



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



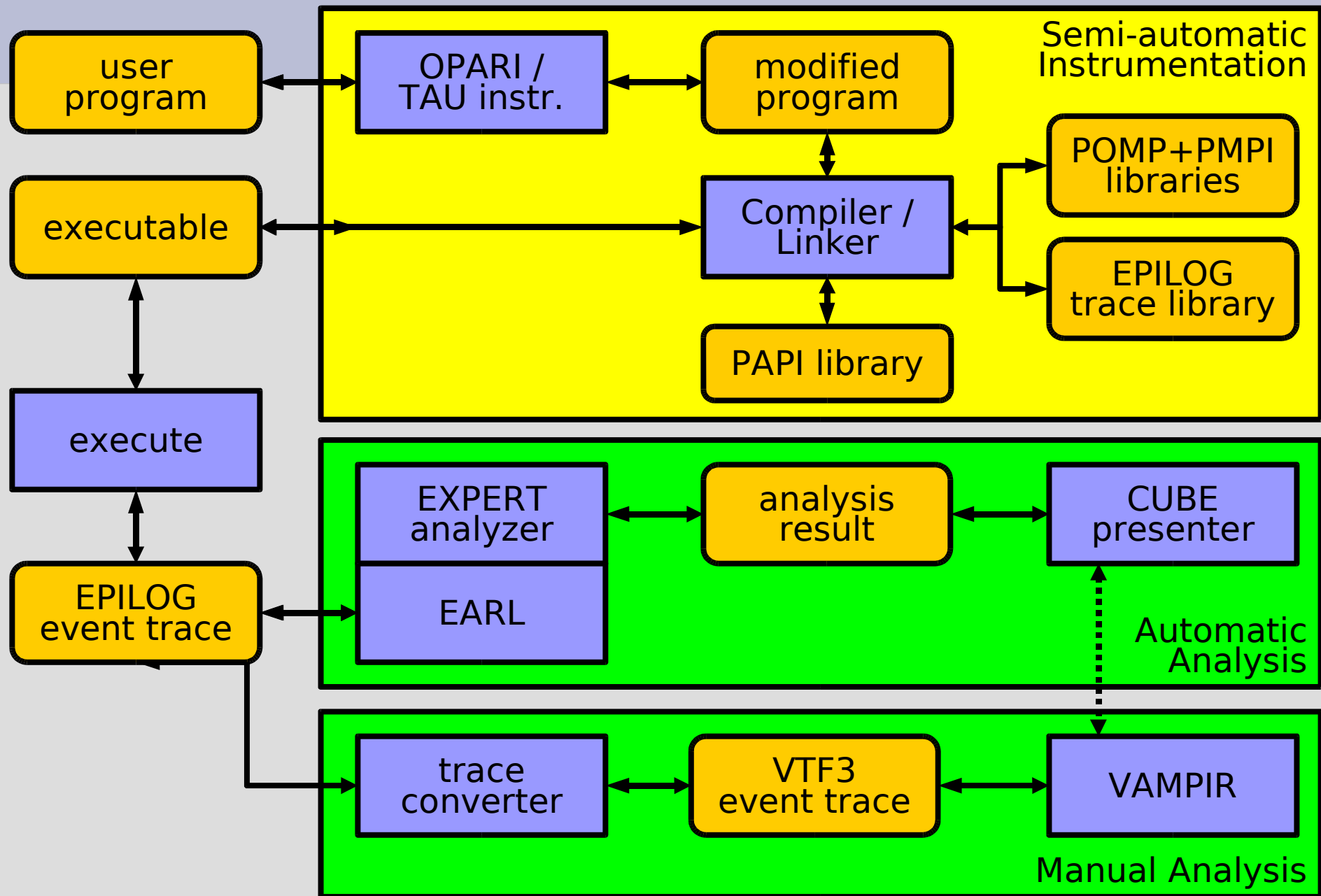
- **K**it for **O**bjective **J**udgement & **A**utomatic **K**nowledge-based detection of bottlenecks
 - Forschungszentrum Jülich
 - Innovative Computing Laboratory, UTK
- Long-term goals
 - Design & implementation of a portable, generic & automatic performance analysis environment
- Current focus
 - Event tracing
 - Parallel computers with SMP nodes
 - MPI, OpenMP & Hybrid programming models



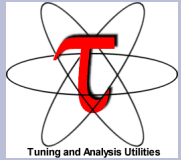
KOJAK supported platforms

- 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



KOJAK tool components



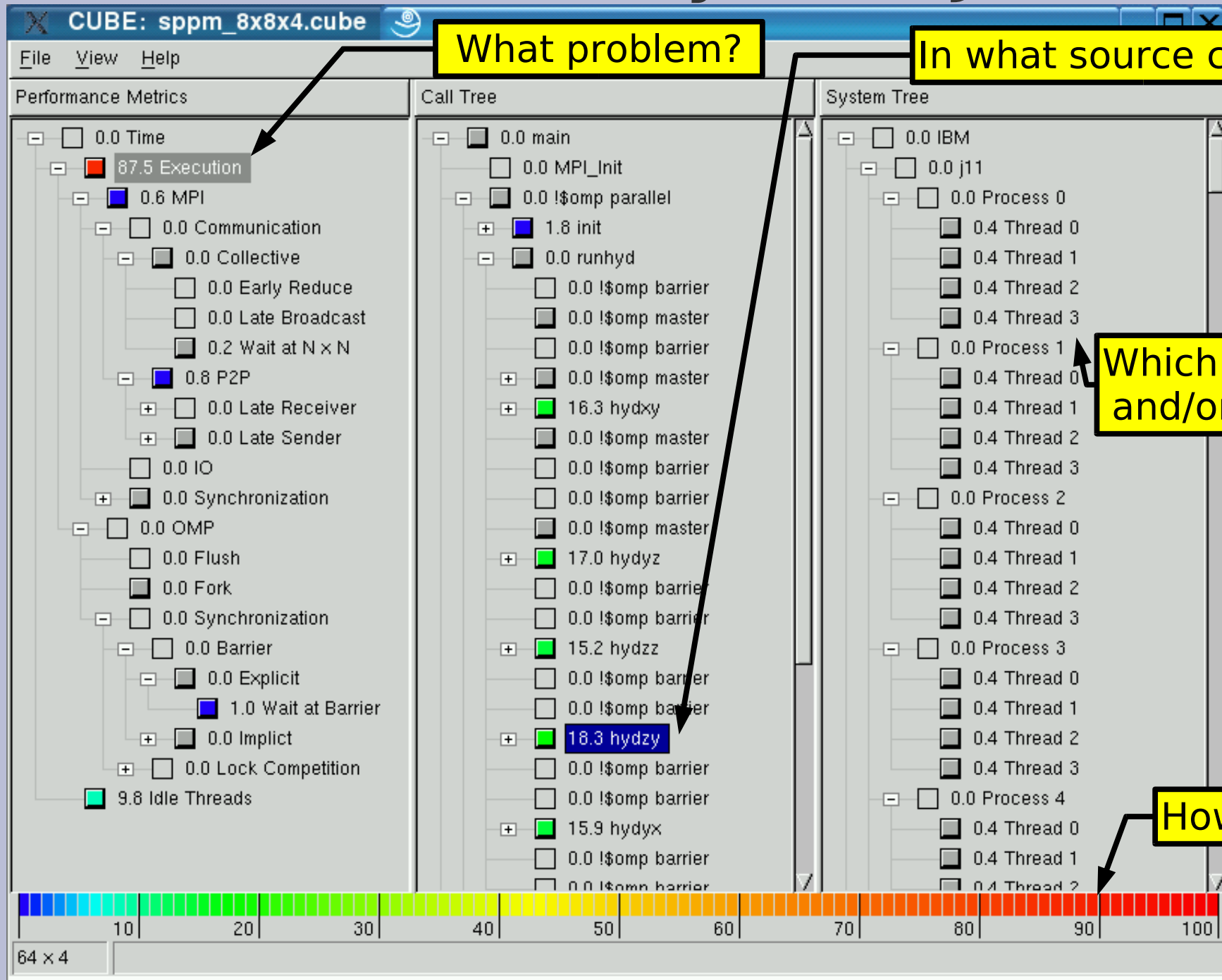
- Instrument user application
 - **EPILOG** tracing library calls
 - User functions and regions:
 - Automatically by **TAU** source instrumentor
 - Automatically by compiler (PGI, Hitachi, NEC, Sun)
 - Manually using **POMP directives**
 - MPI calls: Automatic **PMPI** wrapper library
 - OpenMP: Automatic **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)
 - $ACCESSSES = HITS + MISSES$
 - $ACCESSSES = L1\$_HITS + L2\$_HITS + L3\$_HITS + MEM_HITS$
 - computations
 - $HITS = ACCESSSES - MISSES$
- 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**

