Automatic Performance Analysis of Hybrid MPI/OpenMP Applications

Felix Wolf Bernd Mohr

Forschungszentrum Jülich
Zentralinstitut für Angewandte Mathematik
52425 Jülich, Germany
{f.wolf, b.mohr}@fz-juelich.de

Abstract

The EXPERT *performance-analysis environment provides a complete tracing-based solution for automatic performance analysis of* MPI*, Open*MP*, or hybrid applications running on parallel computers with* SMP *nodes.* EXPERT *describes performance problems using a high level of abstraction in terms of execution patterns that result from an inefficient use of the underlying programming model(s). The set of supported problems can be extended to meet application-specific needs. The analysis is carried out along three interconnected dimensions: class of performance behavior, call-tree position, and thread of execution. Each dimension is arranged in a hierarchy, so that the user can investigate the behavior on varying levels of detail. All three dimensions are interactively accessible using a single integrated view.*

1 Introduction

Coupling SMP systems combines the packaging efficiencies of shared-memory multiprocessors with the scaling advantages of distributed-memory architectures. The result is a computer architecture that can scale more cost-effectively in size. Unfortunately, these systems come at the price of a more complex programming environment to deal with the different modes of parallel execution: shared-memory multithreading vs. distributed-memory message passing. As a consequence, performance optimization becomes more difficult and creates a need for advanced performance tools that are custom made for this class of computing environments.

While performance tools exist for shared-memory systems and for distributed-memory systems, solving performance problems on parallel computers with SMP nodes is not as simple as combining two tools. When dealing with hybrid (MPI/OpenMP) parallel executions, performance problems arise where an integrated view is required. Current state-of-the-art tools such as VGV [10] can capture and visualize these integrated views, but suffer from performance-information overload, unable to abstract performance problems from detailed performance data in an integrated hybrid framework.

The EXPERT performance-analysis environment[1] is able to automatically detect performance problems in event traces of MPI [16], OpenMP [19], or hybrid applications running on parallel computers with SMP nodes as well as on more traditional non-SMP or single SMP systems.

Performance problems are specified in terms of execution patterns that represent situations of inefficient behavior. These patterns are input for an automatic analysis process that recognizes and quantifies the inefficient behavior in event traces. Mechanisms that hide the complex relationships within compound-event specifications allow a simple description of complex inefficient behavior on a high level of abstraction.

The analysis process automatically transforms the event traces into a three-dimensional representation of performance behavior. The first dimension is the kind of behavior. The second dimension describes the behavior's source-code location and the execution phase during which it occurs.

[1] The work on EXPERT is carried out as a part of the KOJAK project [8, 14] and is embedded in the IST working group APART [20].

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

Finally, the third dimension gives information on the distribution of performance losses across different processes or threads. The hierarchical organization of each dimension enables the investigation of performance behavior on varying levels of granularity. Each point of the representation is uniformly mapped onto the corresponding fraction of execution time, allowing the convenient correlation of different behavior using only a single view. In addition, the set of predefined performance problems can be extended to meet individual (e.g., application-specific) needs.

The remainder of this article is organized as follows: First, we describe the overall architecture of our analysis environment in the next section. In Section 3, we present the abstraction mechanisms used to simplify the specification of complex situations representing inefficient performance behavior. After that, we introduce the actual analysis component and how it can be extended to deal with application specific requirements in Section 4. Section 5 proves our concept by applying it to two realistic codes. Finally, we consider related work and conclude the paper.

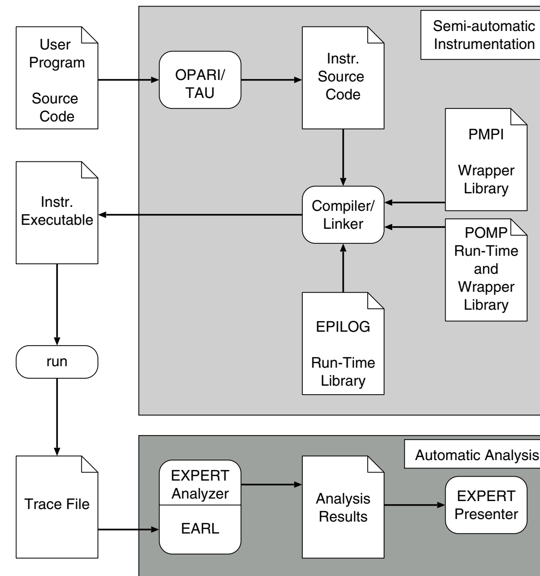

Figure 1. EXPERT overall architecture.

