



Deep Computing

## The IBM High Performance Computing Toolkit

**Advanced Computing Technology Center**

**<http://www.research.ibm.com/actc>**

**[/usr/lpp/ppe.hpct/](#)**

*Rajiv Bendale*

*Jerrold Heyman*

*Kirk E. Jordan*

*Brian Smith*

*Robert E. Walkup*

*[bendale@us.ibm.com](mailto:bendale@us.ibm.com)*

*[jheyman@us.ibm.com](mailto:jheyman@us.ibm.com)*

*[kjordan@us.ibm.com](mailto:kjordan@us.ibm.com)*

*[smithbr@us.ibm.com](mailto:smithbr@us.ibm.com)*

*[walkup@us.ibm.com](mailto:walkup@us.ibm.com)*

## Outline

- **Various Tools for Improved Performance**
- **Performance Decision Tree**
- **IBM HPCToolkit**
- **Remarks**



Deep Computing

# Performance

Compilers  
Libraries  
Tools  
Running

# HPC Tools Available for HPC

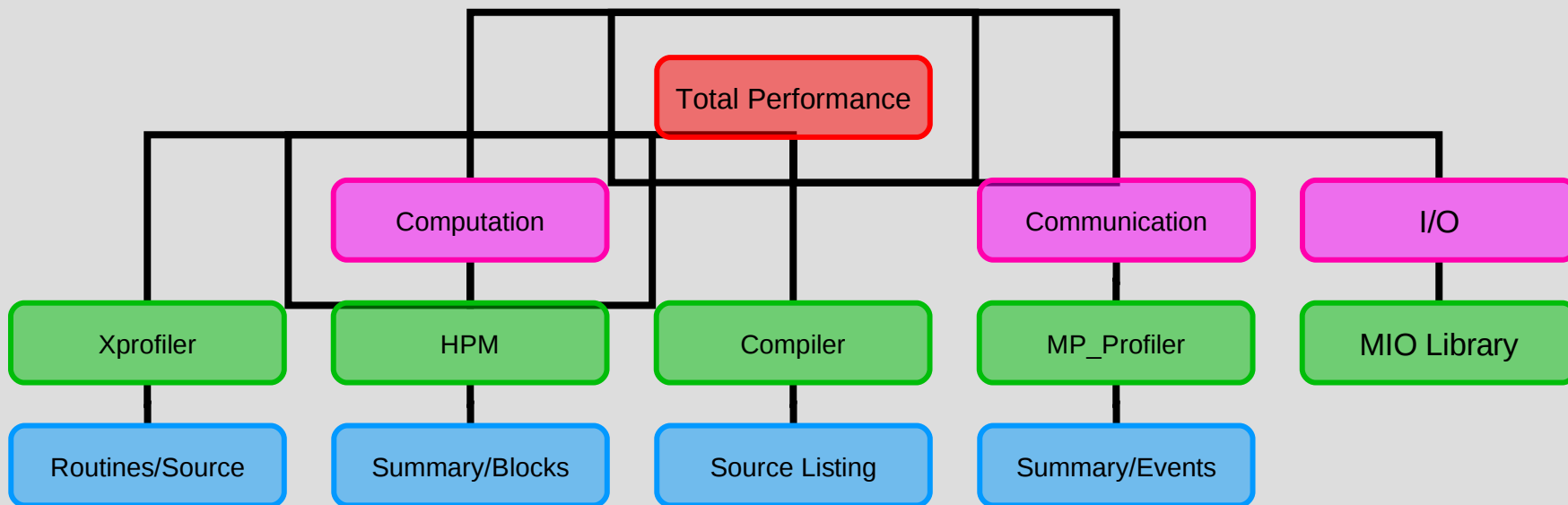
## IBM Software Stack

- **XL Compilers**
  - Externals preserved
  - New options to optimize for specific Blue Gene functions
- **LoadLeveler**
  - Same externals for job submission and system query functions
  - Backfill scheduling to achieve maximum system utilization
- **GPFS**
  - Provides high performance file access, as in current pSeries and xSeries clusters
  - Runs on IO nodes and disk servers
- **ESSL/MASSV**
  - Optimization library and intrinsics for better application performance
  - Serial Static Library supporting 32-bit applications
  - Callable from FORTRAN, C, and C++

