [3] for communication. The Sequoia Toolkit includes an event tracing facility and scripts to process and manage event trace files. The Sequoia tracing facility captures performance data for computation and communication activity in an application. Sequoia records events for each MPI call made by the application, including data such as the number of bytes transferred. For computation, Sequoia uses the Performance Application Programming Interface [4] (PAPI) to collect hardware counter metrics describing the computation that occurs between successive calls to MPI functions.

Using scripts provided with the Sequoia Toolkit, event trace files can be analyzed to compute ratios for characterizing application performance, such as the ratio of the number of floating point operations to the amount of data transferred between MPI tasks. Sequoia trace files can also be used as input to architectural simulators; the Sequoia Toolkit includes a simple simulator that models the system interconnect using the LogP [5] model. We envision using Sequoia event trace files as input for performance modeling approaches such as Modeling Assertions [6]. Because the process

*This research was sponsored by the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC. The Petascale Execution Time Evaluation project is supported by the office of Science of the U.S. Department of Energy. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

of using trace files for simulation is restricted to the processor count of the original trace file parameters, we are actively developing a communication extrapolation tool that can generate synthetic communication traces directly from empirical trace files, which were captured at smaller processor counts.

To facilitate development of tools that use Sequoia event trace files, we are developing the Sequoia Event Analysis Library, or SEAL. SEAL is a C++ library that provides a tool-independent infrastructure for opening Sequoia event trace files, decoding their contents, and dispatching control to tool-specific functions that process events (e.g., to find the total number of bytes transferred during an application run).

In the next section, we describe the Sequoia event tracing library and SEAL. In Section 3, we discuss some of our current characterization work on MPI applications of interest to Oak Ridge National Laboratory and the United States Department of Energy. We discuss related, existing tracing facilities and analysis tools in Section 4, and summarize our work on the Sequoia Toolkit in Section 5.

2. The Sequoia Toolkit

The Sequoia Toolkit includes an event tracing library, scripts for management and basic processing of Sequoia event trace files, and SEAL, the Sequoia Event Analysis Library. In this section, we describe these Sequoia Toolkit components.

2.1. The Sequoia Event Tracing Library

The Sequoia event tracing library captures the behavior of a running application by writing *event records* to event trace files. Sequoia event trace files are ASCII text files containing a chronological sequence of event records describing computation and communication events that occurred during the run of a target application. Although Sequoia includes scripts for basic trace file management and processing, Sequoia trace files are ASCII text files so that common text-processing utilities, like `sed` and `awk`, can be used to manipulate them. Sequoia generates one trace file per MPI process, but toolkit scripts can be used to merge these files into a composite event trace file. Metadata describing the format of event trace records is kept in a separate file called `events.stdef`. A portion of the `events.stdef` is shown in Listing 1. This file can be modified to enable the processing of new event types or to extend existing event types. That figure shows the file’s header, several definitions of built-in MPI types, communicators, and operations, and definitions for a few of the functions in the MPI API.

The Sequoia tracing library operates on the MPI profiling interface. Using this interface, the Sequoia library intercepts calls to each MPI function and records information such as the sender or receiver of the operation and the number of bytes transferred. Because the MPI profiling interface is defined as part of the MPI-1 and MPI-2 standards, Sequoia can be used with any compliant MPI implementation. Also, due to the interface’s design, no source code modifications are required to use Sequoia to trace a MPI application. The application does need to be re-linked against the Sequoia library, and small source code modifications may be used to control the event trace volume and to support fine-grained characterization of the computation between communication events.

Sequoia event records can be broadly classified into three categories:

- *MPI event records*—records that describe entry to or exit from a MPI function;
- *Computation event records*—records that describe computation between successive MPI function calls; and
- *Special event records*—records that describe tracing library meta events.

Listing 1: A portion of a Sequoia `events.stdef` file.

```

@ SEQUOIA
@ Command :
@ Version      : 0.9
@ Build date   : Jul 19 2005, 17:10:29
@ ST env var   : [null]
@ Final Trace Dir : .
@ MPI Type     : MPI_BYTE 2 1
@ MPI Type     : MPI_CHAR 4 1
@ MPI Type     : MPI_DOUBLE 14 8
...
@ MPI Comm     : MPI_COMM_WORLD 0 1
@ MPI Comm     : MPI_COMM_SELF 1 1
@ MPI Comm     : MPI_COMM_NULL -1 0
@ MPI Op       : MPI_MAX 0
@ MPI Op       : MPI_MIN 1
...
@ EDEF ts rank thread pcid Init X hostname size starttime
@ EDEF ts rank thread pcid Finalize E
@ EDEF ts rank thread pcid Comp E
@ EDEF ts rank thread pcid Comp X cycles insts loads stores flops fmas ipc
@ EDEF ts rank thread pcid Send E  count datatype dest tag comm
@ EDEF ts rank thread pcid Send X  count datatype dest tag comm
...

