Lehrstuhl für Rechnertechnik und Rechnerorganisation
der Technischen Universität München

Scalable Automated Online Performance Analysis of Applications using Performance Properties

Karl Fürlinger

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. A. Bode

Prüfer der Dissertation:

1. Univ.-Prof. Dr. H. M. Gerndt
2. Univ.-Prof. Dr. H.-J. Bungartz

Die Dissertation wurde am 11. Mai 2006 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 3. Juli 2006 angenommen.

Research Report Series
Lehrstuhl für Rechnertechnik und
Rechnerorganisation (LRR-TUM)
Technische Universität München

http://wwwbode.in.tum.de/

Editor: Prof. Dr. A. Bode

Vol. 33

Scalable Automated Online Performance Analysis of Applications using Performance Properties

Karl Fürlinger

SHAKER
VERLAG
Aachen 2006

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Zugl.: München, Techn. Univ., Diss., 2006

Copyright Shaker Verlag 2006
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN-10: 3-8322-5459-5
ISBN-13: 978-3-8322-5459-9
ISSN 1432-0169

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

Abstract

Analyzing the performance of applications is an important step in the development process of scientific software for high-performance computing systems. Programmers need to ensure that their code performs well and does make good use of the available resources on such systems.

Unfortunately, the task of finding inefficiencies and identifying their reasons can be a time-consuming and difficult challenge – even for sequential programs. For large-scale parallel systems and programs the situation is complicated by several factors. Firstly, parallel programming is inherently more complicated than sequential programming and additional middleware layers may be present that affect the achieved performance. Secondly, the constantly increasing size and complexity of parallel systems and the problems solved on them leads to challenges concerning the scalable collection, visualization, and analysis of performance data.

This work presents an approach to performance analysis and a software tool called *Periscope* that tries to avoid some of the problems outlined above in order to effectively detect inefficiencies on large-scale parallel computing systems. The approach is based on the notion of performance properties that formally capture situations of inefficient execution. By formalizing what constitutes inefficient behavior, the detection process can be automated. The component which performs the automated performance analysis in Periscope is called an *agent* and several agents distributed over the parallel machine cooperate in the overall analysis process for an application.

The decomposition of the tool into a number of agents keeps the analysis process scalable. As an application's size increases in number of processors or SMP nodes used, the tool's size can also increase in number of analysis agents employed. The distribution also solves problems related to the management of large amounts of performance data. The agents analyze data close to the origin in the application and thus problems with a centralized collection, storage, and processing are avoided. Finally, the Periscope approach allows for online analysis. That is, the performance analysis process can be initiated at any time while the program is still running, as opposed to a post-mortem approach where performance data is analyzed after program termination.

Acknowledgments

Many people have contributed to the success of this work. First and foremost, many thanks are directed to Prof. Michael Gerndt and Prof. Arndt Bode for providing such a supportive environment for conducting research. It is my belief that there are few places that can match the freedom and support for research that I had the opportunity to receive.

As any research, this work builds on the previous ideas of many people and I would like express my gratitude for their excellent work. Most notably, the work of Bernd Mohr and Felix Wolf was very helpful and enlightening in many situations. Additionally, numerous helpful people – too many to be named individually – have offered their advice when it was needed and supported me in many different ways.

Lastly, I owe the deepest gratitude to my parents for their continuous, unwavering support over so many years.

Karl Fürlinger
Munich, Germany
May 2006

Contents

1 Introduction 1

1.1 Performance Analysis . . . 2

1.2 The Benefit of Automation . . . 5

1.3 Contribution of This Work . . . 6

2 Existing Approaches for Automated Performance Analysis 9

2.1 Paradyn . . . 9

2.2 Expert . . . 11

2.3 Aksum . . . 13

2.4 MATE . . . 13

2.5 Poirot . . . 14

2.6 Projections:Expert . . . 15

2.7 Statistical Performance Data Analysis . . . 15

2.7.1 Index of Dispersion Analysis . . . 16

2.7.2 Statistical Scalability Analysis . . . 16

3 Performance Properties 19

3.1 The APART Specification Language (ASL) . . . 19

3.2 OpenMP Properties . . . 22

3.2.1 Data Model . . . 22

3.2.2 Property Specification . . . 25

3.3 Properties of MPI Applications . . . 32

3.3.1 Data Model . . . 32

3.3.2 Property Specification . . . 32

4 The Periscope Monitoring Approach 37
4.1 Instrumentation . . . 37
4.1.1 OpenMP Instrumentation . . . 38
4.1.2 MPI Instrumentation . . . 38
4.2 The PSC Monitoring Library . . . 40
4.2.1 Data Structures Allocated by the Monitoring Library . . . 41
4.3 The Periscope Common Memory Access Abstraction . . . 44
4.4 Runtime Information Producer . . . 47

5 The Periscope Performance Analysis Tool 49
5.1 The Periscope Registry Service . . . 49
5.1.1 Registry Data . . . 50
5.1.2 Registry Service Protocol . . . 52
5.2 The Periscope Node-Level Agents . . . 54
5.2.1 The Design and Implementation of the Node-level agents . . . 54
5.2.2 Search for Performance Properties . . . 57
5.3 The Periscope High-Level Agents . . . 57
5.3.1 Agent Communication Infrastructure . . . 58
5.4 The Periscope Frontend . . . 60
5.4.1 Instantiation of the Agent Hierarchy – Periscope Startup . . . 61
5.4.2 Usage Example . . . 67

