

PAPI and Hardware Performance Analysis Tools

Lawrence Livermore National Laboratory Visit, April, 2003

Philip Mucci, Research Consultant

Innovative Computing Laboratory/UTK

Performance Evaluation Research Center/LBNL

mucci@cs.utk.edu

<http://icl.cs.utk.edu/projects/papi>





PAPI provides two standardized APIs to access the underlying performance counter hardware

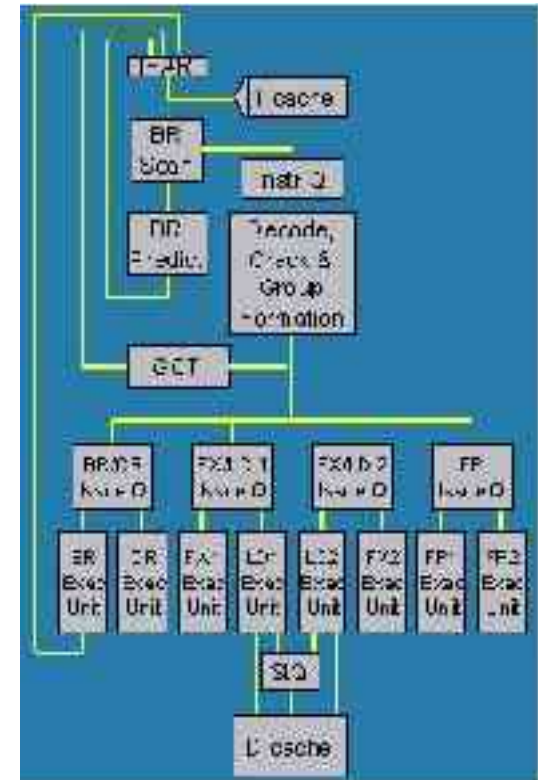
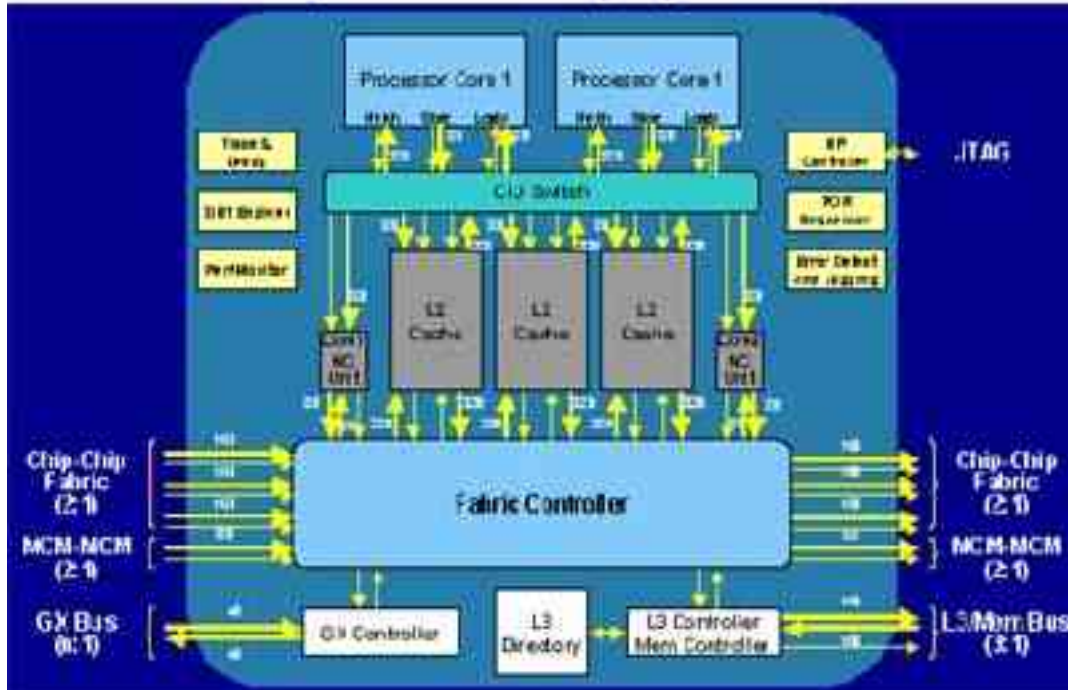
- A low level interface designed for tool developers and expert users.
- The high level interface is for application engineers.

- PAPI
- Performance Monitoring Hardware
- Performance Analysis Tools
 - Dynaprof
 - Other tools
- Trends

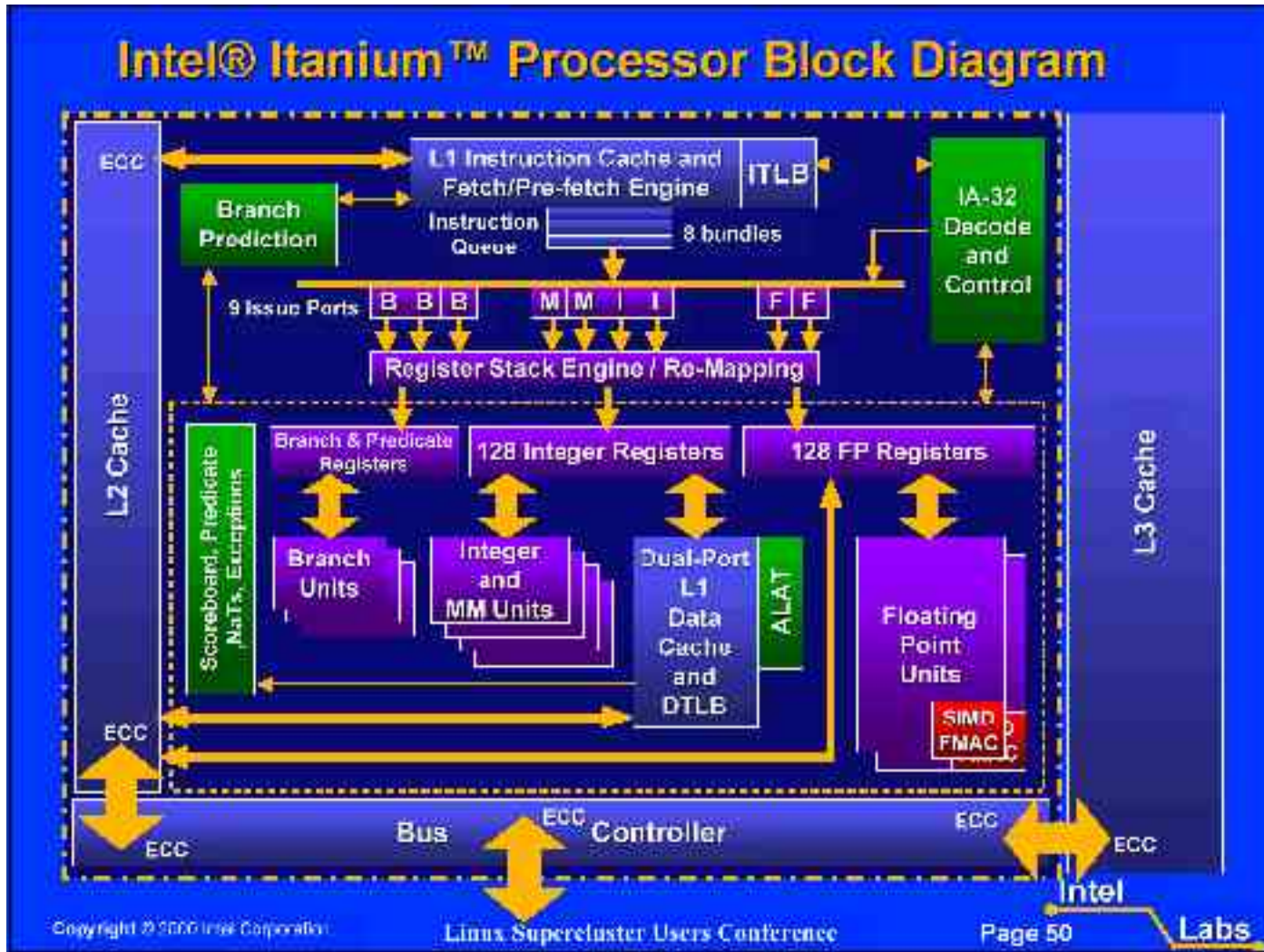
- To understand why the application performs as it does.
 - Optimize the application's performance.
 - Evaluate the algorithms efficiency.
 - Generate an application signature.
 - Develop a performance model.
- Data is NOT PORTABLE, but the interface is...

- Small number of registers dedicated for performance monitoring functions.
 - AMD Athlon, 4 counters
 - Pentium \leq III, 2 counters
 - Pentium IV, 18 counters
 - IA64, 4 counters
 - Alpha 21x64, 2 counters
 - Power 3, 8 counters
 - Power 4, 8 counters
 - UltraSparc II, 2 counters
 - MIPS R14K, 2 counters