2 Overall Architecture

The EXPERT performance-analysis environment is depicted in Figure 1. The different components are represented as boxes with rounded corners and their inputs and outputs are represented as paper sheets with the upper-right corner turned down. The arrows illustrate the whole performance-analysis process from instrumentation to result presentation.

The EXPERT analysis process is composed of two parts: a semi-automatic multi-level instrumentation of the user application followed by an automatic analysis of the generated performance data. The first subprocess is called semi-automatic because it requires the user to slightly modify the makefile. To begin the process, the user supplies the application's source code, written in either C, C++, or Fortran, to OPARI [18], which performs automatic instrumentation of OpenMP constructs and redirection of OpenMP-library calls to instrumented wrapper functions on the source-code level. Instrumentation of user functions is done either on the source-code level using TAU [21] or using a compiler-supplied profiling interface. Instrumentation for MPI events is accomplished with the PMPI [15] wrapper library, which generates MPI-specific events by intercepting calls to MPI functions. All MPI, OpenMP, and user-function instrumentation call the EPILOG run-time library, which provides mechanisms for buffering and trace-file creation. At the end of the instrumentation process the user has a fully instrumented executable.

Running this executable generates a trace file in the EPILOG format. After program termination, the trace file is fed into the EXPERT analyzer. The analyzer uses EARL [23] to provide a high-level view of the raw trace file. We call this view the *enhanced event model*, and it is where the actual analysis takes place. The analyzer generates an analysis report, which serves as input for the EXPERT presenter.

Currently, the software necessary to generate event traces has been successfully installed on two parallel computers with SMP nodes: the PC-based ZAMpano [13] and the HITACHI SR8000-F1 [3]. Instrumentation is done using the unpublished profiling interface of the PGI [12] compiler or of the proprietary HITACHI compiler [9], respectively. The analysis components run on Linux.

3 Abstraction Mechanisms

EARL maps a raw EPILOG trace of "basic" events onto the enhanced event model. The enhanced event model provides abstractions that al-

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 **IEEE**
IEEE COMPUTER SOCIETY

low compound events representing inefficient behavior to be easily described (see [25] for details). The model considers an event trace as a chronologically sorted sequence of primitive events. Depending on the event type, each event is characterized by a set of attributes. The event types are organized in a hierarchy. There are programming-model–independent event types representing simple region enters and exits. Types indicating point-to-point and collective communication cover the MPI model. OpenMP event types comprise fork and join operations, lock synchronization operations, and - similar to MPI - an event type indicating the collective execution of parallel constructs.

Additionally, EARL provides two types of abstractions on top of the basic part of the model:

- State sequences
- Pointer attributes

State sequences map individual events onto a set of events that represent one aspect of the parallel system's execution state at the moment when the event happens. An example is the *message queue* containing all events of sending messages currently being transferred. Pointer attributes connect corresponding events, so that one can define compound events along a path of corresponding events. An example is is an attribute pointing from a message-receipt event to the corresponding send event. An essential part of the enhanced model is the dynamic call tree, which is computed from all region-enter and region-exit events. As an additional pointer attribute, EARL provides a link from each enter event to the first enter event visiting the same node in the call tree. This provides a simple means to associate a performance-relevant compound event with the corresponding execution phase of the parallel program.

4 Analysis Component

The design of the analyzer is based on the specifications and terminology presented in [6]. The analyzer attempts to prove *performance properties* for one execution of a parallel application and to quantify them according to their influence on the performance. A performance property characterizes a class of performance behavior and is specified in terms of a *compound event*, which the analyzer tries to detect in an event trace. A compound event is a set of events matching a specific execution pattern, whose constituents are connected by relationships and constraints. For each property, EXPERT calculates a *severity* measure indicating the fraction of execution time spent on that property and, thus, allows the correlation of different properties in a single view.

The run-time events of a parallel application occur on multiple time lines - one for each control flow (i.e, thread). EXPERT regards all control flows as being mapped to different CPUs at any time, that is, processes or threads running on the same SMP node do not share a CPU. EXPERT describes the severity of a particular performance property in terms of wall-clock interval sets that may be distributed across different time lines. All interval sets are subsets of the CPU-*reservation time*, which is the time from the first to the last event multiplied by the number of threads. The severity is returned in percentage of the CPU-reservation time.

The analyzer is implemented in Python using EARL for trace access. Its architecture is based on the idea of separating the analysis process from the specification of the performance properties; that is, the performance properties are not hard-coded and specified separately.

