SCALEA: A Performance Analysis Tool for Distributed and Parallel Programs *

Hong-Linh Truong, Thomas Fahringer

Institute for Software Science, University of Vienna
Liechtensteinstr. 22, A-1090 Vienna, Austria
{truong,tf}@par.univie.ac.at

Published in Proc. of the 8th International Euro-Par Conference(Euro-Par 2002) , Paderborn, Germany, August 2002

Abstract. In this paper we present SCALEA, which is a performance instrumentation, measurement, analysis, and visualization tool for parallel and distributed programs that supports post-mortem and online performance analysis. SCALEA currently focuses on performance analysis for OpenMP, MPI, HPF, and mixed parallel/distributed programs. It computes a variety of performance metrics based on a novel overhead classification. SCALEA also supports multiple experiment performance analysis that allows to compare and to evaluate the performance outcome of several experiments. A highly flexible instrumentation and measurement system is provided which can be controlled by command-line options and program directives. SCALEA can be interfaced by external tools through the provision of a full Fortran90 OpenMP/MPI/HPF frontend that allows to instrument an abstract syntax tree at a very high-level with C-function calls and to generate source code. A graphical user interface is provided to view a large variety of performance metrics at the level of arbitrary code regions, threads, processes, and computational nodes for single- and multi-experiments.
Keywords: performance analysis, instrumentation, performance overheads

1 Introduction

The evolution of distributed/parallel architectures and programming paradigms for performance-oriented program development challenge the state of technology for performance tools. Coupling different programming paradigms such as message passing and shared memory programming for hybrid cluster computing (e.g. SMP clusters) is one example for high demands on performance analysis that is capable to observe performance problems at all levels of a system while relating low-level behavior to the application program.

In this paper we describe SCALEA, a performance instrumentation, measurement, and analysis system for distributed and parallel architectures that currently focuses on OpenMP, MPI, HPF programs, and mixed programming paradigms such as OpenMP/MPI. SCALEA seeks to explain the performance behavior of each program by computing a variety of performance metrics based on

* This research is supported by the Austrian Science Fund as part of the Aurora Project under contract SFBF1104.

a novel classification of performance overheads for shared and distributed memory parallel programs which includes data movement, synchronization, control of parallelism, additional computation, loss of parallelism, and unidentified overheads. In order to determine overheads, SCALEA divides the program sources into code regions (ranging from the entire program to single statement) and locates whether performance problems occur in those regions or not. A highly flexible instrumentation and measurement system is provided which can be precisely controlled by program directives and command-line options. In the center of SCALEA's performance analysis is a novel dynamic code region call graph (DRG - [9]) which reflects the dynamic relationship between code regions and their subregions and enables a detailed overhead analysis for every code region. Moreover, SCALEA supports a high-level interface to traverse an abstract syntax tree (AST), to locate arbitrary code regions, and to mark them for instrumentation. The SCALEA overhead analysis engine can be used by external tools as well.

A data repository is employed in order to store performance data and information about performance experiments which alleviates the association of performance information with experiments and the source code. SCALEA also supports multi-experiment performance analysis that allows to examine and compare the performance outcome of different program executions. A sophisticated visualization engine is provided to view the performance of programs at the level of arbitrary code regions, threads, processes, and computational nodes (e.g. single-processor systems, Symetric Multiple Processor (SMP) nodes sharing a common memory, etc.) for single- and multi-experiments.

The rest of this paper is organized as follows: Section 2 presents an overview of SCALEA. In Section 3 we present a classification of performance overheads. The next section outlines the various instrumentation mechanisms offered by SCALEA. The performance data repository is described in the following section. Experiments are shown in Section 6. Related work is outlined in Section 7, followed by conclusions in Section 8.

2 SCALEA Overview

SCALEA is a performance instrumentation, measurement, and analysis system for distributed memory, shared memory, and mixed parallel programs. Figure 1 shows the architecture of SCALEA which consists of several components: SCALEA Instrumentation System (SIS), SCALEA Runtime System, SCALEA Performance Data Repository, and SCALEA Performance Analysis & Visualization System. All components provide open interfaces thus they can be used by external tools as well.

