# KOJAK – A Tool Set for Automatic Performance Analysis of Parallel Programs

Bernd Mohr[1] and Felix Wolf[2]

[1] Forschungszentrum Jülich,
Zentralinstitut für Angewandte Mathematik,
52425 Jülich, Germany
b.mohr@fz-juelich.de

[2] Innovative Computing Laboratory,
Computer Science Department,
University of Tennessee,
Knoxville, TN 37996
fwolf@cs.utk.edu

**Abstract.** Today's parallel computers with SMP nodes provide both multithreading and message passing as their modes of parallel execution. As a consequence, performance analysis and optimization becomes more difficult and creates a need for advanced performance tools that are custom made for this class of computing environments. Current state-of-the-art tools provide valuable assistance in analyzing the performance of MPI and OpenMP programs by visualizing the runtime behavior and calculating statistics over the performance data. However, the developer of parallel programs is still required to filter out relevant parts from a huge amount of low-level information shown in numerous displays and map that information onto program abstractions without tool support.

The KOJAK project (Kit for Objective Judgement and Knowledge-based Detection of Performance Bottlenecks) is aiming at the development of a generic automatic performance analysis environment for parallel programs. Performance problems are specified in terms of execution patterns that represent situations of inefficient behavior. These patterns are input for an analysis process that recognizes and quantifies the inefficient behavior in event traces. Mechanisms that hide the complex relationships within event pattern specifications allow a simple description of complex inefficient behavior on a high level of abstraction.

The analysis process 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. 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.

## 1 Short Description of the KOJAK Tool Set

Figure 1 gives an overview about the architecture of the current prototype and its components. The KOJAK 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 and execute the application manually.

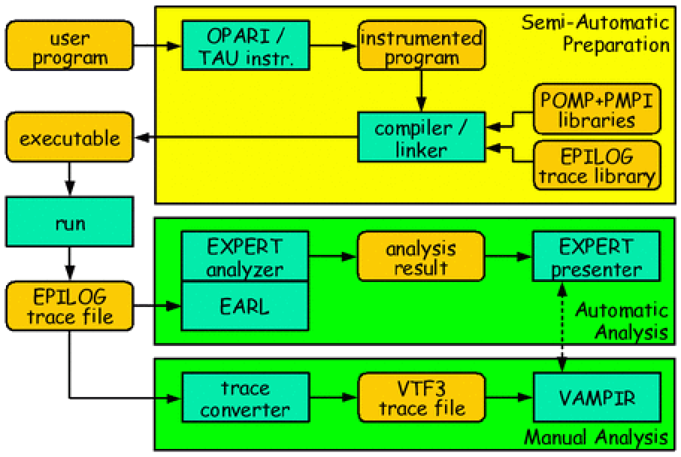
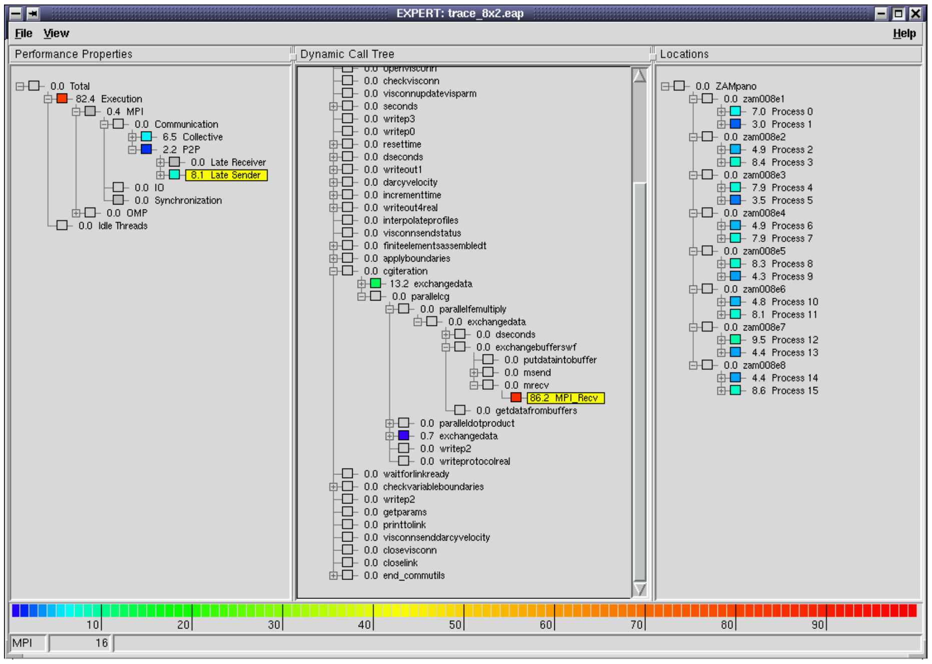


