The Design of a Tool for Parallel Program Performance Analysis and Tuning

Anna Hondroudakis and Rob Procter
Department of Computer Science
Edinburgh University
Mayfield Road
Edinburgh EH9 3JZ
Scotland
Email: ah@dcs.ed.ac.uk
Tel. +44 131 650 5115
Fax. +44 131 667 7209

Abstract
The implementation of efficient parallel programs can be very difficult, and numerous computer-based tools have been developed to help with the task of performance analysis and tuning. We describe the initial steps in the design of such a tool. Our aim is to show how its design has evolved out of an explicit attempt to understand the human factors issues in parallel program implementation. By adopting this approach, we seek to look beyond the current fashion for graphical user interfaces and data visualization techniques. Our emphasis is on gaining a better understanding of the problems inherent in program analysis and tuning and using this as a foundation for developing more usable tools.

Introduction
Parallel program performance analysis and tuning is concerned with achieving efficient utilisation of system resources. One common technique is to collect trace data and then analyse it for possible causes of poor performance. The program source code is then modified in the light of the analysis. In practice, 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 [CG88, RC89, DFH93]. 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. The tuner faces equally difficult problems due to the fact that desired levels of performance may only be achieved after many iterations of the unit tuning cycle.

This paper describes VISPAT (VISualization for Performance Analysis and Tuning), a tool for parallel program performance analysis and tuning. Numerous examples of such tools already exist, many of which employ the latest graphical techniques for user interface design and visualization of trace data [RAM+92, HE91, HIM91, Moh91, MHJ91, Sto88, BB92, FR92]. In spite of this, however, their design often appears to rest upon largely unstated assumptions concerning the nature of the tuning task, and evidence for their effectiveness is often lacking. Graphical user interfaces are not a panacea for usability problems and, in fact, their use can increase the scope for poor design [Bro94]. In contrast, our approach to the design of VISPAT has been to seek a better understanding of the nature of the tuning task and the needs of tuners through the application of user-centered design methodologies.

The results of this investigation highlighted the importance of two specific user requirements. The first was for the tool to provide help in achieving source code reference. It reflects the difficulties of tuning in the small, i.e. the detection, diagnosis and remedying of a specific instance of inefficiency as the outcome of the unit tuning cycle. The tuner should be able to visualize the trace data in ways which significantly reduce the effort necessary to relate the observed behaviour to the program's source code. The second requirement was for the tool to provide assistance in the management of tuning in the large, i.e. the lengthy series of repetitions of the unit tuning cycle that may be required to achieve the desired overall level of performance. The typical tuning problem is associated with a number of inter-dependent program performance parameters for which some 'optimum' set of values has to be found. Tuning in the large may be thought of as a heuristic search through the multi-dimensional performance space defined by these parameters. As a consequence, the tuner must evaluate and compare the results of a succession of unit tuning cycles. To expedite the outcome, the tuner may need help in managing the search process.

The first section briefly describes the programming environment within which VISPAT is to be used. The second gives an overview of how the tool's requirements were elicited from users and evolved into a specification, and the third section describes the design of VISPAT in some detail. Finally, we discuss the next stage in its development.

The Programming Environment

The Common High-Level Interface to Message Passing (CHIMP) library was developed at the Edinburgh Parallel Computer Centre (EPCC) and is available on a number of parallel and distributed platforms [BCMT93]. It provides for the transfer of messages between processes on a MIMD parallel computer. Blocking, non-blocking and multi-casting communications are available. CHIMP has a two-level naming scheme for processes, consisting of symbolic and numeric components. Each process has a number of Service Access Points (SAPs) through which it can receive and send messages. Processes with common algorithmic features are functionally grouped. Every functional group (SAP group) has a name and each process's SAP in the SAP group is identified by a membership number.