6 Evaluation 71
6.1 A RIP-based Profiler for OpenMP Applications . . . 71
6.1.1 Overhead Analysis based on the ASL Data Model . . . 72
6.1.2 Scalability Analysis . . . 75
6.1.3 Experiments . . . 75
6.2 Monitoring Overhead of Periscope . . . 80
6.3 The APART Test Suite (ATS) . . . 82
6.3.1 OpenMP Inefficiency Patterns . . . 83
6.3.2 MPI Inefficiency Patterns . . . 88
6.4 A Parallel Quicksort Implementation . . . 91

6.5 A Mixed-Parallel Jacobi Solver 93
6.6 The NAS Parallel Benchmark Suite 94
6.6.1 MPI 94
6.6.2 OpenMP 96

7 Summary and Outlook 99

List of Figures

1.1 A schematic illustration of the measure-analyze-modify cycle. 3

2.1 Paradyn's search history graph uses color-coding to indicate the status of tested hypotheses (reproduced after [64]). 10

2.2 Expert's result viewer with three hierarchical displays (reproduced after [82]). 12

3.1 An ASL property specification for the `WaitAtBarrier` property. 20

3.2 The basic ASL data model used by Periscope, represented as an UML-like class diagram. Both static (e.g., the region nesting) as well as dynamic performance data (e.g., region summaries) are supported by the ASL model. 21

3.3 The `OmpPerf` structure contains summary (profiling) data for OpenMP constructs. 23

3.4 The times and counts defined for the various OpenMP region types in the ASL `OmpPerf` data model for OpenMP applications. 24

3.5 An example for the `OmpPerf` data structure for an OpenMP code fragment consisting of a parallel region that contains a critical section, executed by four threads. 26

3.6 The `MpiPerf` structure holds summary data for MPI calls. 33

4.1 Instrumentation added to an OpenMP parallel region by Opari. The original source code is shown in **boldface** and the indentation has been optimized. 39

4.2 MPI instrumentation using the library interposition technique. 39

4.3 Schematic illustration of the Periscope monitoring approach. The monitor is separated into two components (PSC library and Runtime Information Producer). 40

4.4 Data structures maintained by the PSC library in a common memory region. 41

4.5 Structure of the event records used by Periscope. For each program execution event (e.g., entering or exiting a region, sending a message) an event record is assembled and stored in a ring-buffer. 44

4.6 The buffers and tables holding performance data are either allocated in physically shared memory (a) or an RDMA facility is used to create a shadow copy (b). 45

4.7 The RIP transforms the event records and data from the context table delivered by the PSC library into C++ objects corresponding to this class model. 48

5.1 Schematic illustration of the components of the Periscope performance analysis system. 50

5.2 Example entries created in the Periscope registry service for the performance analysis of the OpenMP application “310.wupwise.psc” running on node “opt33”. The hierarchy of agents encoded via the `tag` key is shown in Fig. 5.3. 50

5.3 Periscope agent hierarchy corresponding to the registry entries shown in Fig. 5.2 (some keys omitted for brevity). 51

5.4 The set of keys supported by the Periscope registry service. 51

5.5 Schematic illustration of the structure of the Periscope node-level agents. . 55

5.6 The layered design of the Periscope node-level agents. 56

5.7 Example agent hierarchy graph for 16 MPI processes that are executed on 8 nodes (`opt01`,. . . ,`opt08`). The maximum number of child agents (the `maxfan` parameter) was set to four. 62

5.8 The startup process of the Periscope tool. 63

6.1 Example `ompP` output for an OpenMP `CRITICAL` region. `R00002` is the region identifier, `cpp_qsomp1.cpp` is the source code file and `156-177` denotes the extent of the construct in the file. Execution times and counts are reported for each thread individually, and summed over all threads in the last line. 72

6.2 The timing categories reported by `ompP` for the different OpenMP constructs and their categorization as overheads by `ompP`’s overhead analysis. (S) corresponds to synchronization overhead, (I) represents overhead due to imbalance, (L) denotes limited parallelism overhead, and (M) signals thread management overhead. 73

6.3 Example overhead analysis report generated by `ompP`, the columns related to the thread management category (`Mgmt`) are omitted due to space limitations. 76

6.4 The OpenMP constructs found in the NAS parallel benchmarks version 3.2. 77

6.5 Speedup achieved by the NAS benchmark programs relative to the 2-processor execution. 78

6.6 Scaling of the total runtime and the division into work and overhead categories for the NAS OpenMP parallel benchmarks (BT, CG, EP, FT, IS, and LU). 79

6.6 Scaling of the total runtime and the division into work and overhead categories for the NAS OpenMP parallel benchmarks (MG and SP). 80

6.7 A detailed scalability analysis at the level of individual parallel regions of the BT application. The four most important parallel regions are analyzed with respect to the work performed and the overheads incurred for an increasing number of threads. 81

6.8 Monitoring overhead of the Periscope monitoring infrastructure for various event rates. 82

6.9 Monitoring overhead incurred for several applications from the NAS benchmark suite. The occasional decrease reflects the natural runtime variation for multiple program runs. 83

6.10 Performance properties identified by Periscope. This table lists all discovered performance properties, even such with very low severity values. . . . 97

6.11 The three most severe performance properties with source-code location and severity value, identified by Periscope (only two properties were found for EP). 97