SIS uses the front-end and unparser of the VFC compiler [1]. SIS supports automatic instrumentation of MPI, OpenMP, HPF, and mixed OpenMP/MPI programs. The user can select (by directives or command-line options) code regions and performance metrics of interest. Moreover, SIS offers an interface for other tools to traverse and annotate the AST at a high level in order to specify code regions for which performance metrics should be obtained. SIS also generates an *instrumentation description file* [9] to relate all gathered performance data back to the input program.

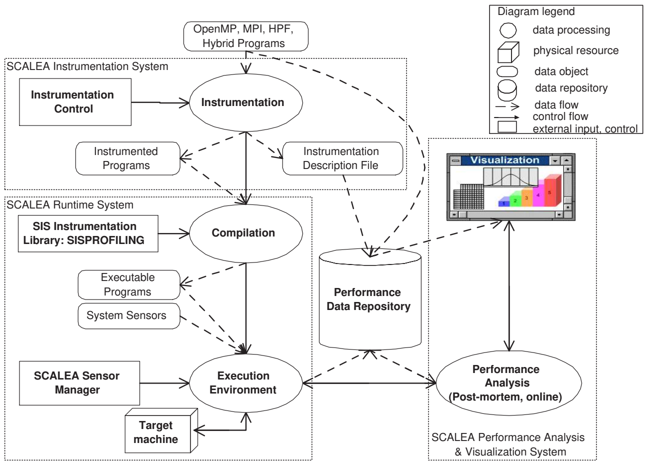

Fig. 1. Architecture of SCALEA

The SCALEA runtime system supports profiling and tracing for parallel and distributed programs, and sensors and sensor managers for capturing and managing performance data of individual computing nodes of parallel and distributed machines. The SCALEA profiling and tracing library collects timing, event, and counter information, as well as hardware parameters. Hardware parameters are determined through an interface with the PAPI library [2].

The SCALEA performance analysis and visualization module analyzes the raw performance data which is collected post-mortem or online and stored in the performance data repository. It computes all user-requested performance metrics, and visualizes them together with the input program. Besides single-experiment analysis, SCALEA also supports multi-experiment performance analysis. The visualization engine provides a rich set of displays for various metrics in isolation or together with the source code.

The SCALEA performance data repository holds relevant information about the experiments conducted.

In the following we provide a more detailed overview of SCALEA.

3 Classification of Temporal Overheads

In previous work [9], we presented a preliminary and very coarse grain classification of performance overheads which has been stimulated by [3]. Figure 2 shows our novel and substantially refined overhead classification which includes:

– *Data movement* shown in Fig. 2(b) corresponds to any data transfer within local memory (e.g. cache misses and page faults), file I/O, communication (e.g. point to point or collective communication), and remote memory access (e.g. put and get). Note that the overhead *Communication of Accumulate Operation* has been stimulated by the *MPI_Accumulate* construct which is employed to move and combine (through reduction operations) data at remote sites via remote memory access.

