The Tuner’s Workbench: An Environment for Supporting Tuning in the Large

Citation for published version (APA):

Published in:
host publication

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester's Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.

Download date:10. Oct. 2020
The Tuner’s Workbench: A Tool to Support Tuning in the Large

Anna Hondroudakis and Rob Procter
Dept. of Computer Science, Edinburgh University,
Edinburgh EH9 3JZ, Scotland
Email: ah|rnp@dcs.ed.ac.uk Tel. +44 131 650 5115

Abstract. The evident difficulties in creating efficient parallel programs have encouraged the development of computer-based support tools for performance analysis. Our work focuses on gaining a better understanding of the task of performance analysis and tuning, and using this to establish where computer-based support can be most effectively applied. We distinguish between the difficulties inherent in tuning in the small and in tuning in the large and relate these to the design of the Tuner’s workbench.

1. Introduction

Parallel program performance analysis and tuning is concerned with achieving efficient utilization of machine resources. One common technique is to collect trace data and then analyse it for possible causes of poor performance. The source code is then modified in the light of the analysis. In practice, tuning a parallel program is a much more difficult task than this brief sketch suggests. Our work has been to identify what aspects of tuning would most benefit from computer-based support. Through the use of various techniques — primarily observation, questionnaires, interview and case history reviews — we have made a close study of what tuners actually do. From the data gathered, a set of user requirements have been identified and two tuning support tools designed and implemented. These are currently in the process of being evaluated by tuners.

The two tools address requirements arising out of what we consider to be two distinct phases of the overall tuning task: tuning in the small and tuning in the large. In contrast to the former, the latter has received relatively little attention. The first section of this paper reviews the issues raised by tuning in the small and subsequent sections discuss those raised by tuning in the large, and how tools can help solve them.

2. Tuning in the Small

Tuning in the small refers to the unit tuning cycle, the sequence of basic steps taken by the tuner in attempting to improve performance i.e. problem detection, analysis and resolution. One difficulty for the tuner is that the amount of trace data produced may be very large, requiring sophisticated visualization mechanisms to present it in a form which aids analysis. This aspect of tuning is well supported by numerous trace visualization tools [1, 5, 6, 12, 11, 14, 15]. In order to resolve the performance problem,
however, the tuner must relate the low-level account of program behaviour provided by the trace data to the high-level representation of program behaviour contained within the source code.

VISPAT (VISualization for Performance Analysis and Tuning) addresses this need by providing the tuner with a simple, but effective, mechanism for establishing source code reference, based upon the concept of hierarchical program phases [7]. It was obvious from our studies of tuning, however, that tuning is often a very protracted process, often requiring a lengthy series of repetitions of the unit tuning cycle. It is this aspect of tuning which we refer to as tuning in the large.

3. Tuning in the Large

The resolution of a typical performance problem consists of a number of iterations of the unit tuning cycle, where each one is guided by the outcome(s) of its predecessor(s). Tuning in the large may therefore be thought of as a heuristic search within the program's performance space (defined by the set of performance-determining parameters) which continues until some optimum (or near optimum) outcome is obtained. Crucial to the efficient conclusion of such tasks is the availability of appropriate search management information. In the case of tuning, this information is generated as a by-product of each individual unit tuning cycle, or tuning experiment. Its importance points to the need to maintain adequate documentation of the task as it progresses.

The tuner may utilize records of tuning experiments in a number of ways. For example, the tuner may wish to keep for later reference all or part of the records of several experiments in order to document:

1. the problem under investigation and its manifestation in the trace data,
2. the parameter values, and the path through the performance space,
3. performance metrics associated with each set of parameter values, and
4. the state of the program when tuning was concluded.

At any time during the course of tuning, the tuner may need to consult tuning experiment records in order to find out:

1. the parameter values that produced the best overall performance so far,
2. the parameter values that produced the best value for one specific performance metric, and
3. whether the current performance is better than that achieved with a specific set of parameter values.

Examples of complete tuning problem case histories might be kept to create an archive which tuners may consult as a source of ideas and strategies for tackling new problems.

