Automated Performance Analysis using ASL Performance Properties

Karl Fürlinger and Michael Gerndt

Institut für Informatik,
Lehrstuhl für Rechnertechnik und Rechnerorganisation,
Technische Universität München
{Karl.Fuerlinger, Michael.Gerndt}@in.tum.de

Abstract. We present our approach for automating performance analysis of parallel applications based on the idea of ASL performance properties. Our tool *Periscope* automatically searches for inefficiencies specified as ASL properties, leveraging a set of agents arranged in a hierarchy.

1 Introduction

Performance analysis of applications can be a complicated and time-consuming task. Tools and methodologies have therefore been developed that try to automate the process of locating inefficiencies in applications and explaining their reason. With growing size and complexity of applications and high-performance computing systems, automation becomes essential, as manual analysis becomes increasingly infeasible. In this work we present our approach, and our tool Periscope, for automated performance analysis based on the notion of ASL performance properties.

The rest of this paper is organized as follows: Sect. 2 introduces the concept of ASL performance properties and gives some examples. Sect. 3 then outlines our properties-based performance analysis tool Periscope. In Sect. 4 we present results from conducting a performance analysis with Periscope on several OpenMP benchmarks. We discuss related work in Sect. 5 and conclude in Sect.6.

2 Performance Properties

Performance properties formalize what can be regarded as a situation of inefficient execution, given a set of performance observations (measurements) for an application. A property's specification is given in a formal language called ASL (Apart Specification Language) and has three main constituents: *condition* checks the existence of the property, *confidence* quantifies the certainty that the property holds and *severity* denotes how large the negative impact on the performance is.

An example for a property describing imbalance in a parallel loop is shown in Fig. 1. The specification of severity, confidence, and condition can refer to

```
property ImbalanceInParallelLoop(ParPerf pd) {
  let
     imbal = pd.exitBarT[0]+...+pd.exitBarT[pd.threadC-1];

  condition  : (pd->reg.type==LOOP || pd->reg.type==PARALLEL_LOOP &&
               (imbal > 0);
  confidence : 1.0;
  severity   : imbal / RB(pd.exp);
}
```

Fig. 1. The ASL specification of the `ImbalanceInParallelLoop` property.

elements of a data model (`ParPerf` in this case), that depend on the particular programming model and the instrumentation used. In Fig. 1, `ParPerf` contains summary data for OpenMP regions, `type` denotes the type of the construct that the region represents, `exitBarT` refers to the summed time spent in the exit barrier of the construct and `threadC` gives the number of threads executing the region. The `ParPerf` structure has a number of other entries and a several other properties can be formulated using these entries. A more detailled discussion can be found in [1].

3 The Periscope Tool

Periscope is a tool which automatically searches for performance properties during the execution of a target application (online operation). Performance data is analyzed on-the-fly by distributed components of Periscope called agents. The agents are arranged in a hierarchy, as shown in Fig. 2. On the lowest level, node-level agents are responsible for the detection of properties on a single node (assuming a clustered architecture composed of several SMP nodes). The higher level agents aggregate the results of the lower-level agents and pass it on towards the root of the agent tree (the tool's front-end). The front-end displays the search results to the user and takes a user's commands such as to display the agent hierarchy graph and to control the search process for performance properties.

4 Evaluation

To evaluate the usability of the Periscope approach we have analyzed inefficiencies in size "C" version of the OpenMP version of the NAS parallel benchmarks. The NAS benchmark suite consists of five kernels (EP, MG, CG, FT, and IS) and three simulated CFD applications (LU, BT, and SP).

The following table shows which performance properties were discovered in each of the NAS benchmarks, the numbers count the different instances a property was detected.

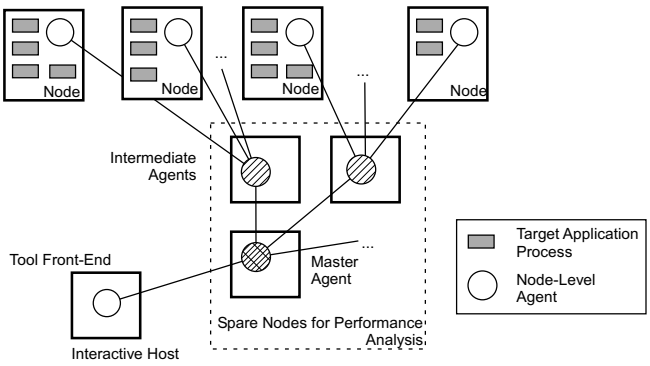

Fig. 2. Periscope agents are arranged in a hierarchy. At the lowest level, node-level agents detect performance properties. Intermediate agents integrate the results of the node-level agents. A single master agent forms the connection to the tool's front-end.

