Hardware Performance Monitoring: Current and Future

Shirley Moore

shirley@cs.utk.edu

VI-HPS Inauguration

4July 2007

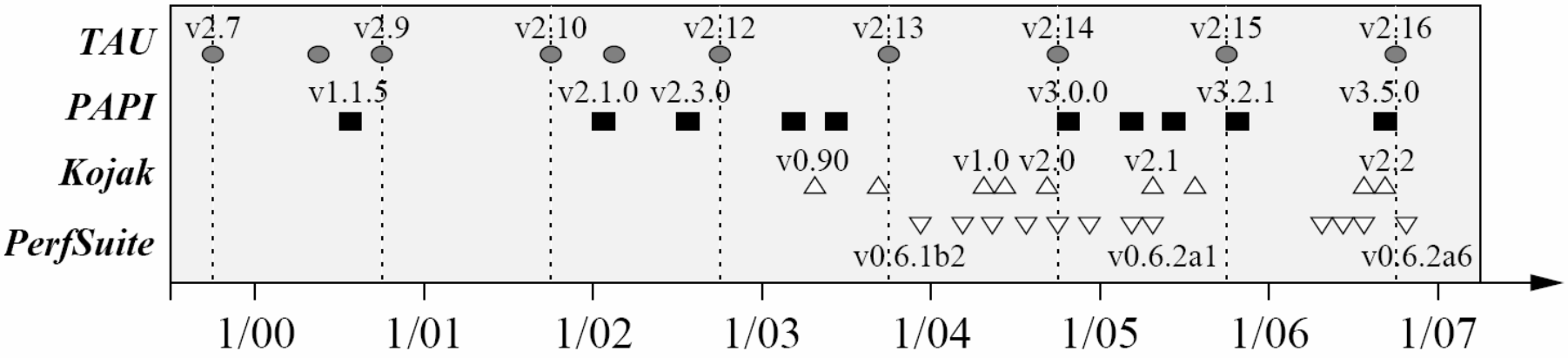
ICL UT
INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Computer Science Department

History of PAPI

- http://icl.cs.utk.edu/papi/
- Started as a Parallel Tools Consortium project in 1998
- Goal was to produce a specification for a portable interface to the hardware performance counters available on most modern microprocessors.

Timeline



Timeline of releases for each tool represented in the project. The vertical dashed lines indicate SC conference dates where the tools are regularly demonstrated. TAU's v1.0 release occurred at SC'97.

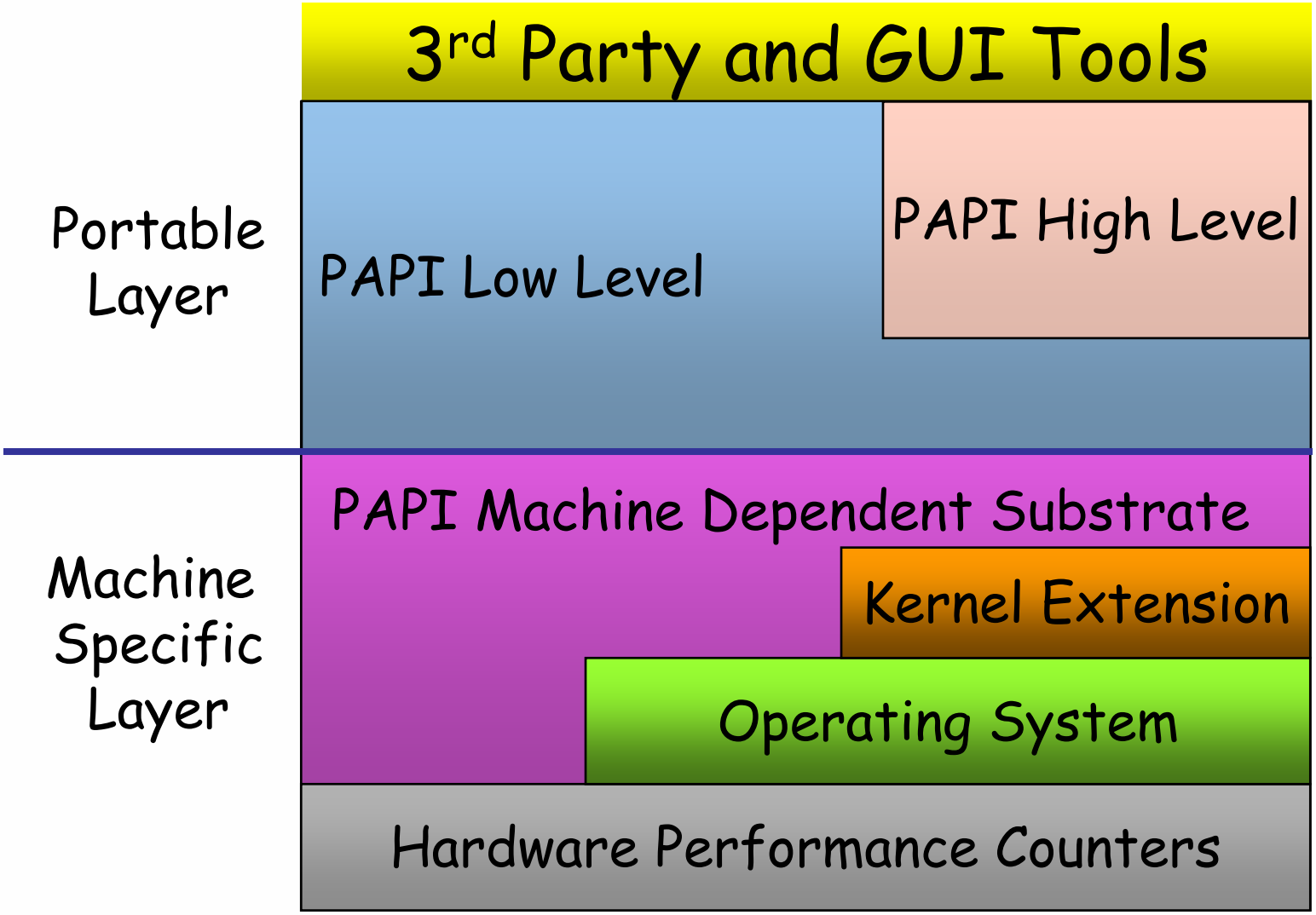
SDCI HPC Improvement:
High-Productivity Performance Engineering
(Tools, Methods, Training) for NSF HPC Applications
Allen D. Malony, Sameer Shende, Shirley Moore, Nick Nystrom, Rick Kufrin

PAPI Counter Interfaces

PAPI provides 3 interfaces to the underlying counter hardware:

1. The low level interface manages hardware events in user defined groups called *EventSets*, and provides access to advanced features.
2. The high level interface provides the ability to start, stop and read the counters for a specified list of events.
3. Graphical and end-user tools provide facile data collection and visualization.