```

Table 1

Fields common to all Sequoia trace file event records.

Name	Explanation
ts	Event timestamp
rank	MPI rank in which event occurred
thread	ID of the thread in which event occurred (<i>reserved</i>)
pcid	Program counter id (<i>reserved</i>)
type	Type of event
mode	Whether record specifies a transition <i>into</i> a state ('E') or <i>out of</i> a state ('X')

These three categories are described in more detail in the remainder of this section. Regardless of an event record's category, certain fields are present at the start of each record. These fields are described in Table 1. The values of the `type` and `mode` fields determine the type and number of the remaining fields in the event record. In effect, these two fields allow Sequoia-based tools to look up the correct event record definition from the `events.stdef` file. The `thread id` and `pcid` fields are reserved for future use.

2.1.1. MPI Event Records

Sequoia MPI event records indicate an entry into or exit from a MPI library routine. Each MPI event record contains data relevant to the specific MPI function that was called. For instance, the event record for the `MPI_Init` call includes fields for the name of the host running the MPI process, the total number of MPI processes in the run, and the start time of the application. The record for

Table 2

Computation event record fields, depending on the value of the `ST` environment variable.

Value	Fields
c 0	cycles instructions loads stores fp-ops fp-mas ipc
c 1	cycles instructions loads stores <i>reserved reserved</i> ipc
c 2	cycles instructions <i>reserved reserved</i> fp-ops fp-mas ipc

Table 3

Explanation of computation event record fields.

Name	Explanation
cycles	Number of CPU cycles in the computation interval
instructions	Number of instructions executed
loads	Number of load instructions executed
stores	Number of store instructions executed
fp-ops	Number of floating point operations performed
fp-mas	Number of floating point multiply-add instructions performed
ipc	Instructions executed per cycle

the `MPI_Send` call includes fields for the number of items sent, the type of the data sent (using an integer encoding specified in the `events.stdef` file), the receiving process' MPI rank, and the message tag and MPI communicator used for the operation.

2.1.2. Computation Event Records

Sequoia event records for computation capture hardware performance counter data for the computation that occurred in a MPI process between successive communication events. By default, Sequoia generates one pair of computation event records that describe the computation between successive calls to routines in the MPI interface. With small modifications to the application source code as described in Section 2.1.3, hardware counter data can be recorded with finer granularity. A computation entry event record is generated when a process exits any MPI routine. When the process makes its next MPI routine call, Sequoia reads the hardware counters and generates a computation exit record that captures the values.

For portability, Sequoia uses the PAPI library for obtaining the hardware counter data. Sequoia also resets the hardware counters after sampling, so that the values in computation event records indicate differences between samples instead of cumulative values. Sequoia uses the `ST` environment variable to specify which hardware counters it will sample for computation event records. If `ST` is not set, "c 0" is the default value. Table 2 shows the event record fields used for the three recognized `ST` values; Table 3 explains the possible event record fields.

2.1.3. Special Event Records

Sequoia can be used to trace MPI applications without modifying their source code, however, in this mode, Sequoia may generate trace files of unmanageable size, especially when tracing long-running applications that transition frequently between computation and communication operations. However, given the regular structure of many scientific applications, insight into the behavior of scientific applications can often be obtained by observing a small number of iterations of an application's main loop (e.g., a small number of simulation time steps in a climate simulation code).

To enable the user to control the trace event record volume, Sequoia tracing can be disabled and then re-enabled as an application process executes. Sequoia event tracing is disabled when the `MPI_Pcontrol` function is called with an argument of 0, and re-enabled when `MPI_Pcontrol` is called with an argument of 1. The action of disabling and re-enabling Sequoia tracing is recorded with a special event trace record without parameters.

By default, Sequoia generates one pair of computation event records for all computation between successive MPI calls in a process. However, the computation that occurs in that interval may not be uniform. To allow users to trace application computation at finer granularity, Sequoia can insert `Mark` event records into the event trace. Inserting a `Mark` event record into the event trace terminates the current computation interval (hence sampling the hardware performance counters as described in Section 2.1.2) and starts a new computation interval. To generate `Mark` event records, the user inserts `MPI_Pcontrol` calls into the application source code with an argument greater than 1. This argument value is recorded as part of the `Mark` event record and can be considered to be a computation phase identifier by analysis tools.