Property	BT	CG	EP	FT	IS	LU	MG	SP
`ImbalanceAtBarrier`					1	3		
`ImbalanceInParallelLoop`	12	13	1	8	2	9	12	16
`ImbalanceInParallelRegion`	6	9	1		2	8	2	5
`UnparallelizedInSingleRegion`						3		
`UnparallelizedInMasterRegion`	4					13	2	5
`CriticalSectionContention`								1

The `Imbalance*` properties refer to the fact that threads perform a different amount of work (prior to a barrier, in a parallel loop or in a parallel region, respectively). The `Unparallelized*` properties refer to the usage of `single` and `master` constructs resulting in serialization of the execution. `CriticalSection-Contention` captures the situation that several threads contend to enter a critical section, resulting in waiting time for some threads. A number of other properties were tested but not detected by Periscope in the NAS benchmarks.

The following table shows the five most severe inefficiencies discovered in the NAS benchmarks by Periscope. Severity referst to the percentage of total exectution time lost due to the inefficiency.

Benchmark	Property	Region	Severity (%)
MG	`ImbalanceInParallelLoop`	`mg.f 608--631`	8.31
FT	`ImbalanceInParallelLoop`	`ft.f 606--625`	6.76
BT	`ImbalanceInParallelLoop`	`rhs.f 177--290`	4.46
BT	`ImbalanceInParallelLoop`	`y_solve.f 40--394`	3.53
BT	`ImbalanceInParallelLoop`	`rhs.f 299--351`	3.47

5 Related Work

Expert [4] is a tool for automated post-mortem performance analysis of C/C++ and Fortran applications. The execution of an instrumented application gener-

ates a trace file, which is scanned for patterns of inefficient execution by Expert. The detected inefficiencies are displayed using a viewer with three panes, the first giving the kind of inefficiency and the other two detailing its location (with respect to program resources and machine organization). In contrast to Expert, Periscope is an online tool and performance analysis can be conducted during the execution of the application. The set of bottlenecks covered by the two tools is somewhat similar, with Expert having the advantage of having full trace information available while Periscope's properties currently rely on profiling data only.

Paradyn [2] is an automated online performance analysis tool leveraging dynamic instrumentation techniques. Paradyn looks for performance problems starting with a root hypothesis. In each step of the search process the currently tested hypothesis is then refined along one of the dimensions of the W^3 (why, where, when) search model. In comparison to Periscope, Paradyn has the advantage of using dynamic instrumentation and is thus able to tailor instrumentation overhead to the current hypothesis. Until recently [3] data analysis and the search for bottlenecks was performed centrally at the tool's front-end, a limiting factor for scalability. In contrast, in Periscope the analysis process is performed inherently distributed by the node-level agents.

6 Conclusion

We have presented Periscope, our tool for automated performance analysis based on the idea of capturing situations of inefficient execution in the form of ASL properties. Using Periscope, developers can quickly locate the most severe inefficiencies and discover their reasons. We have tested Periscope on several OpenMP benchmarks and have discovered bottlenecks of up to 8% in overall execution time.

References

1. Karl Fürlinger and Michael Gerndt. Performance analysis of shared-memory parallel applications using performance properties. In *Proceedings of the 2005 International Conference on High Performance Computing and Communications (HPCC-05)*, September 2005. Accepted for publication.
2. Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam, and Tia Newhall. The Paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, 1995.
3. Philip C. Roth and Barton P. Miller. The distributed performance consultant and the sub-graph folding algorithm: On-line automated performance diagnosis on thousands of processes. 2005. Submitted for Publication.
4. Felix Wolf and Bernd Mohr. Automatic performance analysis of hybrid MPI/OpenMP applications. In *Proceedings of the 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2003)*, pages 13–22. IEEE Computer Society, February 2003.