

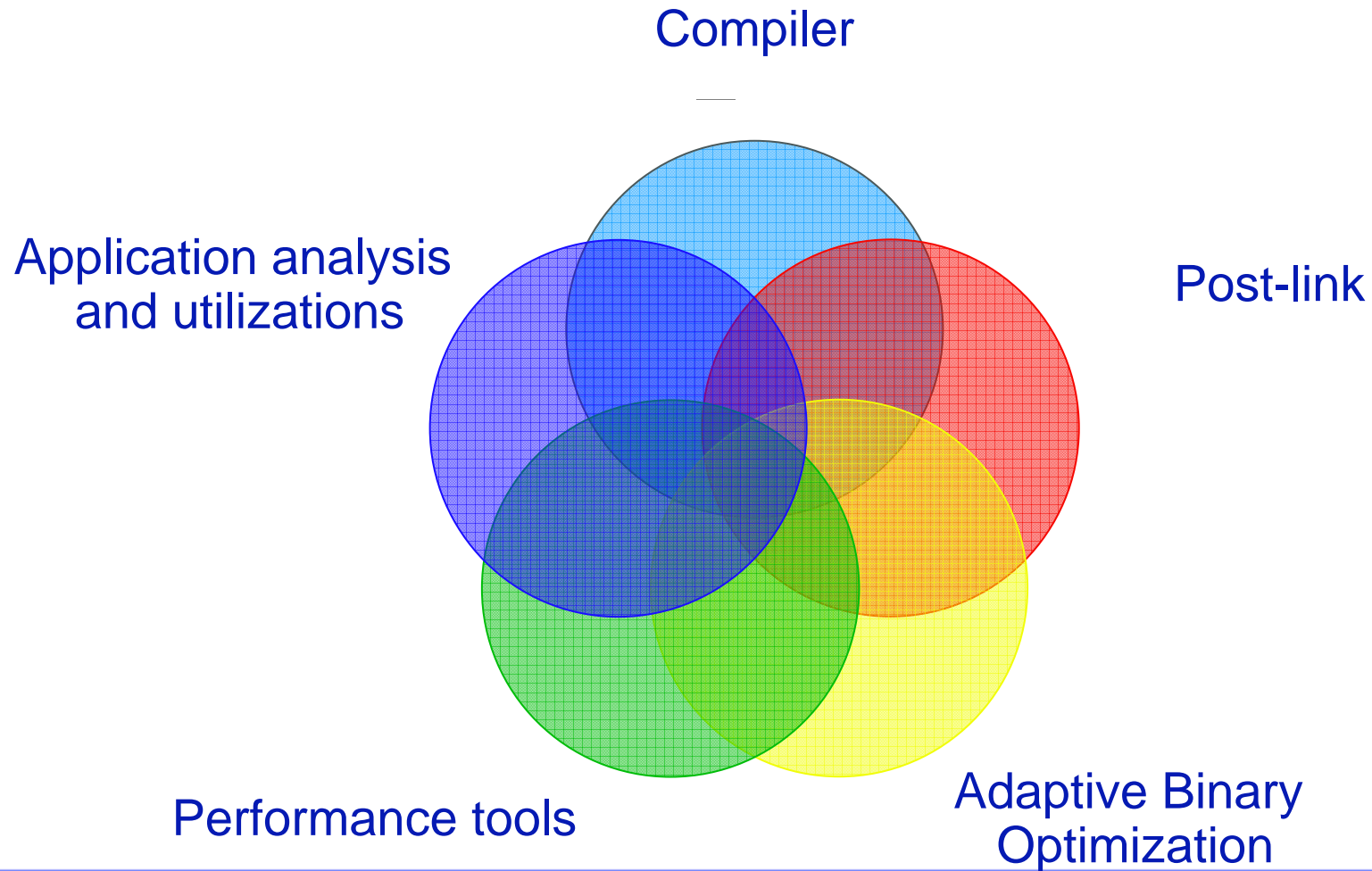


IBM Haifa interests items

Bilha Mendelson (bilha@il.ibm.com)



Performance issues:





GCC – open source compiler

Actively contributing, leading and maintaining optimizations:

- ◆ Automatic vectorization, to exploit data-level parallelism (DLP, SIMD)
 - ◆ Exploiting inner-most, outer, intra-iteration
- ◆ Data-layout transformations, to improve data spatial locality
 - ◆ Transposing multi-dimensional matrices
 - ◆ Struct field reordering and peeling
- ◆ Automatic parallelisation, to exploit thread-level parallelism
 - ◆ Based on OpenMP infrastructure and gomp library
 - ◆ Inner-most DOALL and reduction loops
- ◆ Modulo-scheduling
- ◆ Architecture-specific optimization
 - ◆ PowerPC970, Cell-SPU, PowerPC750CL and more