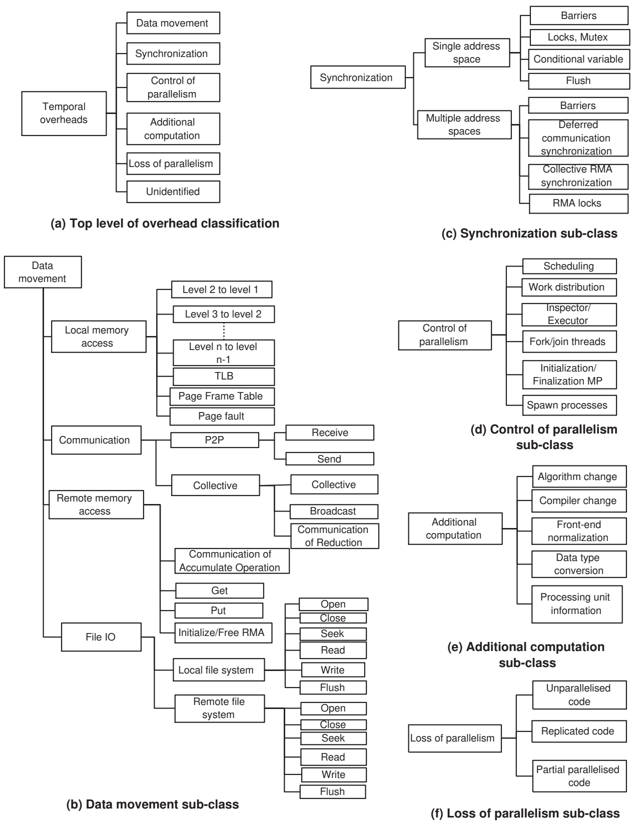

Fig. 2. Temporal overheads classification

- *Synchronization* (e.g. barriers and locks) shown in Fig. 2(c) is used to coordinate processes and threads when accessing data, maintaining consistent computations and data, etc. We subdivided the synchronization overhead into single address and multiple address-space overheads. A single address space corresponds to memory parallel systems. For instance, any kind of OpenMP synchronization falls into this category. Whereas multi-address space synchronization has been stimulated by MPI synchronization, remote memory locks, barriers, etc.
- *Control of parallelism* (e.g. fork/join operations and loop scheduling) shown in Fig. 2(d) is used to control and to manage the parallelism of a program which is commonly caused by code inserted by the compiler (e.g. runtime library) or by the programmer (e.g. to implement data redistribution).
- *Additional computation* (see Fig. 2(e)) reflects any change of the original sequential program including algorithmic or compiler changes to increase par-

allelism (e.g. by eliminating data dependences) or data locality (e.g. through changing data access patterns). Moreover, requests for processing unit identifications, or the number of threads that execute a code region may also imply additional computation overhead.

- *Loss of parallelism* (see Fig. 2(f)) is due to imperfect parallelization of a program which can be further classified: unparallelized code (executed by only one processor), replicated code (executed by all processors), and partially parallelized code (executed by more than one but not all processors).
- *Unidentified overhead* corresponds to the overhead that is not covered by the above categories.

4 SCALEA Instrumentation System (SIS)

SIS provides the user with three alternatives to control instrumentation which includes command-line options, SIS directives, and an instrumentation library combined with an OpenMP/HPF/MPI frontend and unparser. All of these alternatives allow the specification of performance metrics and code regions of interest for which SCALEA automatically generates instrumentation code and determines the desired performance values during or after program execution. In the remainder of this paper we assume that a code region refers to a single-entry single-exit code region. A large variety of predefined mnemonics are provided by SIS for instrumentation purposes. The current implementation of SCALEA supports 49 code region and 29 performance metric mnemonics:

- code region mnemonics: arbitrary code regions, loops, outermost loops, procedures, I/O statements, HPF INDEPENDENT loops, HPF redistribution, OpenMP parallel loops, OpenMP sections, OpenMP critical, MPI send, receive, and barrier statements, etc.
- performance metric mnemonics: wall clock time, cpu time, communication overhead, cache misses, barrier time, synchronization, scheduling, compiler overhead, unparallelized code overhead, HW-parameters, etc. See also Fig. 2 for a classification of performance overheads considered by SCALEA.

The user can specify arbitrary code regions ranging from the entire program unit to single statements and name (to associate performance data with code regions) these regions which is shown in the following:

!SIS$ CR *region_name* **BEGIN**
    code region
!SIS$ END CR

In order to specify a set of code regions $R = \{r_1, ..., r_n\}$ in an enclosing region r and performance metrics which should be computed for every region in R, SIS offers the following directive:

!SIS$ CR *region_name* [,*cr_mnem-list*] [**PMETRIC** *perf_mnem-list*] **BEGIN**
    code region r that includes all regions in R
!SIS$ END CR

The code region r defines the *scope* of the directive. Note that every (code) region in R is a sub-region of r but r may contain sub-regions that are not in R.

The code region (*cr_mnem-list*) and performance metric (*perf_mnem-list*) mnemonics are indicated as a list of mnemonics separated by commas. One of the code region mnemonics (CR_A) refers to arbitrary code regions. Note that the above specified directive allows to indicate either only code region mnemonics or performance metric mnemonics, or a combination of both. If in a SIS directive d only code region mnemonics are indicated, then SIS is instrumenting all code regions that correspond to these mnemonics inside of the scope of d. The instrumentation is done for a set of default performance metrics which can be overwritten by command-line options.