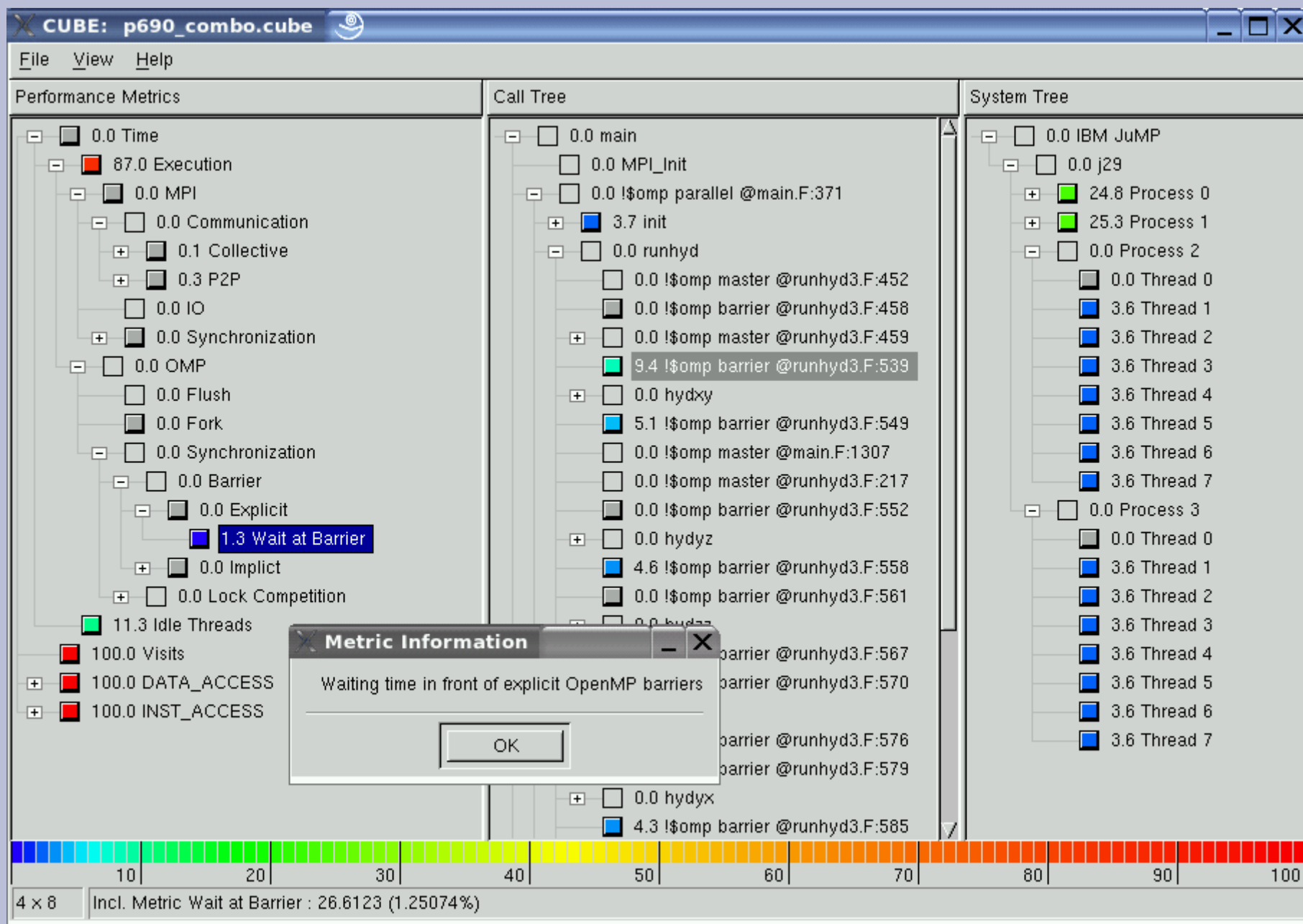
- DATA_HIT_L1\$
 - DATA_STORE_INTO_L1\$
 - DATA_LOAD_FROM_L1\$
- DATA_HIT_L2\$
 - DATA_STORE_INTO_L2\$
 - DATA_LOAD_FROM_L2\$
- DATA_HIT_L3\$
 - DATA_STORE_INTO_L3\$
 - DATA_LOAD_FROM_L3\$
- DATA_HIT_MEM
 - DATA_STORE_INTO_MEM
 - DATA_LOAD_FROM_MEM

- **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

The screenshot shows the CUBE performance analysis tool interface. The main window is titled "CUBE: p690_combo.cube" and contains three panes: Performance Metrics, Call Tree, and System Tree.

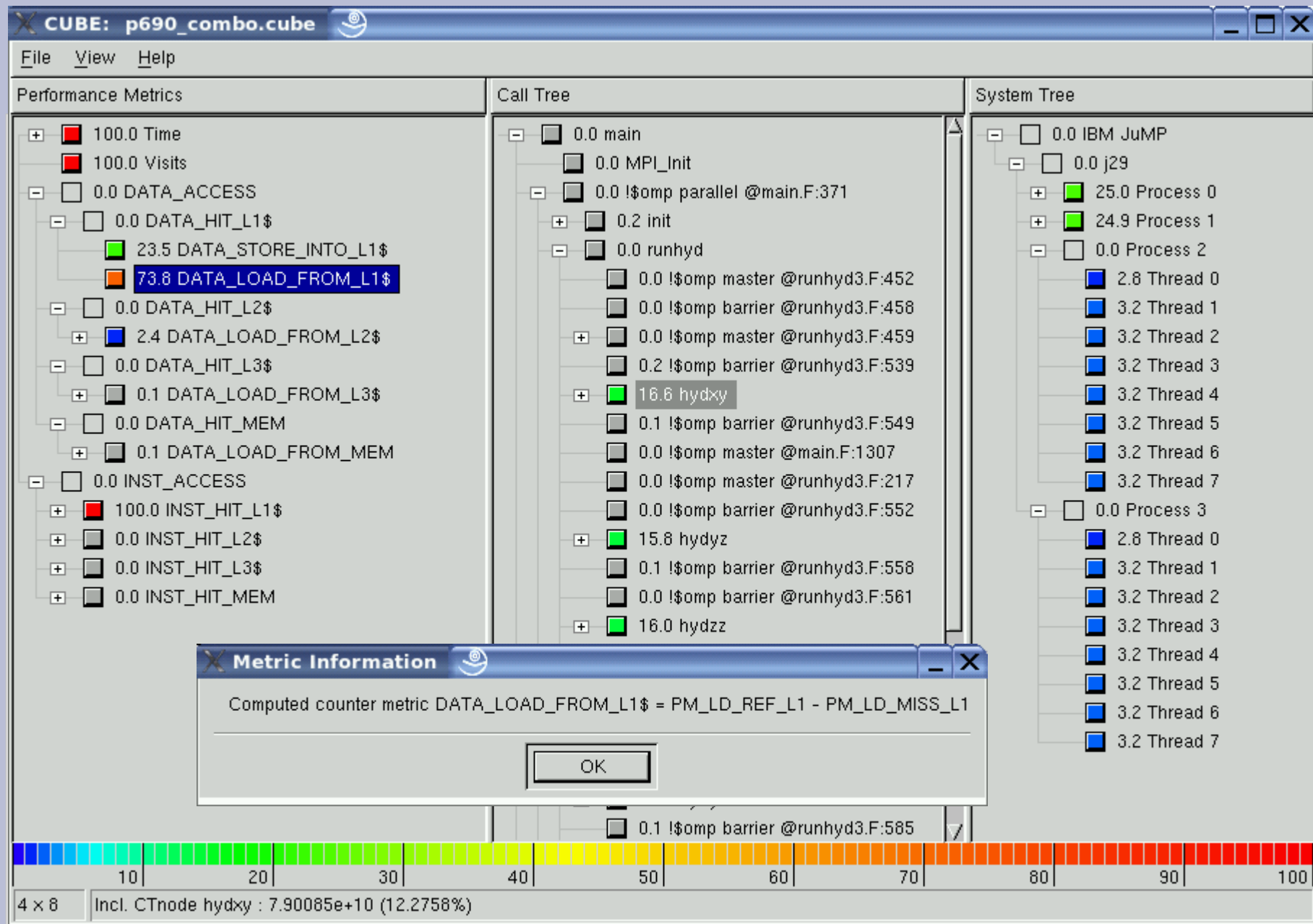
- Performance Metrics:** A tree view showing various metrics. The "DATA_ACCESS" metric is highlighted in blue, with a value of 2.4. Other metrics include "Time" (0.0), "Execution" (87.0), "MPI" (0.0), "Communication" (0.4), "IO" (0.0), "Synchronization" (0.0), "OMP" (0.0), "Flush" (0.0), "Fork" (0.0), "Synchronization" (1.3), "Idle Threads" (11.3), "Visits" (100.0), "DATA_HIT_L1\$" (97.3), "DATA_HIT_L2\$" (2.4), "DATA_HIT_L3\$" (0.1), "DATA_HIT_MEM" (0.1), "INST_ACCESS" (0.0), "INST_HIT_L1\$" (100.0), "INST_HIT_L2\$" (0.0), "INST_HIT_L3\$" (0.0), and "INST_HIT_MEM" (0.0).
- Call Tree:** A tree view showing the call stack. The "hydxy" function is highlighted in green, with a value of 16.9. Other functions include "main" (0.0), "MPI_Init" (0.0), "!\$omp parallel @main.F:371" (0.0), "init" (0.1), "runhyd" (0.0), "!\$omp master @runhyd3.F:452" (0.0), "!\$omp barrier @runhyd3.F:458" (0.0), "!\$omp master @runhyd3.F:459" (0.0), "!\$omp barrier @runhyd3.F:539" (0.0), "!\$omp barrier @runhyd3.F:549" (0.0), "!\$omp master @main.F:1307" (0.0), "!\$omp master @runhyd3.F:217" (0.0), "!\$omp barrier @runhyd3.F:552" (0.0), "hydzy" (16.5), "!\$omp barrier @runhyd3.F:558" (0.0), "!\$omp barrier @runhyd3.F:561" (0.0), "hydzz" (16.3), "!\$omp barrier @runhyd3.F:567" (0.0), "!\$omp barrier @runhyd3.F:570" (0.0), "hydzy" (16.2), and "!\$omp barrier @runhyd3.F:576" (0.0).
- System Tree:** A tree view showing the system hierarchy. The "j29" process is highlighted in green, with a value of 24.9. Other processes include "Process 0" (24.9), "Process 1" (25.0), "Process 2" (0.0), "Process 3" (0.0), and "Thread 0" through "Thread 7" (each 3.2).