PAPI Implementation



PAPI Hardware Events

- Preset Events
  - Standard set of over 100 events for application performance tuning
  - No standardization of the exact definition
  - Mapped to either single or linear combinations of native events on each platform
  - Use *papi_avail* utility to see what preset events are available on a given platform
- Native Events
  - Any event countable by the CPU
  - Same interface as for preset events
  - Use *papi_native_avail* utility to see all available native events
- Use *papi_event_chooser* utility to select a compatible set of events

PAPI Preset Events

- Of ~100 events, over half are cache related:

```
PAPI_L1_DCH:     Level 1 data cache hits
PAPI_L1_DCA:     Level 1 data cache accesses
PAPI_L1_DCR:     Level 1 data cache reads
PAPI_L1_DCW:     Level 1 data cache writes
PAPI_L1_DCM:     Level 1 data cache misses

PAPI_L1_ICH:     Level 1 instruction cache hits
PAPI_L1_ICA:     Level 1 instruction cache accesses
PAPI_L1_ICR:     Level 1 instruction cache reads
PAPI_L1_ICW:     Level 1 instruction cache writes
PAPI_L1_ICM:     Level 1 instruction cache misses

PAPI_L1_TCH:     Level 1 total cache hits
PAPI_L1_TCA:     Level 1 total cache accesses
PAPI_L1_TCR:     Level 1 total cache reads
PAPI_L1_TCW:     Level 1 total cache writes
PAPI_L1_TCM:     Level 1 cache misses

PAPI_L1_LDM:     Level 1 load misses
PAPI_L1_STM:     Level 1 store misses
```

- **Repeat for Levels 2 and 3…**

PAPI Preset Events (ii)

- Other cache and memory events:

Shared cache	PAPI_CA_SNP:	Requests for a snoop
	PAPI_CA_SHR:	Requests for exclusive access to shared cache line
	PAPI_CA_CLN:	Requests for exclusive access to clean cache line
	PAPI_CA_INV:	Requests for cache line invalidation
	PAPI_CA_ITV:	Requests for cache line intervention
TLB	PAPI_TLB_DM:	Data translation lookaside buffer misses
	PAPI_TLB_IM:	Instruction translation lookaside buffer misses
	PAPI_TLB_TL:	Total translation lookaside buffer misses
	PAPI_TLB_SD:	Translation lookaside buffer shootdowns
	PAPI_LD_INS:	Load instructions
	PAPI_SR_INS:	Store instructions
Resource Stalls	PAPI_MEM_SCY:	Cycles Stalled Waiting for memory accesses
	PAPI_MEM_RCY:	Cycles Stalled Waiting for memory Reads
	PAPI_MEM_WCY:	Cycles Stalled Waiting for memory writes
	PAPI_RES_STL:	Cycles stalled on any resource
	PAPI_FP_STAL:	Cycles the FP unit(s) are stalled

PAPI Preset Events (iii)

- Program flow:

Branches

```
PAPI_BR_INS:    Branch instructions
PAPI_BR_UCN:    Unconditional branch instructions
PAPI_BR_CN:     Conditional branch instructions
PAPI_BR_TKN:    Conditional branch instructions taken
PAPI_BR_NTK:    Conditional branch instructions not taken
PAPI_BR_MSP:    Conditional branch instructions mispredicted
PAPI_BR_PRC:    Conditional branch instructions correctly predicted

PAPI_BTAC_M:    Branch target address cache misses
```

Conditional Stores

```
PAPI_CSR_FAL:   Failed store conditional instructions
PAPI_CSR_SUC:   Successful store conditional instructions
PAPI_CSR_TOT:   Total store conditional instructions
```

PAPI Preset Events (iv)

- Timing, efficiency, pipeline:

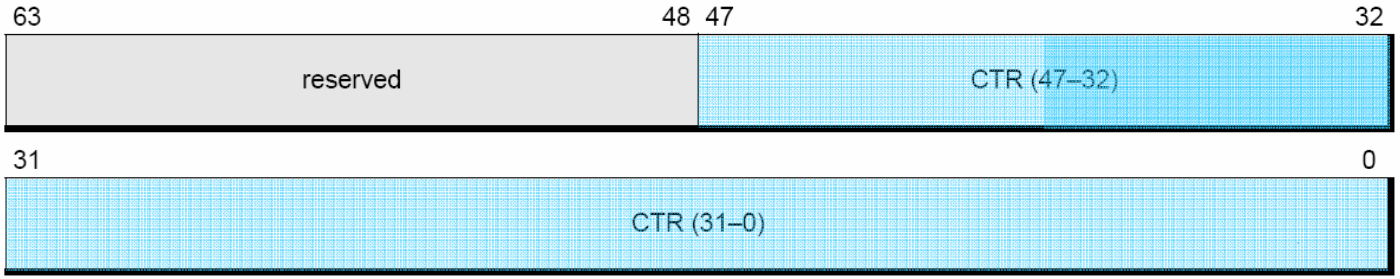
```
PAPI_TOT_CYC:    Total cycles

PAPI_TOT_IIS:    Instructions issued
PAPI_TOT_INS:    Instructions completed
PAPI_INT_INS:    Integer instructions
PAPI_LST_INS:    Load/store instructions completed
PAPI_SYC_INS:    Synchronization instructions completed

PAPI_BRU_IDL:    Cycles branch units are idle
PAPI_FXU_IDL:    Cycles integer units are idle
PAPI_FPU_IDL:    Cycles floating point units are idle
PAPI_LSU_IDL:    Cycles load/store units are idle

PAPI_STL_ICY:    Cycles with no instruction issue
PAPI_FUL_ICY:    Cycles with maximum instruction issue
PAPI_STL_CCY:    Cycles with no instructions completed
PAPI_FUL_CCY:    Cycles with maximum instructions completed

PAPI_HW_INT:     Hardware interrupts
```

PAPI Preset Events (v)