Power 4 Diagram

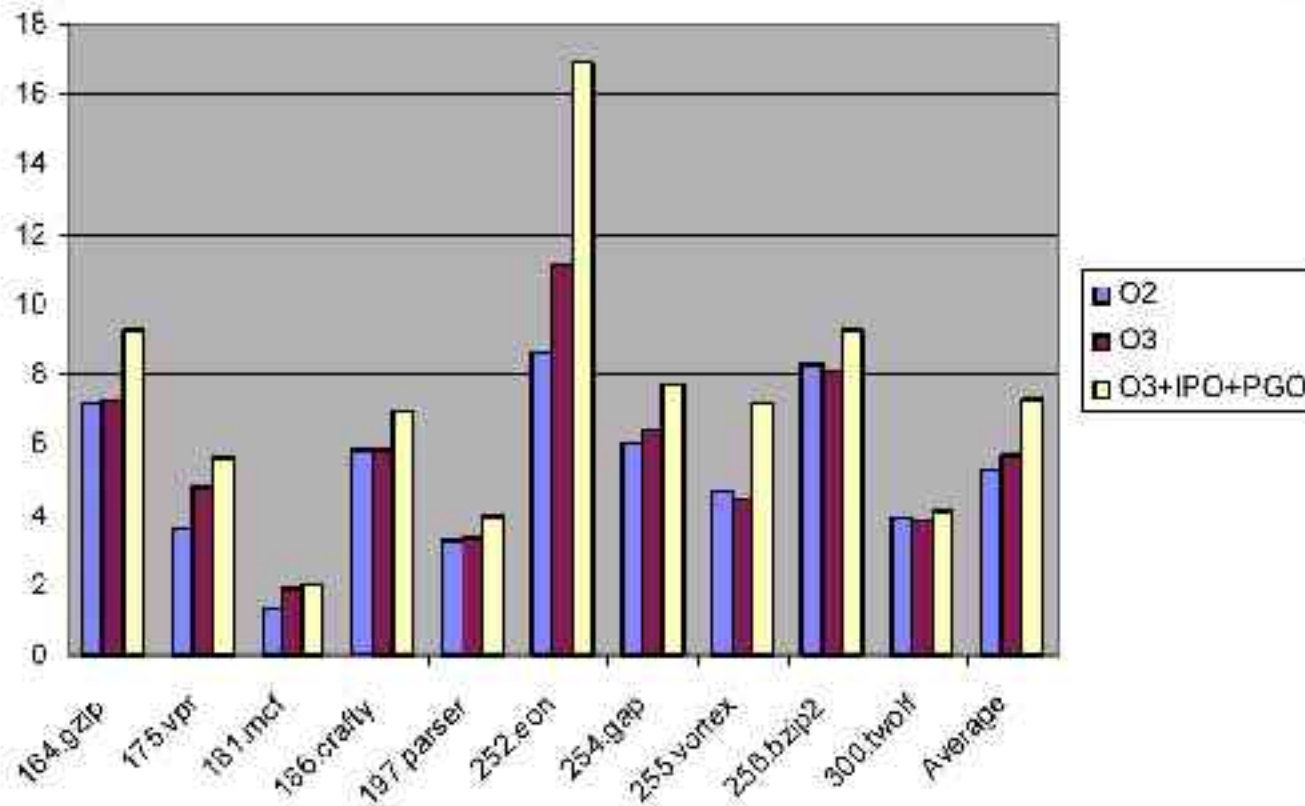


IA64 Block Diagram

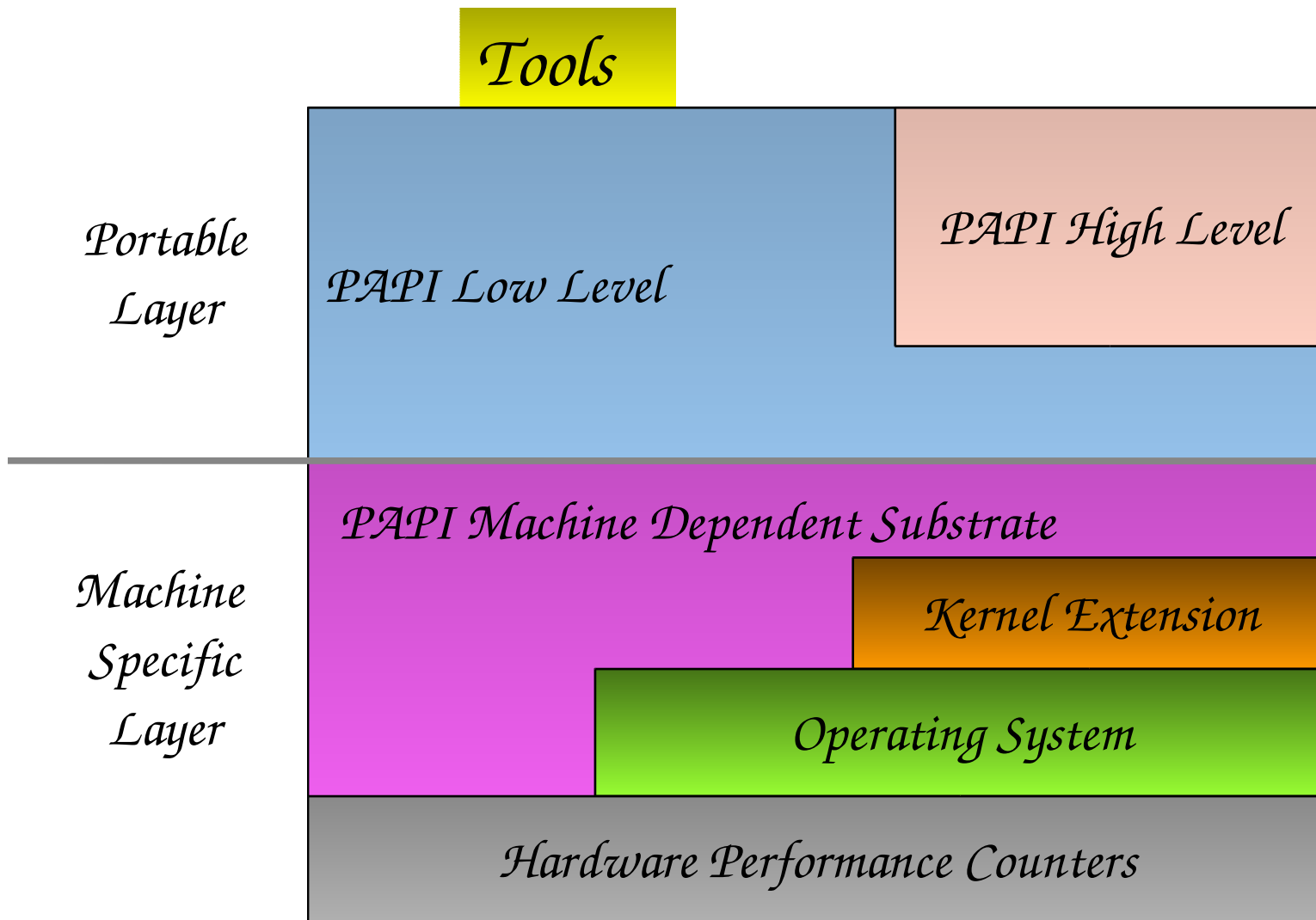


Importance of Optimization

Example: Speed up from Static Compiler Optimization on Itanium-1 in 2002 (SpecInt)



PAPI Implementation



- Proposed standard set of event names deemed most relevant for application performance tuning
- No standardization of the actual definition
- Mapped to native events on a given platform

- PAPI supports approximately 100 preset events.
 - Preset events are mappings from symbolic names to machine specific definitions for a particular hardware event.
 - Example: **PAPI_TOT_CYC**
 - PAPI also supports presets that may be derived from the underlying hardware metrics
 - Example: **PAPI_L1_DCM**

Sample Preset Listing

```
> tests/avail
```

```
Test case 8: Available events and hardware information.
```

```
-----
Vendor string and code   : GenuineIntel (-1)
Model string and code   : Celeron (Mendocino) (6)
CPU revision            : 10.000000
CPU Megahertz           : 366.504944
-----
```

Name	Code	Avail	Deriv	Description (Note)
PAPI_L1_DCM	0x80000000	Yes	No	Level 1 data cache misses
PAPI_L1_ICM misses	0x80000001	Yes	No	Level 1 instruction cache
PAPI_L2_DCM	0x80000002	No	No	Level 2 data cache misses
PAPI_L2_ICM misses	0x80000003	No	No	Level 2 instruction cache
PAPI_L3_DCM	0x80000004	No	No	Level 3 data cache misses
PAPI_L3_ICM misses	0x80000005	No	No	Level 3 instruction cache
PAPI_L1_TCM	0x80000006	Yes	Yes	Level 1 cache misses
PAPI_L2_TCM	0x80000007	Yes	No	Level 2 cache misses
PAPI_L3_TCM	0x80000008	No	No	Level 3 cache misses
PAPI_CA_SNP	0x80000009	No	No	Requests for a snoop
PAPI_CA_SHR	0x8000000a	No	No	Requests for shared
PAPI_CA_CLN	0x8000000b	No	No	Requests for clean c
PAPI_CA_INV	0x8000000c	No	No	Requests for cache l

```
.
.
```

http://icl.cs.utk.edu/projects/papi/files/html_man/papi_presets.html



- PAPI supports native events:
 - An event countable by the CPU can be counted even if there is no matching preset PAPI event.
 - The developer uses the same API as when setting up a preset event, but a CPU-specific bit pattern is used instead of the PAPI event definition

- Meant for application programmers wanting coarse-grained measurements
- As easy to use as IRIX calls
- Requires no setup code
- Restrictions:
 - Allows only PAPI presets
 - Not thread safe
 - Only aggregate counters

- `PAPI_num_counters()`
 - _ Returns the number of available counters
- `PAPI_start_counters(int *cntrs, int alen)`
 - _ Start counters
- `PAPI_stop_counters(long_long *vals, int alen)`
 - _ Stop counters and put counter values in array
- `PAPI_accum_counters(long_long *vals, int alen)`
 - _ Accumulate counters into array and reset
- `PAPI_read_counters(long_long *vals, int alen)`
 - _ Copy counter values into array and reset counters
- `PAPI_flops(float *rtime, float *ptime,
 long_long *flpins, float *mflops)`
 - _ Wallclock time, process time, FP ins since start,
 - _ Mflop/s since last call

- Increased efficiency and functionality over the high level PAPI interface
- Approximately 60 functions
(http://icl.cs.utk.edu/projects/papi/files/html_man/papi.html#4)
- Thread-safe (SMP, OpenMP, Pthreads)
- Supports both presets and native events

- API Calls for:
 - Counter multiplexing
 - Callbacks on user defined overflow value
 - SVR4 compatible profiling
 - Processor information
 - Address space information
 - Static and dynamic memory information
 - Accurate and low latency timing functions
 - Hardware event inquiry functions
 - Eventset management functions
 - Simple locking operations

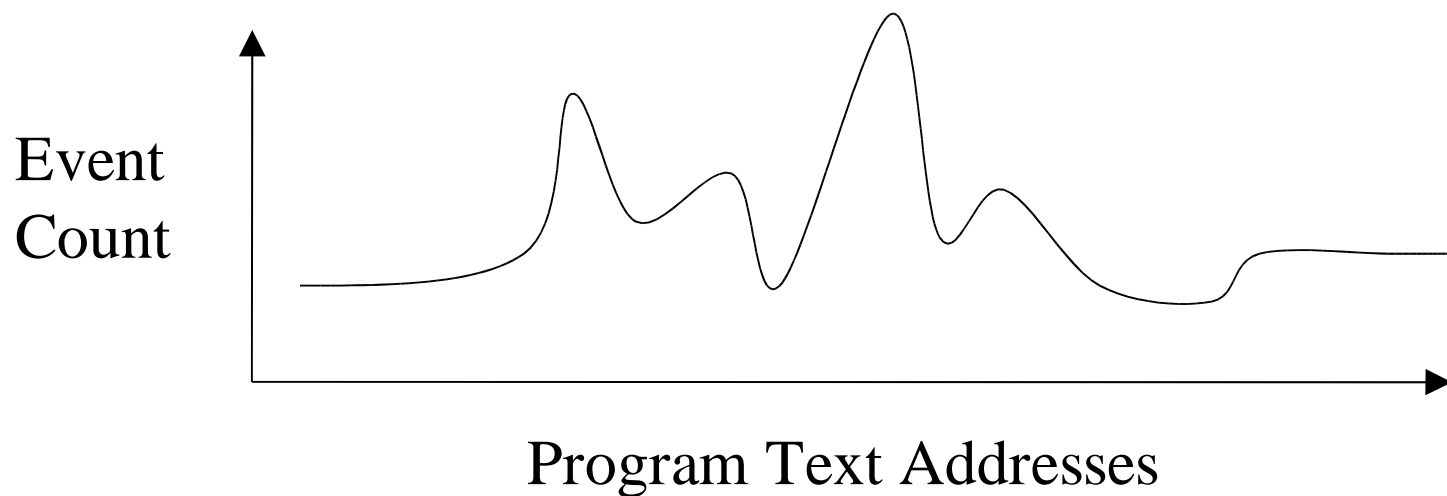
- Multiplexing allows simultaneous use of more counters than are supported by the hardware.
 - This is accomplished through timesharing the counter hardware and extrapolating the results.
- Users can enable multiplexing with one API call and then use PAPI normally.
 - Implementation was based on MPX done by.....
John May!

Interrupts on Counter Overflow

- PAPI provides the ability to call user-defined handlers when a specified event exceeds a specified threshold.
- For systems that do not support counter overflow at the hardware level, PAPI emulates this in software at the user level.
 - Code must run a reasonable length of time.

- On overflow of hardware counter, dispatch a signal/interrupt.
- Get the address at which the code was interrupted.
- Store counts of interrupts for *each* address.
- Vendor/GNU **prof** and **gprof** (**-pg** and **-p** compiler options) use interval timers.

Results of Statistical Profiling



- The result: A probabilistic distribution of where the code spent its time and why.

- <http://icl.cs.utk.edu/projects/papi/>
 - Software and documentation
 - Reference materials
 - Papers and presentations
 - Third-party tools
 - Mailing lists

- Additional Platforms
 - IBM PPC604, 604e, Power 3
 - Intel x86
 - Sun UltraSparc I/II/III
 - SGI MIPS R10K/R12K/R14K
 - Compaq Alpha 21164/21264 with DADD/DCPI
 - Itanium
 - Itanium 2
 - Power 4
 - AIX 5, Power 3, 604e
- Enhancements
 - Static/dynamic memory info
 - IA64 hardware profiling
 - Misc bug fixes
- Sample Tools
 - Perfometer
 - Trapper
 - Dynaprof

- Using lessons learned from years earlier
 - Substrate code: 90% used only 10% of the time
 - In practice, it was never used
- Redesign for:
 - Robustness
 - Feature set
 - Simplicity
 - Portability to new platforms

- Multiway multiplexing
 - Use all available counter registers instead of one per time slice. (Just 1 additional register means 2x increase in accuracy)
- Superb performance
 - Pentium 4, a `PAPI_read()` costs 230 cycles.
 - Register access alone costs 100 cycles.
- Programmable events
 - Event Thresholding
 - Instruction matching
 - Per event counting domains

- Third-party interface
 - Allows PAPI to control counters in other threads of execution
- Internal timer/signal/thread abstractions
- Additional internal layered API to support robust extensions
- Advanced profiling functions for Event Address Sampling. (branch, cache, etc...)