A "Metric Information" dialog box is open in the foreground, displaying the formula for the composed counter metric DATA_ACCESS:

$$\text{Composed counter metric DATA_ACCESS} = \text{DATA_HIT_L1\$} + \text{DATA_HIT_L2\$} + \text{DATA_HIT_L3\$} + \text{DATA_HIT_MEM}$$

The dialog box has an "OK" button and a color scale at the bottom ranging from 0 to 100.

sPPM computed counter metrics



sPPM native counter metrics

The screenshot displays the CUBE performance analysis tool interface, showing three main panels: Performance Metrics, Call Tree, and System Tree. A dialog box titled "Metric Information" is open at the bottom, providing details about a specific counter metric.

Performance Metrics:

- 100.0 Time
- 100.0 Visits
- 0.0 DATA_ACCESS
 - 0.0 DATA_HIT_L1\$
 - 23.5 DATA_STORE_INTO_L1\$
 - 73.8 DATA_LOAD_FROM_L1\$
 - 0.0 DATA_HIT_L2\$
 - 0.0 DATA_LOAD_FROM_L2\$
 - 2.4 PM_DATA_FROM_L2
 - 0.0 PM_DATA_FROM_L25_SHR
 - 0.0 PM_DATA_FROM_L25_MOD
 - 0.0 PM_DATA_FROM_L275_SHR**
 - 0.0 PM_DATA_FROM_L275_MOD
 - 0.0 DATA_HIT_L3\$
 - 0.0 DATA_LOAD_FROM_L3\$
 - 0.1 PM_DATA_FROM_L3
 - 0.0 PM_DATA_FROM_L35
 - 0.0 DATA_HIT_MEM
 - 0.0 DATA_LOAD_FROM_MEM
 - 0.1 PM_DATA_FROM_MEM
 - 0.0 INST_ACCESS
 - 100.0 INST_HIT_L1\$

Call Tree:

- 0.1 main
 - 2.3 MPI_Init
 - 0.1 !\$omp parallel @main.F:371
 - 0.6 init
 - 0.5 runhyd
 - 0.0 !\$omp master @runhyd3.F:452
 - 0.0 !\$omp barrier @runhyd3.F:458
 - 0.1 !\$omp master @runhyd3.F:459
 - 0.0 !\$omp barrier @runhyd3.F:539
 - 18.4 hydxy
 - 0.0 !\$omp barrier @runhyd3.F:549
 - 0.1 !\$omp master @main.F:1307
 - 0.0 !\$omp master @runhyd3.F:217
 - 0.0 !\$omp barrier @runhyd3.F:552
 - 17.5 hydzy
 - 0.0 !\$omp barrier @runhyd3.F:558
 - 0.0 !\$omp barrier @runhyd3.F:561
 - 11.2 hydzz
 - 0.0 !\$omp barrier @runhyd3.F:567
 - 0.0 !\$omp barrier @runhyd3.F:570
 - 19.3 hydzy
 - 0.0 !\$omp barrier @runhyd3.F:576

System Tree:

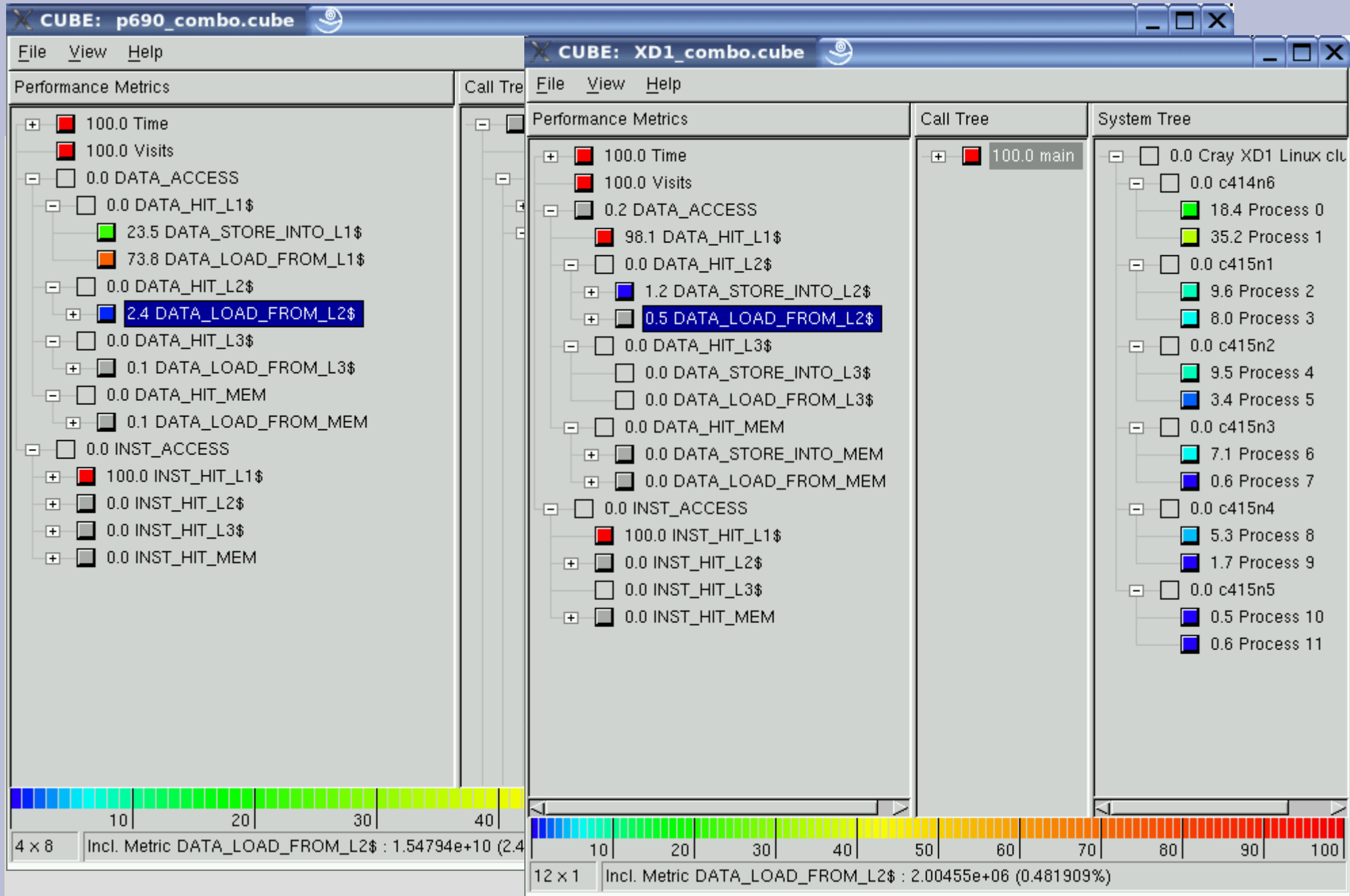
- 0.0 IBM JuMP
 - 0.0 j29
 - 22.8 Process 0
 - 22.9 Process 1
 - 0.0 Process 2
 - 5.2 Thread 0
 - 2.2 Thread 1
 - 2.5 Thread 2
 - 2.3 Thread 3
 - 2.3 Thread 4
 - 2.7 Thread 5
 - 4.0 Thread 6
 - 5.3 Thread 7
 - 0.0 Process 3
 - 5.4 Thread 0
 - 2.0 Thread 1
 - 1.9 Thread 2
 - 3.5 Thread 3
 - 4.1 Thread 4
 - 4.2 Thread 5
 - 2.8 Thread 6
 - 3.7 Thread 7

Metric Information Dialog:

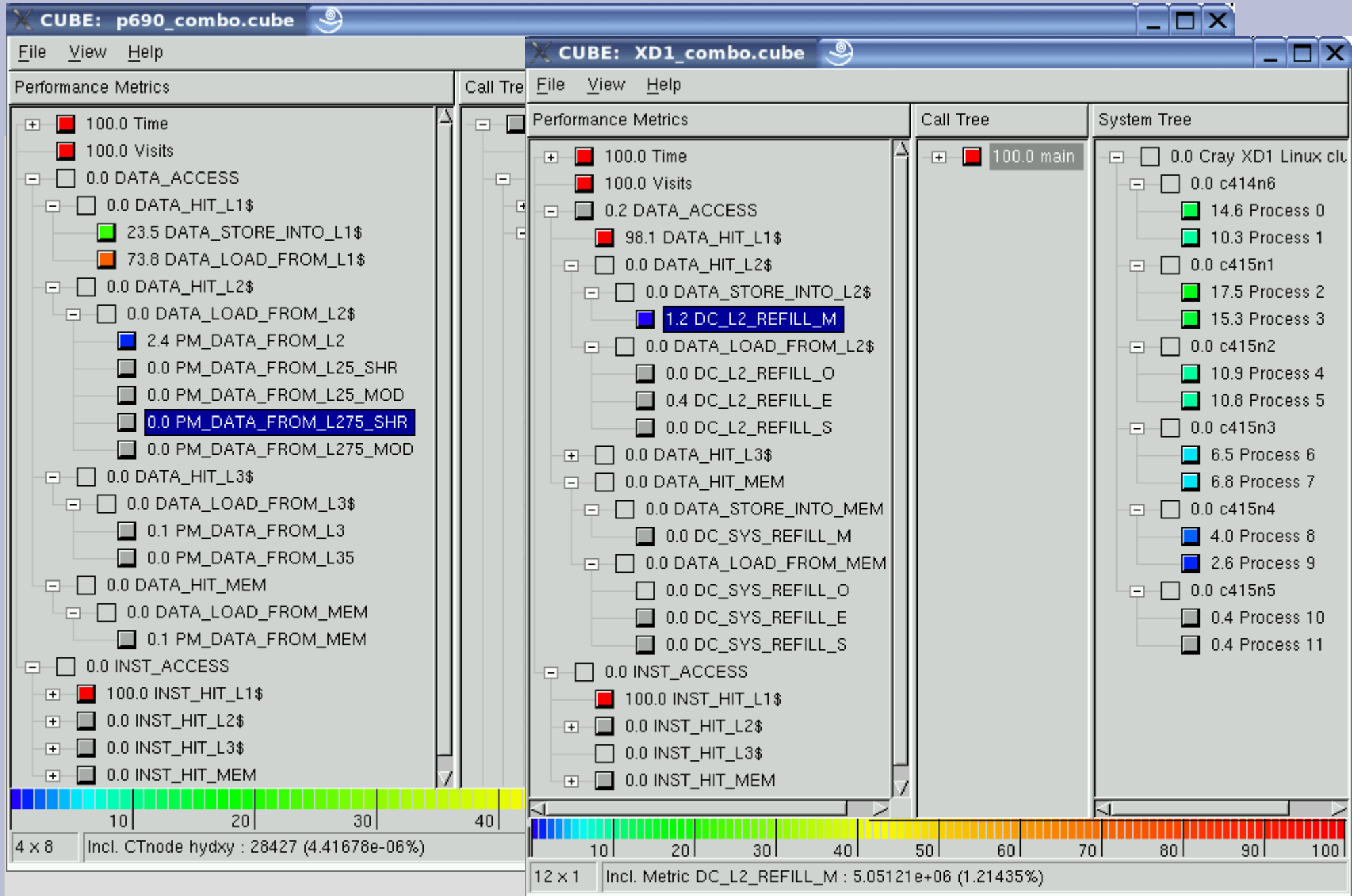
Measured counter metric PM_DATA_FROM_L275_SHR : DL1 was reloaded with shared (T) data from the L2 of another MCM due to a demand load.