In coordination with other compiler teams within IBM and elsewhere



Cell/B.E. Streaming Framework

An architecture independent parallel programming environment based on streaming methods that frees the programmer from handling parallelism missions

◆ Main tasks

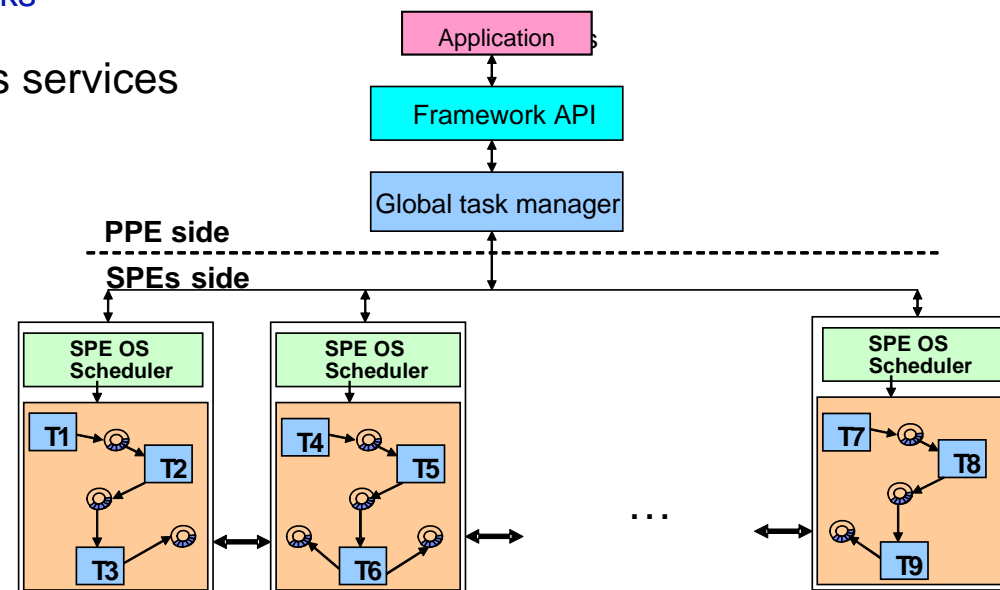
- ◆ Ease the programming environment of the Cell/B.E - **Designed for any parallel platform**
- ◆ Handle **data streaming for real-time and non-real-time** applications
- ◆ Handle **scheduling and load balancing**
- ◆ Enable **code reuse from various sources** in a consistent manner
- ◆ Provide **API for tasks activation and control** of the streaming process
- ◆ Enable future compilers to use the framework (ACOTES)



Cell/B.E. Streaming Framework - Main Characteristics

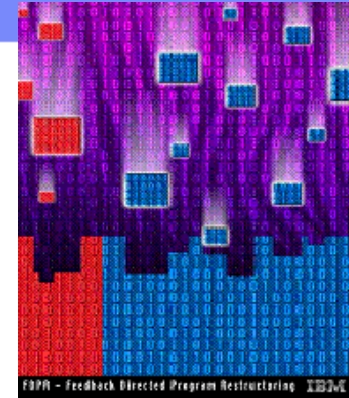
- ◆ An application level framework
 - ◆ An application creates ANY data dependency graphs instead of loops
 - ◆ can be done by a compiler
 - ◆ Basic building blocks: Tasks – local environment
 - ◆ All the parallelism process is handled by the framework
 - ◆ Streaming and control data are loosely coupled
- ◆ Manages a set of tasks that together compose an application
- ◆ Provides well defined means to stream data between the tasks that may run on different processors (mainly SPEs)
 - ◆ Streaming buffers and communication blocks
 - ◆ hidden from the programmer
- ◆ Implements real-time and non real-times services for running tasks on SPEs
 - ◆ Real-time scheduling
 - ◆ based on frames
 - ◆ Non-real-time scheduling
 - ◆ when data is ready
 - ◆ Data driven synchronization
 - ◆ when data is available