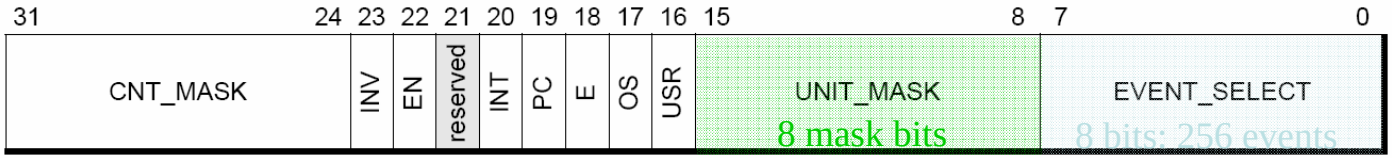
- Floating point:

```
PAPI_FP_INS:   Floating point instructions
PAPI_FP_OPS:   Floating point operations
PAPI_FML_INS:  Floating point multiply instructions
PAPI_FAD_INS:  Floating point add instructions
PAPI_FDV_INS:  Floating point divide instructions
PAPI_FSQ_INS:  Floating point square root instructions
PAPI_FNV_INS:  Floating point inverse instructions
PAPI_FMA_INS:  FMA instructions completed
PAPI_VEC_INS:  Vector/SIMD instructions
```

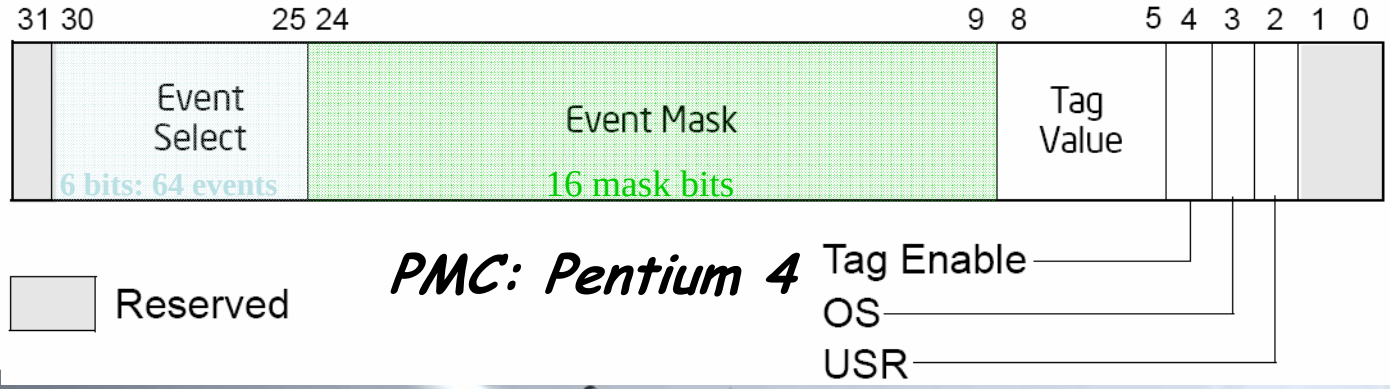
What's a Native Event?



PMD: AMD Athlon, Opteron



PMC: Intel Pentium II, III, M, Core; AMD Athlon, Opteron



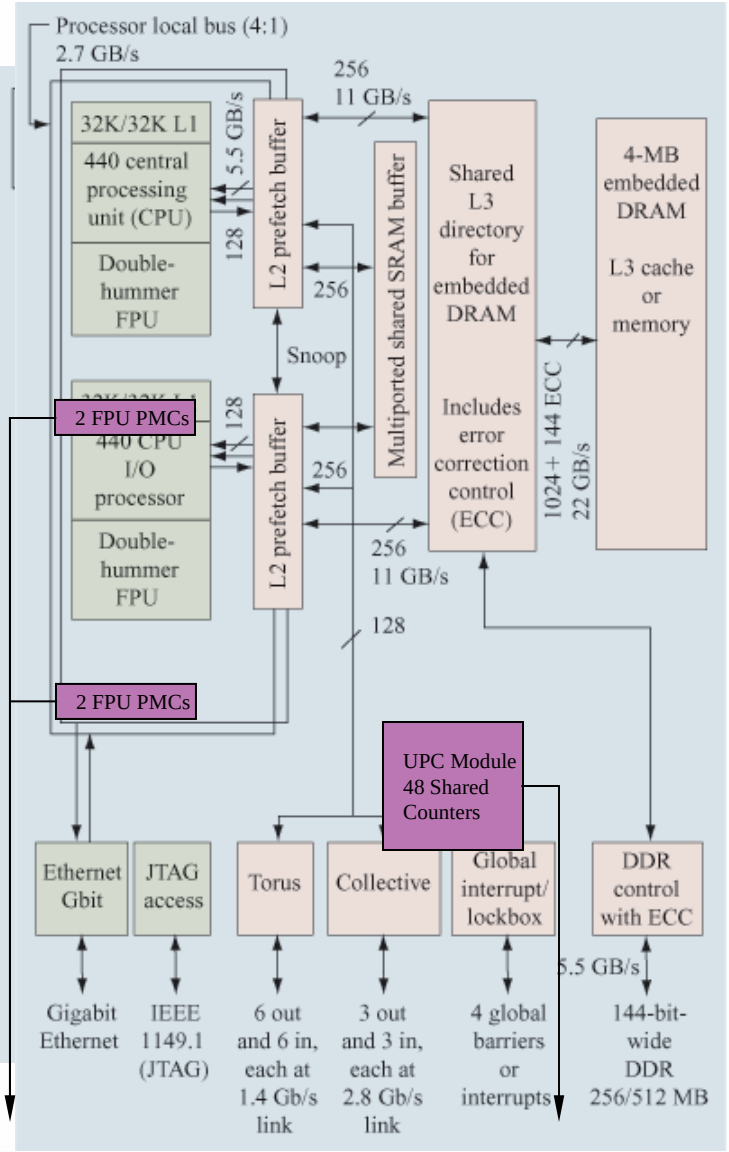
Intel Pentium Core: L2_ST

```
...
{ .pme_name = "L2_ST",
  .pme_code = 0x2a,
  .pme_flags = PFMLIB_CORE_CSPEC,
  .pme_desc =  "L2 store requests",
  .pme_umasks = {
    { .pme_uname = "MESI",
      .pme_udesc = "Any cacheline access",
      .pme_ucode = 0xf\
    },
    { .pme_uname = "I_STATE",
      .pme_udesc = "Invalid cacheline",
      .pme_ucode = 0x1\
    },
    { .pme_uname = "S_STATE",
      .pme_udesc = "Shared cacheline",
      .pme_ucode = 0x2\
    },
    { .pme_uname = "E_STATE",
      .pme_udesc = "Exclusive cacheline",
      .pme_ucode = 0x4\
    },
    { .pme_uname = "M_STATE",
      .pme_udesc = "Modified cacheline",
      .pme_ucode = 0x8\
    }
```

```
    { .pme_uname = "SELF",
      .pme_udesc = "This core",
      .pme_ucode = 0x40\
    },
    { .pme_uname = "BOTH_CORES",
      .pme_udesc = "Both cores",
      .pme_ucode = 0xc0\
    }
   },
   .pme_numasks = 7
},

...
```