OK

p690 vs XD1 generic HWC metrics



p690 vs XD1 native HWC metrics



DATA_ACCESS metric hierarchies

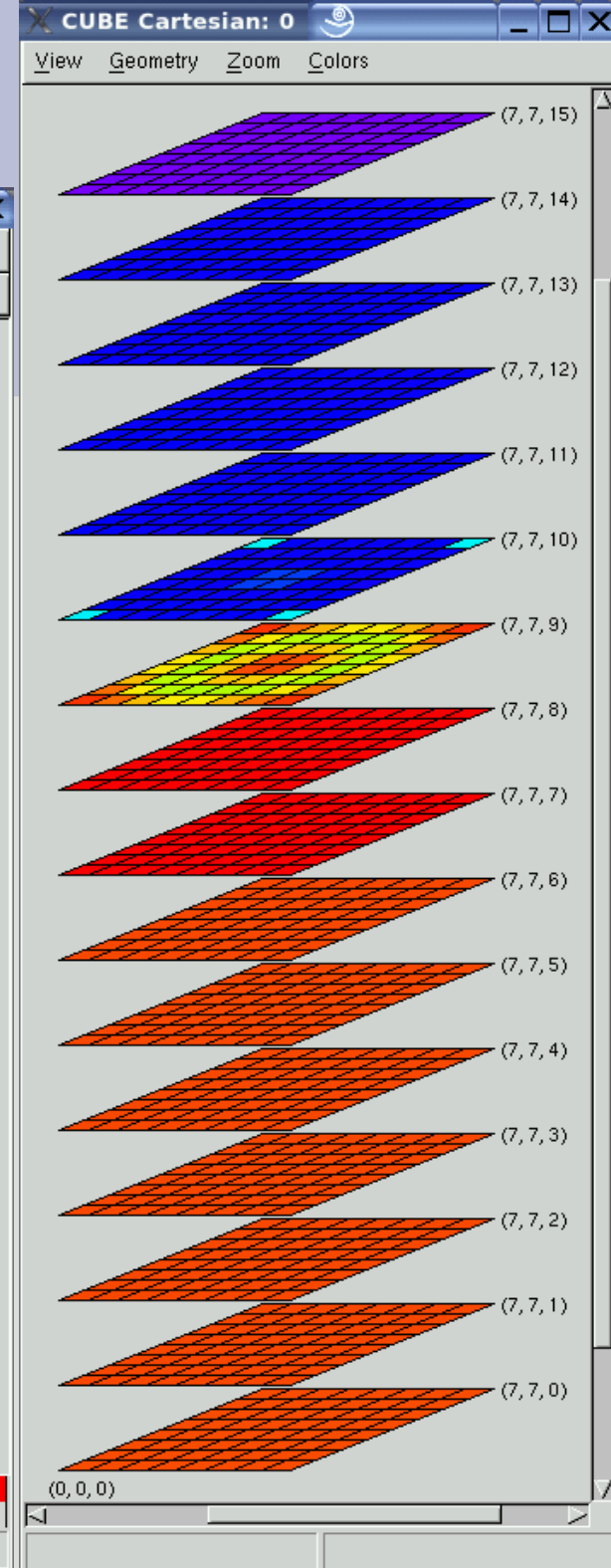
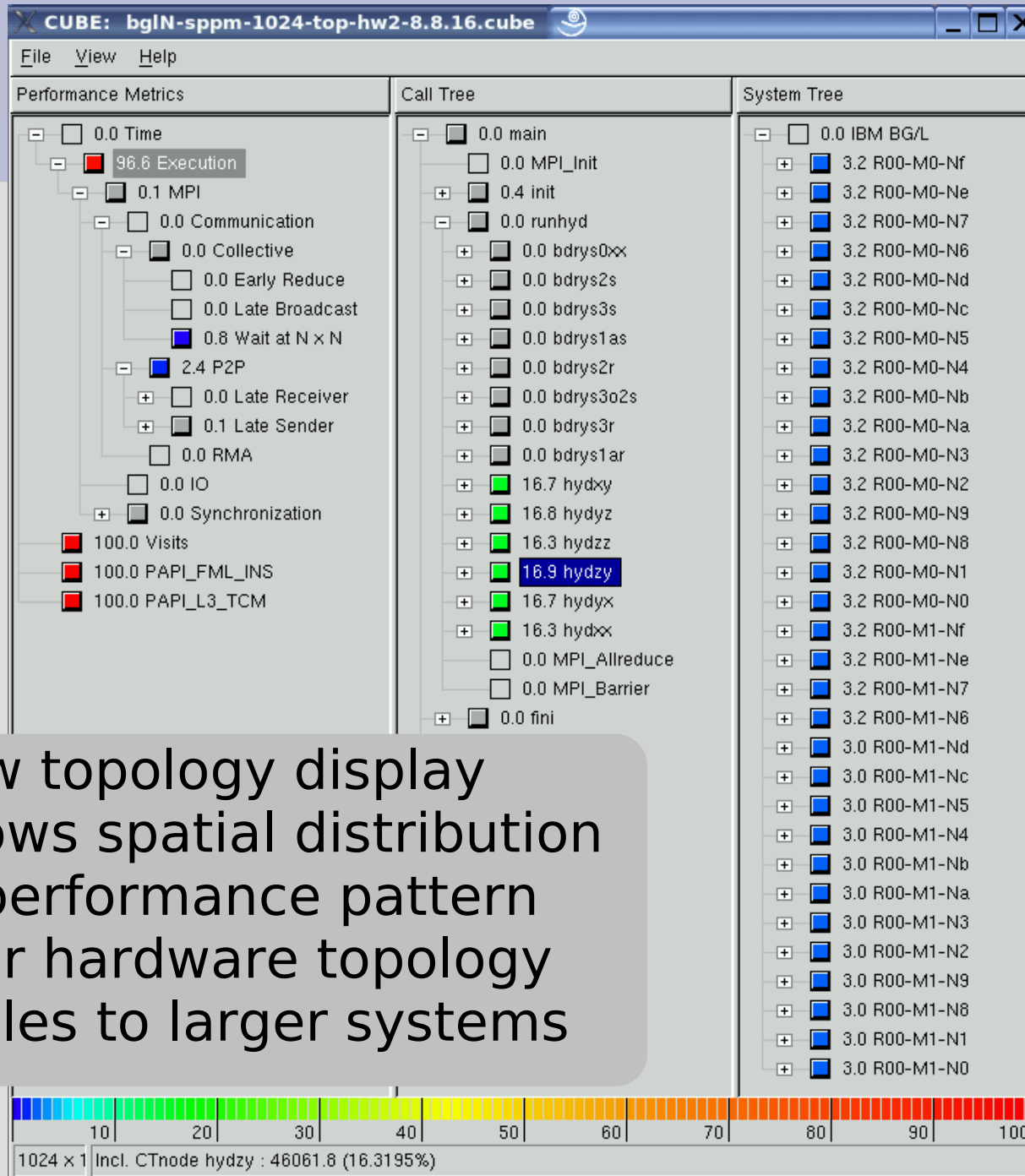
POWER4:

- **DATA_ACCESS**
 - DATA_HIT_L1\$
 - DATA_STORE_INT0_L1\$
 - DATA_LOAD_FROM_L1\$
 - DATA_HIT_L2\$
 - DATA_STORE_INT0_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_INT0_L3\$
 - DATA_LOAD_FROM_L3\$
 - PM_DATA_FROM_L3
 - PM_DATA_FROM_L35
 - DATA_HIT_MEM
 - DATA_STORE_INT0_MEM
 - DATA_LOAD_FROM_MEM
 - PM_DATA_FROM_MEM

Opteron:

