Performance Evaluation and Visualization with VISPAT

Anna Hondrouidakis\(^1\), Rob Procter\(^1\) and Kesavan Shanmugam\(^2\)

\(^1\) Department of Computer Science, Edinburgh University, Mayfield Road, Edinburgh, EH9 3JZ, UK
\(^2\) Edinburgh Parallel Computing Centre, Edinburgh University, Mayfield Road, Edinburgh, EH9 3JZ, UK

Abstract. A tool for performance analysis of parallel programs implemented using the MPI message passing standard is presented. The paper discusses the way information about program execution is gathered, processed and visualized by the graphical front end of the tool. Emphasis is placed on demonstrating how the tool helps the tuner to reduce the volume of data that has to be examined and to relate the behaviour of the program to the source code.

1 Introduction

Programmers who normally write sequential code often encounter major difficulties when applying data and program decomposition techniques to parallel programs. Communication and synchronization issues among usually large numbers of processes increase the amount of effort involved in parallel programming. To counter these problems, program performance visualization and evaluation environments have been developed and have now become crucial tools in parallel program implementation.

One common technique for performance analysis tools is to collect trace data and then visualize it in order to reveal possible causes of poor performance. The program source code is then modified in the light of the analysis. In practice, however, tuning is much more difficult than this brief sketch suggests.

The amount of trace data produced may be very large, particularly in the case of massively parallel systems. For this data to be useful, the tuner needs appropriate tools to process and present it in a comprehensible form. It can often be very difficult to relate the low-level account of program behaviour provided by the trace data to the source code. Compounding this problem is the tendency for parallel programming environments to provide progressively higher-level programming facilities [1]. Whilst this is of great assistance in the design and coding stages of program implementation, it often makes tuning more difficult, with the tuner having to relate low-level events to increasingly abstract program representations.

This paper describes VISPAT [3], a tool for Visualization of Performance Analysis and Tuning of parallel programs as it has evolved to cater for programs using the MPI standard. Numerous examples of such tools already exist [2, 5].
VISPAT’s originality lies in the emphasis it gives to source code reference and trace data filtering. The following three sections describe respectively how the trace data is gathered, processed and visualized.

2 Programming Environment

MPI is intended to be the standard message passing interface for parallel application and library programming [4]. It caters for point to point communication between pairs of processes and collective operations between groups of processes. Its more advanced features provide among other things, for the manipulation of process groups and their topological structure. A local implementation of the MPI standard has been developed at the Edinburgh Parallel Computing Centre (EPCC).

MPI programs can be linked with the instrumented version of the MPI library. The resulting executable generates a trace file for each process in the program that can subsequently be processed by the Trace Processing Engine (TPE) and visualized by the front end of VISPAT. Each instrumented MPI function corresponds to a phase in the execution of the process calling this function. A user-annotated logical part of the source code is also a phase. Instrumentation can be turned on or off in specific parts of the program so as to perturb the instrumented program as little as possible and to reduce the amount of trace data gathered.

3 Trace Processing Engine

The Trace Processing Engine reads in all the trace files from the processes in the program and analyses them in order to generate the appropriate data structures that are used by the visualization component.

Apart from creating the data structures, TPE has facilities which act as a query mechanism on these data structures. The query mechanism can be used to access the information of the data structures and to drive the visualization displays of VISPAT. The query mechanism also contributes to the extensibility of VISPAT. The information generated by trace data queries can be used (with suitable formatting) by new visualization schemes or by displays incorporated from other performance analysis and visualization tools.

4 Performance Visualization

The design of the visualization component of VISPAT was to a large extent driven by the requirement for source code reference (ref). The control flow of each process’s code consists of time-grouped sequences of interesting events, or phases. Groupings can cover many layers of abstraction — e.g., top level phases can consist of a series of sub-phases and so on. There is a one-to-one relationship between the structure of program phases and the structure of the trace files. This
is carried through into VISPAT’s data visualization facilities. This is the means by which VISPAT enables the tuner to relate the behaviour of the program to the source code.