Not PPE centric

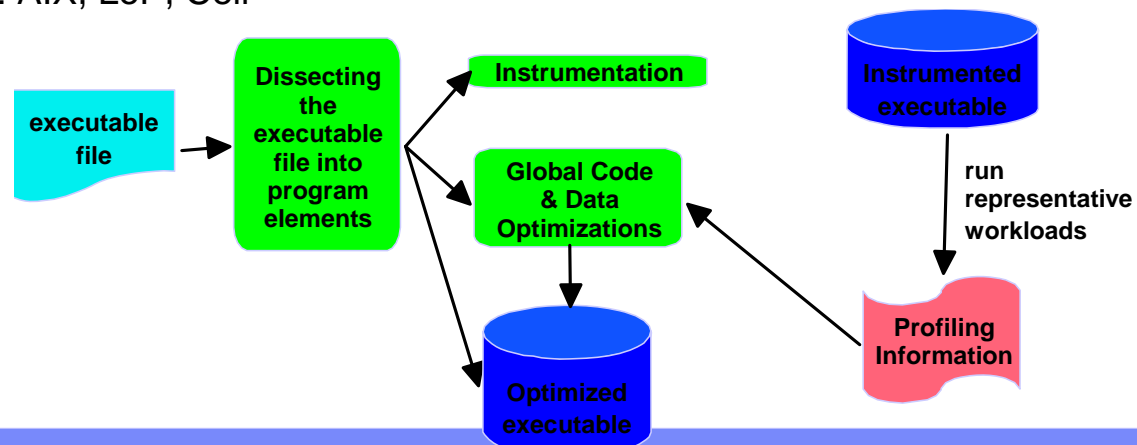




Post-link optimizer - FDPR-Pro



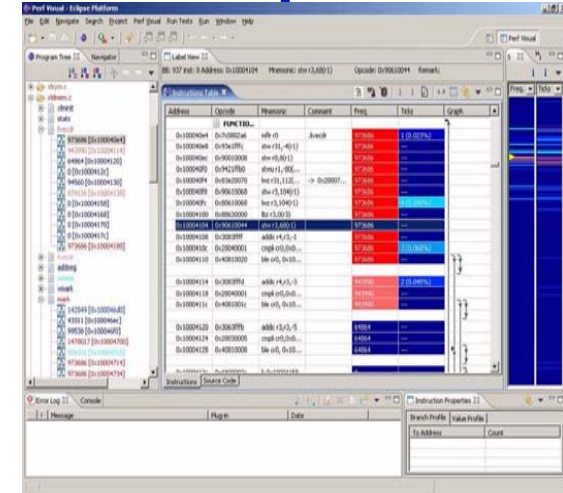
- ◆ Feedback-directed post-link optimization tool that operates directly on binary executables
 - ◆ Operates in three phases:
 - ◆ Phase 1: **Code instrumentation**
 - ◆ Basic block level
 - ◆ Phase 2: **Profile information gathering**
 - ◆ Selection of "right" input set (workload)
 - ◆ Accumulation over several input sets
 - ◆ Phase 3: **Global Code & Data Optimizations**
- ◆ Profile based; Whole program view
- ◆ The optimizations include
 - ◆ Code Reordering, Static Data Reordering, Branch Folding, Branch Prediction, Inlining, Constant and Copy Propagation, Dead Code Elimination, etc.
- ◆ Available on different platforms: AIX, LoP, Cell





Additional performance tools based on post-link optimizer

- ◆ Adaptive Binary Optimization system
- ◆ **Code Analyzer** – eclipse plugin GUI that can display performance analysis and bottlenecks
 - ◆ Releasing to alphaworks as part of Visual Performance Analyzer (VPA)
- ◆ **ESTO** – Expert System for Tuning Optimizations – Machine-Learning based tool
 - ◆ Helps us improve performance gain for customers by tuning optimization options of FDPR-Pro and various compilers
- ◆ **Bprober**– utility for supporting program instrumentation on binary executable files
 - ◆ Static-mode version that is run on given executables
 - ◆ Started working on dynamic-mode version that can instrument and replace code at run-time