## Other Software

- **TotalView Technologies TotalView**
  - Parallel Debugger
- **Lustre File System**
  - Enablement underway at LLNL
- **FFT Library**
  - FFTW Tuned functions by TU-Vienna
- **Performance Tools**
  - Total View
  - HPC Toolkit
  - Paraver
  - Kojak

# Performance Decision Tree

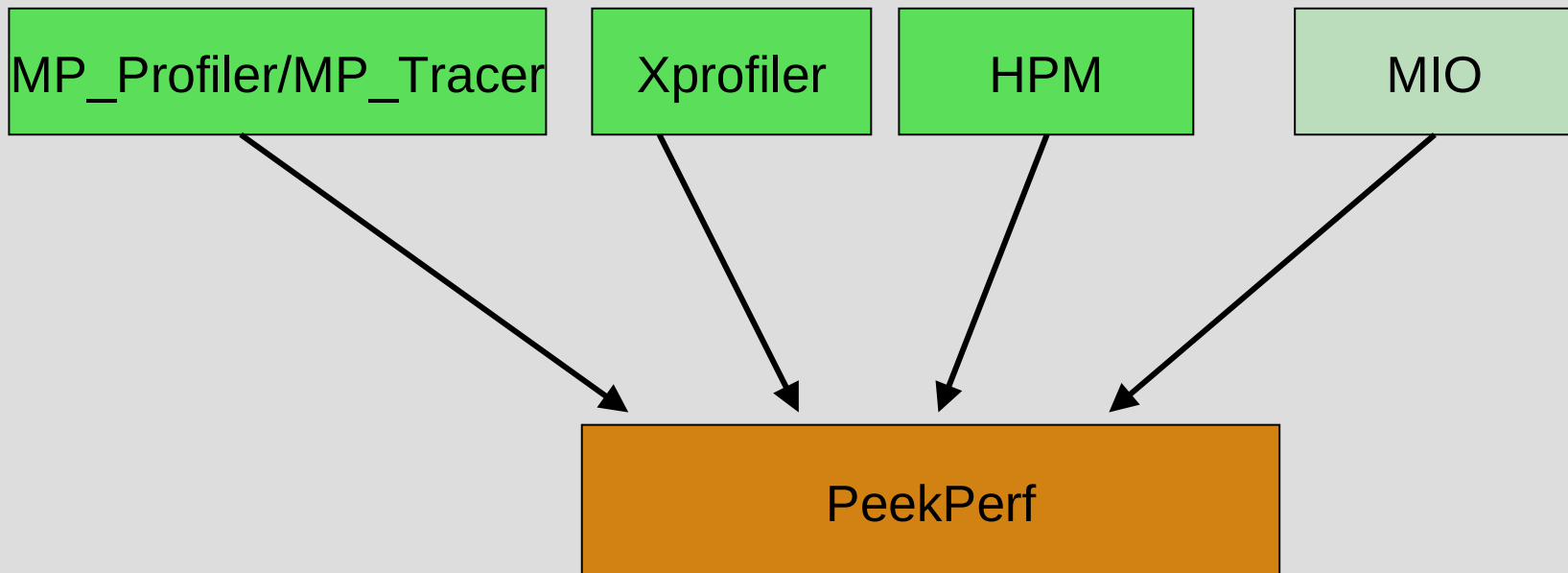


# IBM High Performance Computing Toolkit - What is it?

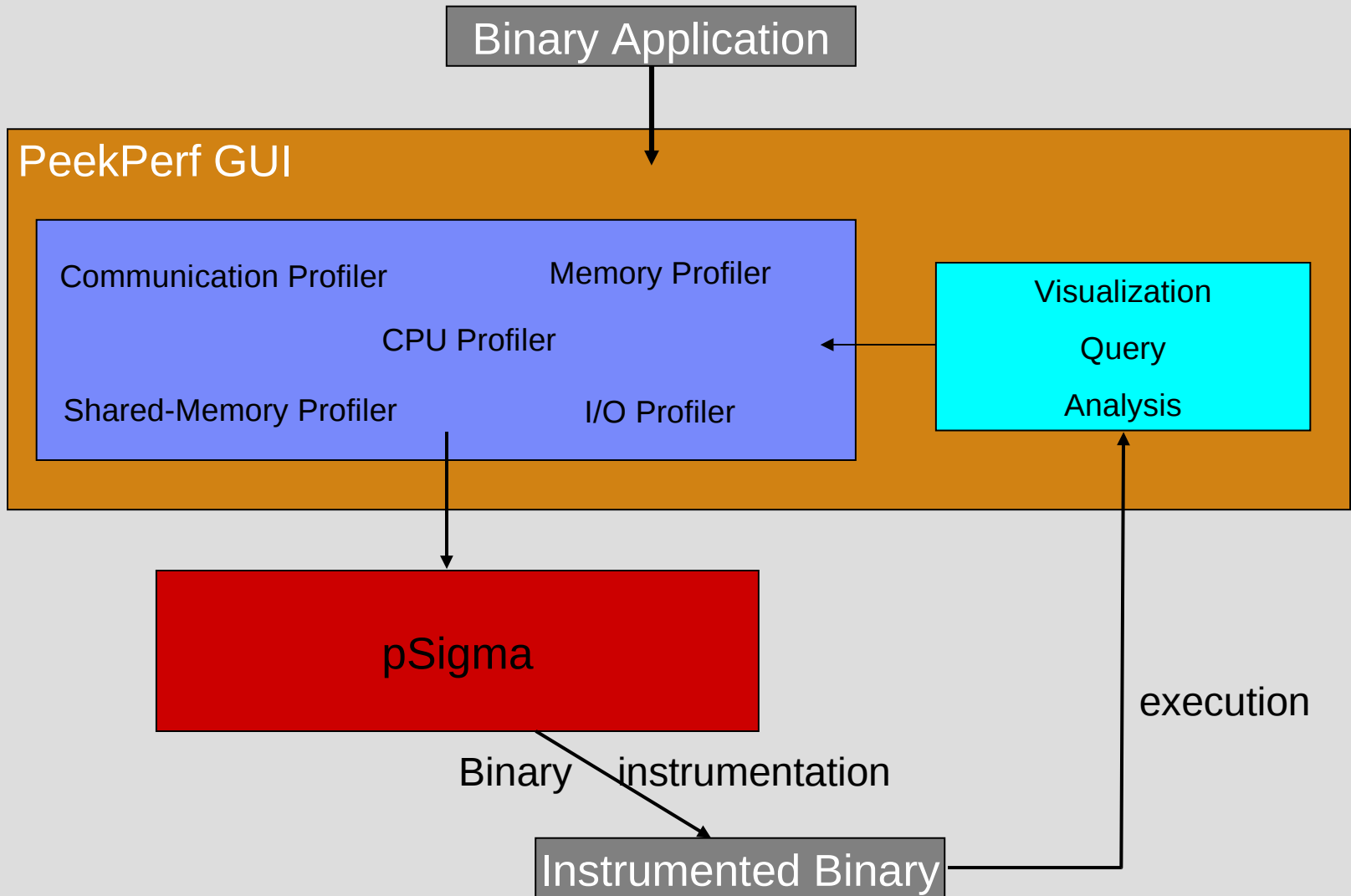
- **IBM long-term goal:**
  - An automatic performance tuning framework
    - Assist users to identify performance problems
  - A common application performance analysis environment across all HPC platforms
  - Look at all aspects of performance (communication, memory, processor, I/O, etc) from within a single interface
- **Where we are: one consolidated package**
  - One consolidate package (Blue Gene, AIX, Linux/Power)
  - Operate on the binary and yet provide reports in terms of source-level symbols
  - Dynamically activate/deactivate data collection and change what information to collect
  - One common visualization GUI

## IBM High Performance Computing Toolkit

- **MPI performance: MPI Profiler/Tracer**
- **CPU performance: Xprofiler, HPM**
- **Threading performance: OpenMP profiling**
- **I/O performance: I/O profiling**
- **Visualization and analysis: PeekPerf**