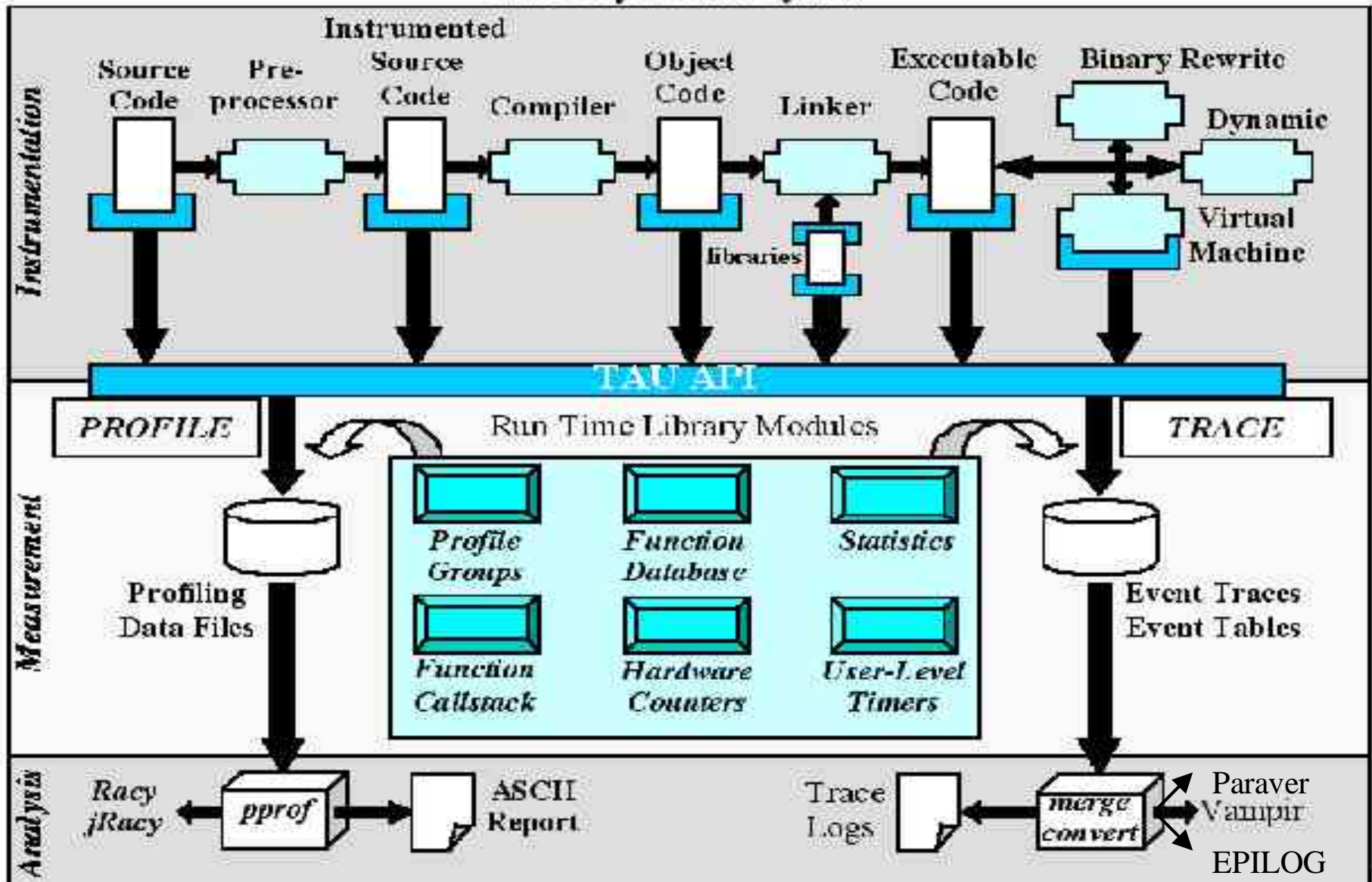
- System-wide counting
- High level API made thread safe
- New language bindings
 - Java
 - Lisp
 - Matlab
- New tools to be included: papirun and papiprof from Rice.

- Release expected May, 2003
- Additional platforms
 - Cray X-1
 - AMD Opteron/K8
 - Nec SX-6?
 - Blue Gene (BG/L)?

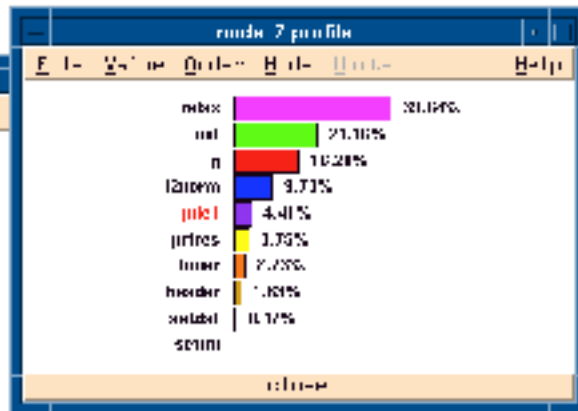
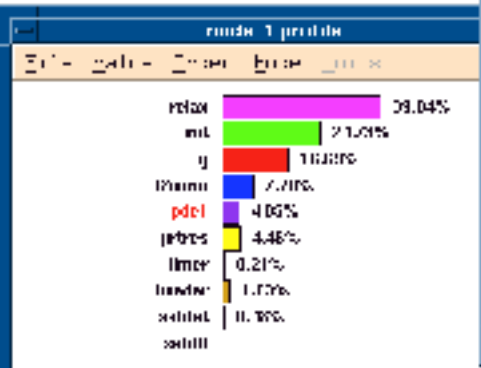
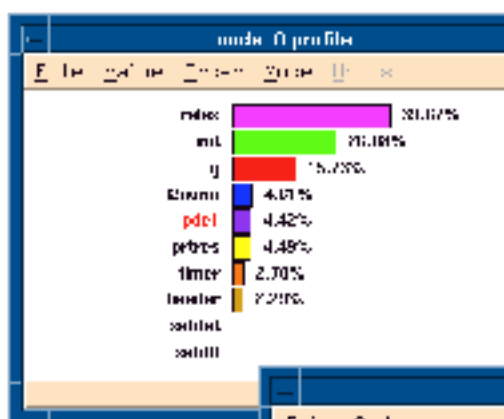
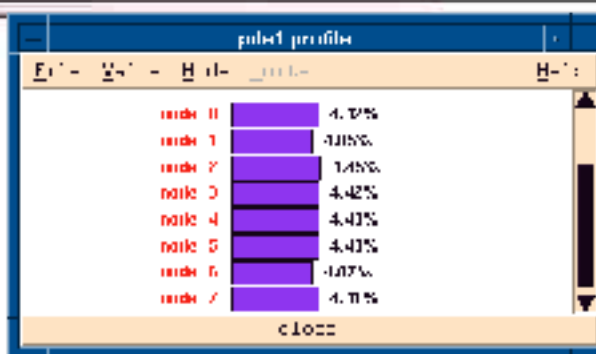
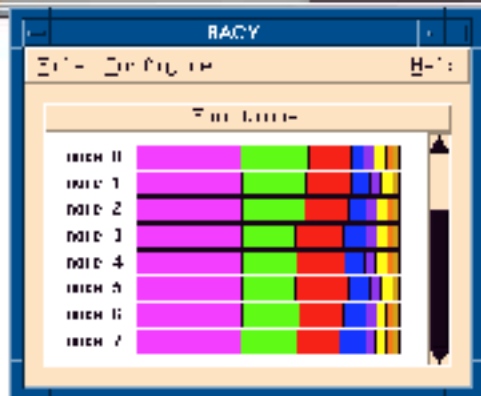
- From Barton Miller's Group
- DynInst based dynamic discovery of bottlenecks
- Different visualization plugins
- Supports all forms of parallelism
- New version will do discovery based on hardware metrics
 - Memory stall time

- From Allen Maloney's Group at U. Oregon
- Source or DynInst based
- Different visualization plugins
- Supports all forms of parallelism
- Integration with Vampir

TAU Performance System Architecture



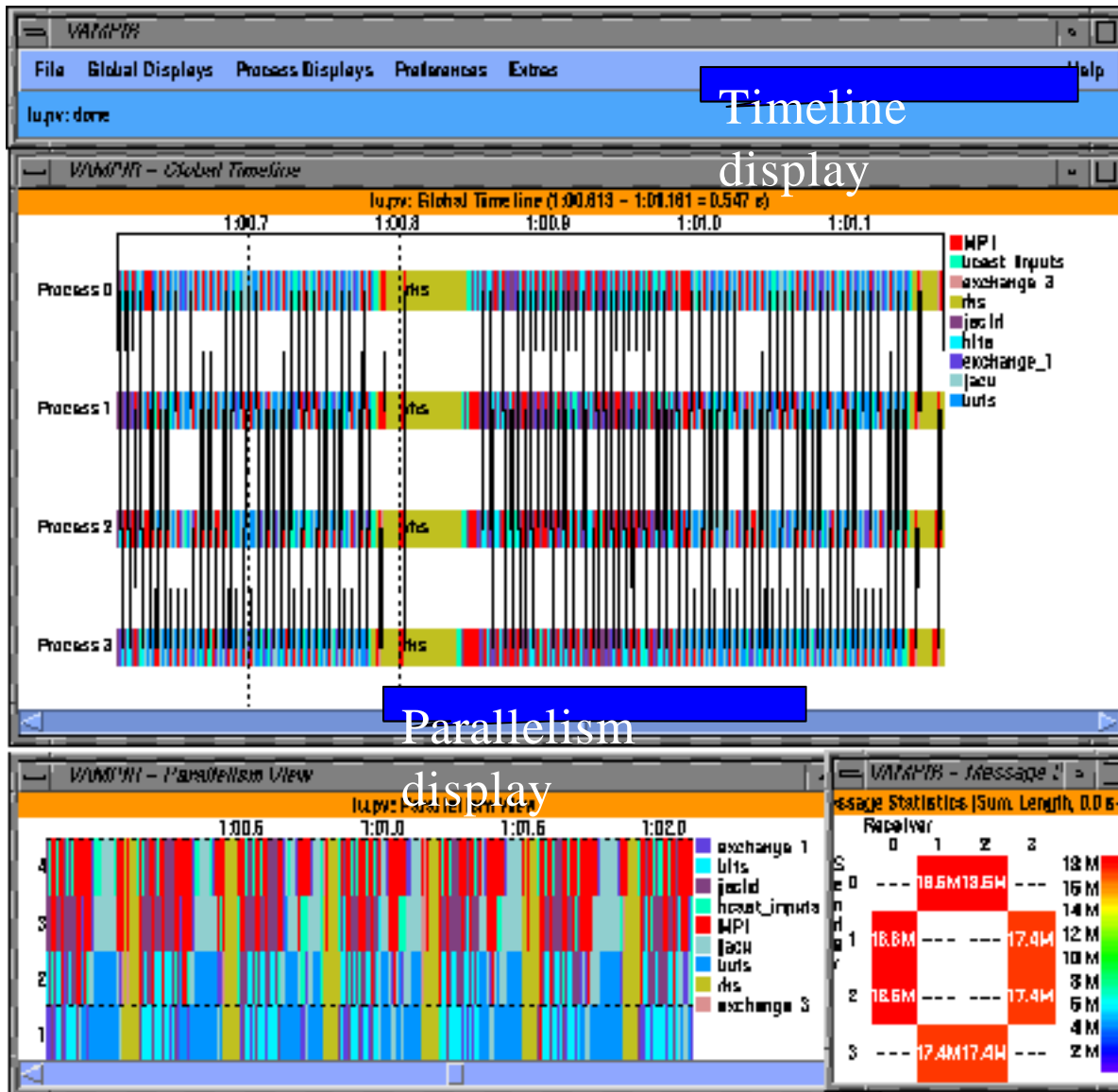
TAU Screenshot



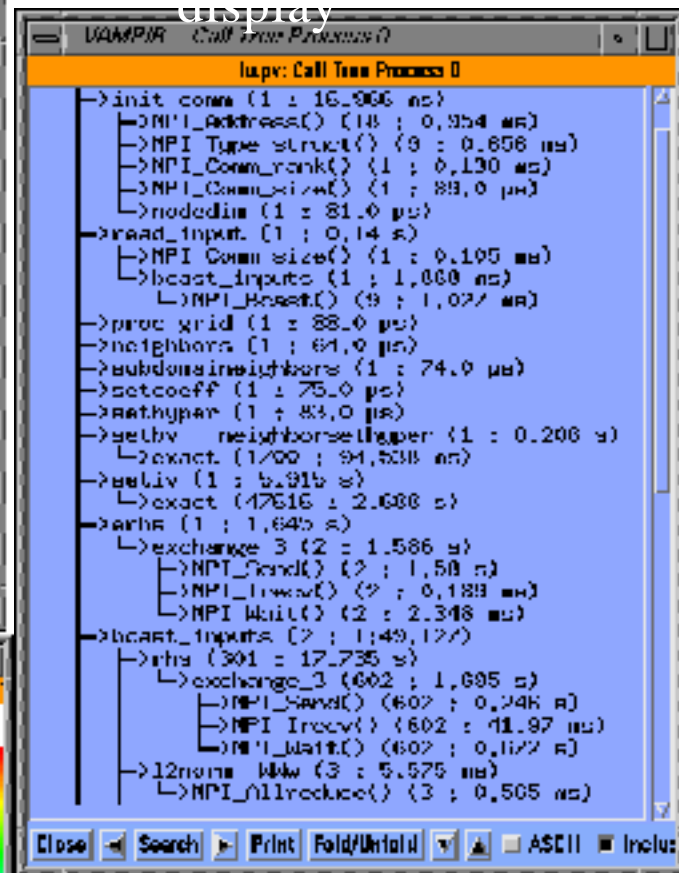

Run 8 profile table showing detailed function call statistics:

Call	Time	Count	Call Rate	Call Rate / All Rate	Function
24-1-70	2.4-1	1,004	1	24-1-70	relax
24-1-70	2.370	2,370	1	24-1-70	init
10-1-70	10.790	1,065	20,110	10-1-70	g
10-1-70	7.00	790	2	767077	l2norm
10-1-70	2.00	9,110	1	6130727	mk1
10-1-70	2.00	277	2	227639	prfres
10-1-70	14.00	147	2	24800	timer
10-1-70	9.00	111	1	141117	header
10-1-70	1.00	45	1	17047	sechit
10-1-70	0.00	0	1	17	sechit0

Vampir (NAS Parallel Benchmark – LU)



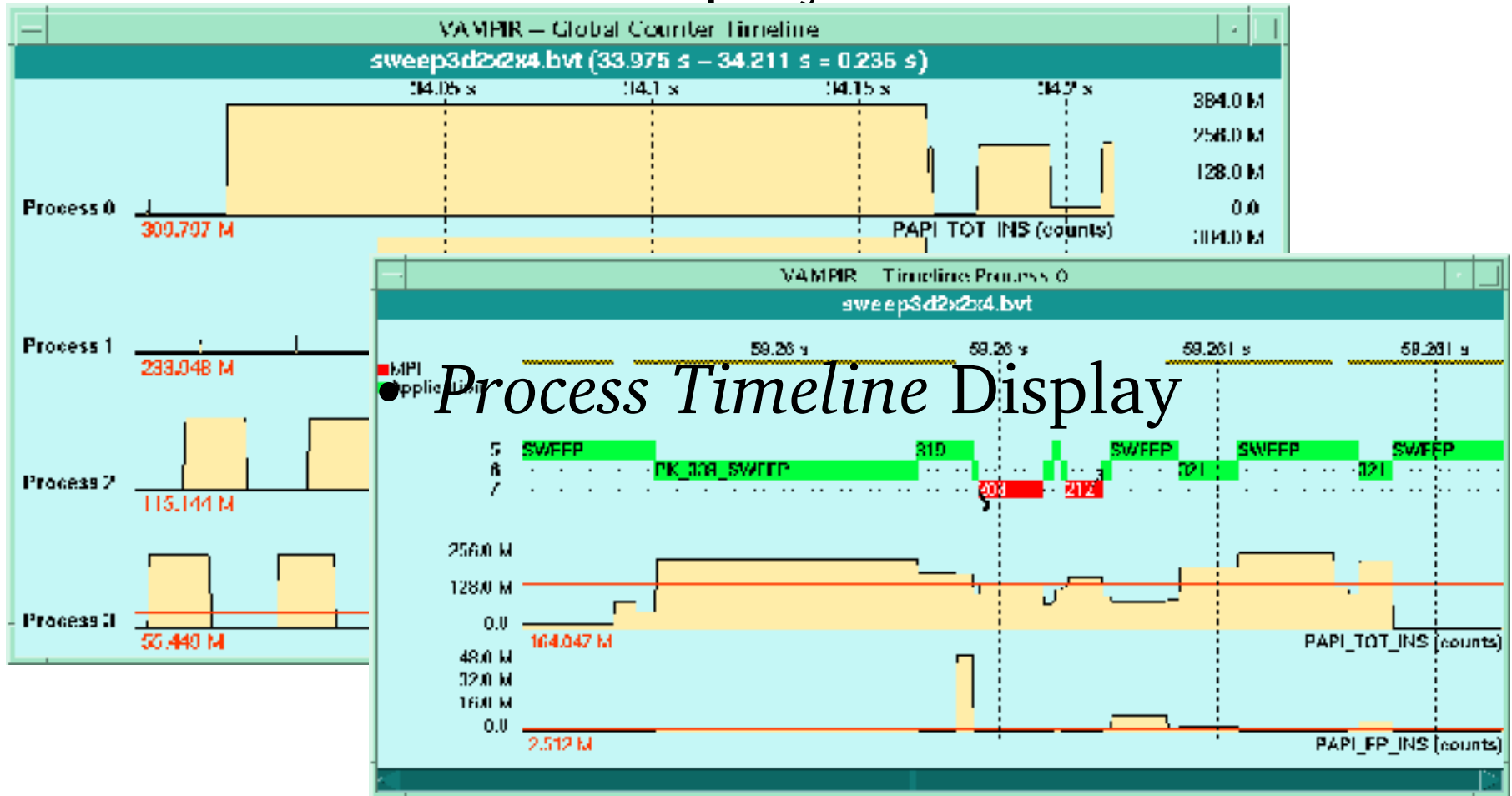
Callgraph display



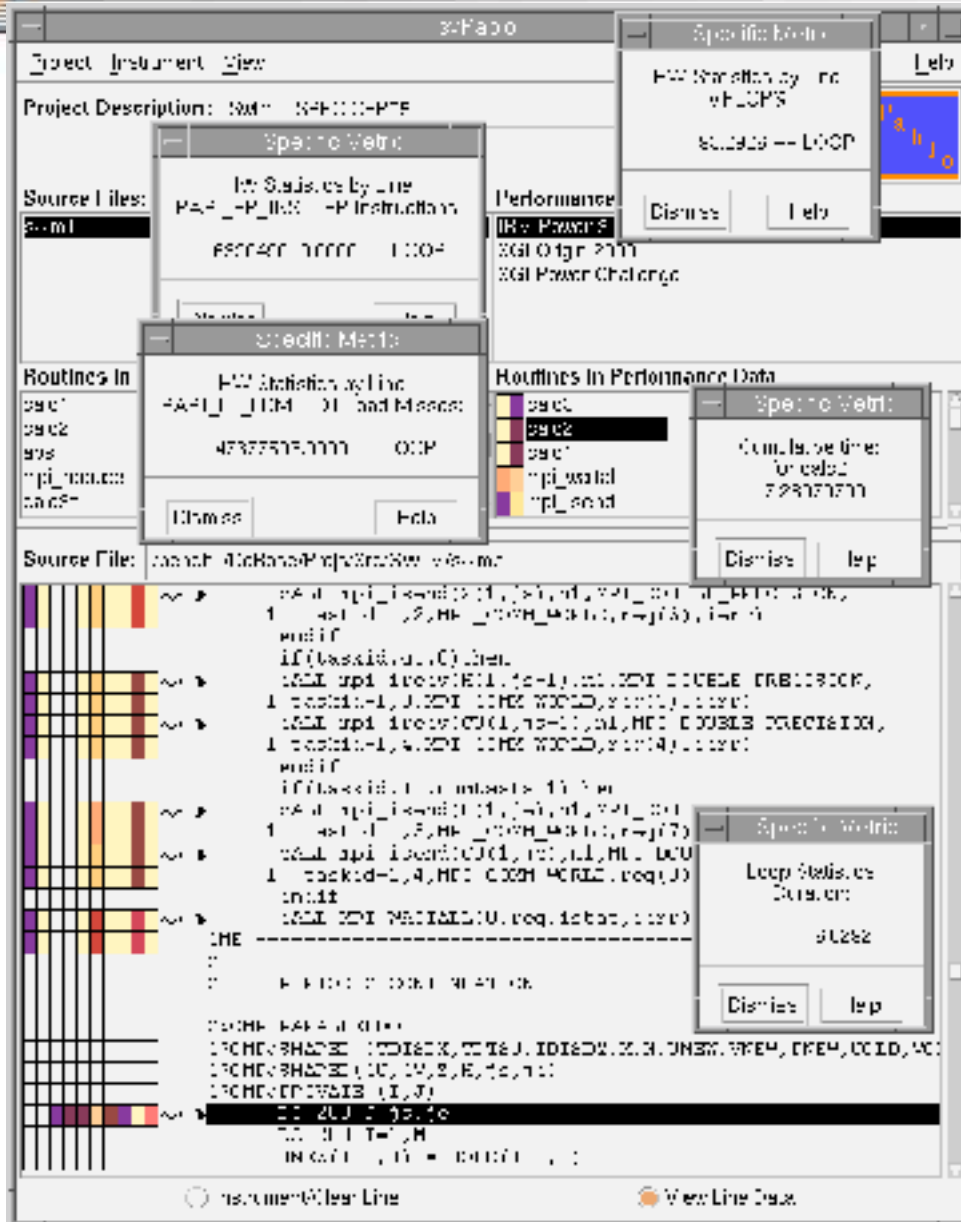
Communications

Vampir v3.x: HPM Counter

- *Counter Timeline Display*



SvPablo from UIUC



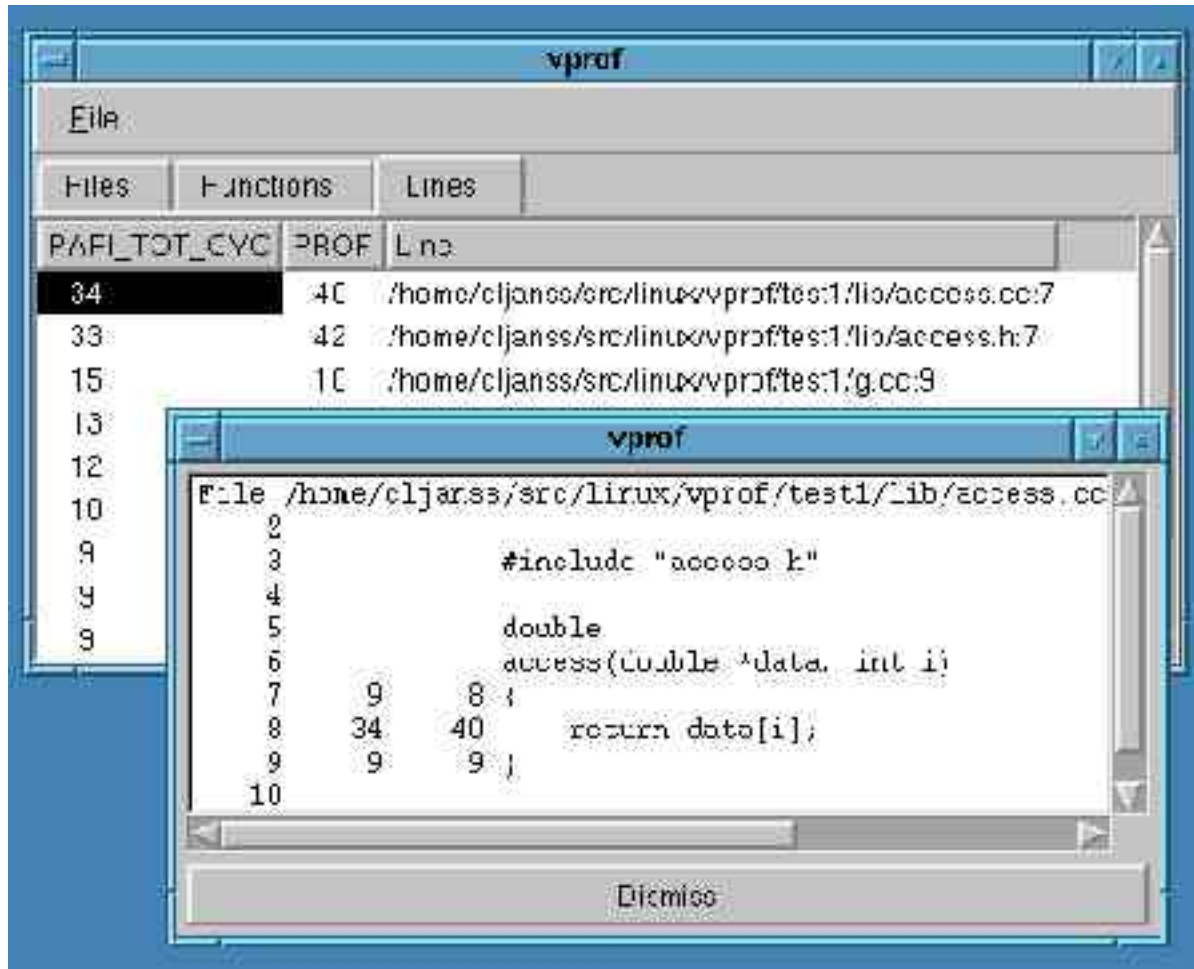
The screenshot displays the SvPablo application window. At the top, it shows the 'Project Description' and 'Source Files' (including 's-m1'). The main area is divided into several sections:

- Performance:** A table showing metrics like 'IR: Power9', 'RGI Ogr 200', and 'RGI Power Challenge'.
- Routines in Performance Data:** A list of routines such as 'pbc1', 'pbc2', 'pbc3', 'mpi_wait', and 'mpi_send'.
- Source File:** A code editor showing F90 code with a color-coded performance overlay on the left side of the lines.
- Spinfo Metric:** A floating window showing 'IR Statistics by Line' for a specific line, including 'Pbc1_Pbc1' and 'Pbc1_Instructions'.
- Spinfo Metric:** Another floating window showing 'Compile time' for a specific routine, with a value of '72837570'.
- Spinfo Metric:** A third floating window showing 'Loop Statistics Summary' for a specific loop, with a value of '90282'.