If only performance metric mnemonics are indicated in a directive d then SIS is instrumenting those code regions that have an impact on the specified metrics. This option is useful if a user is interested in specific performance metrics but doesn't know which code regions may cause these overheads. If both code region and performance metrics are defined in a directive d, then SIS is instrumenting these code regions for the indicated performance metrics in the scope of d. Feasibility checks are conducted by SIS, for instance, to determine whether the programmer is asking for OpenMP overheads in HPF code regions. For these cases, SIS outputs appropriate warnings.

All previous directives are called local directives as the scope of these directives is restricted to part of a program unit (main program, subroutines or functions). The scope of a directive can be extended a full program unit by using the following syntax:

!SIS$ CR [*cr_mnem-list*] [**PMETRIC** *perf_mnem-list*]

A global directive d collects performance metrics – indicated in the `PMETRIC` part of d – for all code regions – specified in the `CR` part of d – in the program unit which contains d. A local directive implies the request for performance information restricted to the scope of d. There can be nested directives with arbitrary combinations of global and local directives. If different performance metrics are requested for a specific code region by several nested directives, then the union of these metrics is determined. SIS supports command-line options to instrument specific code regions for well-defined performance metrics in an entire application (across all program units).

Moreover, SIS provides specific directives in order to control tracing/profiling. The directives MEASURE ENABLE and MEASURE DISABLE allow the programmer to turn on and off tracing/profiling of a specific code region.

!SIS$ MEASURE DISABLE
code region
!SIS$ MEASURE ENABLE

SCALEA also provides an interface that can be used by other tools to exploit SCALEA's instrumentation, analysis and visualization features. We have developed a C-library to traverse the AST and to mark arbitrary code regions for instrumentation. For each code region, the user can specify the performance metrics of interest. Based on the annotated AST, SIS automatically generates an instrumented source code.

In the following example we demonstrate some of the directives as mentioned above by showing a fraction of the application code of Section 6.

```
d1:   !SIS$ CR PMETRIC ODATA_SEND, ODATA_RECV, ODATA_COL
      call MPI_BCAST(nx, 1,MPI_INTEGER, mpi_master,MPI_COMM_WORLD,mpi_err)
      ...
d2:   !SIS$ CR comp_main, CR_A, CR_S PMETRIC WTIME, L2_TCM BEGIN
          ...
d3:       !SIS$ CR init_comp BEGIN
               dj=real(nx,b8)/real(nodes_row,b8)
               ...
d4:       !SIS$ END CR
               ...
d5:       !SIS$ MEASURE DISABLE
               call bc(psi,i1,i2,j1,j2)
d6:       !SIS$ MEASURE ENABLE
               ...
               call do_force(i1,i2,j1,j2)
               ...
d7:   !SIS$ END CR
```

Directive d_1 is a global directive which instructs SIS to instrument all send, receive and collective communication statements in this program unit. Directives d_2 (begin) and d_7 (end) define a specific code region with the name *comp_main*. Within this code region *comp_main*, SCALEA will determine wall clock times (*WTIME*) and the total number of L2 cache misses (*L2_TCM*) for all arbitrary code regions (based on mnemonic *CR_A*) and subroutine calls (mnemonic *CR_S*) as specified in d_2. Directives d_3 and d_4 specify an arbitrary code region with the name *init_comp*. No instrumentation as well as measurement is done for the code region between directives d_5 and d_6.

5 Performance Data Repository

A key concept of SCALEA is to store the most important information about performance experiments including application, source code, machine information, and performance results in a data repository. Figure 3 shows the structure

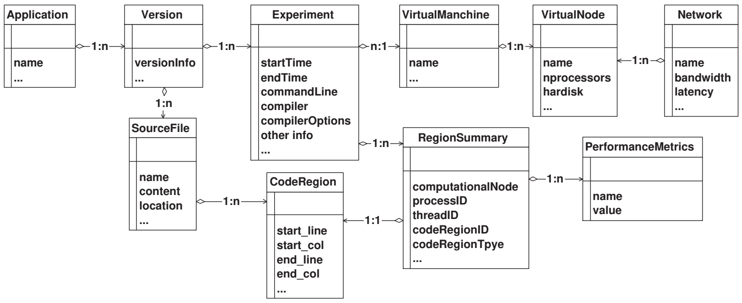

Fig. 3. SCALEA Performance Data Repository

of the data stored in SCALEA's performance data repository. An *experiment* refers to a sequential or parallel execution of a program on a given target architecture. Every experiment is described by *experiment-related data*, which includes information about the application code, the part of a machine on which the code has been executed, and performance information. An application (program) may have a number of implementations (code versions), each of them consists of a set of source files and is associated with one or several experiments.