4.1 Specification of Performance Properties

The performance properties are specified in form of *patterns*. Patterns are Python classes, which are responsible for detecting compound events indicating inefficient behavior. They provide a common interface making them exchangeable from the perspective of the tool. The specifications use the abstractions provided by EARL and, for this reason, are very simple.

The analysis process follows an event driven approach. EXPERT walks sequentially through the event trace and invokes for each single event call-back methods of the pattern instances and supplies the event as an argument. A pattern can provide a different call-back method for each event type. The call-back method itself then tries to locate a compound event representing an inefficiency, thereby following links (i.e., pointer attributes) emanating from the supplied event or investigating state sequences. This mechanism allows the simple specification of very complex performance-relevant situations and an explanation of inefficiency that is based on the terminology of the programming model.

The common interface also provides a method to launch a configuration dialog for the input of pattern-specific parameters before the analysis process as well as a method to launch a presentation

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

dialog for the display of pattern-specific results afterward, which allows the treatment of pattern-specific performance criteria.

EXPERT organizes the performance properties in a hierarchy. The upper levels of the hierarchy (i.e., those that are closer to the root) correspond to more general behavioral aspects such as time spent in MPI functions. The deeper levels correspond to more specific situations such as time lost due to blocking communication.

Figure 2 shows the complete hierarchy of performance properties being currently supported by EXPERT. We shall briefly discuss some of the most interesting ones in Section 4.1.1 and 4.1.2.

4.1.1 Examples of MPI Performance Properties

Late Sender This property refers to the time wasted, when a call to a blocking receive operation (e.g, `MPI_Recv` or `MPI_Wait`) is posted before the corresponding send operation is executed.

Late Receiver This property refers to the inverse case. A send operation blocks until the corresponding receive operation is called. This can happen for several reasons. Either the MPI implementation is working in synchronous mode by default or the size of the message to be sent exceeds the available MPI-internal buffer space and the operation blocks until the data is transferred to the receiver.

Messages in Wrong Order This property, which has been motivated by [11], deals with the problem of passing messages out of order. For example, the sender may send messages in a certain order, but the receiver may expect their arrival in the reverse order. The implementation locates such situations by querying the message queue each time a message is received and by looking for older messages with the same target as the current message. This situation can be a specialization of either *Late Sender* or *Late Receiver*.

Wait at N x N Collective communication operations that send data from all processes to all processes exhibit an inherent synchronization, that is, no process can finish the operation until the last process has started. The time until all processes have entered the operation is measured and used to compute the severity. Note that this property requires to identify all parts of individual collective-operation instances in the event stream.

4.1.2 Examples of OpenMP Performance Properties

Wait at Barrier The time spent on waiting for the last participant in implicit (i.e., compiler-generated) or explicit (i.e., user-specified) OpenMP barrier synchronization. Note that this property requires to identify all parts of individual barrier instances in the event stream.

Lock Synchronization The time a thread waits for a lock that is owned by another thread. This can occur as a result of OpenMP-library calls or at the entry of critical sections.

Idle Threads Idle times on processors caused by sequential execution before or after an OpenMP parallel region.

4.2 Representation of Performance Behavior

Each applied pattern instance computes a two-dimensional severity matrix, which contains the severity as a function of the node in the dynamic call tree and the location (i.e., thread). Thus, the complete performance behavior is represented using a three-dimensional matrix, where each cell contains the severity for a specific performance property, call-tree node, and location.

The first dimension describes the kind of inefficient behavior. The second dimension describes both its source-code location and the execution phase during which it occurs. Finally, the third dimension gives information on the distribution of performance losses across different processes or threads, which allows to draw additional conclusions (e.g., load imbalance, see also [24]).

In addition, each of the dimensions is arranged in a hierarchy: the performance properties in a hierarchy of general and more specific ones, the call-tree nodes in their evident hierarchy, and the locations in a hierarchy consisting of the levels machine, node, process, and thread. Thus, it is possible to analyze the behavior on different levels of granularity.

4.3 Presentation of Performance Behavior

The user can interactively access each of the hierarchies constituting a dimension of performance