4. Supporting Tuning in the Large

For the tuner, the difficulties associated with tuning in the large stem from the effort required to manage the tuning process and, in particular, its documentation — the capture, storage and retrieval of tuning experiment records (see Figure 1). The issue is to what extent computer-based support may help the tuner to perform such tasks.
The benefits of good documentation are widely extolled within the software engineering literature [13]. These can be summarized as follows [2]:

1. to support reasoning processes during software design,

2. to facilitate communication amongst the various members of the software development team, and

3. to further the accumulation and development of software design knowledge and experience from project to project.

Some of the many computer-based tools now available to support software documentation (e.g. [3, 4, 8, 9, 16]) might find application within parallel software development. We believe, however, that there are significant qualitative differences in the documentation requirements of parallel software development, which reflect both the experimental, iterative nature of the tuning phase (as opposed to the more orderly and predictable course of conventional software development phases) and the large volumes of information that tuning generates.

To investigate these issues further, we have been conducting a study of tuning practices within a number of organizations engaged in parallel software development.

5. Investigation of Tuning Practices

An initial, informal investigation was conducted through interviews with local tuners and by soliciting views of the wider tuning community via the medium of Internet bulletin boards, the World Wide Web and email. This highlighted a number of issues which are now in the process of being explored more systematically through questionnaires and follow-up interviews.

Almost half of the people who answered work in university establishments and the rest for companies that either manufacture or use parallel machines extensively. Their experience with parallel computing varied from one to eleven years. In this section,
we present some preliminary results, including the responses of twenty four tuners to questionnaires and extracts from interviews.

5.1. Duration of tuning

The time taken to complete a particular tuning task (see Table 1) is a pointer to its complexity and to the importance of documentation. The fact that respondents reported that tuning typically lasted for several days or weeks suggests many iterations of the unit tuning cycle and the handling of large volumes of associated data. In such circumstances, the limitations of human memory would necessitate that the tuner at least document the sequence of changes made to the program to ensure that changes are not inadvertently repeated. One tuner commented:

“Keeping track of the changes; I don’t find it difficult any more because I keep notes of what I do, because I was caught by that once.”

<table>
<thead>
<tr>
<th>Question</th>
<th>less than a day</th>
<th>several days</th>
<th>several weeks</th>
<th>months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration of tuning?</td>
<td>0</td>
<td>9</td>
<td>13</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1: Duration of tuning.

5.2. Size of the parameter set

The size of the performance-determining parameter set is another indicator of the complexity of the task. If the number of parameters is large, it will be difficult for the tuner to establish the relationship between parameter-value pairs and performance without some form of documentation.

Whilst relatively low numbers of parameters predominate in this sample (see Table 2), the complexity of the task may be increased by the fact that parameters are often interdependent. Consequently, the optimum value for each parameter alone is not always the optimum for the parameter set. One tuner suggested performance prediction as a solution to this problem:

“Because I mean it might be the case to try one change by itself or two changes together to see if they work well together. So if you have three you have somehow ...you can’t check everything so you want to predict which are going to work well together.”

<table>
<thead>
<tr>
<th>Question</th>
<th>1-3</th>
<th>5-10</th>
<th>15-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of parameter set?</td>
<td>14</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2: Size of parameter set.
5.3. Documentation methods

All of the respondents reported that they employed some form of documentation (see Table 3). (The totals exceed the number of respondents because some reported using more than one documentation method.) It is interesting to see that documenting the source code and keeping notes either in paper or on-line are almost equally common. Email responses emphasized the variety of documenting methods and tools:

“On pieces of paper. They were not very organized. So I often had to repeat the experiments to be able to carry on from where I stopped.”

“The documentation we did was ad hoc: a file of notes on the various configurations tried, and on the tuning results. This was never written up in any formal way, just kept for informal reference by the programmers.”

“I tend to keep sets of timings for each version in a spreadsheet allowing me to easily compare improvements or degradations between one version and the next.”

“I document the source code and I have an on-line file where I normally write down values of parameters, performance data, program changes and some times gnuplot graphics.”

<table>
<thead>
<tr>
<th>Question</th>
<th>source code</th>
<th>on-line file</th>
<th>paper notes</th>
<th>reports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation methods?</td>
<td>15</td>
<td>9</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3: Documentation methods.