Every source file has one or several static code regions (ranging from the entire program unit to single statement), uniquely specified by *startPos* and *endPos* (position – start/end line and column – where the region begins and ends in the source file). Experiments are associated with the virtual machines on which they have been taken. The virtual machine is part of a physical machine available to the experiment; it is described as a set of computational nodes (e.g. single-processor systems, Symetric Multiple Processor (SMP) nodes sharing a common memory, etc.) connected by a specific network. A region summary refers to the performance information collected for a given code region and processing unit (process or thread) on a specific virtual node used by the experiment. The region summaries are associated with performance metrics that comprise performance overheads, timing information, and hardware parameters. Moreover, most data can be exported into XML format which further facilitates accessing performance information by other tools (e.g. compilers or runtime systems) and applications.

6 Experiments

SCALEA as shown in Fig. 1 has been fully implemented. Our analysis and visualization system is implemented in Java which greatly improves their portability. The performance data repository uses PostgreSQL and the interface between SCALEA and the data repository is realized by Java and JDBC. Due to space limits we restrict the experiments shown in this section to a few selected features for post-mortem performance analysis.Our experimental code is a mixed OpenMP/MPI Fortran program that is used for ocean simulation. The experiments have been conducted on an SMP cluster with 16 SMP nodes (connected by Myrinet) each of which comprises 4 Intel Pentium III 700 MHz CPUs.

6.1 Overhead Analysis for a Single Experiment

SCALEA supports the user in the effort to examine the performance overheads for a single experiment of a given program. Two modes are provided for this analysis. Firstly, the *Region-to-Overhead* mode (see the "Region-to-Overhead" window in Fig. 4) allows the programmer to select any code region instance in the DRG for which all detected performance overheads are displayed. Secondly, the *Overhead-to-Region* mode (see the "Overhead-to-Region" window in Fig. 4) enables the programmer to select the performance overhead of interest, based on which SCALEA displays the corresponding code region(s) in which this overhead occurs. This selection can be limited to a specific code region instance, thread or process. For both modi the source code of a region is shown if the code region instance is selected in the DRG by a mouse click.

6.2 Multiple Experiments Analysis

Most performance tools investigate the performance for individual experiments one at a time. SCALEA goes beyond this limitation by supporting also performance analysis for multiple experiments. The user can select several experiments and performance metrics of interest whose associated data are stored in the data repository. The outcome of every selected metric is then analyzed and visualized

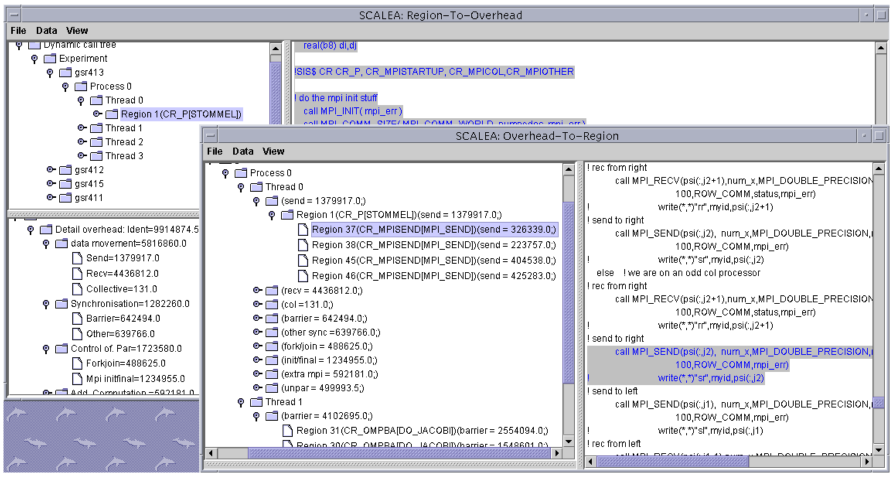

Fig. 4. Region-To-Overhead and Overhead-To-Region DRG View

for all experiments. For instance, in Fig. 5 we have selected 6 experiments (see x-axis in the left-most window) and examine the wall clock, user, and system times for each of them. We believe that this feature is very useful for scalability analysis of individual metrics for changing problem and machine sizes.