At the bottom, there are radio buttons for 'Instrument Clear Line' and 'View Line Data'.

- Source based instrumentation of loops and function calls
- Supports serial and MPI jobs
- Freely available
- Rough F90 parser

Vprof from Sandia National Laboratory



The screenshot shows the vprof application interface. The main window displays a summary table with columns for PAFI_TOT_CYC, PROF, and Line. A secondary window is open, showing the source code for the function 'access' in 'access.c'.

PAFI_TOT_CYC	PROF	Line
34	40	/home/cljanss/src/linux/vprof/test1/lib/access.c:7
33	42	/home/cljanss/src/linux/vprof/test1/lib/access.h:7
15	10	/home/cljanss/src/linux/vprof/test1/g.c:9

```

File /home/cljanss/src/linux/vprof/test1/lib/access.c
2
3      #include "access.h"
4
5      double
6      access(double *data, int i)
7      {
8          return data[i];
9      }
10
  
```

- Based on statistical sampling of the hardware counters
- Must instrument the source
- Ported to other architectures for generalized use
- Serial codes, parallel with modification
- Freely available

- Uses PC-sampling on event thresholds
- Combines numerous modules
 - Collection: papirun, equivalent to “ssrun”
 - Binary loop/CFG recovery: bloop
 - Formatting: papiprof
 - Data display and exploration: hpcview
 - Call stack profiles: csprof
- Data is aggregated into an XML database
- HPCView is a Java applet that generates dynamic HTML

HPCView Screenshot

The screenshot displays the HPCView interface with the following components:

- Source Files:** A tree view on the left showing the file structure. A red label "file pane" is overlaid on this area.
- Source Code:** The main window displays Fortran code for a program named "sweep3dsingle". A red label "source pane" is overlaid on the code.
- Scope Tree:** A tree view at the bottom showing the execution scope hierarchy. A red label "parent scope" points to the top-level scope, "current scope" points to the active scope, and "child scopes" points to the sub-scopes. A red label "flatten/unflatten" is also present near the tree.
- Table:** A table below the scope tree showing process statistics. The table has columns for Location, Program, and various performance metrics (DOUTER, DOUT, DOUT, DOUT, DOUT).

Location	Program	DOUTER	DOUT	DOUT	DOUT	DOUT
1.62e+10	100	5.63e+05	100	1.36e+07	100	1.45e+06

- Tool that allows the user to write functions that get executed at:
 - Process Creation/Deletion
 - Thread Creation/Deletion
- Actually, any function can be “preempted”.
- The object code of the application isn’t modified.
- Works by “preloading” special shared libraries and overloading function calls in cooperation with the run-time linker.

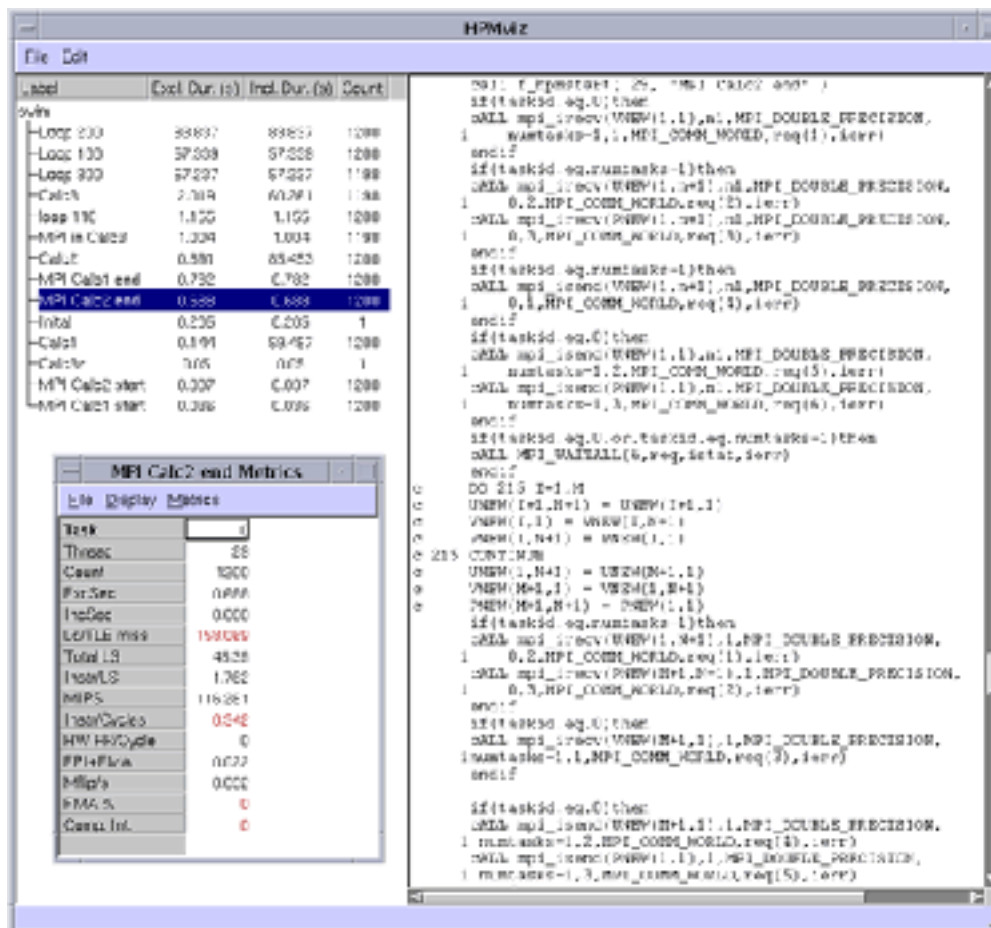
- Gives aggregate counts and derived metrics for unmodified source code.
- 2 phase
 - psrun: Collection, output to XML
 - psprocess: Formatting, output via XML/XSL parser
- Based on
 - PerfSuite from NCSA
- From NCSA (Rick Kufirin) for use on the Itanium 2 TeraGrid IA64 installations.

PerfSuite Hardware Performance Report

Derived Metrics	
Estimated floating point instructions	0.00
Estimated floating point instructions per cycle	0.115
Estimated percentage of all grid-based instructions	0.2195
Estimated average branch ratio	0.019
Estimated cache & branch mispredicted instructions	1.85
Cache references per instruction	0.029
Cache of floating point instructions per cycle (cache way size)	20.118
Cache of integer instructions per cycle	0.013
Cache of cache miss ratio	0.289
Cache of branch instructions	0.017
Cache of branch misratio	0.082
Cache of instruction miss ratio	0.213
Cache of integer instructions per cycle (cache way size)	0.013
Cache of data miss ratio	0.035
Cache of data hit ratio	0.497
Cache of data hit ratio	0.014
Cache of integer hit ratio	0.452
Cache of integer miss ratio	0.673
Cache of integer miss ratio	0.409
Branches used by 1 thread (MIPS)	1192.061
Branches used by 32 threads (MIPS)	1326.512
Branches used by 1 thread (MIPS)	190.037
Branches used by 32 threads (MIPS)	1.2181
Percentage of cycles waiting for memory access	43.1485
MIPS of 1 thread	18.019
MIPS of 32 threads	18.019
MIPS of CPU context	1.5615
MIPS of Effective	1.4441
Access to memory	19.2181
Cache of 1 thread	
Cache of 32 threads	

- Command line utility to gather aggregate counts.
 - PAPI version has been tested on IA32 & IA64
 - User can manually instrument code for more specific information
 - Reports derived metrics like SGI's perfex
- Developed by Luis Derosé at IBM ACTC.
- Hpmviz is a GUI to view results

hpmviz Screenshot



The screenshot shows the hpmviz application window. On the left, there is a table of performance metrics. Below it is a smaller table titled 'MPI Calc2 end Metrics'. On the right, there is a window displaying MPI code.

Loop	Excl Dur. (s)	Incl Dur. (s)	Count
Loop 200	89897	89897	1000
Loop 100	87008	87008	1000
Loop 300	87007	87007	1000
Calc1	2.769	60.791	1000
loop 1%	1.100	1.100	1000
MPI Calc2 start	1.004	1.004	1000
Calc2	0.580	85.483	1000
MPI Calc2 end	0.730	0.730	1000
MPI Calc2 end	0.588	0.588	1000
Initial	0.200	0.200	1
Calc1	0.191	69.467	1000
Calc2	0.05	0.05	1
MPI Calc2 start	0.007	0.007	1000
MPI Calc2 end	0.006	0.006	1000

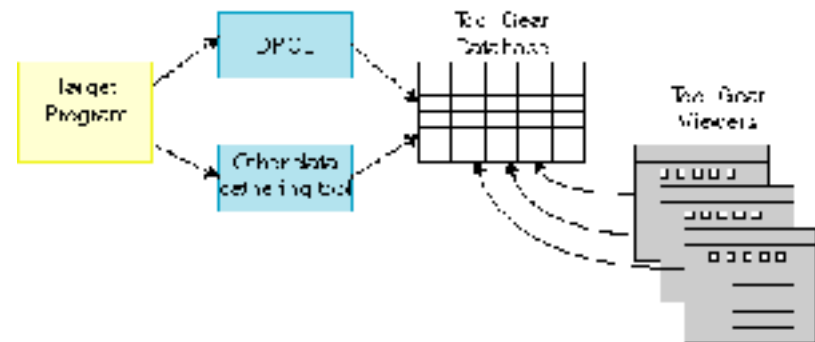
Task	Count
Task	0
Thread	28
Count	1000
ErrSec	0.000
ErrSec	0.000
ErrSec	0.000
LEAVE miss	159005
Total LS	48.28
Total LS	1.760
MIPS	116.281
Total Cycles	0.048
HW HW Cycle	0
FPI+Fwa	0.022
MIPS	0.000
FMA %	0
Cache Hit	0


```

c: f_prestat; do. "MPI Calc2 end"
if(taskid.eq.0)then
  call mpi_send(VIEW(1,1),n1,MPI_DOUBLE_PRECISION,
  i ranktask=1,1,MPI_COMM_WORLD,req(1), ierr)
endif
if(taskid.eq.ranktask-1)then
  call mpi_recv(VIEW(1,n+1),n1,MPI_DOUBLE_PRECISION,
  i 0,2,MPI_COMM_WORLD,req(2), ierr)
  call mpi_send(VIEW(1,n+1),n1,MPI_DOUBLE_PRECISION,
  i 0,3,MPI_COMM_WORLD,req(3), ierr)
endif
if(taskid.eq.ranktask-1)then
  call mpi_send(VIEW(1,n+1),n1,MPI_DOUBLE_PRECISION,
  i 0,1,MPI_COMM_WORLD,req(4), ierr)
endif
if(taskid.eq.0)then
  call mpi_send(VIEW(1,1),n1,MPI_DOUBLE_PRECISION,
  i ranktask=1,2,MPI_COMM_WORLD,req(5), ierr)
  call mpi_send(VIEW(1,1),n1,MPI_DOUBLE_PRECISION,
  i ranktask=1,3,MPI_COMM_WORLD,req(6), ierr)
endif
if(taskid.eq.0.or.taskid.eq.ranktask-1)then
  call MPI_WAITALL(6,req,stat, ierr)
endif
do i=1, N
  VIEW(1,n+1) = VIEW(1,i)
  VIEW(1,i) = VIEW(1,n+1)
  VIEW(1,n+1) = VIEW(1,i)
enddo
CONTINUE
VIEW(1,n+1) = VIEW(1,1)
VIEW(1,n+1) = VIEW(1,n+1)
VIEW(1,n+1) = VIEW(1,1)
if(taskid.eq.ranktask-1)then
  call mpi_recv(VIEW(1,n+1),n1,MPI_DOUBLE_PRECISION,
  i 0,2,MPI_COMM_WORLD,req(1), ierr)
  call mpi_send(VIEW(1,n+1),n1,MPI_DOUBLE_PRECISION,
  i 0,3,MPI_COMM_WORLD,req(2), ierr)
endif
if(taskid.eq.0)then
  call mpi_send(VIEW(1,1),n1,MPI_DOUBLE_PRECISION,
  i ranktask=1,2,MPI_COMM_WORLD,req(3), ierr)
  call mpi_send(VIEW(1,1),n1,MPI_DOUBLE_PRECISION,
  i ranktask=1,3,MPI_COMM_WORLD,req(5), ierr)
endif
if(taskid.eq.0)then
  call mpi_send(VIEW(1,1),n1,MPI_DOUBLE_PRECISION,
  i ranktask=1,2,MPI_COMM_WORLD,req(4), ierr)
  call mpi_send(VIEW(1,1),n1,MPI_DOUBLE_PRECISION,
  i ranktask=1,3,MPI_COMM_WORLD,req(5), ierr)
endif