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

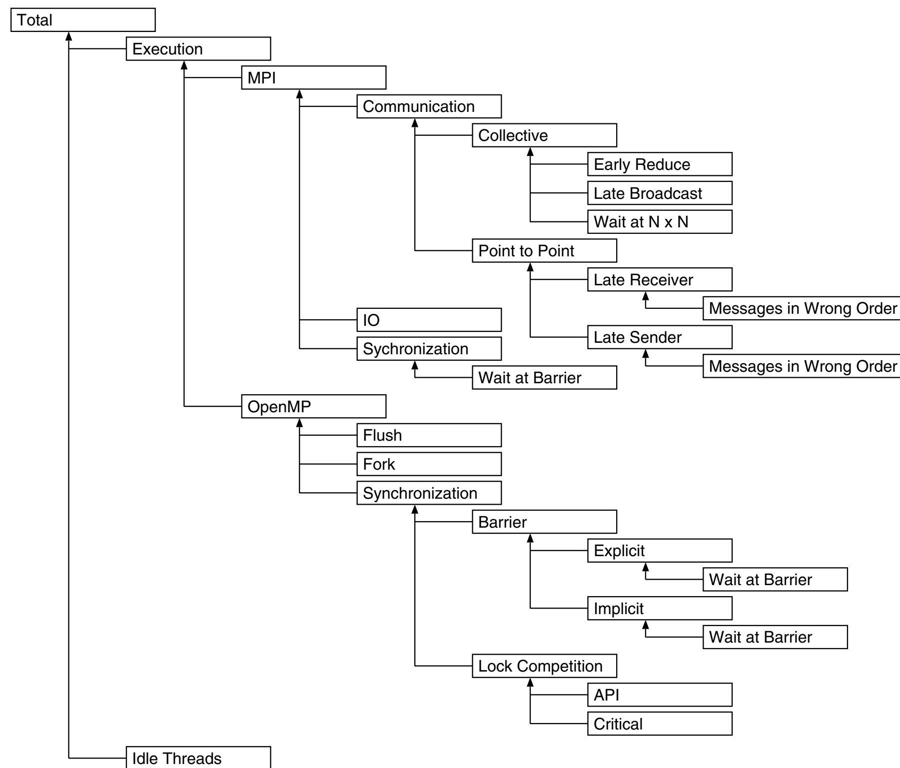

Figure 2. Hierarchy of performance properties.

behavior using a tree browser that labels each node with a weight. EXPERT uses as weight a percentage of the application's CPU-reservation time. The weight that is actually displayed depends on the state of the node, that is, whether it is expanded or collapsed. The weight of a collapsed node represents the whole subtree associated with that node, whereas the weight of an expanded node represents only the fraction that is not covered by its descendants because the weights of its descendants are now displayed separately. This allows the analysis of performance behavior on different levels of granularity.

For example, the call tree may have a node *main* with two children *foo* and *bar* (Figure 3). In the collapsed state, this node is labeled with the weight representing the time spent in the whole program. In the expanded state it displays only the fraction that is spent neither in *foo* nor in *bar*.

The weight is displayed simultaneously using both a numerical value as well as a colored icon. The color is taken from a spectrum representing the whole range of possible weights (i.e., 0 - 100 percent). To avoid distraction, insignificant values below 0.5 percent are displayed in gray. Colors enable the easy identification of nodes of interest even in a large tree, whereas the numerical values enable the precise comparison of individual weights.

Figure 3. Node of the call tree in collapsed and expanded state.

The trees of the different analysis dimensions are interconnected, so that the user can display the call tree with respect to a particular performance property, and the distribution across the locations with respect to a particular node in call tree. In Figure 4, the selections are indicated by framed node labels. Thus, the user can investigate the performance behavior in a scalable but still accurate way along all its interconnected dimensions using only

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY

a single integrated view.

In the default mode, the display represents the severity as a percentage of the total CPU-reservation time. However, always referring to the total CPU-reservation time may limit scalability because values may become very small (e.g., in the case of many locations). For this reason, the presenter offers a *relative view mode*. In this relative view mode, a percentage shown in a tree always refers to the selection in the left neighbor tree.

4.4 Extension Mechanism

EXPERT provides a large set of built-in performance properties, which cover the most frequent inefficiency situations. But sometimes the user may wish to consider application-specific metrics such as iterations or updates per second. In this case, the user can simply write another pattern class that implements an own application-specific performance property according to the common interface of all pattern classes, and place it into a plug-in module.

At startup time, EXPERT dynamically queries the module's name space and looks for newly inserted patterns from which it is now able to build instances. The new patterns are integrated into the graphical user interface and can be used like the predefined ones.

5 Examples

We tested our environment for two realistic code examples, REMO and SWEEP3D, on ZAMpano [13]. Both applications are hybrid MPI/OpenMP applications. CPU reservation was done such that there was one CPU per computational thread or single-threaded process. We consider one event trace per application.

	REMO	SWEEP3D
CPUs	16	16
Size (MB)	170	72
Execution time (sec)	37.2	16.5
Overhead (%)	9.7	6.0
Analysis time (h:m)	9 : 48	3 : 22

Table 1. Trace-file size and overhead.