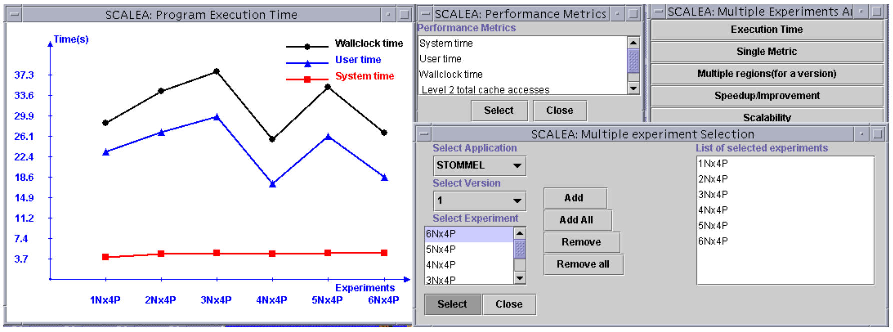

Fig. 5. Multiple Experiment Analysis

7 Related Work

Significant work has been done by Paradyn [6], TAU [5], VAMPIR [7], Pablo toolkit [8], and EXPERT [10]. SCALEA differs from these approaches by providing a more flexible mechanism to control instrumentation for code regions and performance metrics of interest. Although Paradyn enables dynamic insertion of probes into a running code, Paradyn is currently limited to instrumentation of subroutines and functions, whereas SCALEA can instrument - at compile-time only - arbitrary code regions including single statements. Moreover, SCALEA differs by storing experiment-related data to a data repository, by providing multiple instrumentation options (directives, command-line options, and high-level

AST instrumentation), and by supporting also multi-experiment performance analysis.

8 Conclusions and Future Work

In this paper, we described SCALEA, which is a performance analysis tool for OpenMP/MPI/HPF and mixed parallel programs. The main contributions of this paper are centered around a novel design of the SCALEA architecture, new instrumentation directives, a substantially improved overhead classification, a performance data repository, a visualization engine, and the capability to support both single- and multi-experiment performance analysis.

Currently, SCALEA is extended for online monitoring for Grid applications and infrastructures. SCALEA is part of the ASKALON programming environment and tool set for cluster and Grid architectures [4]. SCALEA is used by various other tools in ASKALON to support automatic bottleneck analysis, performance experiment and parameter studies, and performance prediction.

References

1. S. Benkner. VFC: The Vienna Fortran Compiler. *Scientific Programming, IOS Press, The Netherlands*, 7(1):67–81, 1999.
2. S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceeding SC'2000*, November 2000.
3. J.M. Bull. A hierarchical classification of overheads in parallel programs. In *Proc. 1st International Workshop on Software Engineering for Parallel and Distributed Systems*, pages 208–219. Chapman Hall, March 1996.
4. T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, and H.-L. Truong. ASKALON - A Programming Environment and Tool Set for Cluster and Grid Computing. www.par.univie.ac.at/project/askalon, Institute for Software Science, University of Vienna.
5. Allen Malony and Sameer Shende. Performance technology for complex parallel and distributed systems. In *3rd Intl. Austrian/Hungarian Workshop on Distributed and Parallel Systems*, pages 37–46. Kluwer Academic Publishers, Sept. 2000.
6. B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, November 1995.
7. W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, Jan. 1996.
8. D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proc. Scalable Parallel Libraries Conf.*, pages 104–113. IEEE Computer Society, 1993.
9. Hong-Linh Truong, Thomas Fahringer, Georg Madsen, Allen D. Malony, Hans Moritsch, and Sameer Shende. On Using SCALEA for Performance Analysis of Distributed and Parallel Programs. In *Proceeding SC'2001*, Denver, USA, November 2001. IEEE/ACM.
10. Felix Wolf and Bernd Mohr. Automatic Performance Analysis of MPI Applications Based on Event Traces. *Lecture Notes in Computer Science*, 1900:123–??, 2001.