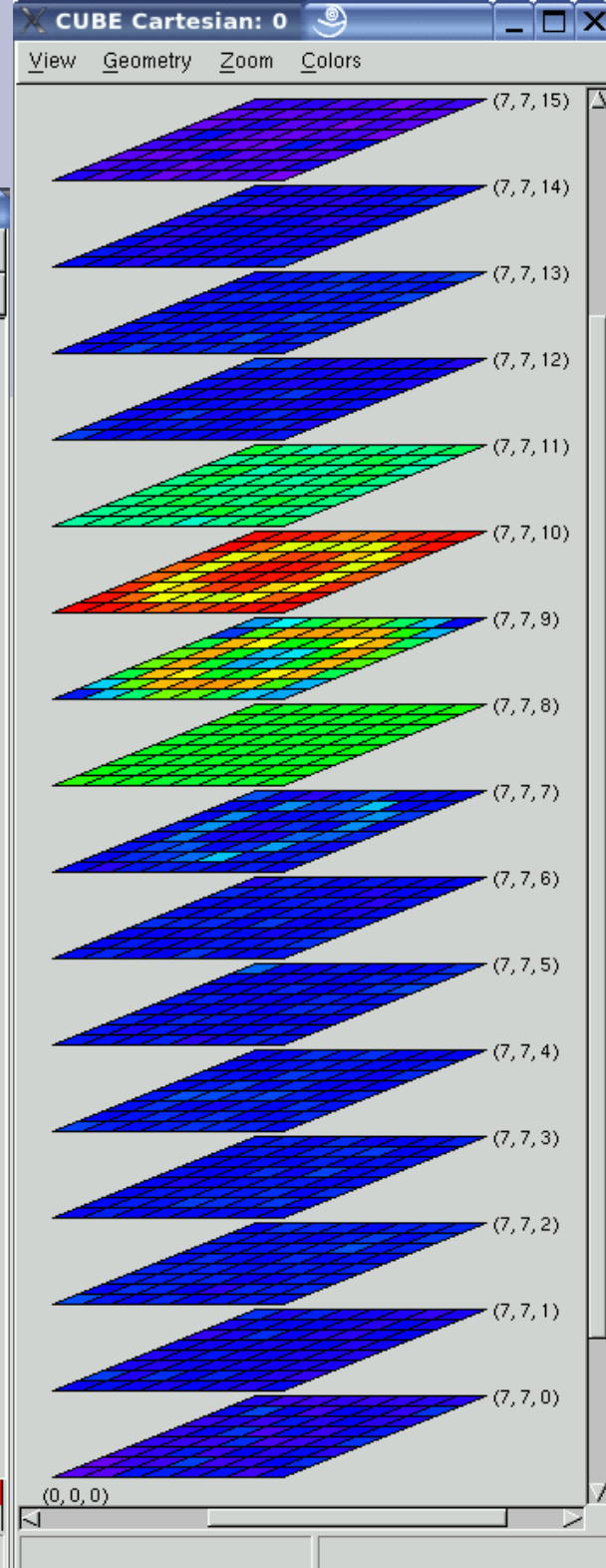
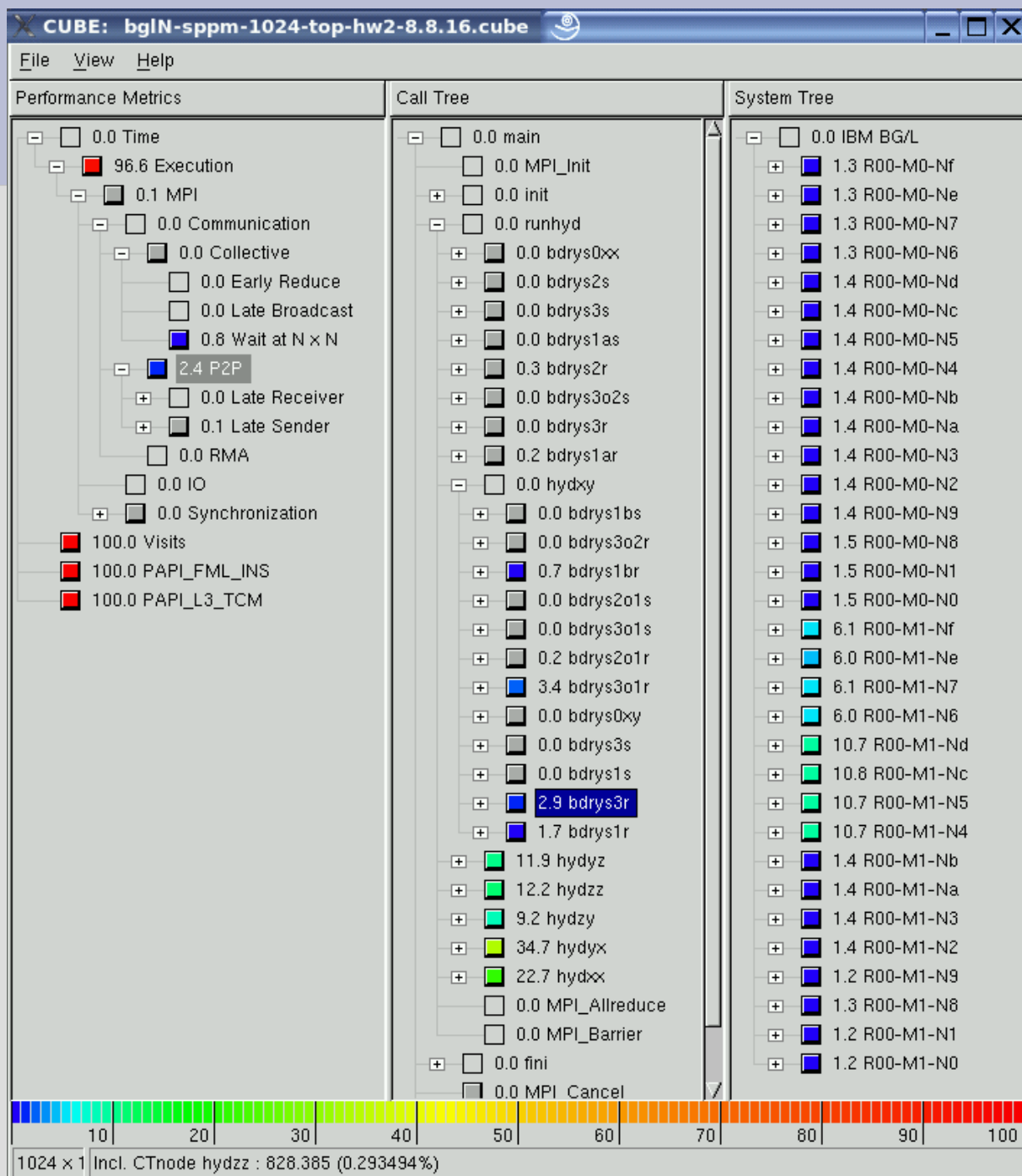
- **DATA_ACCESS**
 - DATA_HIT_L1\$
 - DATA_STORE_INT0_L1\$
 - DATA_LOAD_FROM_L1\$
 - DATA_HIT_L2\$
 - DATA_STORE_INT0_L2\$
 - DC_L2_REFILL_M
 - DATA_LOAD_FROM_L2\$
 - DC_L2_REFILL_O
 - DC_L2_REFILL_E
 - DC_L2_REFILL_S
 - DATA_HIT_L3\$
 - DATA_STORE_INT0_L3\$
 - DATA_LOAD_FROM_L3\$
 - DATA_HIT_MEM
 - DATA_STORE_INT0_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

