A Platform for Scalable Parallel Trace Analysis

Markus Geimer[1], Felix Wolf[1], Andreas Knüpfer[2], Bernd Mohr[1], and Brian Wylie[1]

[1]John von Neumann Institute for Computing (NIC)
FZJ, 52425 Jülich, Germany
{m.geimer, f.wolf, b.mohr, b.wylie}@fz-juelich.de

[2]ZIH, TU Dresden, Germany
andreas.knuepfer@tu-dresden.de

Abstract Automatic trace analysis is an effective method of identifying complex performance phenomena in parallel applications. To simplify the development of complex trace-analysis algorithms, the EARL library interface offers high-level access to individual events contained in a global trace file. However, as the size of parallel systems grows further and the number of processors used by individual applications is continuously raised, the traditional approach of analyzing a single global trace file becomes increasingly constrained by the large number of events. To enable scalable trace analysis, we present a new design of this interface that accesses multiple local trace files in parallel while offering means to conveniently exchange events between processes. This article describes the modified view of the trace data as well as related programming abstractions provided by the new interface and discusses its application in performance analysis.

1 Introduction

Event tracing is a well-accepted technique for post-mortem performance analysis of parallel applications. Time-stamped events, such as entering a function or sending a message, are recorded at runtime and analyzed afterwards with the help of software tools. For example, graphical trace browsers, such as VAMPIR [1], allow the fine-grained investigation of parallel performance behavior using a zoomable time-line display.

However, in view of the large amounts of data usually generated, automatic off-line trace analyzers, such as EXPERT [2], can provide the user with relevant information more quickly by automatically searching traces for complex patterns of inefficient behavior and quantifying their significance. In addition to being faster than a manual analysis performed using the aforementioned trace browsers, this approach is also guaranteed to cover the entire event trace and not to miss any pattern instances.

To simplify the analysis logic incorporated in EXPERT, it has been designed on top of EARL [3], a high-level interface to access individual events from a single global trace file. As opposed to a low-level interface that allows reading individual event records only in a sequential manner, EARL offers random access to individual events. Not to restrict trace-file size, EARL assumes access locality allowing it to buffer the context of recent accesses in main memory while reading events outside this context from file. In

addition, to support the identification of pattern constituents, EARL provides abstractions representing execution state information at the time of a given event as well as links between related events. Unfortunately, sequentially analyzing a single and potentially large global trace file does not scale to applications running on thousands of processors. Even if access locality is exploited, the amount of main memory might not be sufficient to store the current working set of events. Moreover, the amount of trace data might not even fit into a single file, which already suggests to perform the analysis in a more distributed fashion.

To facilitate scalable trace analysis also for very large systems and applications running on them, we have designed a parallel trace-data interface PEARL as a building block for parallel trace analysis algorithms and tools. In this article, we describe the modified view of the trace data in combination with programming abstractions representing this view. We start our discussion with a review of related work in Section 2, followed by a description of the serial interface in Section 3. In Section 4, we detail the programming abstractions offered by the new parallel interface. Finally, in Section 5, we outline the intended usage as a framework for implementing automatic parallel trace analysis.

2 Related Work

A number of approaches address scalable trace analysis: The frame-based SLOG trace-data format supports scalable visualization. Dynamic periodicity detection in OpenMP applications prevents redundant performance behavior from being recorded in the first place. Important to our particular approach has been the distributed trace analysis and visualization server VNG, which already provides parallel trace access mechanisms, albeit targeting a "serial" human client in front of a graphical trace browser as opposed to fully automatic and parallel trace analysis. A tree-based main memory data structure for event traces called CCCG allows potentially lossy compression of trace data while observing previously specified deviation bounds. We are considering CCCGs as an alternate base data structure for our parallel analysis platform. A more detailed description of the above approaches including references can be found in [4].

3 EARL

EARL (Event Analysis and Recognition Library) is a C++ class library that offers a high-level interface to access event traces of MPI, OpenMP, or SHMEM applications. In the context of EARL, an event trace is a single global trace file that includes events from all processes or threads in chronological order. The user is given random access to individual events, plus execution state information at the time of a given event in the form of event sets describing a particular aspect of this state. Such state information can describe the current call stack, messages in transit, or the progress of collective operations. Based on this state information, EARL also provides links between related events, which are called pointer attributes, for example, to allow the easy location of the message send event belonging to a given receive event.

The intended trace analysis process supported by EARL is a sequential traversal of the event trace from beginning to end. As the analysis progresses, EARL updates the

execution-state information and calculates pointer attributes for the most recent event being read, which always point backwards to avoid a costly look-ahead. To make the trace analysis more efficient, EARL buffers the context of the current event so that events within this context can be accessed from main memory. This context includes the last n events (i.e., the history) plus all related execution-state information. To avoid rereading from the very beginning in the case that an event outside the context is requested, EARL stores the complete execution state information in regular intervals in so-called bookmarks. The history size and the bookmark distance can be flexibly configured. However, since these parameters have a significant performance impact with respect to memory consumption and the number of required file accesses and, in addition, these effects are highly application dependent, finding an optimal choice of parameters is a non-trivial task.

