KOJAK

Bernd Mohr
Forschungszentrum Jülich (FZJ)
John von Neumann Institute for Computing (NIC)
Central Institute for Applied Mathematics (ZAM)
52425 Jülich, Germany
b.mohr@fz-juelich.de

KOJAK ↔ TAU ↔ VAMPIR

© 2006 Bernd Mohr, Forschungszentrum Jülich
**Basic Idea**

- "Traditional" Tool
  - For non-standard / tricky cases (10%)
  - For expert users
- Automatic Tool
  - For standard cases (90% ?!)
  - For "normal" users
  - Starting point for experts

- More productivity for performance analysis process!

---

**The KOJAK Project**

- Kit for Objective Judgement and Automatic Knowledge-based detection of bottlenecks
- Forschungszentrum Jülich
- Innovative Computing Laboratory, TN
- Long-term goals
  - Design and Implementation of a Portable, Generic, and Automatic Performance Analysis Environment
- Approach
  - Instrument C, C++, and Fortran parallel applications
    - Based on MPI, OpenMP, SHMEM, or hybrid
  - Collect event traces
  - Search trace for event patterns representing inefficiencies
  - Categorize and rank inefficiencies found
- http://www.fz-juelich.de/zam/kojak/
Example Performance Property:
Wrong Order Late Sender

KOJAK: Basic Pattern Hierarchy
KOJAK: MPI Pattern Hierarchy I

- **Communication**
  - **Collective**
    - Early Reduce
    - Late Broadcast
    - Wait at NxN
  - **Point to Point**
    - Late Receiver
    - Late Sender
    - Msg in Wrong Order
  - **IO**
    - Early Transfer
  - **Synchronization**

KOJAK: MPI Pattern Hierarchy II

- **Synchronization**
  - **Barrier**
    - Wait at Barrier
    - Barrier Completion
  - **RMA Sync.**
    - Window Mngt.
      - Wait at Create
      - Wait at Free
    - **Fence**
      - Wait at Fence
    - **Locks**
    - Active Target
      - Early Wait
      - Late Complete
      - Late Post
KOJAK: OpenMP Pattern Hierarchy

- **OpenMP**
  - **Synchronization**
    - **Barrier**
      - **Explicit**
      - **Implicit**
        - **Wait at Barrier**
    - **Lock Competition**
      - **API**
      - **Critical**
  - **Fork**
  - **Flush**

KOJAK: SHMEM Pattern Hierarchy

- **SHMEM**
  - **Communication**
    - **Collective**
      - **Late Broadcast**
      - **Wait at NxN**
    - **RMA**
  - **Synchronization**
    - **Barrier**
      - **Wait at Barrier**
    - **Point to Point**
      - **Lock Competition**
      - **Wait Until**
    - **Init/Exit**
    - **Memory Mngt.**
**MPI-1 Pattern: Early Reduce**

- Waiting time if the destination process (root) of a collective N-to-1 communication operation enters the operation earlier than its sending counterparts
- Applies to MPI calls `MPI_Reduce()`, `MPI_Gather()`, `MPI_Gatherv()`

**Pattern: Late Broadcast**

- Waiting times if the destination processes of a collective 1-to-N communication operation enter the operation earlier than the source process (root)
- MPI-1: Applies to `MPI_Bcast()`, `MPI_Scatter()`, `MPI_Scatterv()`
- SHMEM: Applies to `shm_membroadcast()`
**Generic Pattern: Wait at #**

- Time spent waiting in front of a collective synchronizing operation call until the last process reaches the operation.
- Pattern instances:
  - Wait at NxN (MPI)
  - Wait at Barrier (MPI)
  - Wait at NxN (SHMEM)
  - Wait at Barrier (SHMEM)
  - Wait at Barrier (OpenMP)
  - Wait at Create (MPI-2)
  - Wait at Free (MPI-2)
  - Wait at Fence (MPI-2)

**MPI-1 Pattern: Barrier Completion**

- Time spent in MPI barriers after the first process has left the operation.
**MPI-1 Pattern: Late Sender / Receiver**

- **Late Sender**: Time lost waiting caused by a blocking receive operation posted earlier than the corresponding send operation.

- **Late Receiver**: Time lost waiting in a blocking send operation until the corresponding receive operation is called.

**MPI-1 Pattern: Wrong Order**

- **Late Sender / Receiver** patterns caused by messages received/sent in wrong order.
- **Sub patterns of Late Sender / Receiver**
**MPI-2 Pattern: Early Transfer**

- Time lost being blocked in a RMA operation until exposure epoch is opened with `MPI_Win_post`.