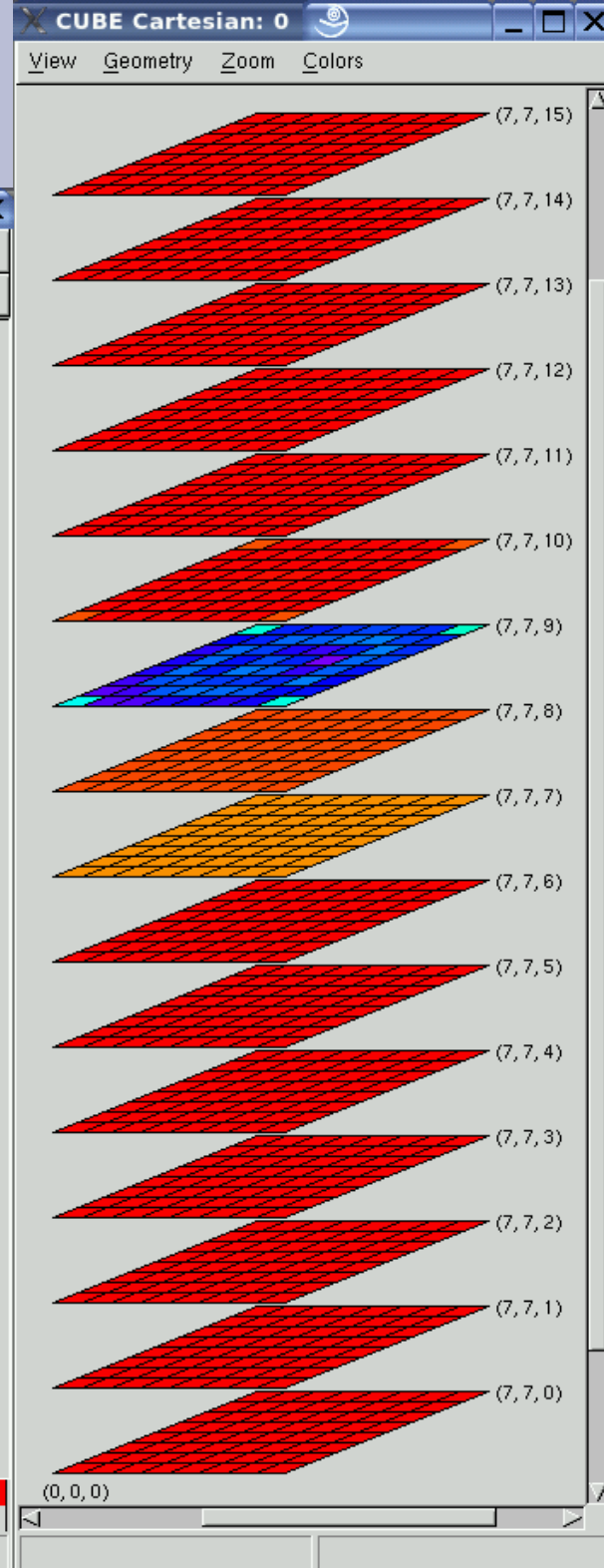
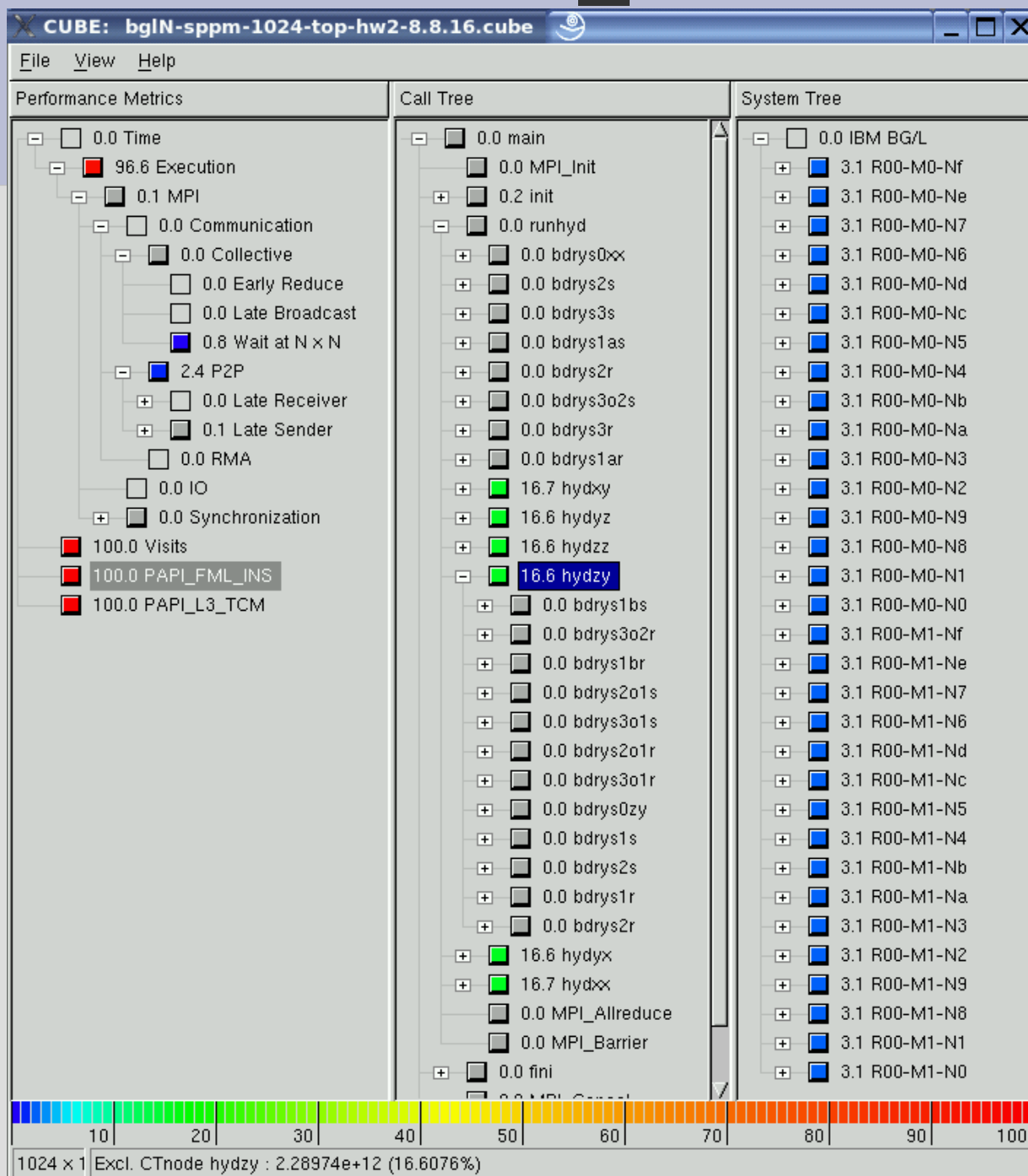


- New topology display
- Shows spatial distribution of performance pattern over hardware topology
- Scales to larger systems

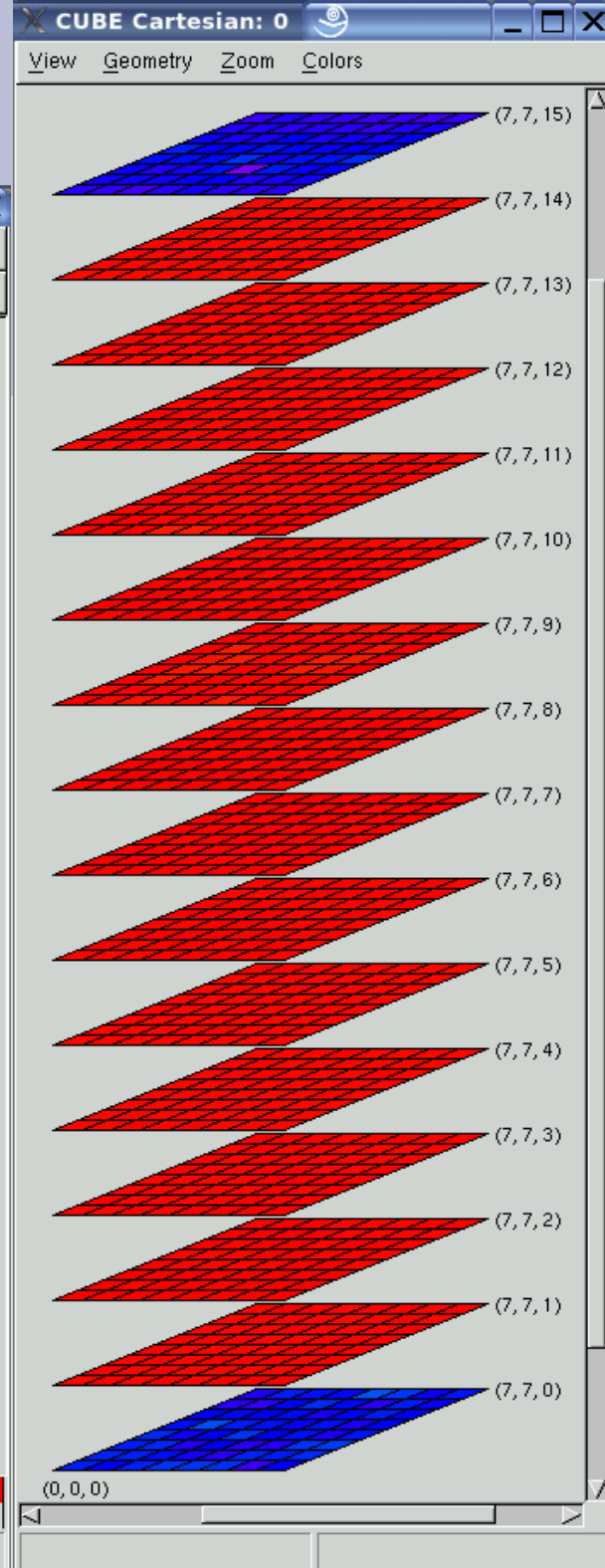
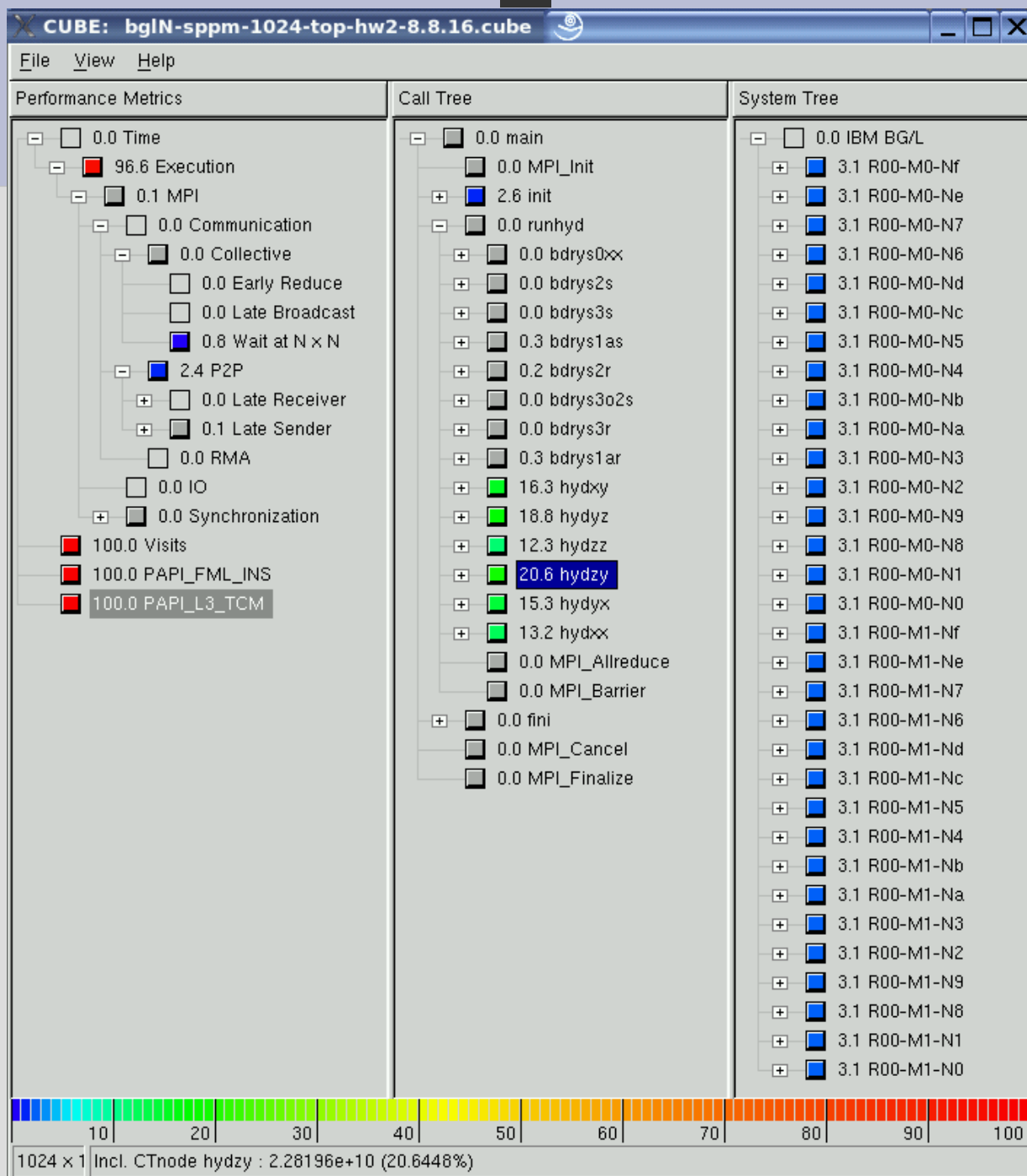
BG/L sPPM MPI P2P