```

- Dynamic instrumentation and analysis suite from LLNL
- Based on DPCL
 - Works only on AIX/Linux
- Front end can accept data from ‘any’ source



File	Line	Action	Start count	Stop count	Time	Flops
testcpnod.c	33	init_index_array();				
testcpnod.c	33	Entry to asfs				
testcpnod.c	33	Before call to init_index_array				
testcpnod.c	33	After call to init_index_array				
testcpnod.c	34	for(i = 0; i < IC; i++) {				
testcpnod.c	35	/* Tiled */				
testcpnod.c	36	init_array();				
testcpnod.c	36	Before call to init_array				
testcpnod.c	36	After call to init_array				
testcpnod.c	37					
testcpnod.c	38	printf("Doing %d flops of tiled test\n", FLOPS);				
testcpnod.c	38	Before call to printf				
testcpnod.c	38	After call to printf				
testcpnod.c	39	/* StartCachePerf(); */				
testcpnod.c	40	do_tiled_cache_test(FLOPS);				
testcpnod.c	40	Before call to do_tiled_cache_test				
testcpnod.c	40	After call to do_tiled_cache_test				
testcpnod.c	41	/* stopcacheprof(); */				
testcpnod.c	42					
testcpnod.c	43	/* Untiled */				
testcpnod.c	44	init_array();				
testcpnod.c	44	Before call to init_array				
testcpnod.c	44	After call to init_array				
testcpnod.c	45					

- A portable tool to dynamically instrument serial and parallel programs for the purpose of performance analysis.
- Simple and intuitive command line interface like GDB.
- Java/Swing GUI.
- Instrumentation is done through the run-time insertion of function calls to specially developed performance probes.



Why the “Dyna” in DynaProf?

- Instrumentation:
 - Functions are contained in shared libraries.
 - Calls to those functions are generated at run-time.
 - Those calls are dynamically inserted into the program’s address space.
- Built on DynInst and DPCL
- Can choose the mode of instrumentation, currently:
 - Function Entry/Exit
 - Call site Entry/Exit
 - One-shot

- Parallel framework based on DynInst
- Asynch./Sync. operation
- Functions for getting data back to tool
- Integrated with POE
- Available on all HPC platforms (and Windows)
- Breakpoints
- Arbitrary ins. points
- CFG and Basic Block decoding

- Popularized by James Larus with EEL: An Executable Editor Library at U. Wisc.
 - <http://www.cs.wisc.edu/~larus/eel.html>
- Technology matured by Dr. Bart Miller and (now Dr.) Jeff Hollingsworth at U. Wisc.
 - DynInst Project at U. Maryland
 - <http://www.dyninst.org/>
 - IBM's DPCL: A Distributed DynInst
 - <http://oss.software.ibm.com/dpcl/>

- Make collection of run-time performance data easy by:
 - Avoiding instrumentation and recompilation
 - Avoiding perturbation of compiler optimizations
 - Providing complete language independence
 - Allowing multiple insert/remove instrumentation cycles

No source code required!

DynaProf Goals 2

- Using the same tool with different probes
- Providing useful and meaningful probe data
- Providing different kinds of probes
- Allowing custom probe development Make collection of run-time performance data easy by:

No source code required!

- perfometerprobe
 - Visualize hardware event rates in “real-time”
- papiprobe
 - Measure any combination of PAPI presets and native events
- wallclockprobe
 - Highly accurate elapsed wallclock time in microseconds.
- The latter 2 probes report:
 - Inclusive
 - Exclusive
 - 1 Level Call Tree



Sample DynaProf Session

```
$/dynaprof
(dynaprof) load tests/swim
(dynaprof) list
DEFAULT_MODULE
swim.F
libm.so.6
libc.so.6
(dynaprof) list swim.F
MAIN__
inital_
calc1_
calc2_
calc3z_
calc3_
(dynaprof) list swim.F MAIN__
Entry
    Call s_wsle
    Call do_lio
    Call e_wsle
    Call s_wsle
    Call do_lio
    Call e_wsle
    Call calc3_
```

```
(dynaprof) use probes/papiprobe
Module papiprobe.so was loaded.
Module libpapi.so was loaded.
Module libperfctr.so was loaded.
(dynaprof) instr module swim.F calc*
swim.F, inserted 4 instrumentation points
(dynaprof) run
papiprobe: output goes to
/home/mucci/dynaprof/tests/swim.1671
```

- Probes export a few functions with loosely standardized interfaces.
- Easy to roll your own.
 - If you can code a timer, you can write a probe.
- DynaProf detects thread model.
- Probes dictate how the data is recorded and visualized.

- For threaded code, use the same probe!
- Dynaprof detects threads and loads a special version of the probe library.
- Each probe specifies what to do when a new thread is discovered.
- Each thread gets the same instrumentation.

- Can count any PAPI preset or Native event accessible through PAPI
- Can count multiple events
- Supports PAPI multiplexing
- Supports multithreading
 - AIX: SMP, OpenMP, Pthreads
 - Linux: SMP, OpenMP, Pthreads

- Counts microseconds using RTC
- Supports multithreading
 - AIX: SMP, OpenMP, Pthreads
 - Linux: SMP, OpenMP, Pthreads

- The wallclock and PAPI probes produce very similar data.
- Both use a parsing script written in Perl.
 - `wallclockrpt <file>`
 - `papiproberpt <file>`
- Produce 3 profiles
 - Inclusive: $T_{function} = T_{self} + T_{children}$
 - Exclusive: $T_{function} = T_{self}$
 - 1-Level Call Tree: $T_{child} = \textit{Inclusive } T_{function}$

```
(dynaprof) use probes/papiprobe PAPI_TOT_CYC, PAPI_TOT_INS
```

```
Module papiprobe.so was loaded.
```

```
Module libpapi.so was loaded.
```

```
Module libperfctr.so was loaded.
```

```
(dynaprof) instr function swim.F calc*
```

```
Swim.F, inserted 3 instrumentation points
```

```
(dynaprof) instr
```

```
calc1_
```

```
calc2_
```

```
calc3_
```

```
calc3z_
```



Swim Benchmark: Cycles & Instructions

Exclusive Profile of Metric PAPI_TOT_INS.

Name	Percent	Total	Call
TOTAL	100	1.723e+09	1
calc2	38.28	6.598e+08	120
calc1	32.31	5.567e+08	120
calc3	22.33	3.847e+08	118
unknown	7.084	1.221e+08	1

Inclusive Profile of Metric PAPI_TOT_INS.

Name	Percent	Total	SubC
TOTAL	100	1.723e+09	0
calc2	39.42	6.793e+08	1680
calc1	35.28	6.08e+08	1800
calc3	22.87	3.942e+08	1652

1-Level Inclusive Call Tree of Metric PAPI_TOT_INS.

Parent/-Child	Percent	Total	Call
TOTAL	100	1.723e+09	1
calc1	100	6.08e+08	120
- fsav	0.02065	1.255e+05	120
- mpi_irecv	0.03132	1.904e+05	120
- mpi_isend	0.05911	3.593e+05	120
- mpi_isend	0.06434	3.912e+05	120
-mpi_waitall	0.9013	5.479e+06	120
- mpi_irecv	0.03132	1.904e+05	120
- mpi_irecv	0.03132	1.904e+05	120
- mpi_isend	0.05356	3.256e+05	120
- mpi_isend	0.05079	3.088e+05	120
-mpi_waitall	6.813	4.142e+07	120
- mpi_irecv	0.03132	1.904e+05	120
- mpi_irecv	0.03132	1.904e+05	120
- mpi_isend	0.07504	4.562e+05	120
- mpi_isend	0.06757	4.108e+05	120
-mpi_waitall	0.161	9.791e+05	120
calc2	100	6.793e+08	120
- fsav	0.01848	1.255e+05	120
- mpi_irecv	0.02804	1.904e+05	120
- mpi_irecv	0.02804	1.904e+05	120

Exclusive Profile of Metric PAPI_TOT_CYC.

Name	Percent	Total	Calls
TOTAL	100	3.181e+09	1
calc2	34.85	1.108e+09	120
calc1	33.48	1.065e+09	120
calc3	26.1	8.301e+08	118
unknown	5.568	1.771e+08	1

Inclusive Profile of Metric PAPI_TOT_CYC.

Name	Percent	Total	SubCa
TOTAL	100	3.181e+09	0
calc2	35.98	1.144e+09	1680
calc1	35.61	1.133e+09	1800
calc3	26.88	8.55e+08	1652

1-Level Inclusive Call Tree of Metric PAPI_TOT_CYC.

Parent/-Child	Percent	Total	Calls
TOTAL	100	3.181e+09	1
calc1	100	1.133e+09	120
- fsav	0.03432	3.887e+05	120
- mpi_irecv	0.07356	8.332e+05	120
- mpi_isend	0.0663	7.51e+05	120
- mpi_isend	0.0739	8.371e+05	120
-mpi_waitall	0.7189	8.143e+06	120
- mpi_irecv	0.1646	1.864e+06	120
- mpi_irecv	0.03407	3.859e+05	120
- mpi_isend	0.1867	2.115e+06	120
- mpi_isend	0.06067	6.872e+05	120
-mpi_waitall	4.22	4.78e+07	120
- mpi_irecv	0.03979	4.506e+05	120
- mpi_irecv	0.03008	3.407e+05	120
- mpi_isend	0.1014	1.148e+06	120
- mpi_isend	0.07568	8.573e+05	120
-mpi_waitall	0.1076	1.219e+06	120
calc2	100	1.144e+09	120
- fsav	0.03382	3.87e+05	120
- mpi_irecv	0.03222	3.687e+05	120
- mpi_irecv	0.03554	4.067e+05	120

Swim Benchmark: Instructions per Cycle

Exclusive Profile of Metric PAPI_TOT_INS.

Name	Percent	Total	Calls
TOTAL	100	1.723e+09	1
calc2	38.28	6.598e+08	120
calc1	32.31	5.562e+08	120
calc3	22.33	3.847e+08	118
unknown	7.084	1.221e+08	1

Exclusive Profile of Metric PAPI_TOT_CYC.

Name	Percent	Total	Calls
TOTAL	100	3.181e+09	1
calc2	34.85	1.108e+09	120
calc1	33.48	1.067e+09	120
calc3	26.1	8.301e+08	118
unknown	5.568	1.771e+08	1

Inclusive Profile of Metric PAPI_TOT_INS.

Name	Percent	Total	SubC
TOTAL	100	1.723e+09	0
calc2	39.42	6.793e+08	1680
calc1	35.28	6.08e+08	
calc3	22.87	3.942e+08	

Inclusive Profile of Metric PAPI_TOT_CYC.

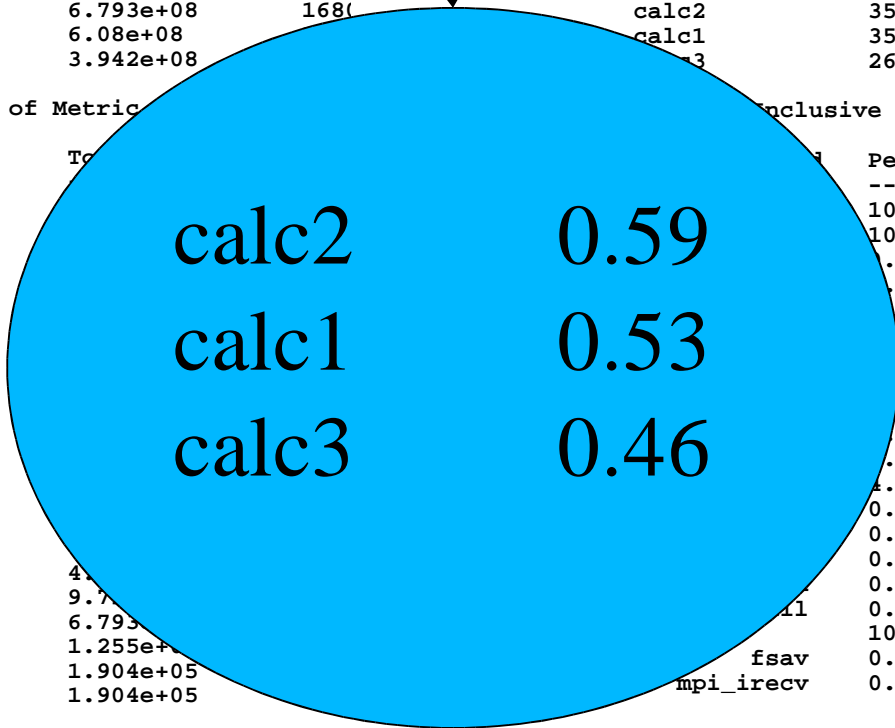
Name	Percent	Total	SubCa
TOTAL	100	3.181e+09	0
calc2	35.98	1.144e+09	1680
calc1	35.61	1.133e+09	1800
calc3	26.88	8.55e+08	1652

1-Level Inclusive Call Tree of Metric PAPI_TOT_INS.