**MPI-2 Pattern: Late Post**

- `MPI_Win_start` (variant A) or `MPI_Win_complete` (variant B) block until exposure epoch is opened with `MPI_Win_post`.
**MPI-2 Pattern: Early Wait**

- **MPI_win_start**
- **RMA Op**
- **MPI_win_complete**

- **MPI_win_post**
- **MPI_win_wait**

- Blocking time of **MPI_win_wait** until access epoch is closed by last **MPI_win_complete**

**MPI-2 Pattern: Late Complete**

- **MPI_win_start**
- **RMA Op**
- **MPI_win_complete**

- **MPI_win_post**
- **MPI_win_wait**

- Blocking time of **MPI_win_wait** until access epoch is closed by last **MPI_win_complete**
  - Sub pattern of Early Wait
  - Portion of blocking time because of unnecessary pause between last RMA operation and last **MPI_win_complete**
Generic Pattern: Lock Competition

- Time waiting for a lock that had been previously acquired by another thread/process
- Pattern instances:
  - Lock competition, API (OpenMP)
  - Lock competition, Critical (OpenMP)
  - Lock competition (SHMEM)

KOJAK Architecture

- user program
- executable
- execute
- EPILOG event trace

- TAU instr. + OPARI
- modified program
- Compiler / Linker
- PAPI library

- EXPERT Analyzer
- EARL
- analysis result
- CUBE Presenter
- Automatic Analysis

- trace converter
- VTF3/OTF/PRV event trace
- VAMPIR or Paraver
- Manual Analysis

© 2006 Bernd Mohr, Forschungszentrum Jülich
**KOJAK Tool Components**

- Instrument user application with EPILOG tracing library calls
  - User functions and regions:
    - Manually using POMP directives or Tracing API calls
    - Automatically by compiler
    - Automatically by TAU source instrumentor
  - MPI calls: Automatically by PMPI Wrapper Library
  - OpenMP: Automatically by OPARI source instrumentor
  - Record HW counters with PAPI
- Analyze measured event trace
  - Automatically with EXPERT trace analyzer (based on EARL trace analysis language) and CUBE result visualizer
  - Manually with VAMPIR or Paraver (+ EPILOG-VTF3/OTF converters) (+ EPILOG-PRV converter)

---

**KOJAK Automatic Analysis Process**

1. `expert <elg-trace-file>`
   - Scans event trace for patterns; if instance found:
     - Determine call path and process/thread affected
     - Calculate severity ::= percentage of total execution time "lost" because of pattern
   - Result stored in "cube" file containing
     - For each pattern: distribution of severity
       - Over all call paths
       - Over machine / nodes / processes / threads
2. `cube <cube-file>`
   - Result presentation via three interconnected tree browsers
     - Pattern hierarchy (general ⇔ specific problem)
     - Region / call tree
     - Location hierarchy
KOJAK: Supported Platforms

- Instrumentation and measurement only (analysis on front-end or workstation)
  - Cray T3E, Cray XD1, Cray X1, and Cray XT3
  - IBM BlueGene/L
  - Hitachi SR-8000
  - NEC SX
- Full support (instrumentation, measurement, and automatic analysis)
  - Linux IA32, IA64, and EMT64/x86_64 based clusters
  - IBM AIX Power3/4/5 based clusters (SP2, Regatta)
  - SGI Irix MIPS based clusters (Origin 2K, Origin 3K)
  - SGI Linux IA64 based clusters (Altix)
  - SUN Solaris Sparc and x86/x86_64 based clusters (SunFire, ...)
  - DEC/HP Tru64 Alpha based clusters (Alphaserver, ...)
Other Features and Future Work

- Other major features
  - Comparative analysis of multiple executions
    - Merge, Mean, and Difference of experiments
  - Holistic analysis
    - Combined hardware counter metrics + time-based analysis
  - Extensive one-sided communication analysis
    - MPI-2 RMA and SHMEM
    - CAF and UPC soon

- Future Work: SCALABILITY
  - Scalable trace format
  - Parallel pattern analyzer
  - Scalable displays

CUBE Performance Algebra

- "Compute" with CUBE result files
  - Mean
  - Merge
  - Diff

- Raised relief / positive values show improvement
- Sunken relief / negative values show degradation
Holistic Performance Analysis

- Collect
  - a series of experiments
  - with different but matching sets of hardware counters
- Merge results using
  - Generic
  - Platform-specific counter hierarchy specification

KOJAK: sPPM run on (8x16x14) 1792 PEs

- New topology display
- Shows distribution of pattern over HW topology
- Easily scales to even larger systems