The Parallel Utilities Library (PUL) is a collection of reusable basic parallel utilities that support program development [BCMT93]. PUL is built on top of CHIMP to take advantage of its portability. The utilities of PUL fall into three different classes: modules
that implement generic functionality independent of the underlying mapping: modules that follow a certain programming paradigm e.g. for decomposition or mapping, and modules that are application-specific. Some of the utilities have a procedural interface which encloses data flow and some can have, in addition, a skeletal interface which encloses control flow. Example modules are PUL-EM (extended message passing), PUL-TF (task farm paradigm) and PUL-RD (regular domain decomposition).

**User Requirements Capture**

To maximise the usability and effectiveness of a new tool, it is essential for the designer to explore users' requirements thoroughly and to gain a sound understanding of the nature of the task that they wish to perform. These goals can only be achieved if designer and users work in partnership within the framework of a user-centred design methodology[]. Where there is uncertainty about user requirements, the designer must be prepared to follow an iterative approach that permits users' views to be incorporated before each new step is taken.

As the first step in the design of VISPAT, a series of meetings was held with CHIMP and PUL users, beginning with “brainstorming sessions” in which the participants were simply encouraged to voice their ideas. These were subsequently ranked in importance by the participants and further categorised as being either high- or low-level requirements. The requirements thus defined were used in a second series of meetings as the basis for an initial design. This was then presented to prospective users in the form of a set of paper-based mock-ups of the user interface. At each meeting, the mock-ups were examined page by page and comments solicited from the group. In this way, requirements and specifications were generated, elaborated and refined, eventually leading to the point where they were deemed by the group to be a sufficient basis for the implementation of a first prototype.

At the same time, the nature of the tuning task was explored through group discussion and through the examination of individual tuning case studies. The unit tuning cycle, or tuning in the small, is shown in Figure 1. The upper half of the figure represents those aspects of the task as it is directly supported by VISPAT. The instrumented program is executed and the trace data then is passed to graphical displays through a series of abstraction and filtering mechanisms. Each display may render a different aspect of the program's behaviour. The lower half of the figure represents those aspects of the task which are indirectly supported by VISPAT. The tuner examines the visualized trace data and considers its relevance in the context of the emerging understanding of the program's behaviour. This in turn guides the tuner in the generation of a hypothesis concerning the relationship of parameters to performance, and the specification of an experiment in which parameter values are manipulated in accordance with the hypothesis. Finally, the tuner begins a fresh new iteration of the unit tuning cycle by re-executing the program with the new parameter values and another set of trace data is collected. It is this repetition of the unit tuning cycle which underlies the notion of tuning in the large.

The outputs of the requirements capture process were eventually categorised into one of four fundamental design issues: trace data format; filtering mechanisms; display options, and the management of the tuning task over time. The first three are concerned with tuning in the small in that they reflect tuners' needs within the unit tuning cycle. The last issue relates to tuning in the large which will be the subject of a future paper.

Within each of these categories, requirements were further classified as either low-
or high-level. Low-level requirements include the spatial arrangement of displays on the screen, and the precise ways in which the tuner might interact with them. High-level requirements reflect issues that define the character and scope of VISPAT. Three of the principal high-level requirements will now be discussed in more detail.

**Trace Data Format**
The trace data format requirements are largely determined by the programming model and the requirement for source code reference. The control flow of each process's code consists of time-grouped sequences of interesting events, or *phases*. A phase occupies a part of the program's execution time and corresponds either to a call of a library function or to a programmer-defined logical part of the source code. A phase is identified by the name of the function, together with a number which distinguishes each unique occurrence of it within the trace data file. It is also characterized by its class which can be either CHIMP, PUL-EM, PUL-TF, PUL-RD or Application (if the phase is programmer-defined).

Groupings cover many layers of abstractions, with the most abstract layer concerned with phases evident in the execution of almost any parallel program — initialization, computation and closing. Each of these top level phases consists of a series of sub-phases, which can have sub-phases of their own and so on. Figure 2 depicts the hierarchy of phases in a program written with PUL. Functions whose names begin with CHIMP, RD and EM are respectively functions of the CHIMP, PUL-RD and PUL-EM Libraries. There is a one-to-one relationship between the structure of phases in a program and the structure of the trace files. This is carried through into VISPAT's data visualization facilities. By this means, VISPAT enables the tuner to relate the behaviour of the program to the source code — a facility found in few other tools ([SAB+92] being one notable exception).

