Scalable Compression and Replay of Communication Traces in Massively Parallel Environments

Michael Noeth\textsuperscript{1}, Jaydeep Marathe\textsuperscript{1}, Frank Mueller\textsuperscript{1}, Martin Schulz\textsuperscript{2}, Bronis R. de Supinski\textsuperscript{2}
\textsuperscript{1} Department of Computer Science, North Carolina State University, Raleigh, NC
\textsuperscript{2} Lawrence Livermore National Laboratory, CASC, Livermore, CA

Abstract

Characterizing the communication behavior of large-scale applications is a difficult and costly task due to code and system complexity as well as the time to execute such codes. An alternative to run actual codes is to gather their communication traces and then replay them, which facilitates application tuning and future procurements. While past approaches lacked lossless scalable trace collection, we contribute an approach that provides near constant-size communication traces regardless of the number of nodes while preserving structural information. We introduce intra- and inter-node compression techniques of MPI events and present results of our implementation. Given this novel capability, we discuss its impact on communication tuning and beyond.

Structure

The poster will be divided into four sections. Section 1 motivates the problem and gives an overview of the tool’s design. Section 2 and 3 detail intra- and inter-node trace compression. Section 4 presents results. The poster will provide additional results omitted here due to space constraints.

1 Introduction/Motivation

Scalability is one of the main challenges to Peta-scale computing. One central problem lies in a lack of scaling of communication. However, understanding the communication patterns of complex large-scale scientific applications is non-trivial. Instead of source-code analysis, we promote a trace-driven approach to analyze MPI communication. While past approaches fail to gather full traces for hundreds of nodes in a scalable manner or only gather aggregate information, we have designed a framework that extracts full communication traces of near constant size regardless of the number of nodes while preserving structural information of the program. In addition, compressed traces can be replayed on-the-fly independent of the original application, which aids performance tuning of MPI communication and facilities projections of network requirements for future large-scale procurements. Our framework (Fig. 1) utilizes the MPI profile layer to intercept MPI calls during application execution. Profiling wrappers trace which MPI function was called along with call parameters within each node. Such intra-node information (task-level) is compressed on-the-fly. Upon application termination, inter-node compression is triggered over all nodes resulting in a single trace file that preserves structural information suitable for lossless replay.

2 Intra-Node/Task-Level Trace Compression

Lossless, yet space-efficient trace compression requires that the structure of events is preserved, \textit{i.e.}, repetitive MPI events in loops with identical parameters should only require near constant size. We exploit regular section descrip-
tion (RSDs) for single loops to express such MPI events in a loop while power-RSDs (PRSDs) are utilized to recursively specify RSDs nested in multiple loops. MPI events may occur in inner-most or outer loops alike in PRDSs. For example, the tuple \( RSD_1: < 100, MPI_{Send1}, MPI_{Recv1} > \) denotes a loop with 100 iterations of alternating send/receive calls with identical parameters (omitted here), and \( PRSD_1 : < 1000, RDS_1, MPI_{Barrier1} > \) denotes 1000 invocations of (a) the former loop (RDS1) followed by (b) a barrier. To effectively compress location-specific parameters, communication endpoints are encoded relative to the task ID of the local node, i.e., as \( \pm c \) for a constant \( c \). Fig. 2 depicts a 2D stencil where 9 communicates with relative neighbors \(-4, -1, +1\) and \(+4\). Furthermore, indeterministic repetitions (e.g., MPI\_waitsome) depending on the number of completed calls are squashed into a single event, which preserves compression capabilities while exploiting specific MPI semantics, even during replay. Handles for asynchronous MPI calls are further recorded in a buffer, which is indexed relative to the last element at asynchronous calls for later cross-node compression.

3 Inter-Node/Cross-Node Trace Compression

Upon application completion, local traces are combined into a single global trace. To guarantee scalability, cross-node compression occurs step-wise and in a bottom-up fashion over a tree. Events and structures (RDS / PRDSs) of nodes are merged when events, parameters, structure and iteration counts match. Task IDs are then encoded as RSDs themselves to specify which subset of nodes participated in some set of events. This allows us to concisely represent cross-node similarities, even for stencil codes. Assuming non-wrap-around communication for the 2D stencil in Figure 2, interior nodes 5, 6, 9 and 10 have an identical communication pattern. Any pair of nodes between corners on the boundary as well as any corner node also have a unique pattern resulting in nine different patterns to record for 2D stencils, regardless of the number of nodes. This approach makes cross-node compression feasible and results in a single near constant-size trace file, which is by far more efficient than storing per-node trace files and later consolidating them.

4 Experimental Results

Fig. 3 depicts the memory requirements (log-scale) of the trace per node for a stencil benchmark and a production-scale code (Raptor, 27-point stencil) for BlueGene/L, a memory-constrained architecture. The final trace is equivalent in size to the memory used by task 0. Our results demonstrate that near constant-size compression is feasible within 25KB and 38-55MB of memory for the stencil code and Raptor, respectively. The 38-55MB range for Raptor is due to minor inefficiencies of cross-node compression currently being addressed. The resulting traces have been replayed and verified to result in identical communication behavior as the original nodes. This opens tremendous opportunities for rapid prototyping of communication tuning as well as for assessing communication needs of future platforms for large-scale procurements. We are currently pursuing this work and making the framework available.

![Fig. 2: Communic. Endpoint Encoding](image)

![Fig. 3: Memory Usage per Node on BlueGene/L](image)