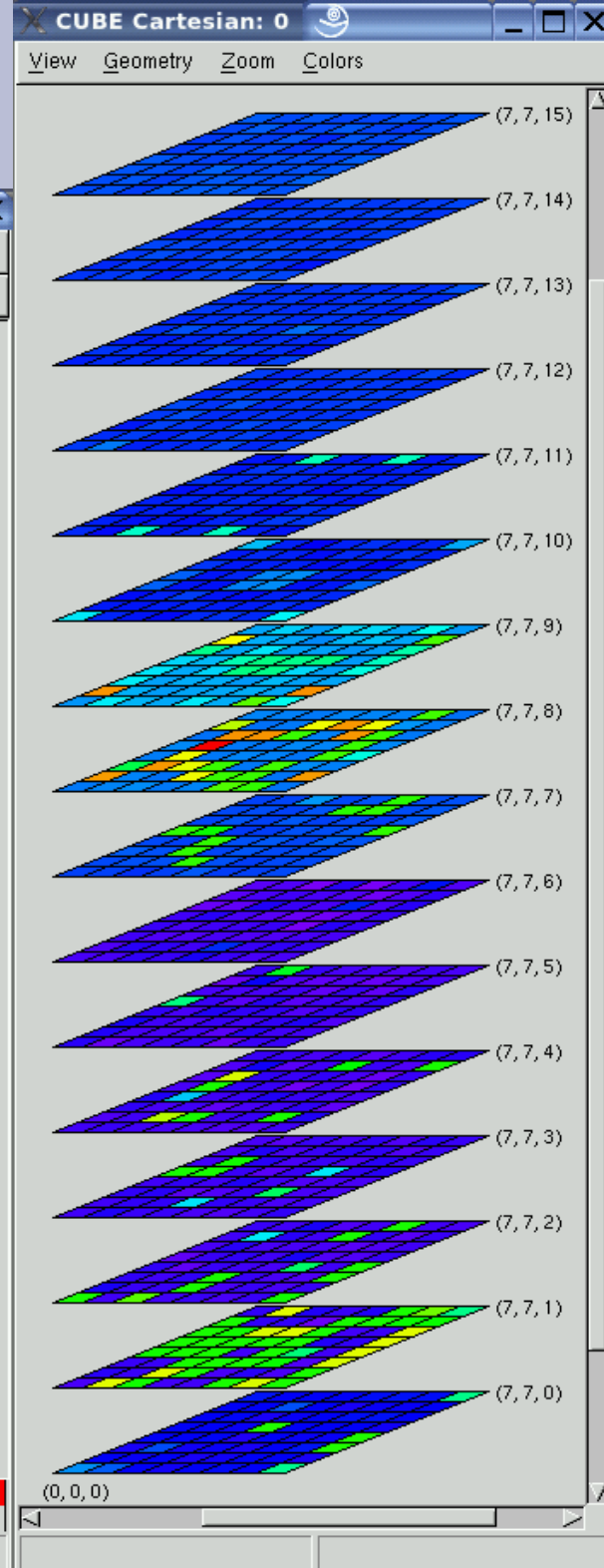
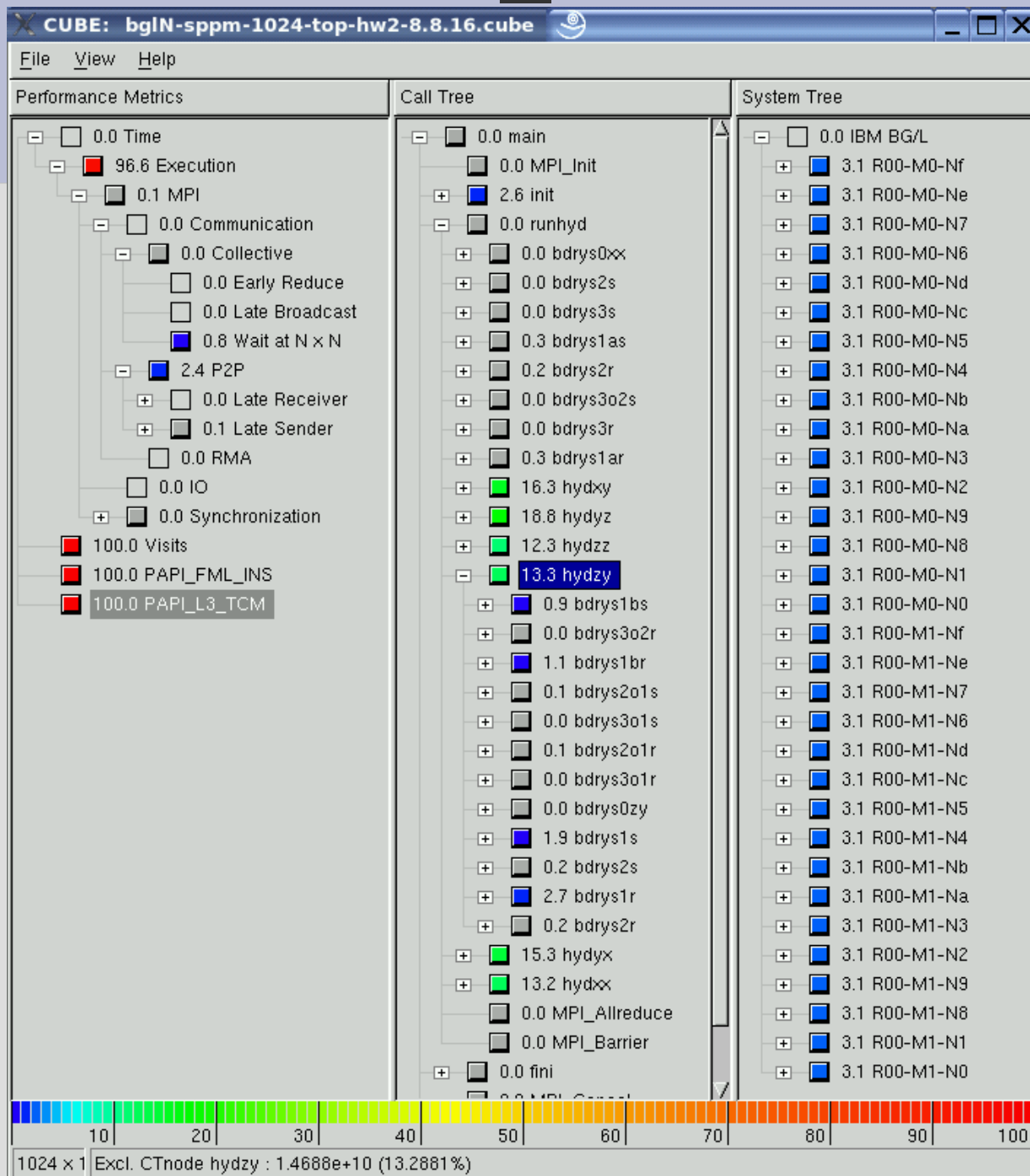
BG/L sPPM FML_INS