**Data Navigation and Filtering**
In this hierarchical presentation of events and phases, a mechanism to help the tuner identify events of interest is important. Data navigation provides the tuner with a means of determining what data will be subsequently visualized by the performance displays. More specifically, the requirements investigation emphasised that 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 through a single user interface mechanism — the Navigation display — a form of Gantt chart (see Figure 3). 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. In the figure, the early phases of a program computing the Optimal Matrix Multiplication Order (OMMO) of 4 matrices [Sed90, WG92] are shown. The program has three types of processes: four workers which calculate costs and store them in a cost matrix; eight matrix line managers, which store and update rows and columns of the cost matrix, and one barrier synchronization manager to allow workers to co-operate in determining the value of a particular cost matrix entry.
Data Visualization

Another high-level requirement is the need for a number of distinct types of visualization format for presenting detailed information relating to program performance.

The Single Event display is a special case of the Navigation display. As its name implies, it shows one particular phase over the program execution time, allowing the tuner to compare relative timings of the phase across processes.

The Animation display (see Figure 4) presents a two-dimensional image of the program, depicting significant events such as communication between processes. Incoming arrowheads indicate where processes have initiated a blocking receive. Outgoing arrowheads on the uppermost process indicate that it has initiated blocking sends. The presence of an arrow body indicates that a communication is in progress.

The Membership Matrix display (see Figure 4) provides access to SAP group membership information for each of the processes in the program. Information about SAP group membership is essential if the tuner is to be able to interpret program behaviour. SAP group membership defines the way processes communicate, and corresponds to an algorithmic view of the program. SAP group names are shown in the first column and processes along the top row. In the example program, the first SAP group is called `synch` and consists of the barrier synchronization manager process. This process has the SAP group membership number zero.

The Statistics display consists of a number of tabular displays providing summary performance data. An instance of the Statistics display is given in Figure 5. The metrics are colour coded and the user can access the actual values of these metrics by interacting with the display.

Together, the Navigation and data visualization displays constitute the principal parts of VISPAT's user interface. They are the means by which the tuner is able to reduce a large volume of trace data into a more manageable form, and use it to extract meaningful information about program behaviour.

Implementation of VISPAT

VISPAT is a postmortem visualization tool for the analysis of parallel programs written with the CHIMP message passing system and the PUL utilities libraries [TW92]. It processes and visualizes trace files that have been produced by instrumented versions of CHIMP and PUL. Trace files are read in by a pre-processor and passed on to the visualizer which then renders the data using the displays described in the previous section.

The current display repertoire consists of the Navigation, Animation and Membership Matrix displays. Plans for future work include development of the single event display and a suite of statistics displays. The tool has been built in a modular way so that new displays can be added easily. Because of its central importance, the Navigation display will now be described in some detail.

Navigation display

The Navigation display renders the parallel event histories of the processes in the program. It is a Gantt chart where the time line is depicted on the horizontal axis and the set of processes on the vertical axis. Each process occupies a horizontal strip of the display.
where all the interesting events of that process are depicted as they take place in time. The
Navigation display provides the context for the visualization of trace and performance
data. When used in combination with VISPA's abstraction mechanisms, this context
also provides the means for achieving source code reference.

The Navigation display enables the tuner to move around within the space of pro-
gram phases. Navigation can be performed in both a horizontal fashion, i.e. forwards and
backwards in time, or in a vertical fashion, i.e. up and down the phase hierarchy. Two
event trace 'playback' modes are provided, logical and temporal. The requirement for dif-
ferent playback modes arises from the effect of non-blocking communications on phase
sequencing. In logical playback, a non-blocking communication finishes when the func-
tion initiating it returns. Logical playback is more coarse-grained and hides some timing
information. Temporal playback takes into account the completion times of non-blocking
communications, and so allows phases to be interleaved on the process time line.

