KOJAK

ICL UT
INNOVATIVE COMPUTING LABORATORY

http://icl.cs.utk.edu/kojak/

Zentralinstitut für Angewandte Mathematik (ZAM)
Forschungszentrum Jülich (FZJ)

http://www.fz-juelich.de/zam/kojak/

KOJAK is an automatic performance evaluation system for parallel applications. It supports developers of these applications in detecting sources of inefficient program behavior and, thus, in writing more efficient code. KOJAK's most attractive feature is its ability to identify the reasons for low performance on a very high abstraction level (e.g., a process was waiting for a message that was sent too late). KOJAK can be used for MPI, OpenMP, and hybrid applications written in C/C++ or Fortran. KOJAK generates event traces from running applications and automatically searches them offline for execution patterns indicating inefficient performance behavior. In this way, it relieves the user from the burden of searching large amounts of data manually. KOJAK is jointly developed by Forschungszentrum Jülich, Germany, and the University of Tennessee, USA.

SUPPORTED PLATFORMS

- Cray T3E, X1, XD1, XT3
- HP Alpha based clusters
- IBM Power3 / Power4 based clusters
- IBM BlueGene/L
- Hitachi SR-8000
- Linux IA-32, IA-64, and EM64T/x86_64 clusters
- NEC SX
- SGI Mips based clusters (O2k, O3k)
- SGI IA-64 based clusters (Altix)
- SUN Solaris Sparc and x86 based clusters
- Generic UNIX workstation (clusters)