5.4. Reasons for documenting tuning

It is interesting that the need to communicate results to colleagues is rated quite highly by respondents (see Table 4). This, together with the last three reasons listed — assisting portability, providing a record for others and report generation — emphasize the collaborative nature of parallel software development, and the importance — as in sequential software development — of sharing knowledge amongst project members. Respondents also stressed the value of accumulating tuning case histories across projects: “to use as a learning aid and for future performance sessions” and:

“It allows me to see what changes made the biggest differences hopefully to help me for the next time I need to tune a parallel program.”

Other email respondents pointed to the difficulties ensuing when projects get interrupted before completion. For example:

“To have a place holder in order to be able to put a project on hold for a period of time and be able to return to that project at a later time without having to reinvent the wheel. We juggle many projects at a time and this is a key not to regress.”
<table>
<thead>
<tr>
<th>Reasons for documenting tuning?</th>
<th>No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>as a reminder of progress</td>
<td>16</td>
</tr>
<tr>
<td>to keep track of changes</td>
<td>15</td>
</tr>
<tr>
<td>to aid communication with colleagues</td>
<td>8</td>
</tr>
<tr>
<td>to aid portability</td>
<td>1</td>
</tr>
<tr>
<td>provide a record for others</td>
<td>2</td>
</tr>
<tr>
<td>report generation</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4: Reasons for documenting tuning.

5.5. Interviews with tuners

To gather a more detailed picture of parallel software tuning projects and practices, we interviewed twenty three members of large supercomputer centres and companies. The following are extracts from interviews with project team members in a large European research institution. A sizeable code (300000 lines) had been developed and this was being parallelized and ported to a number of different vendors’ machines with the goal purpose of identifying the optimum system. Communication between project members was a problem:

“The way we are working here today is from this point of view very haphazard. We have a lot of versions, and this is a significant problem. Although we have a version control system the guys in the parallel world don’t use it because it is installed on a different system from the one they work on … so they are reluctant to use it and we are at the situation where there are lots of tar files lying there and it is slightly difficult to follow what you have.”

The need for a tuning management tool had already been raised and its requirements discussed in some detail:

“We would be interested in having something that integrates your kind of environment where you can query a database of outcomes of experiments. And have a kind of version management included … you have an application and you are going to be running thousands and thousands of data sets with this, or a large number of data sets, you have to get to record that, and maybe you have number of processors as one factor, maybe the optimization level of the compiler etc. And it was exactly the idea, for example the system would automatically record what optimizations you used or when you use a visualization tool the trace files would be linked to makefiles that you used. And it could associate the options for the compiler and the outcome of the experiment. If you extend it you can even create a database of case histories that other people can go and look up, for cases that are similar to his, and say some one has a problem on one machine and he describes a bit of the problem and then somebody could access it and benefit from this experience.”

5.6. Conclusions

We conclude from this investigation that tuning in the large would benefit significantly from computer-based support to help the tuner manage the process and communicate
the rationale of solutions to others [2, 10]. The following section outlines the specification of such a tool.

6. Specification of the Tuning Workbench

The Tuning Workbench (see Figure 2) consists of four main components: a program development environment, a runtime environment, a performance analysis and visualization tool and a tuning experiment record database.

![Diagram of Tuning Workbench](image)

Figure 2: Components of the Tuning Workbench.

It is important that the Tuner’s Workbench should be easy to use and not overly constrain the way in which the tuner documents the task. In this way documentation will be less likely to be considered as an additional overhead. The Workbench is “policy-free” in two other important ways; it does not force the use of specific tuning tools (e.g. visualization tools) or impose any particular strategy for performance analysis upon the tuner. Rather, it is intended to provide an environment in which the tuner’s choice of tools can be used more effectively in support of whatever strategy the tuner feels is most appropriate to the particular problem.

An experiment, the central entity in the experiment database, is a record that represents the outcome of a single unit tuning cycle. Typically it will include a description of why it was done (the hypothesis), what was done (the set of parameter-value pairs), the results (trace data), analysis and conclusions.