PRESET,
PAPI_L2_DCA,
DERIVED_ADD,
L2_LD:SELF:ANY:MESI,
L2_ST:SELF:MESI

PAPI and BG/L



- Performance Counters:
  - 48 UPC Counters
    - shared by both CPUs
    - External to CPU cores
    - 32 bits
  - 2 Counters on each FPU
    - 1 counts load/stores
    - 1 counts arithmetic operations
  - Accessed via blg_perfctr
  - 15 Preset Events
    - 10 PAPI presets
    - 5 Custom BG/L presets
  - 328 native events available

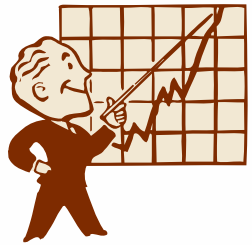
PAPI Data and Instruction Range Qualification

- Implemented a generalized PAPI interface for data structure and instruction address range qualification
- Applied that interface to the specific instance of the Itanium2 platform
- Extended an existing PAPI call, `PAPI_set_opt()`, with the capability of specifying starting and ending addresses of data structures or instructions to be instrumented

```
option.addr.eventset = EventSet;
option.addr.start = (caddr_t)array;
option.addr.end = (caddr_t)(array + size_array);
retval = PAPI_set_opt(PAPI_DATA_ADDRESS, &option);
```

- An instruction range can be set using `PAPI_INSTR_ADDRESS`
- `papi_native_avail` was modified to list events that support data or instruction address range qualification.

Tools that use PAPI

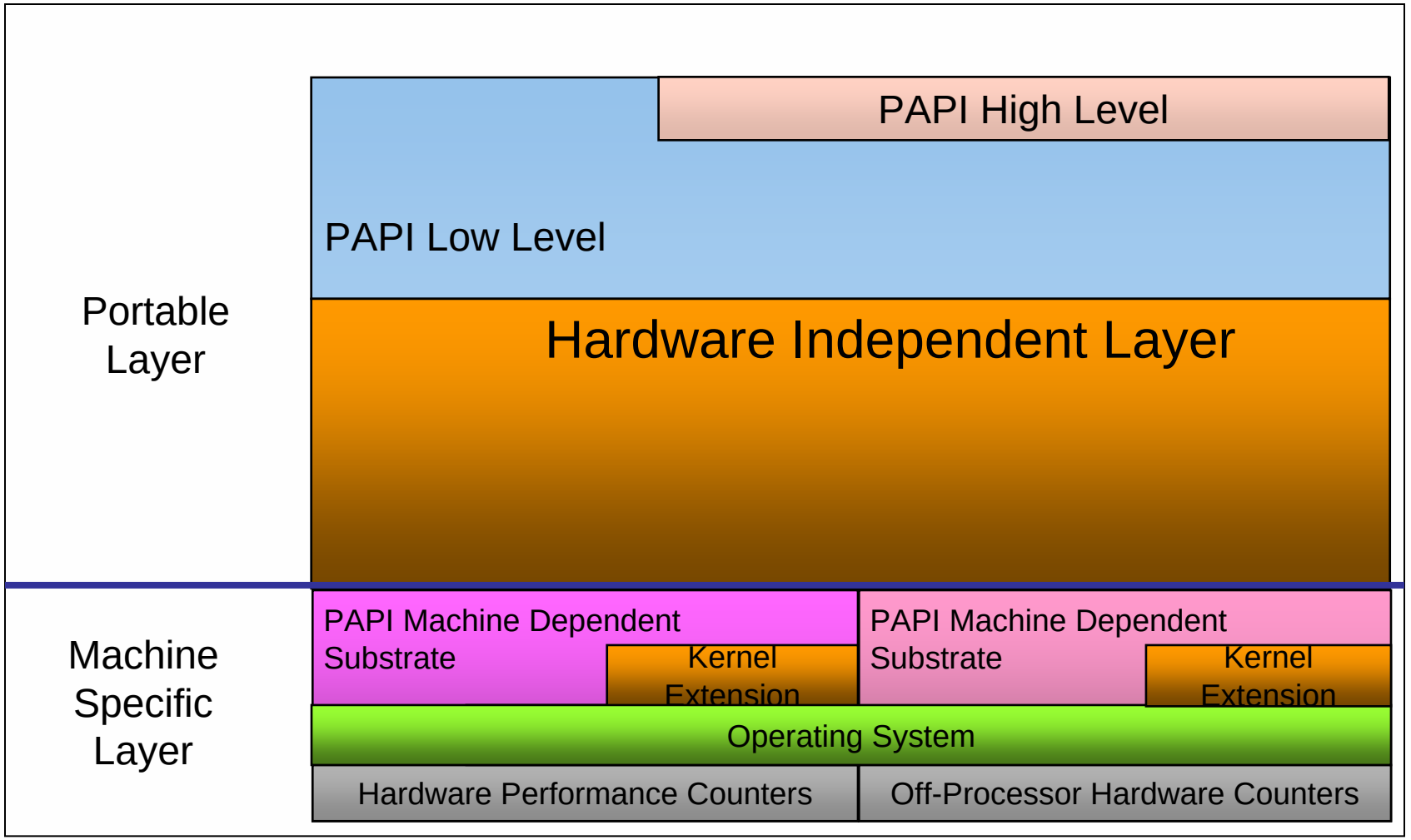


- TAU (U Oregon) http://www.cs.uoregon.edu/research/tau/
- HPCToolkit (Rice Univ) http://hipersoft.cs.rice.edu/hpctoolkit/
- KOJAK (UTK, FZ Juelich) http://icl.cs.utk.edu/kojak/
- PerfSuite (NCSA) http://perfsuite.ncsa.uiuc.edu/
- Titanium (UC Berkeley)
  http://www.cs.berkeley.edu/Research/Projects/titanium/
- SCALEA (Thomas Fahringer, U Innsbruck)
  http://www.par.univie.ac.at/project/scalea/
- Open|Speedshop (SGI)
  http://oss.sgi.com/projects/openspeedshop/
- SvPablo (UNC Renaissance Computing Institute)
  http://www.renci.unc.edu/Software/Pablo/pablo.htm

Component PAPI (PAPI-C)

- Goals:
  - Support simultaneous access to on- and off-processor counters
  - Isolate hardware dependent code in a separable ‘substrate’ module
  - Extend platform independent code to support multiple simultaneous substrates
  - Add or modify API calls to support access to any of several substrates
  - Modify build environment for easy selection and configuration of multiple available substrates
- Will be released as PAPI 4.0