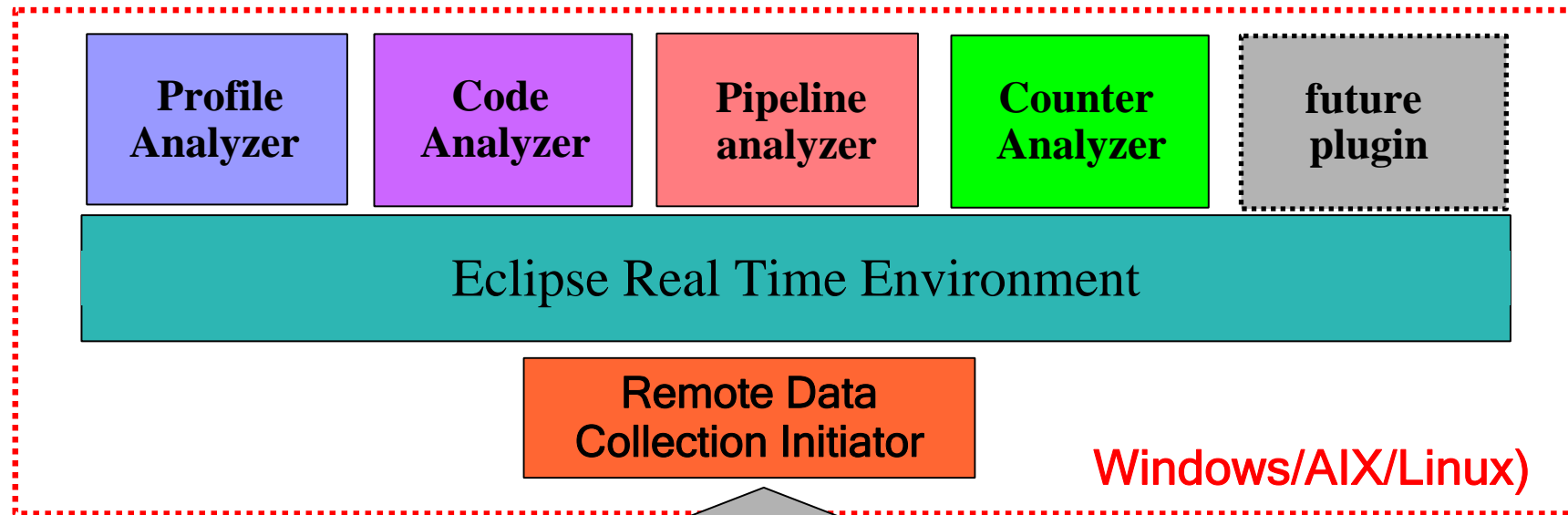
Adaptive Binary Optimization

- ◆ Adaptive performance optimization of long running applications
 - ◆ Addresses dynamic optimization
 - ◆ binary code of statically compiled applications
 - ◆ Does not rely on the application's source code or recompilation
 - ◆ Aims at adapting the application to:
 - ◆ significant changes in application behavior
 - ◆ run-time events
- ◆ Some of the challenges
 - ◆ Low overhead online event-monitoring
 - ◆ Efficient redirection of code execution to optimized code (using ptrace)
 - ◆ Online optimization validation and optimization adaptation
- ◆ Status
 - ◆ Work on change detection published:
 - ◆ "Detecting change in program behavior for adaptive optimization" by N. Peleg and B. Mendelson, PACT 2007
 - ◆ Initial prototype
 - ◆ Start developing infrastructure for dynamic binary code analysis and optimization
 - ◆ Implementing event-driven dynamic optimization

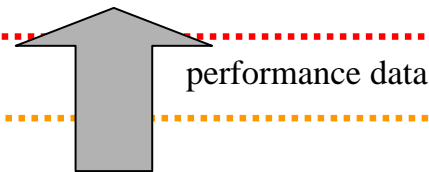


Visual Performance Analyzer

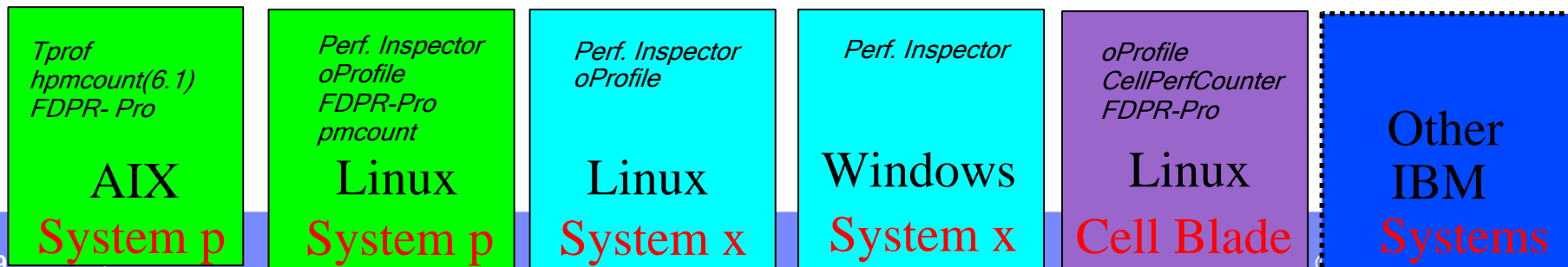
◆ Eclipse based tool set, currently including four plug-in applications working collaboratively



Windows/AIX/Linux)