2.2. The Sequoia Event Analysis Library

Tools that consume Sequoia event trace files as input can have many purposes, but they each share some similar operations: they read Sequoia trace file records, decode the records, and dispatch control to other parts of the code that will process the records. Because this functionality is not specific to any one tool, it can be factored into a library that is shared by many tools. We are developing the Sequoia Event Analysis Library, or SEAL, that provides a tool-independent infrastructure for reading Sequoia event trace files. SEAL provides a C++ API and implements a tool framework for opening Sequoia event trace files, reading event records in order from the trace file, and dispatching control to tool-specific functions that operate on the current event record. A SEAL-based tool need only provide the tool-specific event record processing functions.

3. Application Characterization

The Sequoia Toolkit is under active development, but we have already used it to trace and characterize several scientific applications of interest to Oak Ridge National Laboratory and the U.S. Department of Energy, including:

- *AMBER*—a suite of programs that allow users to perform molecular dynamics operations, particularly on bio-molecules.
- *GYRO*—a simulation of microturbulence in tokamak fusion reactors.
- *HYCOM*—an ocean general circulation model.
- *LAMMPS*—a classical molecular dynamics simulation code.
- *POP*—a 4D ocean modeling code.
- *SPPM*—a 3D gas dynamics simulation code.
- *SWEEP3D*—a neutron transport code.
- *UMT2K*—a 3D photon transport code.

For example, we have used the Sequoia Toolkit to characterize the computation and communication demands of the LAMMPS molecular dynamics code. The Sequoia trace files were gathered during a run on the IBM p690+ at Forschungszentrum Jülich in Germany. Sequoia tracing was enabled for two simulation time steps. Using simple post-processing scripts to accumulate performance data from the Sequoia event trace file, we generated the application characterization shown

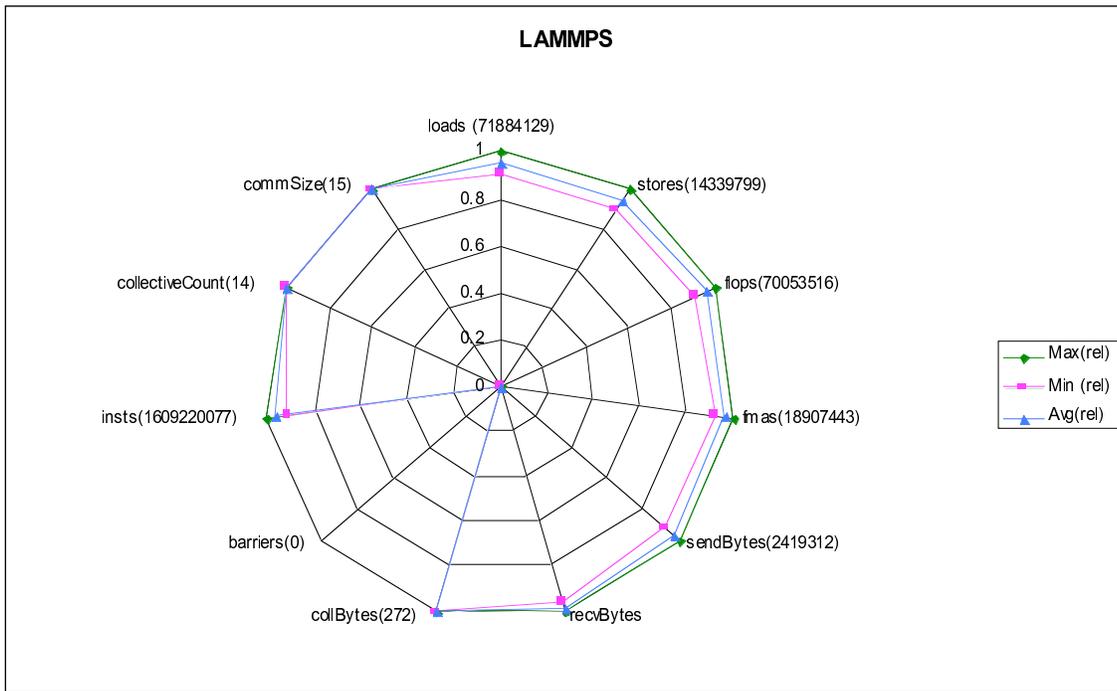


Figure 1. Characterization of LAMMPS for 16 tasks using data collected using the Sequoia Toolkit. Values are normalized using the maximum value observed for any task during the run. Axis labels include the maximum value.