Table 5 summarizes trace-file size and overhead. The first row contains the program name, the second row shows the number of CPUs used, the third row lists the trace-file size, and the fourth row gives the execution time. To estimate the runtime overhead introduced by the instrumentation, the minimum execution time of a series of ten uninstrumented runs was compared to the minimum execution time of a series of ten instrumented runs. The result is listed in the fifth row. Finally, the last row shows the duration of the analysis process carried out on the test platform.

5.1 REMO

REMO [4] is a weather forecast application of the DKRZ (Deutsches Klima Rechenzentrum). It implements a hydrostatic limited area model, which is based on the *Deutschland/Europa* weather forecast model of the German Meteorological Services (Deutscher Wetterdienst (DWD)). We consider an early experimental MPI/OpenMP version of the production code. The application was executed on four nodes with one process per node and four threads per process (4 processes $\times$ 4 threads).

Figure 4 shows the result display of REMO in the default mode, that is, all values and colors represent percentages of the total CPU-reservation time. The property view indicates that one half (i.e., 51.8 percent) of the total CPU-reservation time is idle time (i.e., *Idle Threads*) resulting from OpenMP sequential execution outside of parallel regions. Although during this period the idle threads actually do not execute any code, the time is mapped onto the call paths that have been executed by the master thread during this time. That is to say, for analysis and presentation purposes EXPERT assumes that outside parallel regions the slave threads "execute" the same code as their master thread. This method of call-path mapping helps to identify parts of the call tree that might be optimized in order to reduce the amount of sequential execution.

In the case of REMO, the EXPERT display (Figure 4, middle) allows the easy identification of two call paths as major sources of idle times. The location view (Figure 4, right) shows the distribution of the idle time across the slave threads.

5.2 SWEEP3D

The benchmark code SWEEP3D [2] represents the core of a real ASCI application. It solves a 1-group time-independent discrete ordinates (Sn) 3D Cartesian (XYZ) geometry neutron transport problem. We consider an early experimental MPI/OpenMP version of the original MPI version. While MPI is responsible for parallelism by domain decomposition, OpenMP is responsible for parallelism by multitasking.