Architecture for Support of Multiple Components



PAPI-C Status

- PAPI 3.9 pre-release available with documentation
- Implemented Myrinet substrate (native counters)
- Implemented ACPI temperature sensor substrate
- Working on Inifinband and Cray Seastar substrates (access to Seastar counters not available under Catamount but expected under CNL)
- Asked by Cray engineers for input on desired metrics for next network switch
- Tested on HPC Challenge benchmarks
- Tested platforms include Pentium III, Pentium 4, Core2Duo, Itanium (I and II) and AMD Opteron

PAPI-C New Routines

- PAPI_get_component_info()
- PAPI_num_cmp_hwctrs()
- PAPI_get_cmp_opt()
- PAPI_set_cmp_opt()
- PAPI_set_cmp_domain()
- PAPI_set_cmp_granularity()

PAPI-C Building and Linking

- CPU components are automatically detected by *configure* and included in the build
- CPU component assumed to be present and always configured as component 0
- To include additional components, use configure option

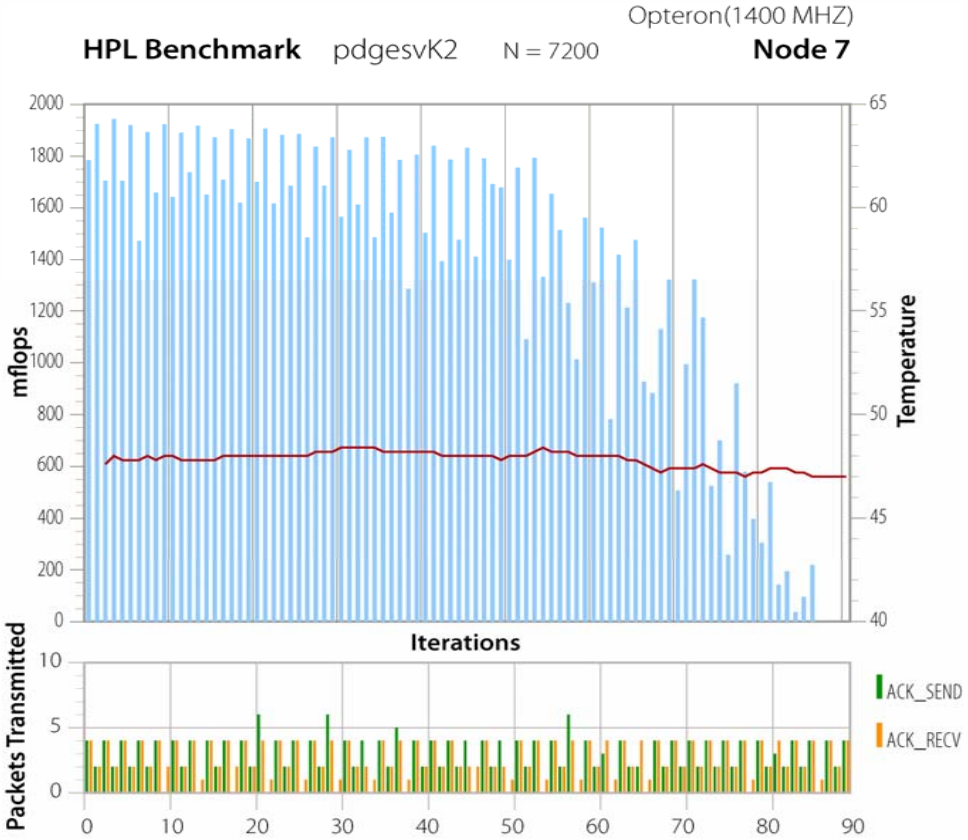
  --with-<cmp> = yes
- Currently supported components
  - with-acpi = yes
  - with-mx = yes
  - with-net = yes
- The make process compiles and links sources for all requested components into a single library

Myrinet MX Counters

LANAI_UPTIME	ACK_NACK_FRAMES_IN_PIPE	REPLY_SEND	ROUTE_DISPERSION
COUNTERS_UPTIME	NACK_BAD_ENDPT	REPLY_RECV	OUT_OF_SEND_HANDLES
BAD_CRC8	NACK_ENDPT_CLOSED	QUERY_UNKNOWN	OUT_OF_PULL_HANDLES
BAD_CRC32	NACK_BAD_SESSION	DATA_SEND_NULL	OUT_OF_PUSH_HANDLES
UNSTRIPPED_ROUTE	NACK_BAD_RDMAWIN	DATA_SEND_SMALL	MEDIUM_CONT_RACE
PKT_DESC_INVALID	NACK_EVENTQ_FULL	DATA_SEND_MEDIUM	CMD_TYPE_UNKNOWN
RECV_PKT_ERRORS	SEND_BAD_RDMAWIN	DATA_SEND_RNDV	UREQ_TYPE_UNKNOWN
PKT_MISROUTED	CONNECT_TIMEOUT	DATA_SEND_PULL	INTERRUPTS_OVERRUN
DATA_SRC_UNKNOWN	CONNECT_SRC_UNKNOWN	DATA_RECV_NULL	WAITING_FOR_INTERRUPT_DMA
DATA_BAD_ENDPT	QUERY_BAD_MAGIC	DATA_RECV_SMALL_INLINE	WAITING_FOR_INTERRUPT_ACK
DATA_ENDPT_CLOSED	QUERY_TIMED_OUT	DATA_RECV_SMALL_COPY	WAITING_FOR_INTERRUPT_TIMER
DATA_BAD_SESSION	QUERY_SRC_UNKNOWN	DATA_RECV_MEDIUM	SLABS_RECYCLING
PUSH_BAD_WINDOW	RAW_SENDS	DATA_RECV_RNDV	SLABS_PRESSURE
PUSH_DUPLICATE	RAW_RECEIVES	DATA_RECV_PULL	SLABS_STARVATION
PUSH_OBSOLETE	RAW_OVERSIZED_PACKETS	ETHER_SEND_UNICAST_CNT	OUT_OF_RDMA_HANDLES
PUSH_RACE_DRIVER	RAW_RECV_OVERRUN	ETHER_SEND_MULTICAST_CNT	EVENTQ_FULL
PUSH_BAD_SEND_HANDLE_MAGIC	RAW_DISABLED	ETHER_RECV_SMALL_CNT	BUFFER_DROP
PUSH_BAD_SRC_MAGIC	CONNECT_SEND	ETHER_RECV_BIG_CNT	MEMORY_DROP
PULL_OBSOLETE	CONNECT_RECV	ETHER_OVERRUN	HARDWARE_FLOW_CONTROL
PULL_NOTIFY_OBSOLETE	ACK_SEND	ETHER_OVERSIZED	SIMULATED_PACKETS_LOST
PULL_RACE_DRIVER	ACK_RECV	DATA_RECV_NO_CREDITS	LOGGING_FRAMES_DUMPED
ACK_BAD_TYPE	PUSH_SEND	PACKETS_RESENT	WAKE_INTERRUPTS
ACK_BAD_MAGIC	PUSH_RECV	PACKETS_DROPPED	AVERTED_WAKEUP_RACE
ACK_RESEND_RACE	QUERY_SEND	MAPPER_ROUTES_UPDATE	DMA_METADATA_RACE
LATE_ACKh	QUERY_RECV		