Platform Specific Data Collectors





Performance Debugging Tools for multicores

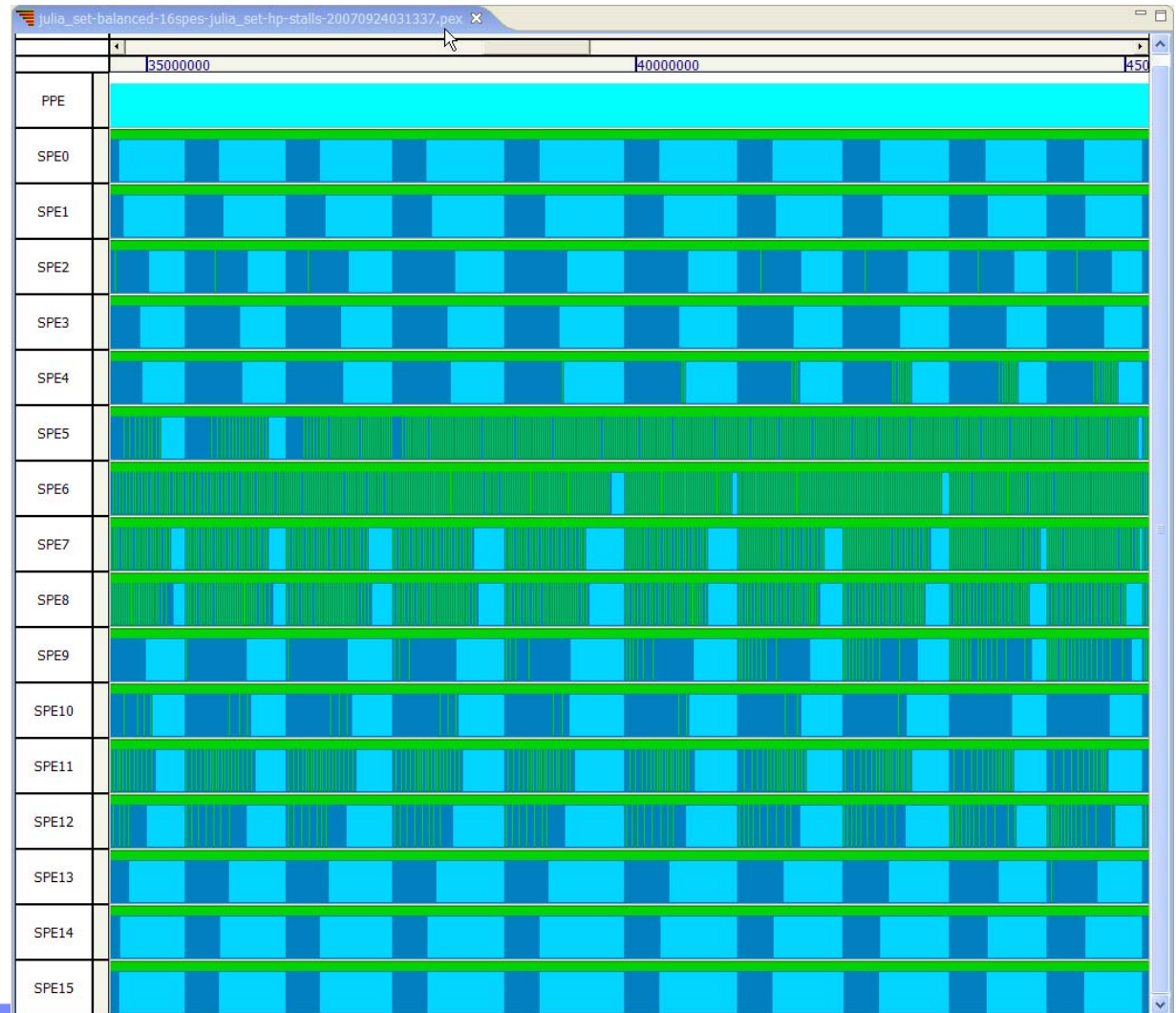
- ◆ Performance debugging using trace data
- ◆ Back-end:
 - ◆ Tracing of events on PPE and SPEs
 - ◆ User-guided selective event tracking
 - ◆ Minimal interference with the application
 - ◆ Very small amount of code added on the SPE
- ◆ Front-end:
 - ◆ Eclipse GUI integrated into the VPA
 - ◆ Time-based visualization of the Cell/B.E. events
 - ◆ Integrated textual/graphical browsing of the trace
- ◆ PDT for tracing the Cell platforms – as part of Cell SDK 3.0
- ◆ TA for analysis and visualizing multicore traces



Trace Analyzer for Cell: Julia Set

- ◆ Zoom-in: 10 iterations of 100
- ◆ Green: computation
- ◆ Dark blue: short stall
- ◆ Light blue: long stall