**Fig. 1.** KOJAK tool architecture

To begin the process, the user supplies the application's source code, written in either C, C++, or Fortran, to OPARI (OpenMP Pragma And Region Instrumentor), which performs automatic instrumentation of OpenMP constructs and redirection of OpenMP-library calls to instrumented wrapper functions on the source-code level based on the POMP API [6]. Instrumentation of user functions is done either on the source-code level using TAU or using a compiler-supplied profiling interface. Instrumentation for MPI events is accomplished using a PMPI wrapper library, which generates MPI-specific events by intercepting calls to MPI functions. All MPI, OpenMP, and user-function instrumentation call the EPILOG (Event Processing, Investigating and LOGging) 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 (Extensible Performance Tool) analyzer. The analyzer uses EARL (Event Analysis and Recognition Language) 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. Figure 2 shows a screendump of the EXPERT presenter.



```
0.0 Total
    82.4 Execution
        0.4 MPI
            0.0 Communication
                6.5 Collective
                2.2 P2P
                    0.0 Late Receiver
                    8.1 Late Sender
            0.0 IO
            0.0 Synchronization
        0.0 OMP
    0.0 Idle Threads
```

```
0.0 checkvisconn
0.0 visconnupdatevisparm
0.0 seconds
0.0 writep3
0.0 writep0
0.0 resettime
0.0 dseconds
0.0 writeout1
0.0 darcyvelocity
0.0 incrementtime
0.0 writeout4real
0.0 interpolateprofiles
0.0 visconnsendstatus
0.0 finiteelementsassembledt
0.0 applyboundaries
0.0 cgiteration
    13.2 exchangedata
    0.0 parallelcg
        0.0 parallelfemultiply
            0.0 exchangedata
                0.0 dseconds
                0.0 exchangebufferswf
                    0.0 putdataintobuffer
                    0.0 msend
                    0.0 mrecv
                        86.2 MPI_Recv
                0.0 getdatafrombuffers
        0.0 paralleldotproduct
        0.7 exchangedata
        0.0 writep2
        0.0 writeprotocolreal
0.0 waitforlinkready
0.0 checkvariableboundaries
0.0 writep2
0.0 getparams
0.0 printtolink
0.0 visconnsenddarcyvelocity
0.0 closevisconn
0.0 closelink
0.0 end_commutils
```

```
0.0 ZAMpano
    0.0 zam008e1
        7.0 Process 0
        3.0 Process 1
    0.0 zam008e2
        4.9 Process 2
        8.4 Process 3
    0.0 zam008e3
        7.9 Process 4
        3.5 Process 5
    0.0 zam008e4
        4.9 Process 6
        7.9 Process 7
    0.0 zam008e5
        8.3 Process 8
        4.3 Process 9
    0.0 zam008e6
        4.8 Process 10
        8.1 Process 11
    0.0 zam008e7
        9.5 Process 12
        4.4 Process 13
    0.0 zam008e8
        4.4 Process 14
        8.6 Process 15
```

**Fig. 2.** EXPERT Presenter Example Screendump. Using the color scale shown on the bottom, the severity of performance problems found (left pane) and their distribution over the program's call tree (middle pane) and machine locations (right pane) is displayed. By expanding or collapsing nodes n each of the three trees, the analysis can performed on different levels of ranularity.