4 Parallel Access to Trace Data

In our new parallel design, which we call PEARL, we no longer assume a single global trace file. Instead, PEARL operates on multiple process-local trace files. In addition, one set of global definitions for static program entities, such as code regions, is required in a separate file. Along the same lines, the parallel analysis itself will be a parallel program using as many CPUs as have been allocated for the target application. In practice, we intend to perform the parallel analysis right after the target application finishes as part of the same job execution. For simplicity, our initial version will support only single-threaded MPI-1 applications, but we plan to extend our approach to alternate programming models, such as OpenMP or MPI-2.

Every analysis process has access to one local trace represented by a C++ class `LocalTrace` and to the global definitions represented by another class `GlobalDefs`. We assume that the internal representation of a local trace is smaller than the memory available to a single process on a parallel machine so that the entire local trace can be kept in main memory, relaxing the aforementioned limitations resulting from strict forward analysis. That is, PEARL provides performance-transparent random access to individual events plus local execution state information. Pointer attributes can point backward and forward, but not to non-local events. Events can be accessed through a class `Event` which provides access to all event attributes. To navigate through the local trace, the class also exposes iterator semantics available through simple `prev()` and `next()` methods and pointer attributes for more sophisticated tasks, such as call stack traversal. In this way, it is possible, for example, to reach enclosing enter and exit events of a given communication event to determine the duration of the entire communication operation. The return values of pointer attributes are always new `Event` (iterator) objects that can be subject to further navigation operations.

To facilitate the cross-process analysis of communication patterns, PEARL provides means to conveniently exchange one or more events between processes. Remote events received from other processes are represented by a class `RemoteEvent`, which provides identical functionality but without iterator semantics. There are generally two modes of exchanging events: point-to-point and collective. Point-to-point exchange allows a remote event to be created with arguments specifying the source process, a communicator, and a message tag. The source process has to invoke a corresponding send

method on the local event object to be transferred. The exchange of multiple events can be accomplished in one batch by collecting local events in an object of class `EventSet` on the sender's side and instantiating an object of class `RemoteEventSet` on the receiver's side by supplying message parameters to the constructor. Each event in the set is identified by a numeric identifier that can be used to assign a role to each member of the set, for example, to distinguish a particular constituent of a pattern. Collective event exchange has the form of a reduce operation that identifies the earliest or latest (min or max) event across a number of processes and instantiates a corresponding remote event at all participating processes. We are currently investigating methods to optimize event exchange in cases where the exact number and types of events to be transferred is difficult to obtain or completely unknown at the receiver's side.

5 A Framework for Parallel Trace Analysis

The strength of EARL has been the provision of truly parallel abstractions that allow the user to access higher-level events, such as messages and collective operations, which requires the ability to match corresponding events across processes. In the case of our parallel interface, this is more difficult, since matching incurs costly communication. To minimize this communication, the intended usage of PEARL is that of a *replay-based* analysis. The central idea behind replay-based analysis is analyzing a communication operation using an operation of the same type. For example, to analyze a point-to-point message, the related events are exchanged in point-to-point mode. To accomplish this, the new analysis process traverses local traces in parallel and meets at the synchronization points of the target application by replaying the original communication. We are currently investigating the scalability properties of the replay in comparison to those of the target application.

References

1. Nagel, W., Weber, M., Hoppe, H.C., Solchenbach, K.: VAMPIR: Visualization and Analysis of MPI Resources. Supercomputer **63, XII** (1996) 69–80
2. Wolf, F., Mohr, B., Dongarra, J., Moore, S.: Efficient Pattern Search in Large Traces through Successive Refinement. In: Proc. of the European Conference on Parallel Computing (Euro-Par), Pisa, Italy, Springer (2004)
3. Wolf, F., Mohr, B.: EARL - A Programmable and Extensible Toolkit for Analyzing Event Traces of Message Passing Programs. In: Proc. of the 7th International Conference on High Performance Computing and Networking Europe (HPCN). Volume 1593 of Lecture Notes in Computer Science., Amsterdam, The Netherlands, Springer (1999) 503–512
4. Wolf, F., Freitag, F., Mohr, B., Moore, S., Wylie, B.: Large Event Traces in Parallel Performance Analysis. In: Proc. of the 8th Workshop 'Parallel Systems and Algorithms' (PASA). Lecture Notes in Informatics, Frankfurt/Main, Germany, Gesellschaft für Informatik (2006)