- ◆ Long stalls at phase synchronization indicate load balancing problem





Example: Apache process creation

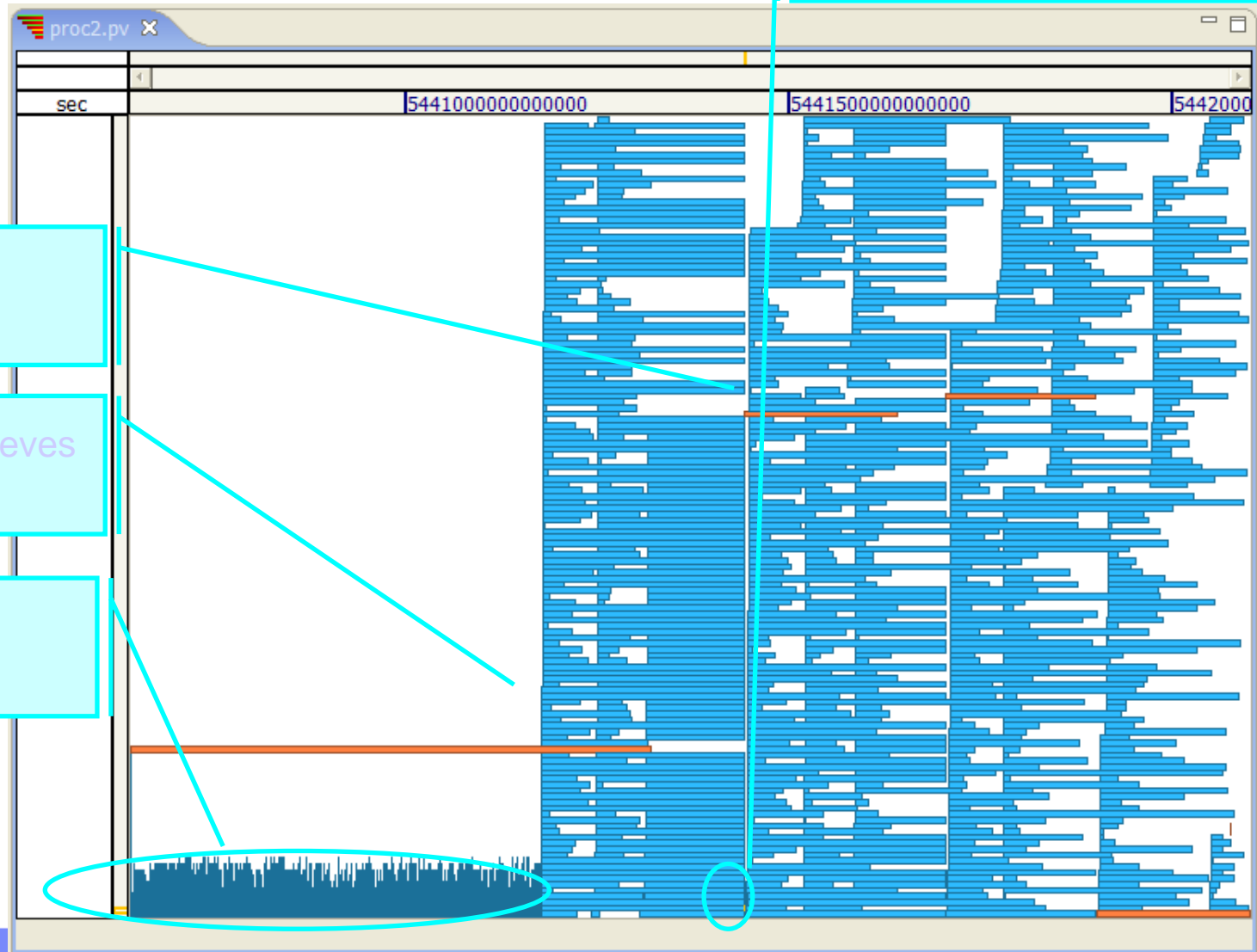
- ◆ Solaris on SunFire T2000
- ◆ Time is on the x axis
- ◆ Blue bars are

3. ab runs again

2. Finally Apache believes it's got work to do

1. 15 K short-living threads

4. Just a few short threads





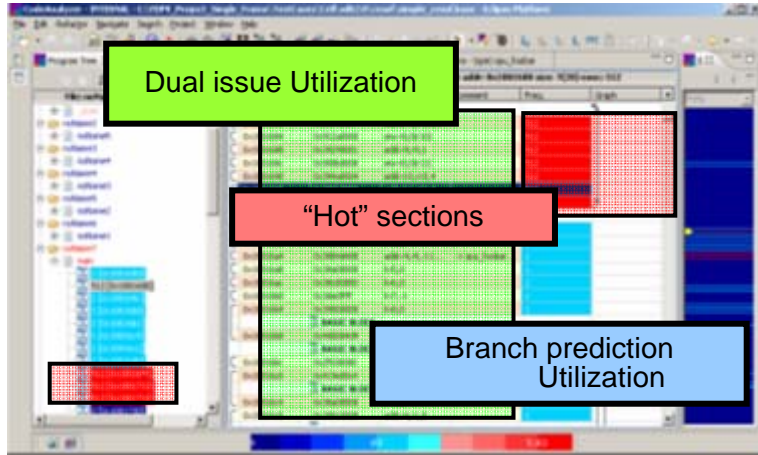
IBM

Thanks!