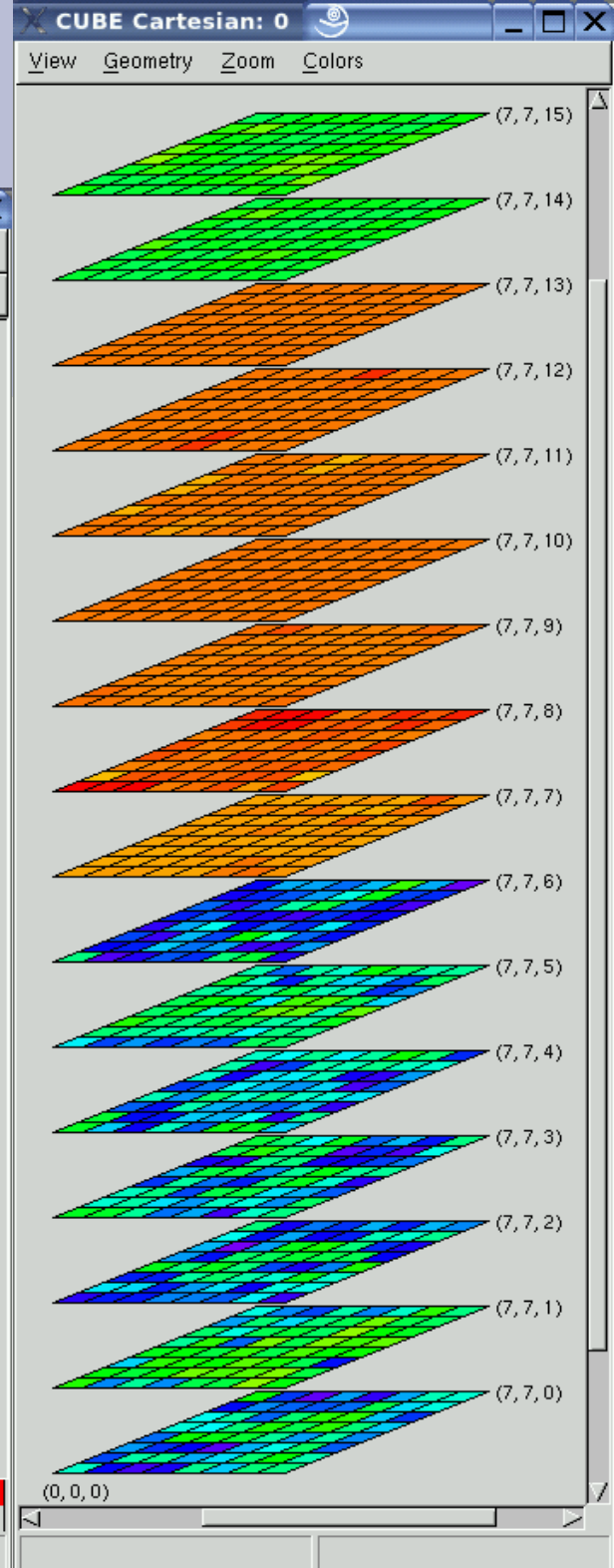
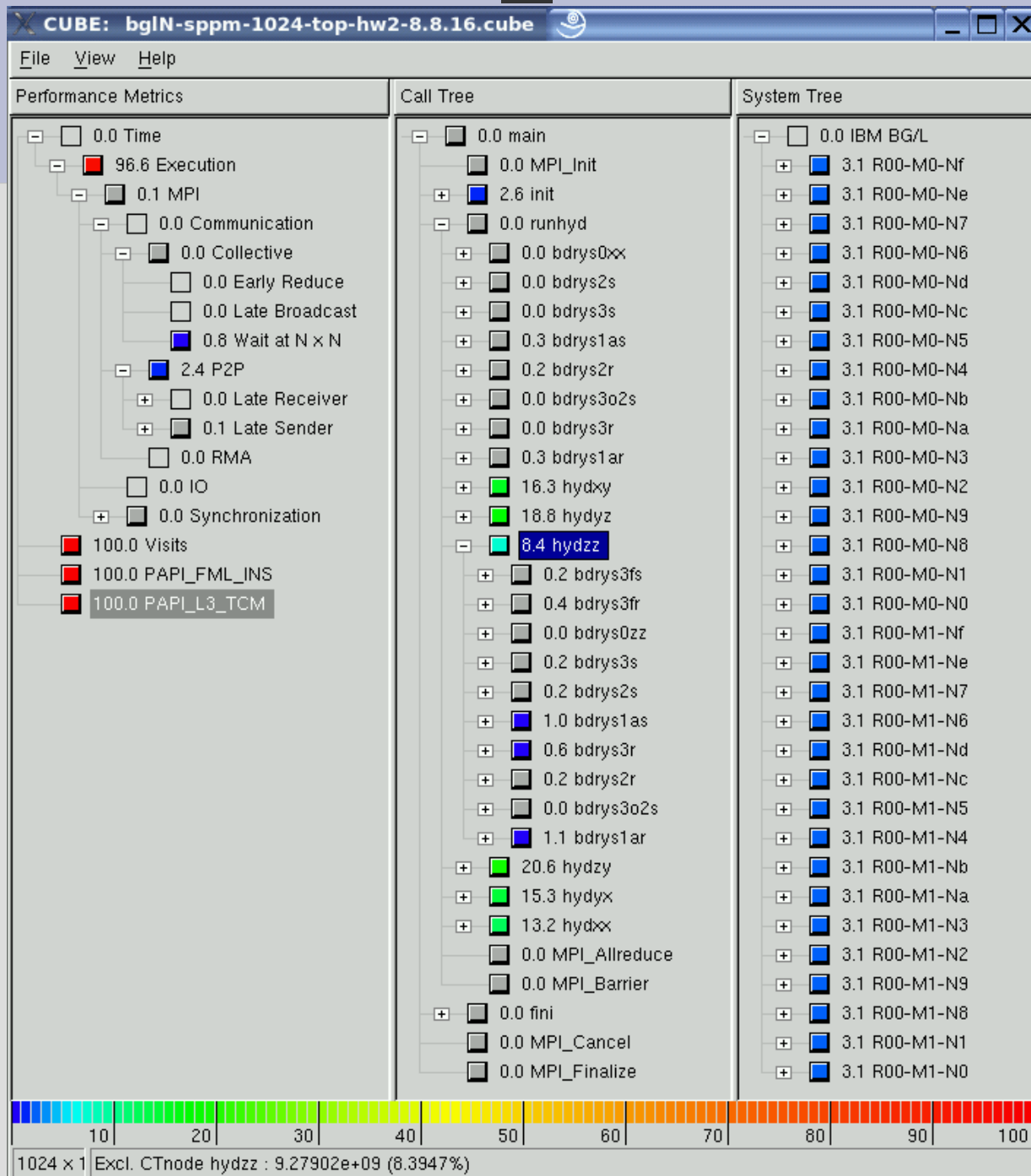
BG/L sPPM L3_TCM



BG/L sPPM L3_TCM.2



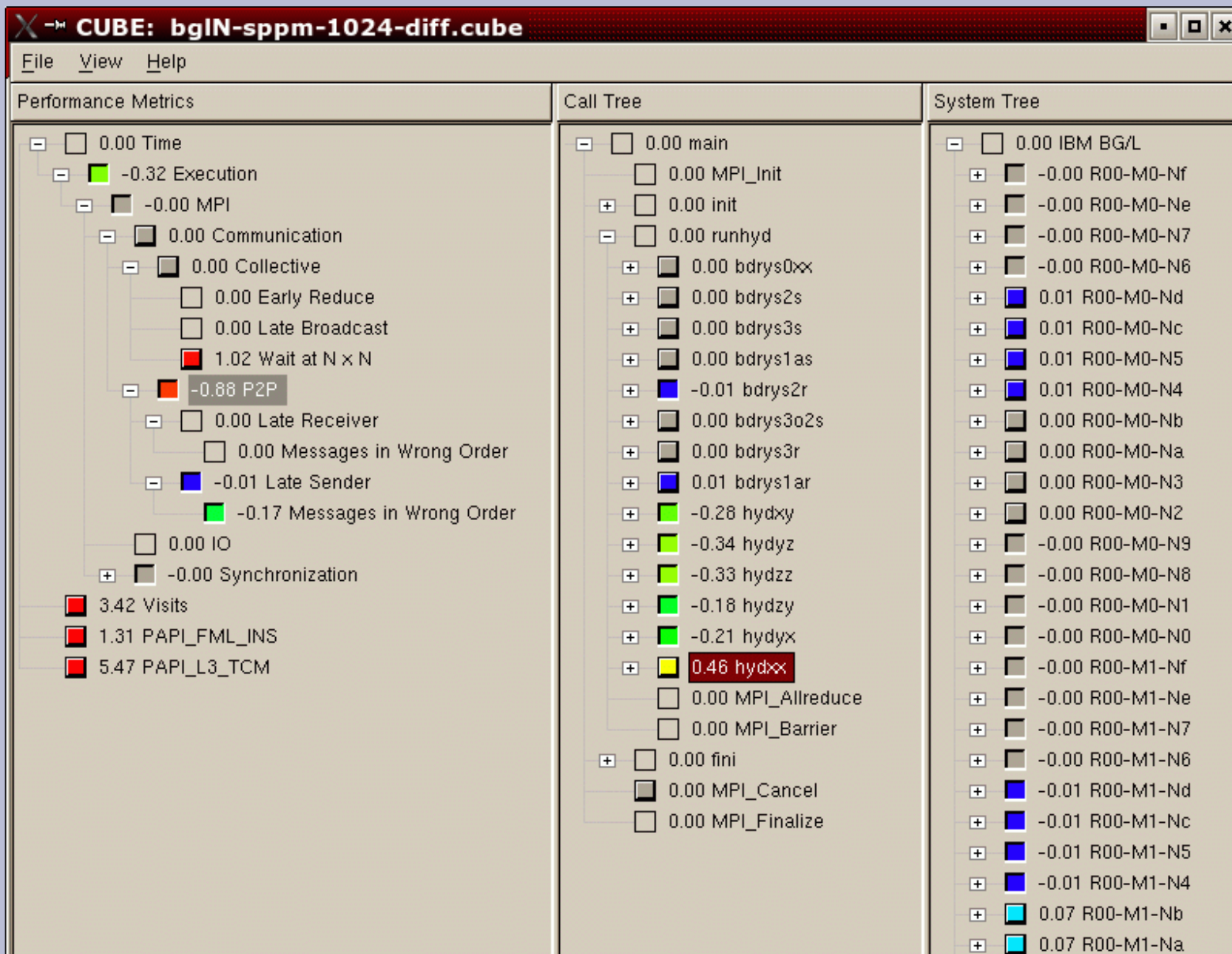
BG/L sPPM L3_TCM.3



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>
- 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