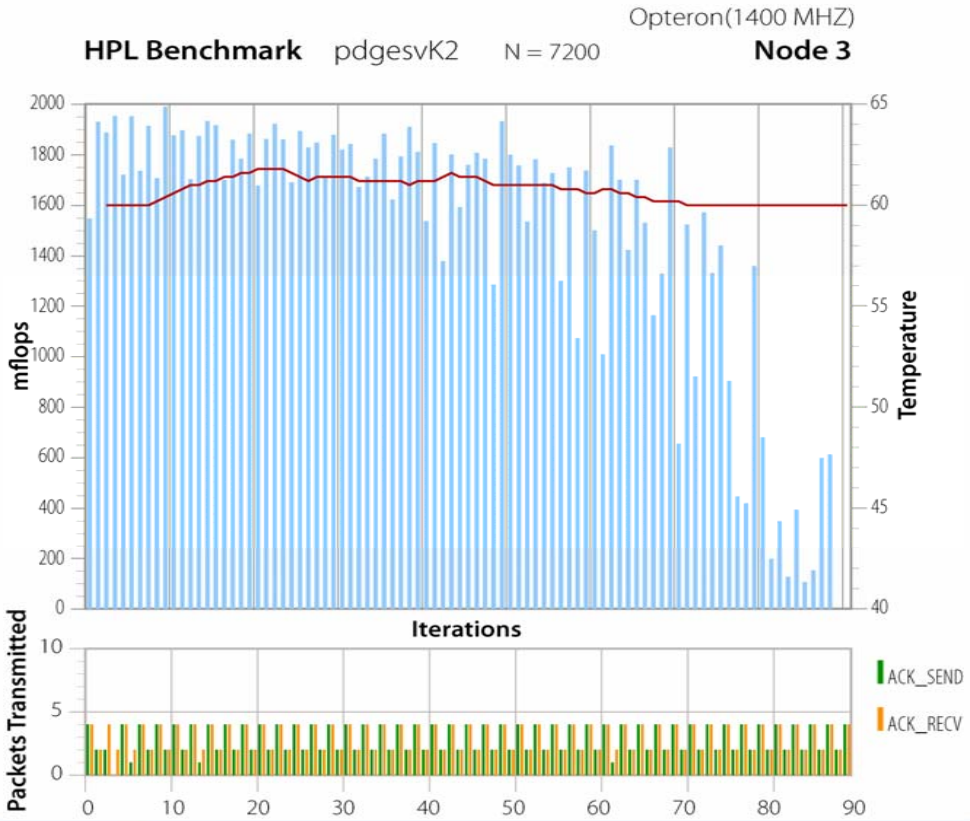
Multiple Measurements

- HPCC HPL benchmark on Opteron with 3 performance metrics:
  - FLOPS; Temperature; Network Sends/Receives
    - Temperature is from an on-chip thermal diode



Multiple Measurements (2)

- HPCC HPL benchmark on Opteron with 3 performance metrics:
  - FLOPS; Temperature; Network Sends/Receives
    - Temperature is from an on-chip thermal diode



Perfctr

- Written by Mikael Petterson
  - Labor of love…
  - First available: Fall 1999
  - First PAPI use: Fall 2000
- Supports:
  - Intel Pentium II, III, 4, M, Core
  - AMD K7 (Athlon), K8 (Opteron)
  - IBM PowerPC 970, POWER4, POWER5

Perfctr Features

- Patches the Linux kernel
  - Saves perf counters on context switch
  - Virtualizes counters to 64-bits
  - Memory-maps counters for fast access
  - Supports counter overflow interrupts where available
- User space library
  - PAPI uses about a dozen calls

Perfctr Timeline

- Steady development
  - 1999 – 2004
- Concerted effort for kernel inclusion
  - May 2004 – May 2005
- Ported to Cray Catamount; Power Linux
  - ~ 2005
- Maintenance only
  - 2005 →

Perfmon

- Written by Stephane Eranian @ HP
- Originally Itanium only
  - Built-in to the Linux-ia64 kernel since 2.4.0
- System call interface
- libpfm helper library for bookkeeping

Perfmon2*

- Provides a generic interface to access PMU
  - Not dedicated to one app, avoid fragmentation
- Must be portable across all PMU models:
  - Almost all PMU-specific knowledge in user level libraries
- Supports per-thread monitoring
  - Self-monitoring, unmodified binaries, attach/detach
  - multi-threaded and multi-process workloads
- Supports system-wide monitoring
- Supports counting and sampling
- No modification to applications or system
- Built-in, efficient, robust, secure, simple, documented

* Slide contents courtesy of Stephane Eranian

Perfmon2

- Setup done through external support library
- Uses a system call for counting operations
  - More flexibility, ties with ctxsw, exit, fork
  - Kernel compile-time option on Linux
- Perfmon2 context encapsulates all PMU state
  - Each context uniquely identified by file descriptor
- int perfmonctl(int fd, int cmd, void *arg, int narg)

PFM_CREATE_CONTEXT	PFM_READ_PMDS	PFM_START
PFM_WRITE_PMCS	PFM_LOAD_CONTEXT	PFM_STOP
PFM_WRITE_PMDS	PFM_UNLOAD_CONTEXT	PFM_RESTART
PFM_CREATE_EVTSET	PFM_DELETE_EVTSET	PFM_GETINFO_EVTSET
PFM_GETINFO_PMCS	PFM_GETINFO_PMDS	
PFM_GET_CONFIG	PFM_SET_CONFIG	

Perfmon2 Features

- Support today for:
  - Intel Itanium, P6, M, Core, Pentium4, AMD Opteron, IBM Power, MIPS
- Full native event tables for supported processors
- Kernel based Multiplexing
  - Event set chaining
- Kernel based Sampling/Overflow
  - Time or event based
  - Custom sampling buffers