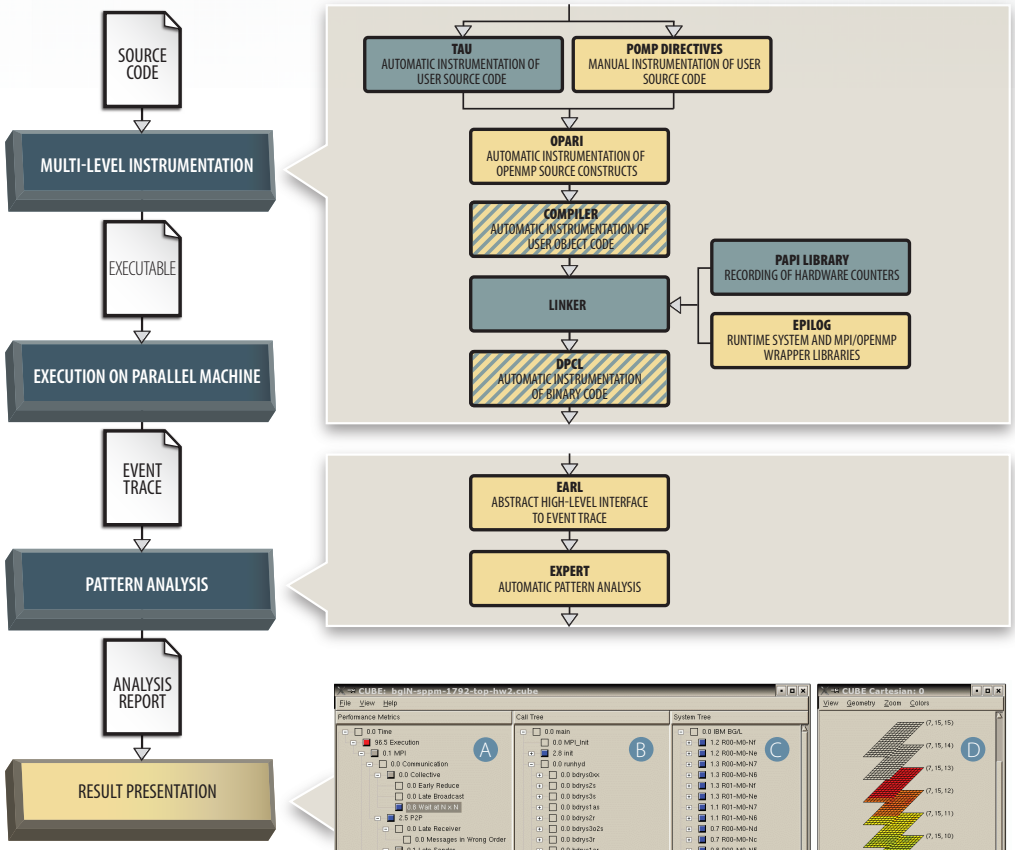
E-MAIL kojak@cs.utk.edu

KOJAK IS A COLLABORATIVE RESEARCH PROJECT

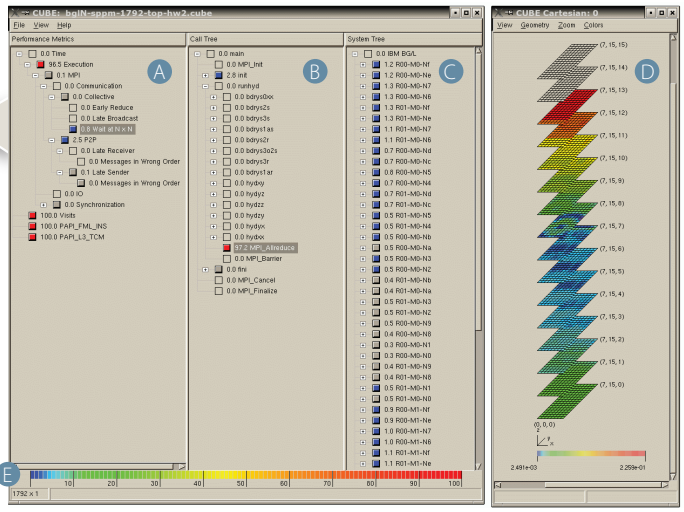
ICL UT
INNOVATIVE COMPUTING LABORATORY

Zentralinstitut für Angewandte Mathematik (ZAM)
Forschungszentrum Jülich (FZJ)

SPONSORED BY



CUBE GUI



- A Which type of problem?
- B Where in the source code? Which call path?
- C Distribution across processes?
- D Where in my physical/virtual topology?
- E How severe is the problem?

THE UNIVERSITY of TENNESSEE
Computer Science Department

ICL UT
INNOVATIVE COMPUTING LABORATORY

CITR
CENTER for INFORMATION TECHNOLOGY RESEARCH

KOJAK

ICL UT
Innovative Computing Laboratory
http://icl.cs.utk.edu/kojak/

Zentralinstitut für Angewandte Mathematik (ZAM)
Forschungszentrum Jülich (FZJ)
http://www.fz-juelich.de/zam/kojak/

Parallel applications often fail to exploit the full power of the underlying computing hardware. Their optimization, however, is extremely difficult due to the inherent complexity of parallel systems. KOJAK is an automatic performance evaluation system that supports developers of these applications in detecting sources of inefficient program behavior and, thus, in writing more efficient code. KOJAK's most attractive feature is its ability to identify the reasons for low performance on a very high abstraction level (e.g., a process was waiting for a message that was sent too late). KOJAK can be used for MPI, OpenMP, and hybrid applications written in C/C++ or Fortran. KOJAK generates event traces from running applications and automatically searches them offline for execution patterns indicating inefficient performance behavior. In this way, it relieves the user from the burden of searching large amounts of data manually. KOJAK is jointly developed by Forschungszentrum Jülich, Germany, and the University of Tennessee, USA.

KOJAK includes tools for instrumentation, event-trace generation, and post-processing of event traces plus a generic browser to display the analysis results. KOJAK has proved to be a very useful performance tool for a broad variety of applications including applications from physics and environmental science, and it has been presented in several tutorials and hands-on workshops.

The basic process of analyzing an application consists of generating an event trace from the running application by either inserting directives into the source code or using automatic instrumentation features that require only a small modification of the make file. The components involved in this part are OPARI and EPILOG. After program termination, the trace file is analyzed offline using EXPERT. To simplify the analysis, EXPERT accesses the trace through the EARL interface, which provides precalculated abstractions supporting the search process. Finally, the analysis results can be viewed in the CUBE performance browser.

OPARI is a source-to-source translation tool that automatically inserts calls to the POMP profiling interface into the source code of OpenMP applications. OPARI works with Fortran, C, and C++ programs. The POMP interface can be implemented by tool builders who want, for example, to monitor the performance of OpenMP applications. OPARI is based on the idea of OpenMP pragma / directive rewriting.

EPILOG is a binary event trace format plus a run-time library for generating event traces of MPI and OpenMP applications. The EPILOG event types cover MPI point-to-point and collective communication as well as OpenMP parallelism change, parallel constructs, and synchronization. A recently added feature is support for MPI-2 and SHMEM one-sided communication. Finally, the library includes capabilities to record data from hardware counters accessed using the PAPI library.

EARL is a generic high-level interface for reading EPILOG event traces. EARL provides random access to single events and computes the execution state at the time of a given event as well as predefined relationships between pairs of related events. EARL can be used for a large number of different trace-analysis tasks.

EXPERT is an automatic analysis tool for EPILOG traces. It identifies execution patterns indicating low performance and quantifies them according to their severity. These patterns target problems resulting from inefficient communication and synchronization as well as from low CPU and memory performance. The analysis process automatically transforms the traces into a compact call-path profile that includes the times spent on the different patterns. The profile can be viewed using the CUBE display.

CUBE is a generic presentation component suitable for displaying a wide variety of performance metrics for parallel programs (see picture on the front). It offers interactive exploration of a multidimensional performance space in a scalable fashion. Scalability is achieved in two ways: hierarchical decomposition of individual dimensions and aggregation across different dimensions. All performance metrics are uniformly accommodated in the same display providing the ability to easily compare the effects of different kinds of performance behavior. A topological view shows the performance behavior relative to virtual or physical process topologies. Multiple experiments can be automatically compared, integrated, or merged using a performance-algebra utility.