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

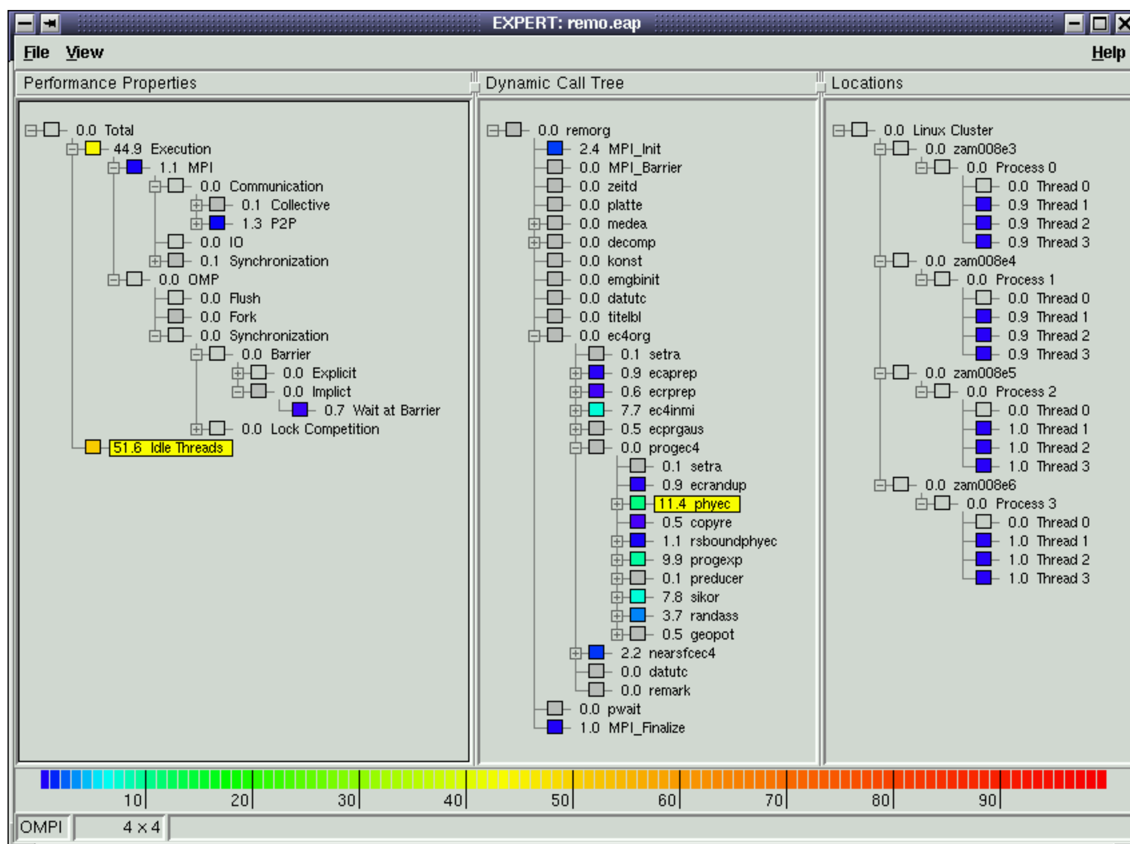

Figure 4. Display of performance behavior in EXPERT for REMO.

The application was executed on four nodes with one process per node and four threads per process (4 processes $\times$ 4 threads). The performance behavior of SWEEP3D exhibits a weak point of hybrid programming, that is, a performance problem resulting from the combination of MPI and OpenMP. MPI calls made outside a parallel region prolong sequential execution and prevent available CPUs from being used by multiple threads. The results are shown in Table 5.2. The call path (a) shown in the table is responsible for most of the losses occurring due to the property *Idle Threads*. However, at the same time this call path exhibits a significant loss due to the property *Late Sender*. Note that *Late Sender* counts the times of the master threads, whereas *Idle Threads* counts the times of the slave threads (3 slaves per master). Taking this into account, reducing *Late Sender* by one percent would speed up the application by four percent because speeding up the master also reduces idle times of the slaves. Obviously, one reason for the *Late Sender* problem at call path (a) is receiving messages in the reverse sending order (*Messages in Wrong Order*). Moreover, a significant amount of time is spent on the implicit (i.e., compiler-generated) OpenMP barrier at the end of call path (b). Expanding the node of the property *Implicit Barrier* (Figure 5, left) reveals that most of that time is lost due to the property *Wait at Barrier*.

6 Related Work

Miller and associates [17] developed automatic on-line performance analysis according to the W^3 Search Model in the well-known Paradyn project. In contrast to our approach, the W^3 model describes performance behavior along the dimensions performance problem, program resource, and time. Performance problems are expressed in terms of a threshold and one or more metrics such as CPU time, blocking time, message rates, I/O rates, or number of active processors. The main

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

Call Paths			
(a) `seep3d → inner_auto → inner → sweep → recv_real → MPI_Recv`			
(b) `driver → inner_auto → inner/sweep → !$omp parallel → !$omp do → !$omp ibarrier`			

Performance Property	Whole Program	(a)	(b)
Idle Threads	37.5	17.5	
Communication	6.5	5.8	
Late Sender	3.2	3.2	
Implicit Barrier (OpenMP)	4.3		3.3
Wait at Barrier (OpenMP, implicit)	2.8		2.6

Table 2. Performance problems found in SWEEP3D in percentage of the total CPU-reservation time.

accomplishments of EXPERT in contrast to Paradyn is the description of performance problems in terms of complex event patterns that go beyond counter-based metrics. Also, the uniform mapping of arbitrary performance behavior onto the CPU-reservation time allows the correlation of different behavior in a single view.

Espinosa [5] implemented an automatic trace analysis tool KAPPA-PI for evaluating the performance behavior of MPI and PVM message-passing programs. Here, behavior classification is carried out in two steps. At first, a list of idle times is generated from the raw trace file using a simple metrics. Then, based on this list, a recursive inference process continuously deduces new facts on an increasing level of abstraction. Finally, recommendations on possible sources of inefficiencies are built from the facts being proved on the one hand and from the results of source-code analysis on the other hand.

Vetter [22] performs automatic performance analysis of MPI point-to-point communication based on machine learning techniques. He traces individual message-passing operations and then, classifies each individual communication event using a decision tree. The decision tree has been previously trained by microbenchmarks that demonstrate both efficient as well as inefficient performance behavior. As opposed to this approach, EXPERT draws conclusions from the temporal relationships of individual events in a platform-independent way, which does not require any training prior to analysis.

JavaPSL [7] has been designed by Fahringer and associates to specify performance properties based on the Java programming language. Whereas EXPERT uses Python to provide a uniform interface to performance properties, JavaPSL exploits similar mechanisms of the Java language, such as polymorphism, abstract classes, and reflection. In contrast to EXPERT, which concentrates on compound-event analysis, JavaPSL puts emphasis on the definition of performance properties based on existing ones (e.g., by defining metaproperties).