Parent/-Child	Percent	Total
TOTAL	100	
calc1	100	
- fsav	0.02065	
- mpi_irecv	0.03132	
- mpi_isend	0.05911	
- mpi_isend	0.06434	
-mpi_waitall	0.9013	
- mpi_irecv	0.03132	
- mpi_irecv	0.03132	
- mpi_isend	0.05356	
- mpi_isend	0.05079	
-mpi_waitall	6.813	
- mpi_irecv	0.03132	
- mpi_irecv	0.03132	
- mpi_isend	0.07504	
- mpi_isend	0.06757	
-mpi_waitall	0.161	
calc2	100	
- fsav	0.01848	
- mpi_irecv	0.02804	
- mpi_irecv	0.02804	

1-Level Inclusive Call Tree of Metric PAPI_TOT_CYC.

Parent/-Child	Percent	Total	Calls
TOTAL	100	3.181e+09	1
calc1	100	1.133e+09	120
- fsav	0.03432	3.887e+05	120
- mpi_irecv	0.07356	8.332e+05	120
- mpi_isend	0.0663	7.51e+05	120
- mpi_isend	0.0739	8.371e+05	120
-mpi_waitall	0.7189	8.143e+06	120
- mpi_irecv	0.1646	1.864e+06	120
- mpi_irecv	0.03407	3.859e+05	120
- mpi_isend	0.1867	2.115e+06	120
- mpi_isend	0.06067	6.872e+05	120
-mpi_waitall	1.22	4.78e+07	120
- mpi_irecv	0.03979	4.506e+05	120
- mpi_irecv	0.03008	3.407e+05	120
- mpi_isend	0.1014	1.148e+06	120
- mpi_isend	0.07568	8.573e+05	120
-mpi_waitall	0.1076	1.219e+06	120
calc2	100	1.144e+09	120
- fsav	0.03382	3.87e+05	120
- mpi_irecv	0.03222	3.687e+05	120



- Graphically monitor rates of an applications performance in near real time.
- Ability to adjust metrics of interest as the application runs.
- Ability to mark functions of interest with different colors.

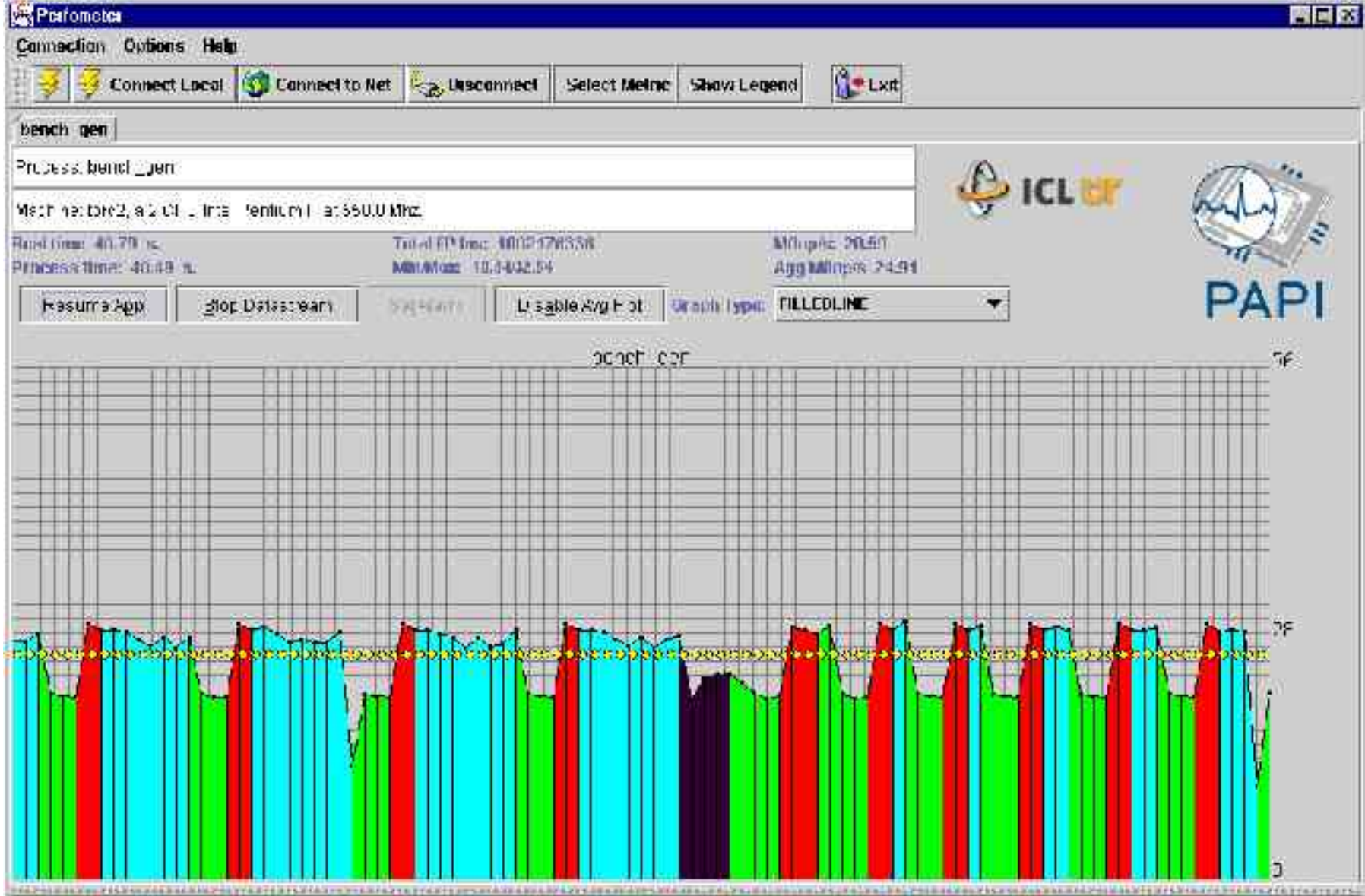


SWIM with perfometerprobe

```
Module perfometerprobe.so was loaded.  
Module libperfometer.so was loaded.  
Module libpapi.so was loaded.  
(dynaprof) instr function swim.F calc1_ 0xff0000  
swim.F, inserted 1 instrumentation points  
(dynaprof) instr function swim.F calc2_ 0x00ff00  
swim.F, inserted 1 instrumentation points  
(dynaprof) instr function swim.F calc3_ 0x0000ff  
swim.F, inserted 1 instrumentation points  
(dynaprof) run  
Module libnss_files.so.2 was loaded.  
Module libnss_nisplus.so.2 was loaded.  
Module libnsl.so.1 was loaded.  
Module libnss_dns.so.2 was loaded.  
Module libresolv.so.2 was loaded.  
Perfometer client awaiting connection on port #33733
```

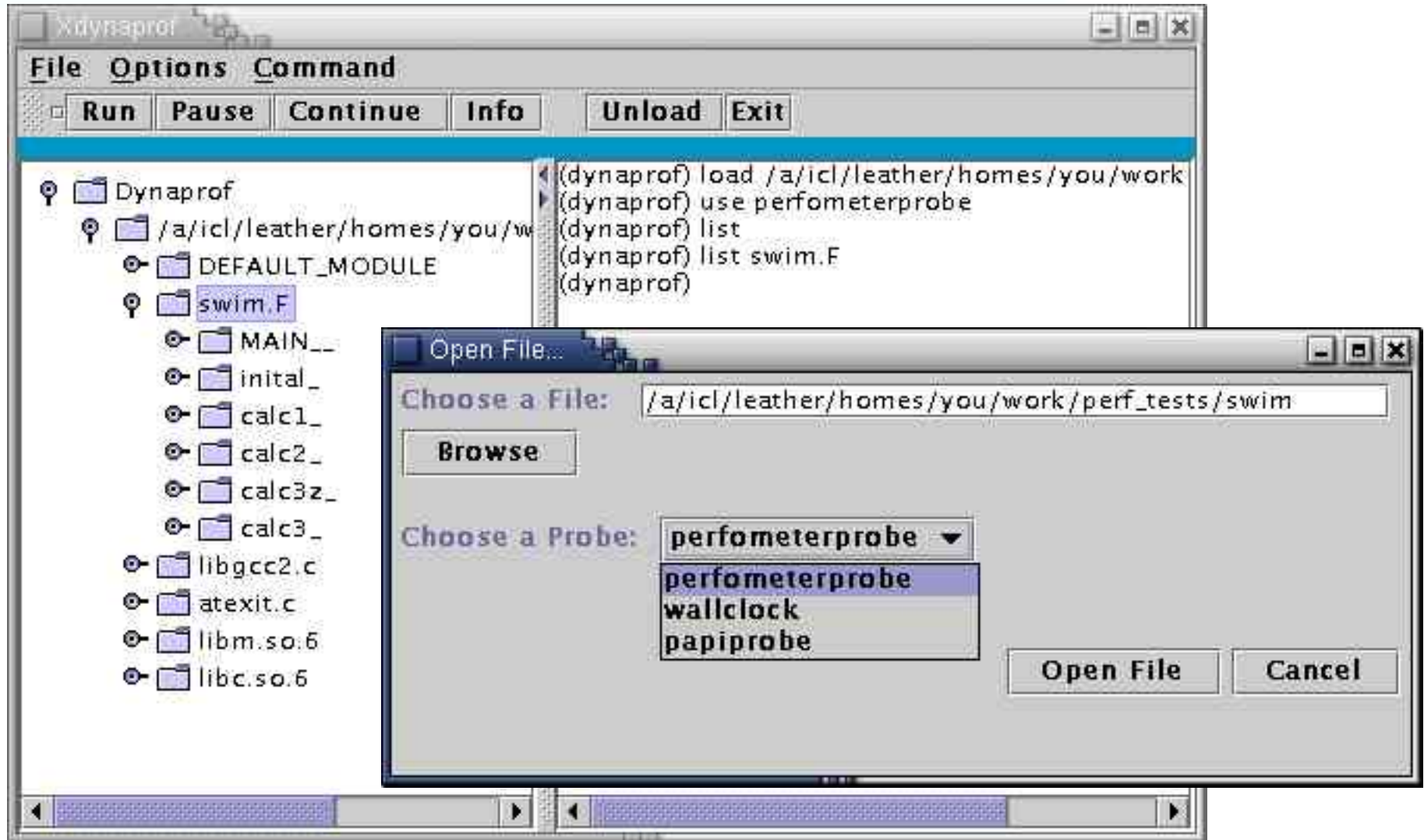


Perfometer Screenshot



- Displays module tree for instrumentation
- Simple selection of probes and instrumentation points
- Single-click execution of common DynaProf commands
- Coupling of probes and visualizers (e.g. perfometer)

DynaProf GUI Screenshot





Dynaprof v0.8 Release

- Supported Platforms
 - Using DynInst
 - Linux 2.x
 - AIX 4.3/5?
 - Solaris 2.8
 - IRIX 6.x
 - Using DPCL (formal MPI support)
 - AIX 4.3
 - AIX 5
- Available as a development snapshot from:
- Includes:
 - Java/Swing GUI
 - User's Guide
 - Probe libraries