In this hierarchical presentation of events and phases, a mechanism to help the tuner identify events of interest and determine what data will be subsequently visualized by the performance displays is important. In particular, the tuner should be able to determine interactively: a region of the trace file (pan over the data); the time grain (zoom in or out of the chosen region); which events will be visible (filter out unwanted events) and, finally, control the level of abstraction (fold or unfold phases).

These requirements were realised in VISPAT through a single user interface mechanism, the Navigation Display (see Figure 1). The Navigation Display determines the context of data visualization. It has a central role because VISPAT’s other displays render data only over the time period and parts of the program that are currently visible within it. The Navigation display renders the parallel event histories of the processes in the program. It is a form of Gantt chart where the time line is depicted on the horizontal axis and the set of processes on the vertical axis. When the Navigation Display is used in combination with VISPAT’s abstraction mechanisms, this context also provides the means for achieving source code reference.

Apart from the Navigation Display, VISPAT’s display set currently includes the Communication Display, the Statistics Display, the Membership Matrix Display and the Profile Display.

The Communication Display presents an animated graphical representation
of the communication events in an MPI-based parallel program. The numerous communication events in a parallel program necessitate filterings that reduce the complexity of the communication space. MPI communicators provide the means of separating the communication space since each communicator specifies a communication context for a given communication operation. Communications that happen within one communication context do not interfere with communications in a different context. VISPAT provides multiple instances of the Communication Display, where each instance depicts the communication events within a particular context.

The abstraction achieved above can be extended to allow for filtering over the processes that participate in the communications of particular communicator(s). A list of single-line textual descriptions of each communication event is also provided along with every instance of the Communication Display. The textual description aims to:

- resolve any ambiguities that might be present in the graphical representation,
- provide timing information about the beginning and completion of events
- and
- supply the tuner with a history list of communication events, so that the current state of the display can be related to previous ones.

5 An Example of the Use of VISPAT

VISPAT has been applied to several performance studies of programs solving a number of the so-called “Cowichan problems” [6]. One of these is the inverted percolation problem. The corresponding program simulates the displacement of one fluid such as oil by another such as water in fractured rock. An \( N \times N \) matrix is filled with randomly generated numbers representing the resistance of the point to its displacement. During each iteration, the process assigned with a member of the matrix examines the four orthogonal neighbours of this cell and chooses to fill the one with the lowest value. The simulation continues until a certain part of the array is filled or some other condition is satisfied. Figure 2 shows the phase tree for each of the processes in the application.

The program in question is written in C++ and uses MPI for communication. It consists of three distinct phases: input, calculation and output. Each of these phases consists of sub-phases. The Navigation Display in Figure 1 reveals that the calculation (Calc) phase is the dominant execution phase. It also shows that the output (Out) phase takes a lot more time to complete on Process 0 than on any of the rest of the processes. By expanding the phase Out for Process 0 (see the Profile Display in Figure 3), it is clear that its sub-phase Write is responsible for most of the delay observed. This was expected because this process first gathers all the partial results from the rest of the processes and outputs them into a file.

The program execution time can be reduced by a factor of ten percent if the result data is output in parallel by every process in the program. Subsequently the tuner can improve the algorithm employed for the calculation of the matrix.
6 Conclusions and Future Work

A prototype tuning tool has been implemented which through the concept of hierarchical phases enables the tuner to relate low-level events in trace data to program source code. In VISPAT, there is one-to-one correspondence between the structure of program phases, the structure of trace files and its visualization mechanisms. VISPAT has been extensively evaluated by prospective users and the feedback gained is being used to refine the functionality and improve usability.

Future work will include the extension of VISPAT's query mechanism to include appropriate filters to extract information from trace data convert it to a suitable format for use with other performance analysis and visualization tools.

References