7 Conclusion

The EXPERT tool environment provides a complete but still extensible solution for automatic performance analysis of MPI, OpenMP, or hybrid applications running on parallel computers with SMP nodes. EXPERT represents performance properties on a very high level of abstraction that goes beyond simple metrics and provides the ability to explain performance problems in terms of the underlying programming model(s). The set of performance-property specifications is embedded in a flexible architecture and can be extended to meet application-specific needs.

The performance behavior is presented along three interconnected dimensions: class of performance behavior, position within the call tree and thread of execution. The last dimensions allows even the effects of different communication patterns among subdomains to be investigated. Each dimension is arranged in a hierarchy, so that the user can view the behavior on varying levels of detail. In particular, the hierarchical structure of hybrid applications and SMP-cluster hardware is reflected this way. Each point of the representation is uniformly mapped onto the corresponding fraction of CPU-reservation time, allowing the convenient correlation of different behavior in a single integrated view. The user can access all three dimensions interactively using a scalable but still accurate tree display. Colors make it easy to identify

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 **IEEE**

IEEE COMPUTER SOCIETY

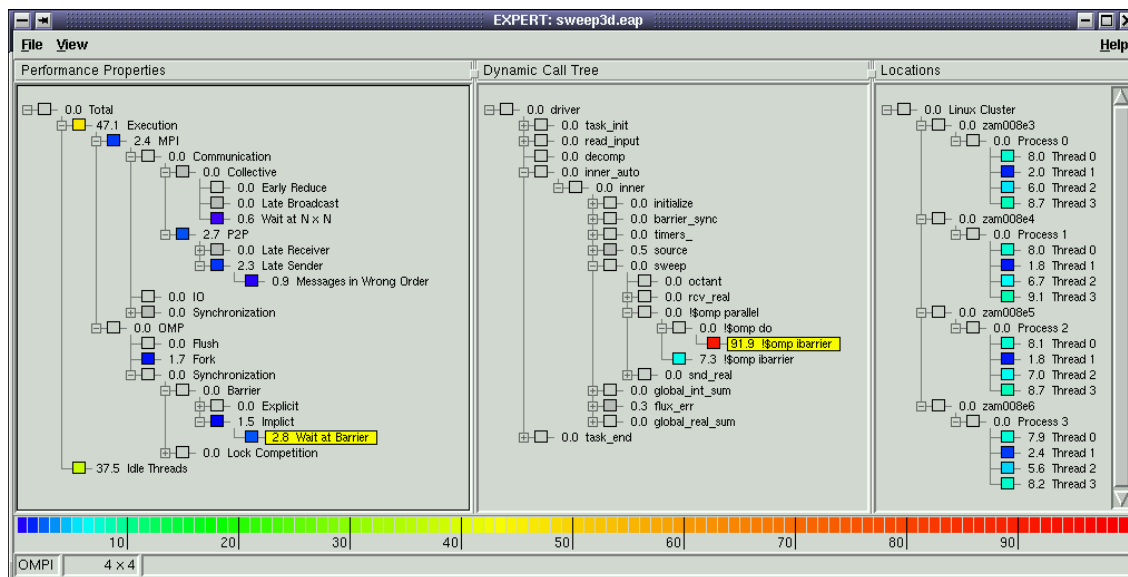

Figure 5. Display of performance behavior in EXPERT for SWEEP3D in the relative view mode.

interesting nodes even in case of large trees.

EXPERT is well suited to analyze a single trace file. But the development process of parallel applications often demands for comparison of trace files representing different execution configurations or development versions. For the future, we intend to integrate mechanisms for comparative performance analysis. In addition, we plan to improve our result presentation by integrating it with an event-trace browser such as VAMPIR [1] to visualize instances of performance problems using time-line diagrams and by adding source-code displays. Finally, we will work on further improving and completing our performance-property catalog.

8 Acknowledgements

We would like to thank all our partners in the IST working group APART for their contributions to this topic. We also would like to thank Arpad Kiss for providing the basic tree browser implementation, DKRZ for giving us access to their application, and Reiner Vogelsang for helping us in conducting our experiments. Finally, we would like to thank Allen Malony and Craig Soules for their helpful comments and suggestions on the language.

References

[1] A. Arnold, U. Detert, and W.E. Nagel. Performance Optimization of Parallel Programs: Tracing, Zooming, Understanding. In R. Winget and K. Winget, editors, *Proc. of Cray User Group Meeting*, pages 252–258, Denver, CO, March 1995.

[2] Accelerated Strategic Computing Initiative [ASCI]. The ASCI sweep3d Benchmark Code. http://www.llnl.gov/asci_benchmarks/.