Traversing the various levels of the phase hierarchy is achieved through unfolding and
folding, with the additional option of filtering. This not only allows for the transfer of the
focus of visualization to a higher or lower level of abstraction, but also for the view to
be limited to specified phases. The data visualization displays show events only for those
phases visible on the Navigation display after all filterings and abstractions have been
applied. In this way the tuner is able to relate trace data to specific parts of the program.

The design of the Navigation display is driven by the requirement for flexible abstrac-
tions over the trace data. Tuners need to be able to view the phases at different levels of
abstraction in order to be able to hide irrelevant information, and distinguish the most
interesting events. Such abstraction mechanisms [BW83] are found in other tuning envi-
ronments where, in a 'bottom up' way, low-level events are abstracted to higher level ones.
Event abstraction in VISPA is more extensive and higher-level events are defined auto-
matically by the PUL and CHIMP library functions. The hierarchical grouping of events
in VISPA reflects the hierarchical grouping of the library functions in the program and
in this way provides the abstractions necessary to limit the focus to the areas of interest
whilst maintaining a close link to the structure of the source code.

The function of the current version of the Navigation display can be categorised into
tree distinct groups: operations on phases; operations on processes, and user interface
operations. The first group consists of the Selection, Unfolding/Folding and Filtering
mechanisms:

Selection — when phases are chosen for expansion or filtering. A selection can be global
or local. Global selection of a phase ensures that not only the current instance but
all subsequent instances of a phase will be selected. Local selection means that only
the current instance of a phase is selected. The view of selected phases can then be
either enhanced or removed.

Unfolding/Folding — where selected phases can be unfolded into their sub-phases (or
sub-phases folded into their parent phases). There are two ways of unfolding a phase.
Unfolding is either simple or it is specific. In the former, a phase can be unfolded into
all its sub-phases independently of the class to which they belong. In the latter, the
parent phase is unfolded only into phases of a nominated class. This class may be
one of the pre-defined CHIMP or PUL classes, or a programmer-defined Application
class. In this manner, the trace data can be abstracted in two orthogonal directions
and the phase hierarchy traversed horizontally and vertically.

Filtering — which realises a more immediate abstraction mechanism by enabling the
user to remove selected phases from the display. The Navigation display in Figure
6 shows the user-defined phases of the OMMO program after filtering has excluded all other phases from Figure 3. It is clear from this that one worker process in the penultimate row has no work to do.

The second category of Navigation display operations focuses on process abstractions: Elimination — whereby those processes whose behaviour is not currently of interest can be removed from the screen.

User interface operations are provided to make the Navigation display more interactive. Apart from the means of changing the time unit and the scale width of the display, the user can hide uninteresting sequences, or the periods where the behaviour of the program has already been analysed.

Conclusions and Future Work

A prototype tuning tool has been implemented which, through the principle of hierarchical phases, enables the tuner to relate low-level events in trace data to program source code. There is a one-to-one correspondence between the structure of program phases, the structure of the trace files, and visualization mechanisms.

A user-centered requirements capture methodology has been employed as a means of identifying the major user requirements of performance analysis and tuning tools. This has enabled the gaining of valuable insights into tuners' needs and the nature of the tuning task itself. Future versions of VISPAT will be implemented following a series of user evaluations of the prototype. This will assist the further refinement of high-level requirements and the identification of new lower-level ones. For example, more data visualization options may be added and the functionality of others improved. Special attention will be given to the provision of scalable representations to cater for programs which run on massively parallel systems.

CHIMP will be replaced gradually by EPCC’s own implementation of the Message Passing Interface standard (MPI) [For93]. A feasibility study will indicate the necessary changes to VISPAT to enable it to be used for tuning programs which use MPI.

Subsequent iterations in the design and evaluation process will also serve as a vehicle for a more detailed investigation of the nature of the tuning task. This will focus on the kinds of task management issues that arise within the context of tuning in the large.

References


Figure 1: The Tuning Model
Figure 2: The phase hierarchy.
Figure 3: The Navigation display.
Figure 4: Membership Matrix and Animation displays
Figure 5: An example of the Statistics display
Figure 6: Programmer-defined phases in the Navigation display