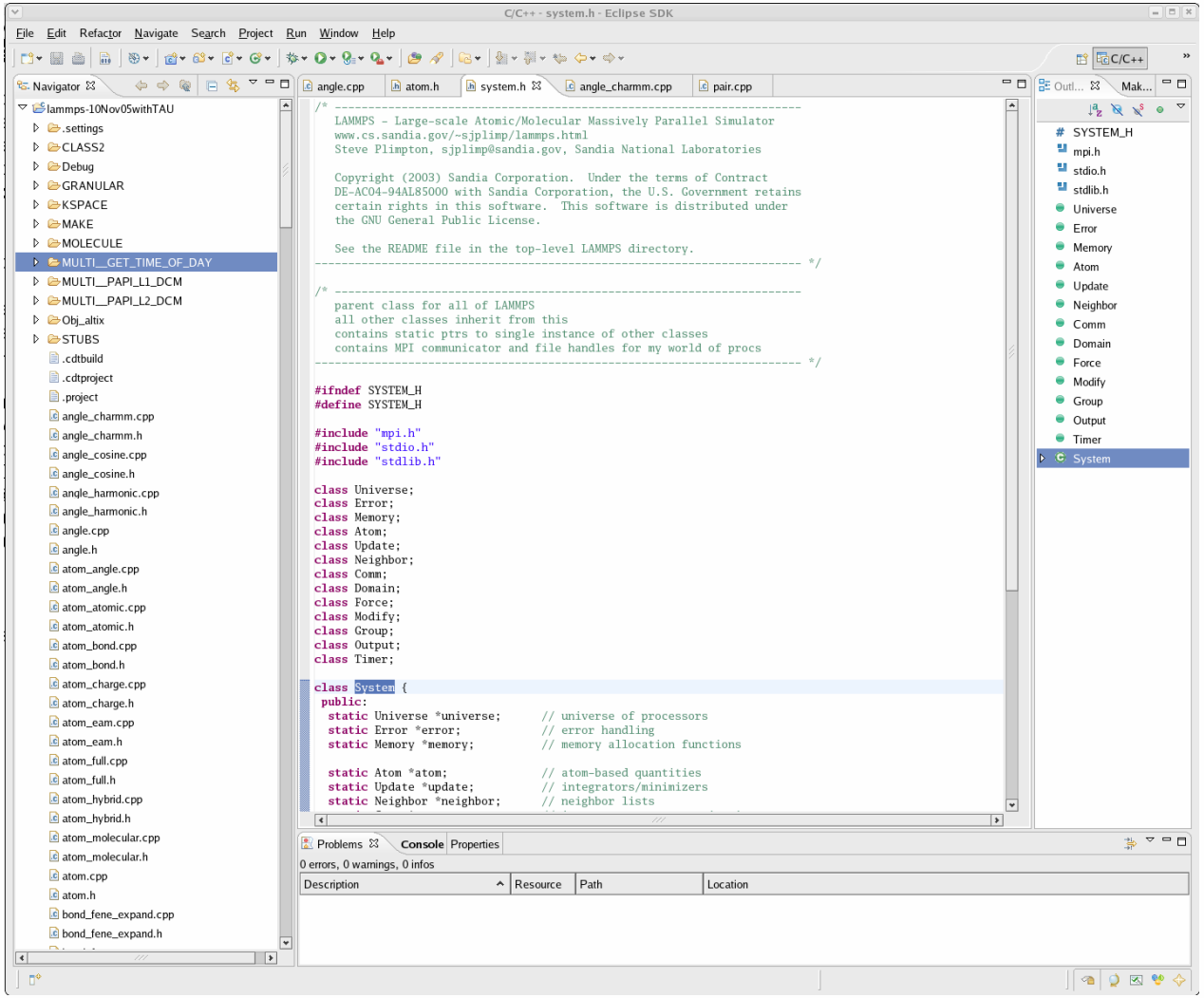
Next Steps

- Kernel integration
  - Discussion underway *now*
  - Possible inclusion in 2.6.22 kernel
- Implementation in Cray CNK, X2
- Cell
  - IBM engineers have started a port
- Leverage libpfm for PAPI native events
  - Migration underway for P6, Core, P4, Opteron
- Begin testing on perfmon2 patched kernels
  - Torc10 currently being tested
  - Woodstock dual-boot?