[3] Leibnitz Rechenzentrum der Bayerischen Akademie der Wissenschaften. HITACHI SR 8000-F1. http://www.lrz-muenchen.de/.

[4] T. Diehl and V. Gülzow. Performance of the Parallelized Regional Climate Model REMO. In *Proc. of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meterology*, pages 181–191, Reading, UK, November 1998. European Centre for Medium-Range Weather Forecasts.

[5] A. Espinosa. *Automatic Performance Analysis of Parallel Programs*. PhD thesis, Universitat Autonoma de Barcelona, September 2000.

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

[6] T. Fahringer, M. Gerndt, B. Mohr, F. Wolf, G. Riley, and J. L. Träff. Knowledge Specification for Automatic Performance Analysis. Technical Report FZJ-ZAM-IB-2001-08, ESPRIT IV Working Group APART, Forschungszentrum Jülich, August 2001. Revised version.

[7] T. Fahringer and C. Seragiotto Junior. Modelling and Detecting Performance Problems for Distributed and Parallel Programs with JavaPSL. In *Proc. of the Conference on Supercomputers (SC2001)*, Denver, Colorado, November 2001.

[8] M. Gerndt, B. Mohr, M. Pantano, and F. Wolf. Performance Analysis for CRAY T3E. In *Proc. of the 7th Euromicro Workshop on Parallel and Disributed Pocessing (PDP'99)*, pages 241–248, 1999.

[9] HITACHI. *HITACHI SR 8000 Compiler*. Technical Manual.

[10] J. Hoeflinger, B. Kuhn, W. Nagel, P. Petersen, H. Rajic, S. Shah, J. Vetter, M. Voss, and R. Woo. An Integrated Performance Visualizer for MPI/OpenMP Programs. In *Proc. of the 3rd European Workshop on OpenMP (EWOMP 2001)*, Barcelona, Spain, September 2001.

[11] J. K. Hollingsworth and M. Steele. Grindstone: A Test Suite for Parallel Performance Tools. Computer Science Technical Report CS-TR-3703, University of Maryland, October 1996.

[12] Portland Group Inc. Product documentation. http://www.pgroup.com/docs.htm.

[13] Forschungszentrum Jülich. ZAMpano (ZAM Parallel Nodes). http://zampano.zam.kfa-juelich.de/.

[14] Research Centre Jülich. KOJAK (Kit for Objective Judgement and Knowledge-based Detection of Performance Bottlenecks). http://www.fz-juelich.de/zam/kojak/.

[15] Message Passing Interface Forum. MPI: A Message Passing Interface Standard, Juni 1995. http://www.mpi-forum.org.

[16] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface, Juli 1997. http://www.mpi-forum.org.

[17] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvine, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The Paradyn Parallel Performance Measurement Tool. *IEEE Computer*, 28(11):37–46, 1995.

[18] B. Mohr, A. Malony, S. Shende, and F. Wolf. Design and Prototype of a Performance Tool Interface for OpenMP. *The Journal of Supercomputing*, 23:105–128, 2002.

[19] OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface - Version 2.0, November 2000. http://www.openmp.org.

[20] IST Working Group APART (Automatic Performance Analysis: Real and Tools). Homepage. http://www.fz-juelich.de/apart/.

[21] S. Shende, A. D. Malony, J. Cuny, K. Lindlan, P. Beckman, and S. Karmesin. Portable Profiling and Tracing for Parallel Scientific Applications using C++. In *Proc. of the SIGMETRICS Symposium on Parallel and Distributed Tools*, pages 134–145. ACM, August 1998.

[22] J. Vetter. Performance Analysis of Distributed Applications using Automatic Classification of Communication Inefficencies. In *Proc. of the 14th International Conference on Supercomputing*, pages 245–254, Santa Fe, New Mexico, May 2000.

[23] F. Wolf. EARL - Eine programmierbare Umgebung zur Bewertung paralleler Prozesse auf Message-Passing-Systemen. Master's thesis, RWTH Aachen, Forschungszentrum Jülich, Jül-Bericht 3551, June 1998.

[24] F. Wolf and B. Mohr. Automatic Performance Analysis of MPI Applications Based on Event Traces. In *Proc. of the European Conference on Parallel Computing (Euro-Par)*, pages 123–132, Munich (Germany), August 2000.

[25] F. Wolf and B. Mohr. Specifying Performance Properties of Parallel Applications Using Compund Events. *Parallel and Distributed Computing Practices (Special Issue on Monitoring Systems and Tool Interoperability)*, In press.

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE