KOJAK

Hardware Counter Performance Analysis of Parallel Programs

Brian J. N. Wylie & Bernd Mohr
John von Neumann Institute for Computing
Forschungszentrum Jülich
B.Wylie@fz-juelich.de
Outline

• Introduction to KOJAK toolkit
  • Instrumentation, measurement & analysis
  • OpenMP, MPI & hybrid parallelism
    • Example: ASCI Purple sPPM
• Hardware counter metric profiling
  • PMAPI, hpmcount, PAPI v3, ...
  • KOJAK HWC metrics
    • HWC metric generalisation/customisation/structuring
• In-progress/future developments
• Summary
The KOJAK project

- **Kit for Objective** Judgement & **Automatic** Knowledge-based detection of bottlenecks
  - Forschungszentrum Jülich
  - Innovative Computing Laboratory, UTK

- **Long-term goals**
  - Design & implementation of a portable, generic & automatic performance analysis environment

- **Current focus**
  - Event tracing
  - Parallel computers with SMP nodes
  - MPI, OpenMP & Hybrid programming models
KOJAK supported platforms

• Full support for instrumentation, measurement, and automatic analysis
  • Linux IA32, IA64 & IA32_64 clusters (incl. XD1)
  • IBM AIX POWER3 & 4 clusters (SP2, Regatta)
  • SGI Irix MIPS clusters (Origin 2K, 3K)
  • Sun Solaris SPARC clusters (SunFire, ...)
  • DEC/HP Tru64 Alpha clusters (Alphaserver, ...)

• Instrumentation and measurement only
  • IBM BlueGene/L
  • Cray T3E, Cray X1
  • Hitachi SR-8000, NEC SX
KOJAK architecture

- User program
- Executable
- Execute
- EPILOG event trace

Semi-automatic Instrumentation:
- OPARI / TAU instr.
- Modified program
- Compiler / Linker
- PAPI library
- POMP+PMPI libraries
- EPILOG trace library

Automatic Analysis:
- EXPERT analyzer
- Analysis result
- EARL
- CUBE presenter

Manual Analysis:
- Trace converter
- VTF3 event trace
- VAMPIR

Manual Analysis:
- EPILOG event trace
KOJAK tool components

- **Instrument user application**
  - **EPILOG** tracing library calls
  - User functions and regions:
    - Automatically by **TAU** source instrumentor
    - Automatically by compiler (PGI, Hitachi, NEC, Sun)
    - Manually using **POMP** directives
  - **MPI** calls: Automatic **PMPI** wrapper library
  - **OpenMP**: Automatic **OPARI** source instrumentor
  - Record hardware counter metrics via **PAPI**

- **Analyze measured event trace**
  - Automatically with EARL-based **EXPERT** trace analyzer and **CUBE** analysis result browser
  - Manually with **VAMPIR** (via **EPILOG-VTF3** converter)
Automatic analysis process

- Scans event trace sequentially
  - If trigger event: call search function of pattern
  - If match:
    - Determine call path and process/thread affected
    - Calculate severity ::= percentage of total execution time “lost” due to pattern

- Analysis result
  - For each pattern: distribution of severity
    - Over all call paths
    - Over machine / nodes / processes / threads
  - CUBE presentation via 3 linked tree browsers
    - Pattern hierarchy (general ⇨ specific problem)
    - Region / call tree
    - Location hierarchy
sPPM OMP/MPI hybrid (JuMP)

What problem?

In what source context?

Which processes and/or threads?

How severe?
ASCI Purple sPPM

- Simplified piecewise parabolic method solution of a 3D gas dynamic problem on a uniform Cartesian mesh
  - 12,000 lines of F77 (via M4 and CPP macros) and a little C in 16 source files
  -Explicitly parallelised with OpenMP and MPI, used independently or as OMP/MPI hybrid
  - Part of ASCI Purple benchmark suite
- Considered to perform very well
  - Good processor performance, and excellent multithreading & message-passing efficiency
CUBE performance algebra

- Cross-experiment analysis required
  - Different execution configuration
  - Different measurement tools
  - Different non-deterministic behaviours

- Arithmetic operations on CUBE instances
  - merge, mean, difference, ...
  - Obtain CUBE instance as a result
  - Displayed like ordinary CUBE instance
    - Special rendering of negative values
Hardware counter profiling

- Multiple counters provided by modern microprocessors & computer systems
- Offer low-overhead access to detailed execution characteristics
  - operation of functional units, cache/memory, network interconnect, I/O, ...
- Potentially valuable, but limited usability
  - processor and/or system-specific events, interfaces & restrictions
Hardware counter profiling tools

- **Platform-specific**
  - AIX/POWER: hpmcount, libhpm, PMAPI
  - Solaris: collect/analyzer, har, cputrack, libcpc
  - Linux: oprofile, perfctr
  - ...

- **Multi-platform “portable”**
  - PCL, PAPI, ...
  - Standardised APIs & predefined events
  - Accuracy & reliability to be verified with care
    - PAPI v3.0 (POWER4/AIX) required corrections to event definitions & for multithreaded accesses
  - Basis for high-level tools: VAMPIR, KOJAK, ...
HWC analysis challenges

- Interpretation of event counts & attributions
  - Often highly architecture-specific & obscure
  - Generally poorly or incompletely documented
  - Sometimes unreliable
- Incomplete analyses due to partial sets of counters accessible simultaneously
  - Must collect & integrate multiple measurements
- Comparative multi-platform analyses
  - Architectural characteristics reflected in provision of counters
KOJAK HWC metrics analysis

- Define hierarchical structures for metrics
  - specifies relationships between metrics
  - supports additional metric derivations
    - compositions (exclusively additive)
      - ACCESSES = HITS + MISSES
      - ACCESSES = L1$_HITS + L2$_HITS + L3$_HITS + MEM_HITS
    - computations
      - HITS = ACCESSES – MISSES
- Multiple hierarchies/relations are valuable
  - platform-specific extension to generic hierarchy
    - DATA_LOAD_FROM_L2$
      = PM\_DATA\_FROM\_L2$
        + PM\_DATA\_FROM\_L25\_MOD + PM\_DATA\_FROM\_L25\_SHR
        + PM\_DATA\_FROM\_L275\_MOD + PM\_DATA\_FROM\_L275\_SHR
      = DC\_L2\_REFILL\_O + DC\_L2\_REFILL\_E + DC\_L2\_REFILL\_S
Instruction metrics hierarchy

- **INSTRUCTION**
  - BRANCH
    - COND_BRANCH
    - UNCOND_BRANCH
  - FLOATING_POINT
    - FP_ADD
    - FP_MUL
    - FP_FMA
    - FP_DIV
    - FP_INV
    - FP_SQRT
  - INTEGER
  - MEMORY
    - LOAD
    - STORE
    - SYNCH
  - VECTOR

- All instructions completed broken down by type
- FLOATING_POINT may be measured directly or computed/approximated from available constituents
- Not all types of instruction counted on all processors
- What about POWER4 FP_STORE and FP_LOAD?
Cache metric hierarchies

• **DATA_ACCESS**
  - DATA_HIT_L1$
    - DATA_STORE_INTO_L1$
    - DATA_LOAD_FROM_L1$
  - DATA_HIT_L2$
    - DATA_STORE_INTO_L2$
    - DATA_LOAD_FROM_L2$
  - DATA_HIT_L3$
    - DATA_STORE_INTO_L3$
    - DATA_LOAD_FROM_L3$
  - DATA_HIT_MEM
    - DATA_STORE_INTO_MEM
    - DATA_LOAD_FROM_MEM

• **INST_ACCESS**
  - INST_HIT_L1$
  - INST_HIT_L2$
  - INST_HIT_L3$
  - INST_HIT_MEM

• Exploit available counters as far as possible
• Attempt consistency across diverse chip architectures
• Some counts may need to be computed, e.g., Hits = Accesses – Misses
sPPM hybrid OMP & MPI metrics
sPPM composed counter metrics

Composed counter metric $DATA_{ACCESS} = DATA_{HIT\_L1\$} + DATA_{HIT\_L2\$} + DATA_{HIT\_L3\$} + DATA_{HIT\_MEM}$
sPPM computed counter metrics

Computed counter metric DATA_LOAD_FROM_L1$ = PM_LD_REF_L1 - PM_LD_MISS_L1

4 x 8  Incl. CNode hydxy : 7.90085e+10 (12.2758%)
sPPM native counter metrics

Measured counter metric PM_DATA_FROM_L275_SHR: DL1 was reloaded with shared (T) data from the L2 of another MCM due to a demand load.
p690 vs XD1 generic HWC metrics
p690 vs XD1 native HWC metrics
## DATA_ACCESS metric hierarchies

### POWER4:
- **DATA_ACCESS**
  - **DATA_HIT_L1**
  - **DATA_STORE_INTO_L1**
  - **DATA_LOAD_FROM_L1**
  - **DATA_HIT_L2**
  - **DATA_STORE_INTO_L2**
  - **DATA_LOAD_FROM_L2**
    - **PM_DATA_FROM_L2**
    - **PM_DATA_FROM_L25_MOD**
    - **PM_DATA_FROM_L25_SHR**
    - **PM_DATA_FROM_L275_MOD**
    - **PM_DATA_FROM_L275_SHR**
  - **DATA_HIT_L3**
  - **DATA_STORE_INTO_L3**
  - **DATA_LOAD_FROM_L3**
  - **DATA_HIT_MEM**
  - **DATA_STORE_INTO_MEM**
  - **DATA_LOAD_FROM_MEM**
    - **PM_DATA_FROM_MEM**

### Opteron:
- **DATA_ACCESS**
  - **DATA_HIT_L1**
  - **DATA_STORE_INTO_L1**
  - **DATA_LOAD_FROM_L1**
  - **DATA_HIT_L2**
  - **DATA_STORE_INTO_L2**
  - **DATA_LOAD_FROM_L2**
    - **DC_L2_REFILL_M**
    - **DC_L2_REFILL_O**
    - **DC_L2_REFILL_E**
    - **DC_L2_REFILL_S**
  - **DATA_HIT_L3**
  - **DATA_STORE_INTO_L3**
  - **DATA_LOAD_FROM_L3**
  - **DATA_HIT_MEM**
  - **DATA_STORE_INTO_MEM**
    - **DC_SYS_REFILL_M**
  - **DATA_LOAD_FROM_MEM**
    - **DC_SYS_REFILL_O**
    - **DC_SYS_REFILL_E**
    - **DC_SYS_REFILL_S**
BG/L sPPM Execution

- New topology display
- Shows spatial distribution of performance pattern over hardware topology
- Scales to larger systems
BG/L sPPM MPI P2P
<table>
<thead>
<tr>
<th>Performance Metrics</th>
<th>Call Tree</th>
<th>System Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 Time</td>
<td>0.0 main</td>
<td>0.0 IBM BG/L</td>
</tr>
<tr>
<td>96.6 Execution</td>
<td>0.0 MPI</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.1 MPI</td>
<td>0.0 MPI_Init</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.0 Communication</td>
<td>2.6 init</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.0 Collective</td>
<td>0.0 nrunhyd</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.0 Early Reduce</td>
<td>0.0 dbdrys0xx</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.0 Late Broadcast</td>
<td>0.0 dbdrys2s</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.8 Wait at N x N</td>
<td>0.0 dbdrys3s</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>2.4 F2P</td>
<td>0.3 dbdry1as</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.0 Late Receiver</td>
<td>0.2 dbdry3r</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.1 Late Sender</td>
<td>0.0 dbdrys302s</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.0 RMA</td>
<td>0.0 dbdry3r</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.0 IO</td>
<td>0.0 dbdry1as</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>0.0 Synchronization</td>
<td>16.3 hydxy</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>100.0 Visits</td>
<td>10.6 hydzy</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>100.0 PAPI_FML_INS</td>
<td>8.4 hydzz</td>
<td>3.1 R00-M0-NF</td>
</tr>
<tr>
<td>100.0 PAPI_L3_TCM</td>
<td></td>
<td>3.1 R00-M0-NF</td>
</tr>
</tbody>
</table>

1024 x 1 Excl. CTnode hydzz: 9.27302e+09 (6.3347%)
Summary

- KOJAK supports most important HPC/cluster platforms, program languages & paradigms
- Provides automated execution analyses for holistic performance characterisation
  - structured comm/synch & hardware counters
  - scalable machine topology presentation
- Automatic performance analysis with KOJAK
  - [http://www.fz-juelich.de/zam/kojak](http://www.fz-juelich.de/zam/kojak)
  - kojak@fz-juelich.de
BlueGene/L expt comparison

- 1024 PE sPPM MPI process topologies 8x8x16 vs. 16x8x8
- Raised relief / +ve values for improvement
- Sunken relief / -ve values for degradation
Vampir NG: BG/L sPPM 1792 PEs
KOJAK HWC metrics #1

- KOJAK v2.0 defined two ELG_METRICS with severity times derived from measurements
  - FLOATING_POINT inefficiency time for periods where FP instruction rates are <25% of peak
  - L1_D_MISS penalty time for periods where 1st-level data-cache misses above average
  - hard-coded PAPI counter metric definitions
  - subtracted from generic CPU Execution time
- Identified key performance behaviour, but...
  - heuristics significantly exaggerated severities
  - seriously compromised resultant derivation of exclusive CPU Execution time metric
KOJAK HWC metrics #2

- Direct report of counter measurements
  - replaces time-penalty derivation heuristics
  - includes all measured counters in analysis
- Improved HWC measurement specification
  - measurements using native or PAPI names
  - measurement-time specification of aliases
    - METRICS.SPEC file read from various locations, customisable with ELG_METRICS_SPEC variable
- Defined measurement groups of counters
  - similar to PMAPI counter groups for POWER4
- More metrics allowed per execution