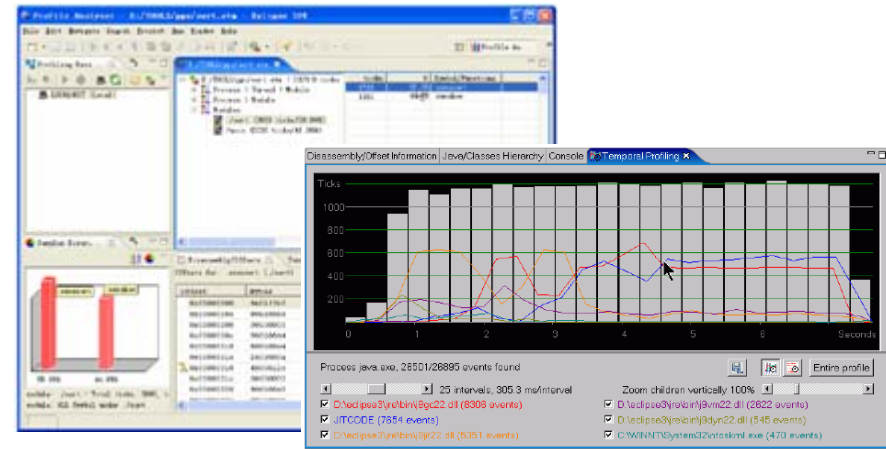


Virtual Performance Analyzer (VPA), Eclipse based

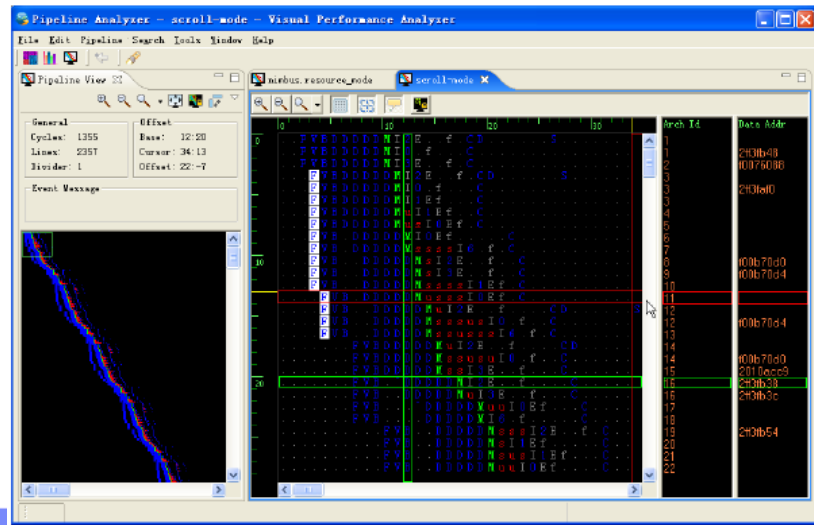
Code Analyzer



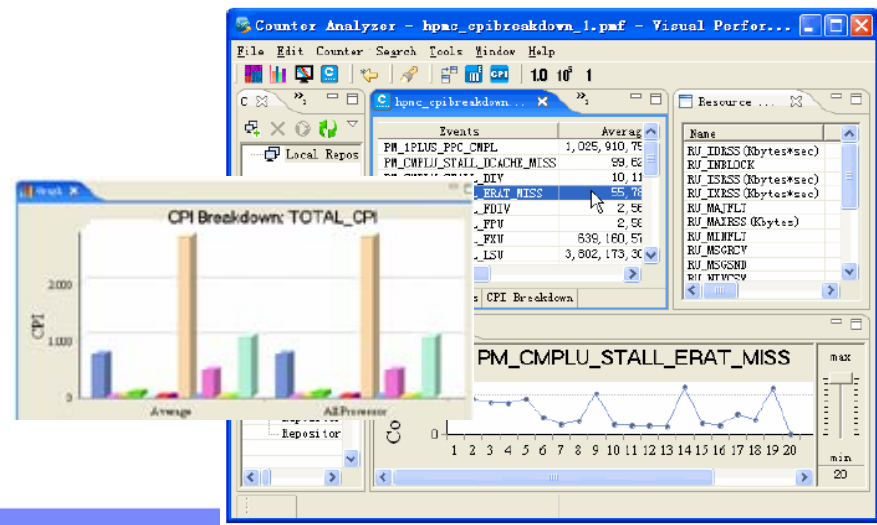
Profile Analyzer



Pipeline Analyzer



Counter Analyzer





Code Analyzer - sample views

Program tree of an executable file

Annotated Basic Block/Disassembly view

Address	Opcode	Mnemonic	Com...	Freq.	Graph
0x10004d08	0x7c0802a6	mflr r0	.mark	142849	
0x10004d0c	0x9421ffa0	stwu r1,-96(r1)		142849	
0x10004d10	0x90010068	stw r0,104(r1)		142849	
0x10004d14	0x90610078	stw r3,120(r1)		142849	
0x10004d1c	0x28030000	cmpli cr0,0,0,r3	.bf	142849	
0x10004d24	0x480001cc	b 0x10004ef0		43311	
0x10004d28	0x38600000	lir3,0		99538	
0x10004d2c	0x90610044	stw r3,68(r1)		99538	
0x10004d30	0x80610078	lwz r3,120(r1)		99538	
0x10004d34	0x90610040	stw r3,64(r1)		99538	
0x10004d38	0x80610040	lwz r3,64(r1)		1478017	
0x10004d3c	0x88630001	lbr r3,1(r3)		1478017	
0x10004d40	0x70630001	andl_ r3,r3,0x1		1478017	
0x10004d44	0x41820008	beq cr0, 0x1000...		1478017	
0x10004d48	0x480000cc	b 0x10004e14		504331	
0x10004d4c	0x80810040	lwz r4,64(r1)		973686	
0x10004d50	0x9081004c	stw r4,76(r1)		973686	
0x10004d54	0x88640001	lbr r3,1(r4)		973686	
0x10004d58	0x60630001	ori r3,r3,0x1		973686	
0x10004d5c	0x5463063e	rlwinm r3,r3,0x0...		973686	
0x10004d60	0x98640001	stb r3,1(r4)		973686	
0x10004d64	0x80610040	lwz r3,64(r1)		973686	
0x10004d68	0x4bffa19	bl 0x10004780	.ll...	973686	
0x10004d6c	0x2c030000	cmpli cr0,0,0,r3,0		973686	
0x10004d70	0x41820048	beq cr0, 0x1000...		973686	
0x10004d74	0x80810040	lwz r4,64(r1)		868467	