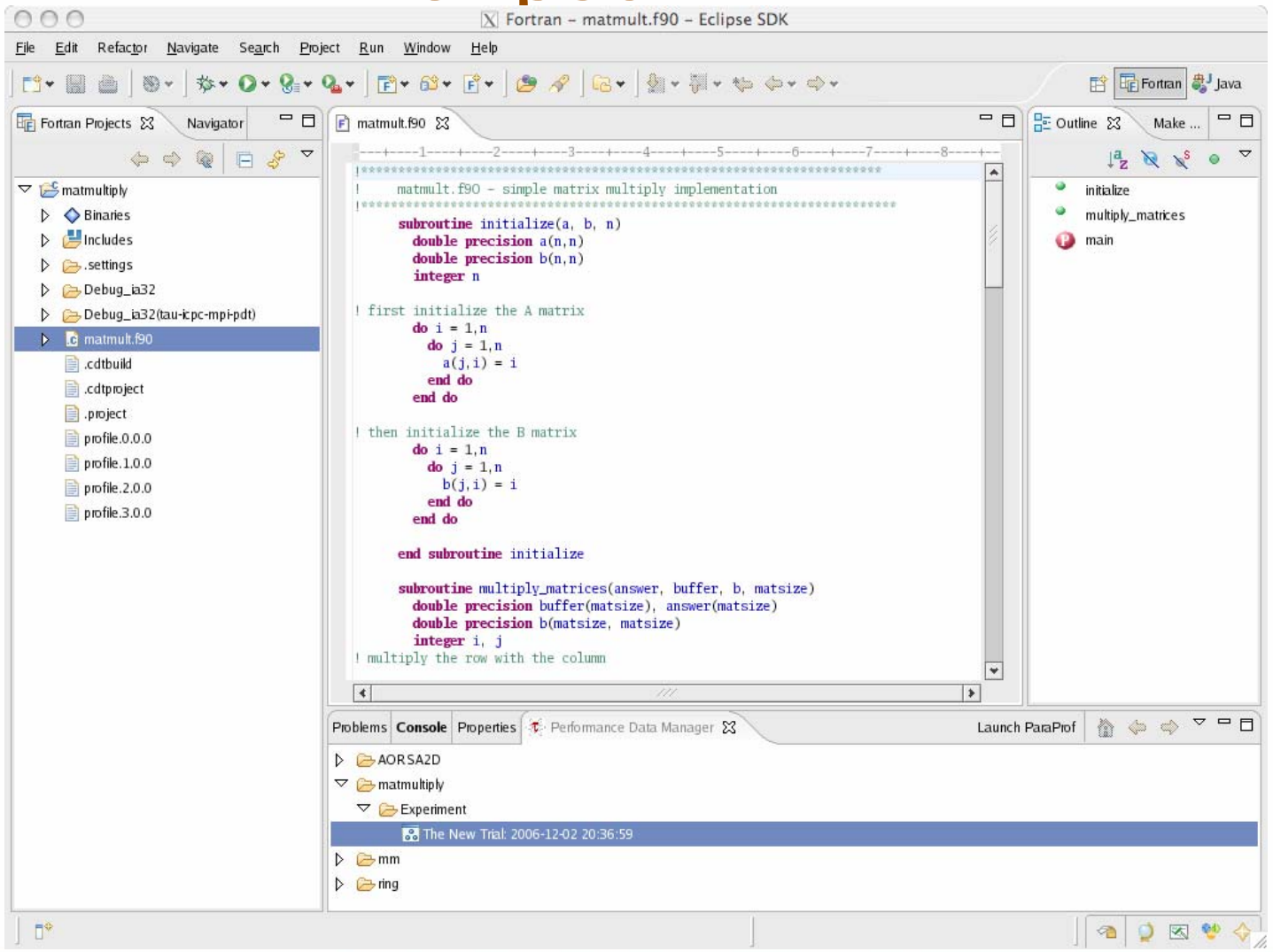
Cell Broadband Engine

- Each Cell contains 1 PPU and 8 SPUs.
  - …and 1 PMU external to all of these.
  - 8 16-bit counters configurable as 4 32-bit counters
  - 1024 slot 128-bit trace buffer
  - 400 native events
- Working with IBM engineers on
  - developing perfmon2 pfmlib layer for Cell BE
  - Linux Cell BE kernel modifications
  - Porting PAPI-C (LANL grant)

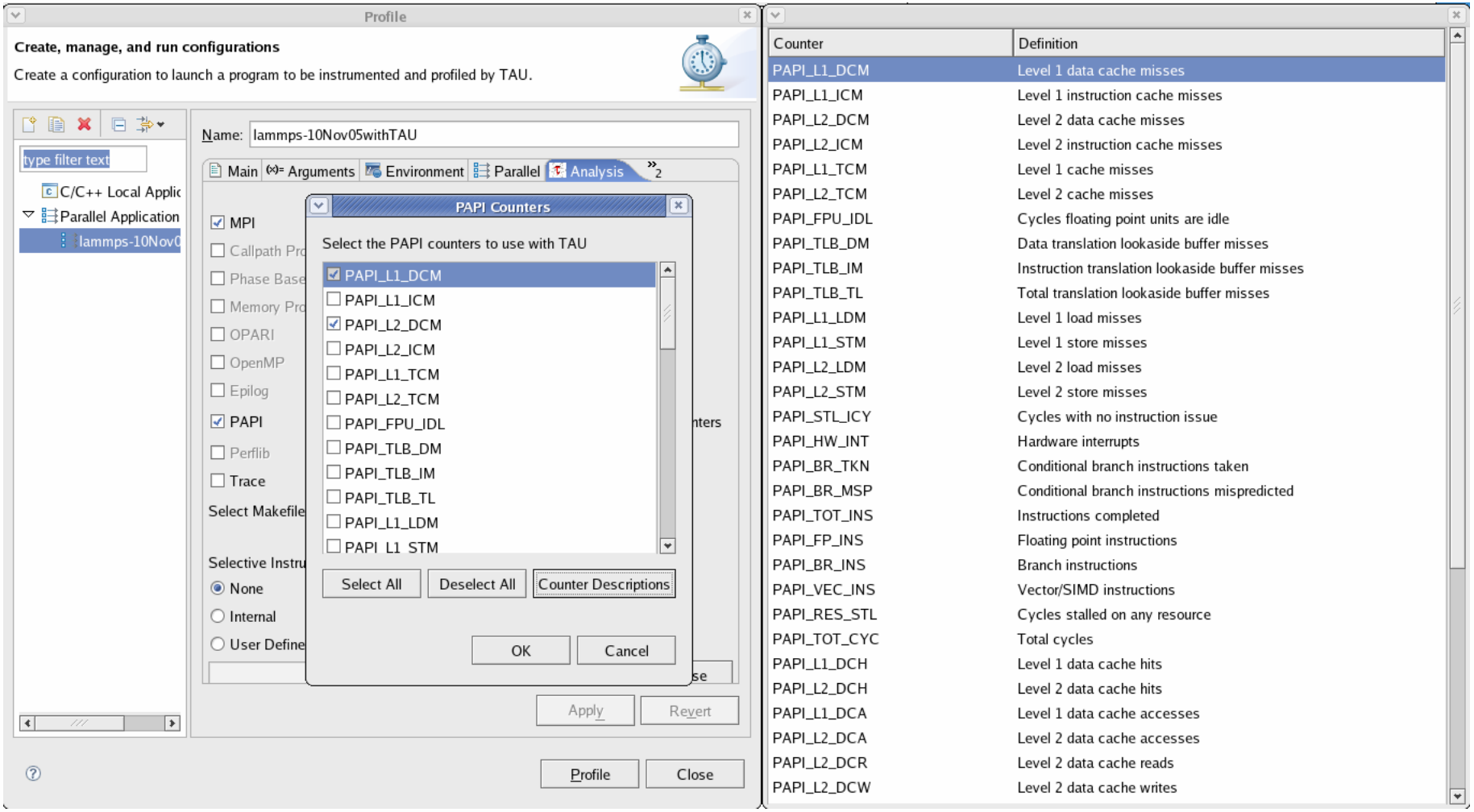
Eclipse PTP IDE



Performance Evaluation within Eclipse PTP



TAU and PAPI Plugins for Eclipse PTP



Conclusions

- PAPI has a long track record of successful adoption and use.
- New architectures pose a challenge for off-processor hardware monitoring as well as interpretation of counter values.
- Integration of perfmon2 into the Linux kernel will broaden the base of PAPI users still further.

Potential VI-HPS Interactions

- Hardware monitoring support for performance analysis tools, debugging, applications
  - Tell us what you need and we will implement or talk to vendors.
  - Error counters may be useful for debugging.
- Deeper understanding of architectures → better mapping of applications onto them