<http://www.cs.utk.edu/~mucci/dynaprof>

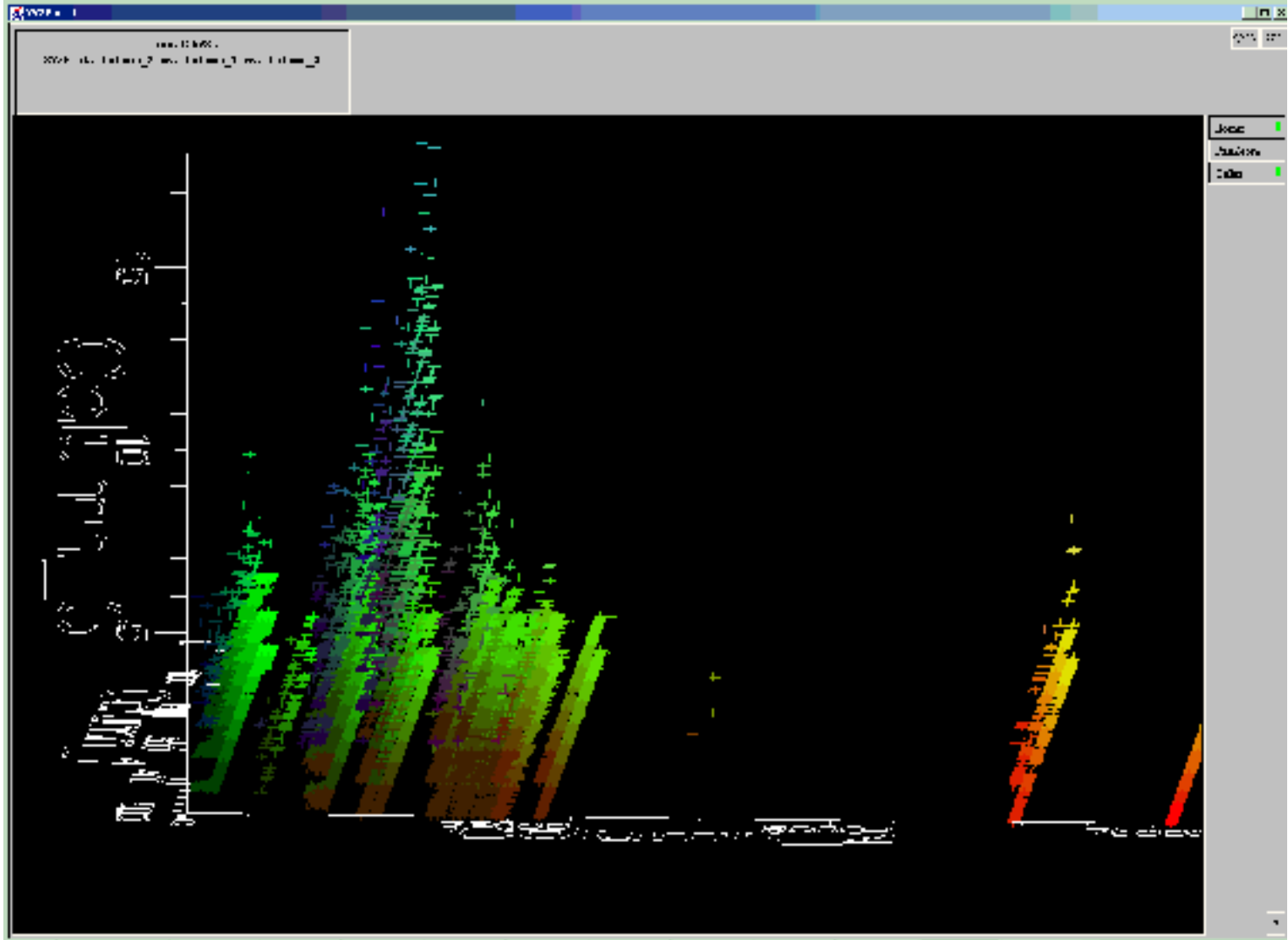
perfapi-devel@ptools.org

- Changes to the DynInst version:
 - Port performance probes to IA64
 - Interactive instrumentation of MPI codes, faster start-up
 - Integration with the Tool Daemon Protocol from U. Wisconsin.
 - New instrumentation point support
 - CFG/Basic Block/Loop level instrumentation exists but is untested
 - Arbitrary start/stop points
 - Arbitrary breakpoints and ‘run-until’ support
 - Support for programs that dynamically load extensions (i.e. Mozilla)
- Documentation:
 - Probe API
 - Tutorial
- Probes
 - **Integration with TAU:**
 - **tauprobe**
 - **TAU/jracy compatible output from papiprobe and wallclockprobe**
 - Additional thread model support: `sproc()` on IRIX
 - Improved data structures for handling for multiple instrumentation cycles
 - Thread support in perfometer probe

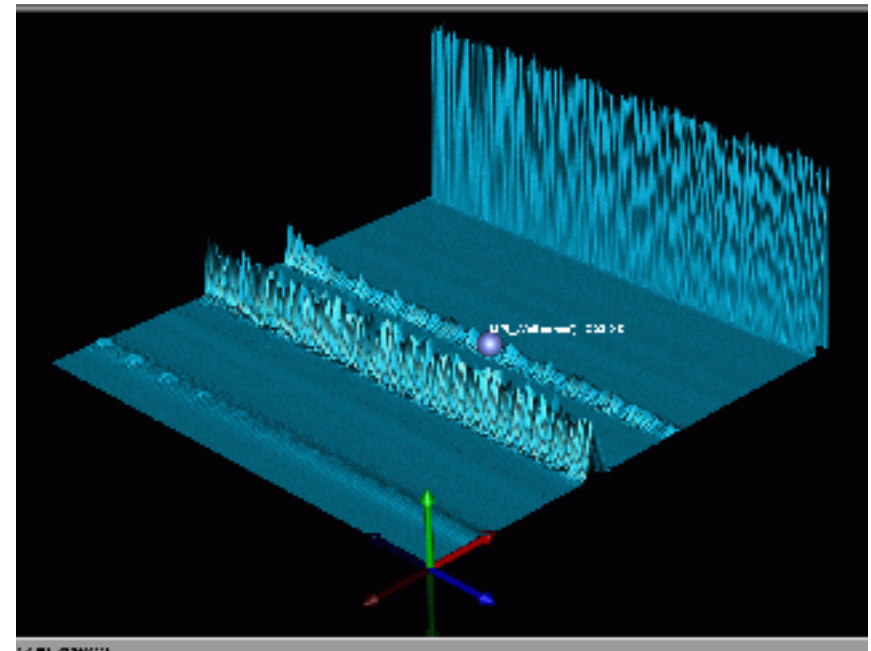
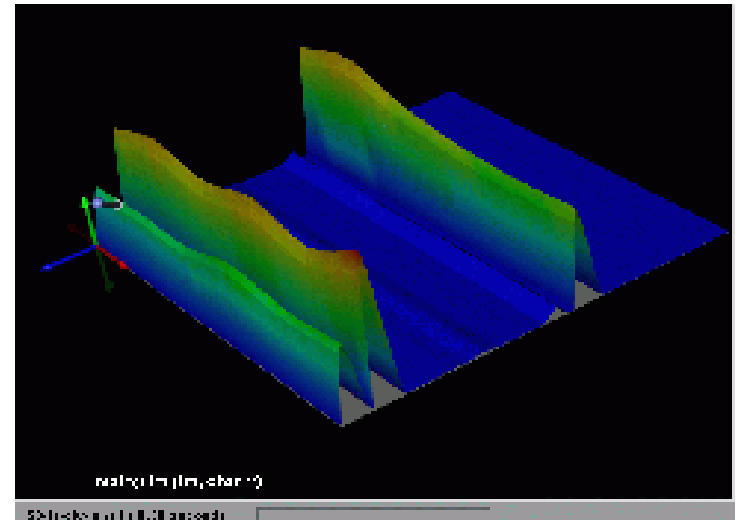
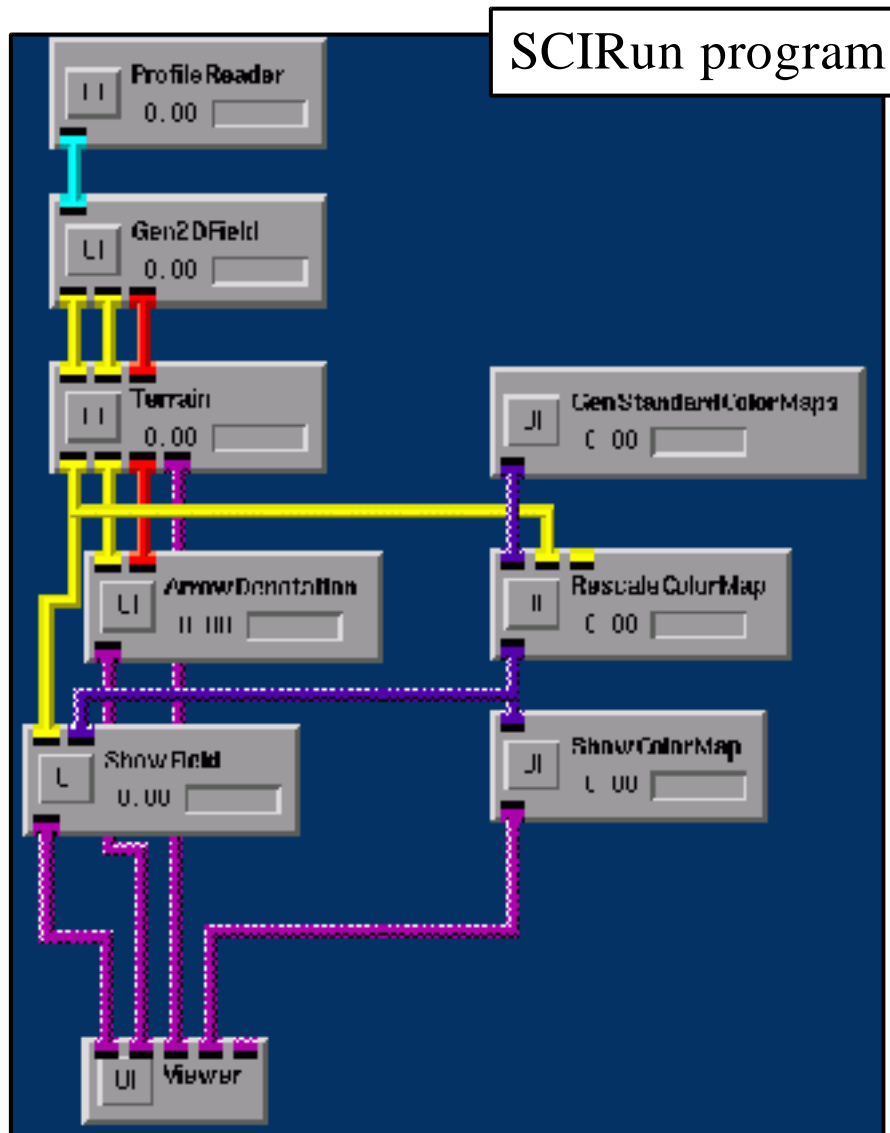
- Statistical profiling is often static
 - Gprof, Quantify, Speedshop, Workshop, Tprof, etc...
- We want to understand all aspects of a program's performance. What about behaviour over time?
- Applications vs. kernels have distinct phases.
 - Initialization
 - Data input
 - Compute
 - Communicate
 - Repeat
 - Data output
 - Finalization

- Workloads on the hardware are most often periodic.
- Open questions:
 - How do we process, visualize and understand this data in a scalable fashion?
 - Can we use this data to optimize an application in the temporal domain?
 - Can we parameterize this data against (t) for performance models?

Sample of Space vs. Time vs. Frequency



2D Field Performance Visualization in TAU & SCIRun



- Most of the infrastructure now exists.
- Many sites are “rolling their own”.
- Can one size fit all?
- 2 types of tools evolving:
 - Simple
 - Comprehensive

- Database of all relevant information regarding the performance of a code.
 - Source code structure
 - Transformations performed during optimization
 - Static and dynamic memory allocation information
 - Derived data types, etc...
- Examples:
 - TAU PDT: Program Database Toolkit
 - HPC Tools: XML Database
 - ToolGear
- This data can be quite large! Remember MPI traces?

Some problems to be solved

- How do we get the data out of the threads/processors/nodes/application?
 - Aggregation
 - Filtering
 - Reduction
 - Tool Daemon Protocol from U. Wisconsin
- How do we correlate performance data from optimized code to the exact line of source code?
 - Software pipelining
 - C++ Templates

- <http://icl.cs.utk.edu/projects/papi>
- <http://www.cs.utk.edu/~mucci/dynaprof>
- <http://www.cs.rice.edu/~dsystem/hpcview>
- <http://aros.ca.sandia.gov/~cljanss/perf/vprof>
- <http://www.alphaworks.ibm.com/tech/hpmtoolkit>
- <http://perfsuite.ncsa.uiuc.edu/psrun>
- <http://software.sci.utah.edu/scirun.html>
- <http://www.paradyn.org>
- <http://www.cs.uoregon.edu/research/paracomp/tau>
- <http://vibes.cs.uiuc.edu/Software/SvPablo/svPablo.htm>
- <http://www.aei-potsdam.mpg.de>
- <http://aros.ca.sandia.gov/~cljanss/perf/vprof>
- <http://www.llnl.gov/asci/projects/asde/toolgear.html>
- <http://www.dyninst.org>
- <http://oss.software.ibm.com/developerworks/opensource/dpcl>