In addition, it is possible to convert EPILOG traces into VTF3 format and analyze them manually with the VAMPIR event trace analysis tool [8]. We are currently working on integrating both tools so that the instance with the highest severity of each performance problem found can be displayed by VAMPIR on request. This would provide the ability to analyze the history of inefficient behavior in a time-line diagram or to do further statistical analysis using VAMPIR's powerful features.

Currently, the measurement components are available for the following platforms:

- Linux IA-32 cluster
- IBM POWER3 AND POWER4 cluster
- SGI MIPS cluster (Onyx, Challenge, Origin 2000, Origin 3000)
- SUN SUN FIRE cluster
- CRAY T3E
- HITACHI SR8000-F1

On Linux clusters and HITACHI SR8000-F1 systems the instrumentation of user functions is done automatically using the unpublished profiling interface of the PGI compiler or of the HITACHI compiler, respectively. For IBM systems, EPILOG provides an automatic binary instrumentor, which has been implemented on top of DPCL. On all other systems, user function instrumentation must be carried out on the source-code level either manually or automatically utilising the TAU instrumententation facilities [4]. The analysis components are based on Python and run on any workstation or laptop.

## 2 Additional Information

- The KOJAK tool suite including the source code can be downloaded from the KOJAK website. The website also provides a variety of technical papers, presentations, and screen dumps showing the analysis of example applications.
  → `http://www.fz-juelich.de/zam/kojak/`
- For more information on EXPERT's analysis and presentation features, see Felix Wolf's Ph.D. thesis [2]. The theoretical aspects can also be found in [3]. A more detailed overview (than this short description) about KOJAK can be found in [1].
- Details on instrumentation of OpenMP applications based on the POMP interface are described in [6] and [7].
- More information on the source-code instrumentation of user functions can be found on the homepages of the TAU [4] and PDT [5] projects.
  → `http://www.cs.uoregon.edu/research/paracomp/tau/`
  → `http://www.cs.uoregon.edu/research/paracomp/pdtoolkit/`
- The KOJAK project is part of the European IST working group APART.
  → `http://www.fz-juelich.de/apart/`

## References

1. F. Wolf, B. Mohr. *Automatic Performance Analysis of Hybrid MPI/OpenMP Applications*. 11th Euromicro Conference on Parallel, Distributed and Network Based Processing, 2003.
2. F. Wolf. *Automatic Performance Analysis on Parallel Computers with SMP Nodes*. Dissertation, NIC Series, Vol. 17, Forschunszentrum Jülich, 2002.
3. F. Wolf and B. Mohr. Specifying Performance Properties of Parallel Applications Using Compound Events. *Parallel and Distributed Computing Practices (Special Issue on Monitoring Systems and Tool Interoperability)*, Vol. 4, No. 3.
4. 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.
5. K. A. Lindlan, J. Cuny, A. Malony, S. Shende, B. Mohr, R. Rivenburgh, C. Rasmussen. A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates. In *Proc. of Supercomputing 2000*, Dallas, TX, 2000.
   → `http://www.sc2000.org/techpapr/papers/pap.pap167.pdf`
6. 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.
7. B. Mohr, A. Malony, H.-Ch. Hoppe, F. Schlimbach, G. Haab, J. Hoeflinger, S. Shah. A Performance Monitoring Interface for OpenMP. In *Proc. of 4th European Workshop on OpenMP (EWOMP 2002)*, Rome, Italy, 2002.
   → `http://www.caspur.it/ewomp2002/prog.html`
8. Pallas GmbH. Visualization and Analysis of MPI Programs
   → `http://www.pallas.de/e/products/vampir/`