Annotated Source view

```

/* descend as far as we can */
while (TRUE) {

    /* check for this node being marked
    if (this->n_flags & MARK)
    break;

    /* mark it and its descendants */
    else {

        /* mark the node */
        this->n_flags |= MARK;

        /* follow the left sublist if there
        if (livecar(this)) {
            this->n_flags |= LEFT;
            tmp = prev;
            prev = this;
            this = car(prev);
            rplacd(prev,tmp);

        /* otherwise, follow the right subli
        else if (livecdr(this)) {
            this->n_flags &= ~LEFT;
            tmp = prev;
            prev = this;
            this = cdr(prev);
            rplacd(prev,tmp);
        }
        else
        break;
    }
}

/* backup to a point where we can contin
while (TRUE) {

```

Performance Comments

Description	File	Function	Address
Hot Function Call to: 0x10004d08	xldmem.c	.vmark	0x10004cb4
Hot Function Call to: 0x10004780	xldmem.c	.mark	0x10004d68
Hot Function Call to: 0x100046a0	xldmem.c	.mark	0x10004dbc
Hot Function Call to: 0x100046a0	xldmem.c	.mark	0x10004e38
Hot Function Call to: 0x10004d08	xldmem.c	.gc	0x10004f80
Hot Function Call to: 0x10005160	xldmem.c	.consd	0x10005860
Hot Function Call to: 0x10005160	xldmem.c	.consa	0x100058c0
Hot Function Call to: 0x10005160	xldmem.c	.cons	0x10005924

Detailed instruction information

Slot 0	Slot 1	Slot 2	Slot 3	Slot 4
CRLogical_odd	CRLogical_even			
mtspr_0_odd/m...				
FXU_0_odd	FXU_1_odd	FXU_1_even	FXU_0_even	
LSU_0_odd	LSU_1_odd	LSU_1_even	LSU_0_even	
FPU_0_odd	FPU_1_odd	FPU_1_even	FPU_0_even	Branch_any