The implementation of the Workbench is based upon existing hypermedia and database technologies. The experiment record can contain information in a variety of formats, including textual (e.g. lists of variable-value pairs representing parameters and performance metrics), static graphical (e.g. snapshots of displays generated by the trace data visualization tool) and dynamic graphical (e.g. animated displays from the trace visualization tool).
The structure of the tuning experiment database is straightforward. Experiment records are simply nodes within a larger hypermedia document that is the complete database. A sequence of experiment records defines the path followed to tune a program and these are organized as a hierarchical tree structure with branches that correspond to experiments that share a common hypothesis. For any hypothesis, the most recent experiment is at the leaf of the branch corresponding to this hypothesis.

Prototypical and other interesting tuning problems can be selected for inclusion within the set of tuning case histories, which can be retrieved at a later time for consultation. The availability of tuning knowledge of this type can be extremely useful, not only to others within a particular organization, but to the parallel software development community at large. In this way, the tuner can benefit from others' experience and save time and effort when faced with a similar tuning problem. Such practices are already commonplace. For example, tuners routinely use Internet news bulletin boards to seek out advice from others who have solved similar problems.

7. A prototype Tuner's Workbench

A prototype of the Tuner’s Workbench has been implemented. Its functionality covers the sub-tasks presented in Figure 1.

**Tuning record management** provides for the creation and management of tuning experiment records. Experiments are grouped into projects. For every project, parameters and metrics that determine and express the performance of the software can be chosen from a database of parameters and metrics. New parameters and metrics can be created or old ones deleted. Tuning experiments can be created, opened deleted or compared to one another. The Revision Control System (RCS) is used to keep and manage the different versions of trace, source and binary files.

**Assess tuning progress** The most successful experiment(s) can be retrieved by querying the experiment database. Searches can be narrowed by specifying one or more performance metrics and/or parameters. Each experiment record is linked to its predecessor and successor nodes and these can be immediately accessed by the activation of the appropriate links.

Users can query the experiment database in two ways. First, they can retrieve the experiment having the lowest or highest parameter or metric value of interest; second, they can retrieve a listing of values of a certain parameter or metric over the entire contents of the experiment database.

**Case history selection** allows users to add tuning projects to, and retrieve them from, the case history database. A case history document consists of a description of the tuning problem and one or more relevant experiment records. To retrieve a case history document, the tuner specifies criteria such as the performance problem, the machine, the languages and tools used etc. Once a case history document has been retrieved, the tuning project can be reviewed by examining its trace files etc. using the appropriate tools.
7.1. World Wide Web and The Tuner’s Workbench

The World Wide Web (WWW) opens up new possibilities for disseminating information. Using WWW Forms\(^1\), WWW can provide a user interface to application engines that exist at remote sites. For this reason, it was decided to use WWW to host the prototype Tuner’s Workbench. This makes the Workbench and its tuning experiment database accessible to a wide audience through readily available WWW viewers. The process of refining the prototype workbench is also enhanced as it can be evaluated by that same audience. Part of the WWW prototype workbench user interface is shown in Figure 3.

8. Conclusions

The focus of this work has been the investigation of the nature of the parallel program tuning. It has highlighted aspects of tuning that relate to tuning in the large and which are neglected by existing tools. We have examined how, in practice, tuners manage tuning in the large and the implications of the lack of appropriate computer-based support. We have proposed the Tuner’s Workbench to address this omission and implemented a prototype which is accessible on the World Wide Web\(^2\).

By making the Tuner’s Workbench available in this way, we hope to enlist the help of tuners from a wide range of organizations. We will use the feedback thus obtained to evaluate the prototype and identify areas for improvement and further development.

References


\(^1\)http://www.nscia.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html
\(^2\)http://www.tardis.ed.ac.uk/~anna/tuning.html


Create a tuning experiment

Project name = heuristic

Creation date = Jan 15 11:21:04 1995

Previous version = 1.1

Current version = 1.1.1

Next version = 1.2

Trace File Name = Recurs.Parti.Bisection

Experiment Name = Rec.Part.Bise/NoOpt

Edit

Run

PARAMETERS THAT AFFECT THE PERFORMANCE OF THE PROGRAM

Name of parameter/ Value of parameter

1. Number of processors 3
2. Decomposition technique Recur/Partition Bisection
3. Compiler flags switched off
4. Communication patterns N/SEWE but not F/B

Figure 3: WWW prototype Tuning Workbench user interface.