in Figures 1–3. For each performance metric in Figure 1, we plot the minimum, maximum, and average value for each computation or communication interval represented in the event trace file (i.e., for each pair of MPI event records or computation event records in the event trace file for each application process). To improve readability of the chart, all values were normalized to the interval $[0,1]$. We characterize LAMMPS using several common performance ratios in Figures 2 and 3. In Figure 2, we compare LAMMPS running with 16, 128, and 512 processes for one user-defined phase of the program’s run. The axes show the number of point-to-point communication operations (p2p), bytes transferred per floating point operation for point-to-point communication (p2bpf), number of collective communication operations (coll), bytes transferred per floating point operation for collective operations (collbpf), loads and stores per floating point operation (lspf), and instructions per cycle (ipc). For readability, the values are normalized using performance data across all user-defined phases. Figure 3 shows the same data but normalized using only values from the phase being analyzed. The characterization shows that LAMMPS places significantly different demands on a computing system at the three process counts we considered.

4. Related Work

Event tracing is a well-established approach for collecting performance data that describes an application’s behavior, and trace analysis tools are often developed in conjunction with a trace capture library. Examples of current performance analysis tools that adopt this approach include Vampir and Vampirtrace [7], the KOJAK [8] automated performance analysis tool with its EPILOG tracing facility, and the Tuning and Analysis Utilities [9] (TAU) that has its own tracing facility but also can operate on Vampir event trace files (or can convert them to a format it can recognize). We are

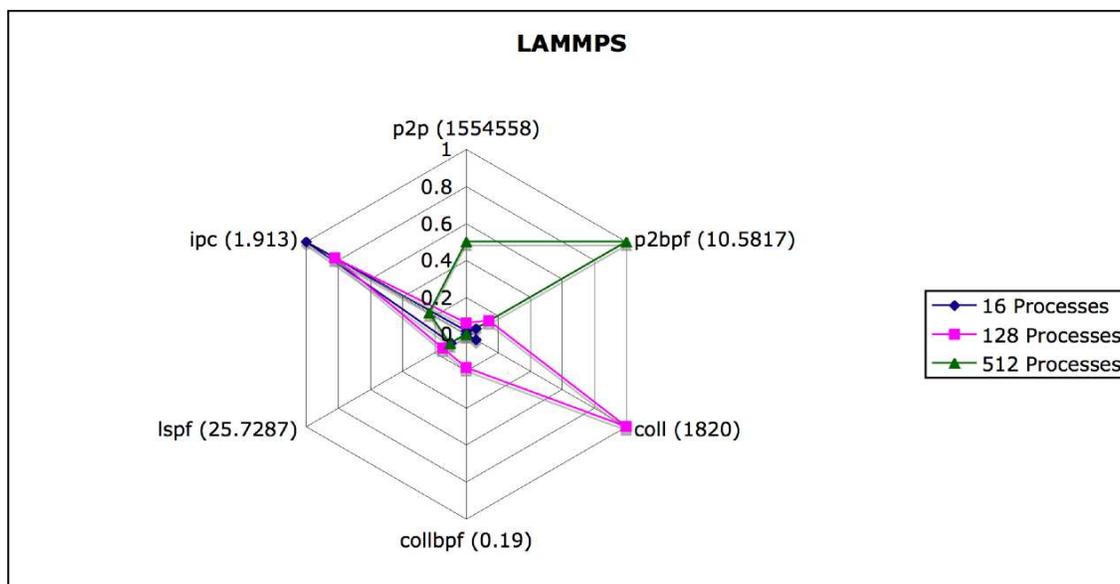


Figure 2. Characterization of LAMMPS phase 5
Metric values are normalized using values from all phases.

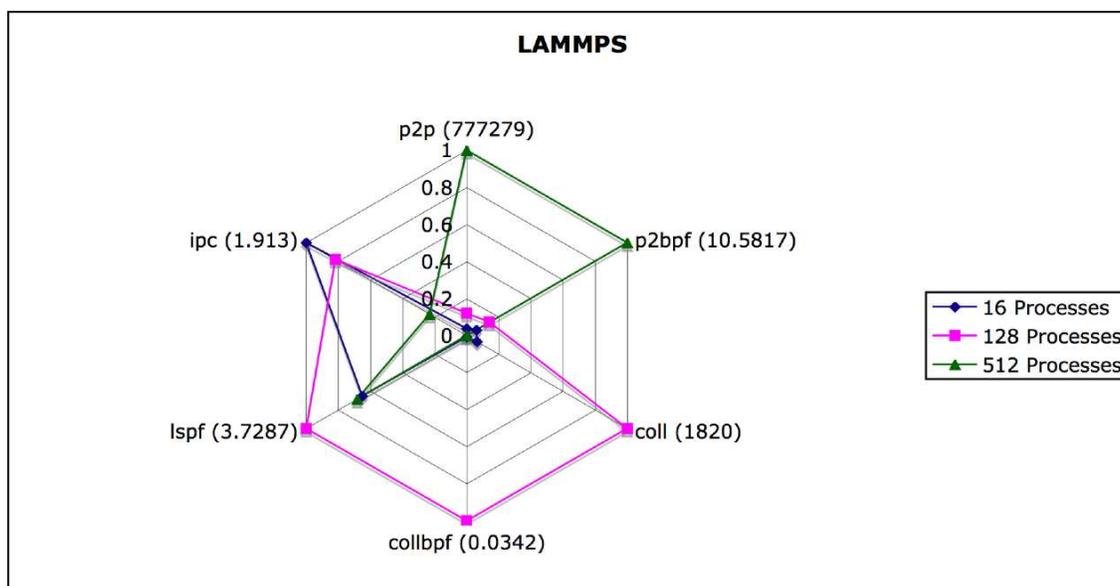


Figure 3. Characterization of LAMMPS phase 5, normalized within that phase.
This figure shows the same data as Figure 2, but the values are normalized using the phase 5 values only.

actively working on the integration of the Sequoia Toolkit with some of these tools. For example, we are currently working on a converter between the Sequoia event trace file format and EPILOG, to allow analysis of Sequoia event traces by the KOJAK tool set.

5. Summary

As a starting point for understanding the demands of future applications on future computing architectures, the Future Technologies Group at Oak Ridge National Laboratory seeks to understand the behavior of current scientific applications. To support this effort, we are developing the Sequoia Toolkit. Sequoia includes an event tracing library that records data about both MPI communication events and the computation that occurs between those communication events. Sequoia also includes scripts for management and basic analysis of Sequoia event trace files. Recently, we have started work on the Sequoia Event Analysis Library, a C++ library that facilitates the development of tools for analyzing Sequoia event traces. We have traced and analyzed scientific applications from a variety of domains, including computational biology, materials science, nuclear fusion, and climate modeling.

We continue to refine and extend the Sequoia Toolkit to make application characterization easier for the end user, to support new analyses, and to improve integration with existing analysis tools. For instance, we are modifying Sequoia to sample hardware counter data for communication entry events in addition to communication exit events, to support analysis of the internal computational requirements of MPI routines. Also, we are actively working on a utility that converts Sequoia event trace files to formats that enable analysis and visualization using the popular TAU and Vampir tools.

References

- [1] ORNL Future Technologies Group. <http://www.csm.ornl.gov/ft/>, October 2005.
- [2] J. S. Vetter and M. O. McCracken. Statistical scalability analysis of communication operations in distributed applications. In *PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 123–132, New York, NY, USA, 2001. ACM Press.
- [3] W. Gropp, R. Thakur, and E. Lusk. *Using MPI-2: Advanced Features of the Message Passing Interface*. MIT Press, Cambridge, MA, USA, 1999.
- [4] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, November 2000. IEEE Computer Society.
- [5] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, E. E. Santos, K. E. Schauer, R. Subramonian, and T. von Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, 1996.
- [6] S. R. Alam and J. S. Vetter. Multiresolution performance modeling with modeling assertions, in submission 2005.
- [7] W.E. Nagel, A. Arnold, M. Weber, H.C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.
- [8] B. Mohr and F. Wolf. KOJAK - a tool set for automatic performance analysis of parallel programs. In *Euro-Par*, volume 2790 of *Lecture Notes in Computer Science*, pages 1301–1304, Heidelberg, 2003. Springer-Verlag.
- [9] S. Shende, A. D. Malony, J. Cuny, P. Beckman, S. Karmesin, and K. Lindlan. Portable profiling and tracing for parallel, scientific applications. In *SPDT '98: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*, pages 134–145, New York, NY, USA, 1998. ACM Press.