# Structure of the HPC toolkit





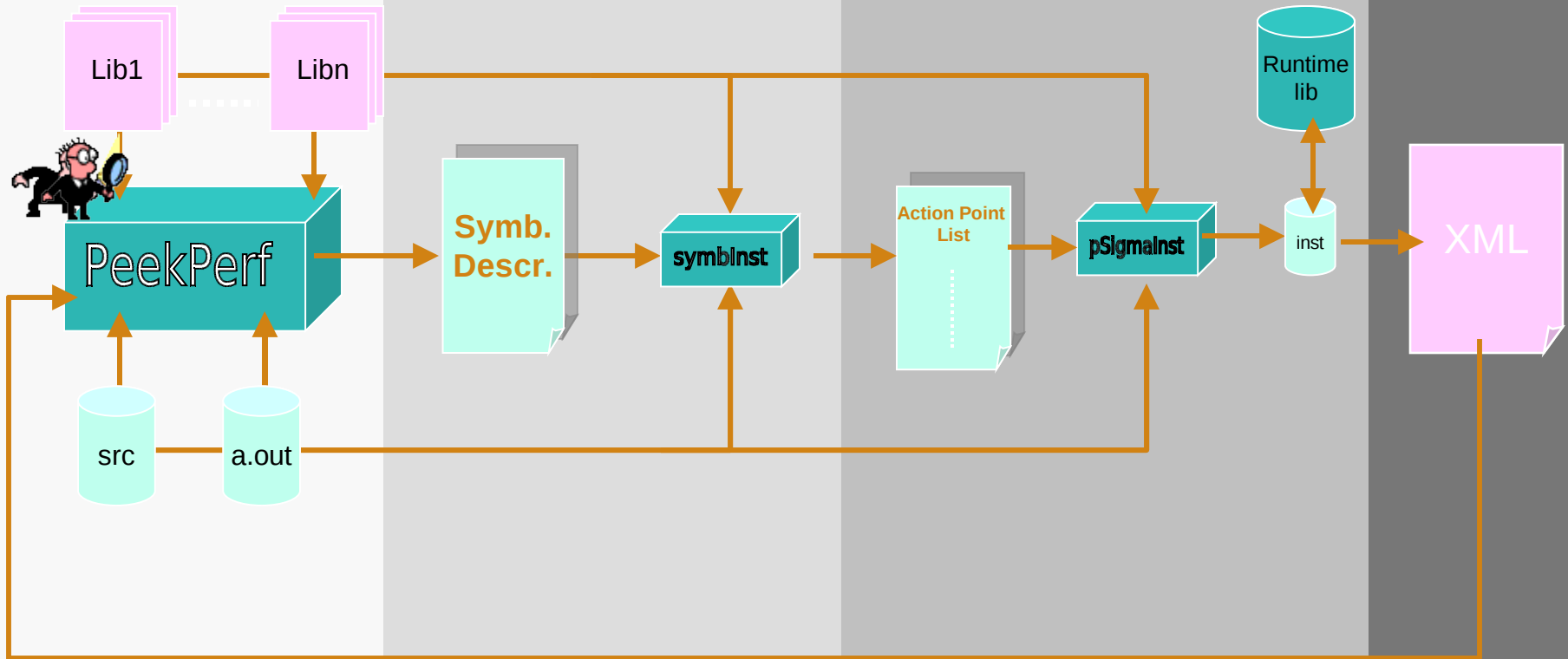
# PeekPerf: Graphical Instrumentation, Visualization and Analysis

Instrumentation  
Visualization  
Analysis

Symbolic Binary  
Instrumentation

Action Point Binary  
Instrumentation

Visualization



# Message-Passing Performance

## MPI Profiler/Tracer

- Implements wrappers around MPI calls using the PMPI interface
  - start timer
  - call pmpi equivalent function
  - stop timer
- Captures MPI calls with source code traceback
- No changes to source code, but MUST compile with -g
- Does not synchronize MPI calls
- Compile with -g and link with libmpitrace.a
- Generate XML files for peekperf

## MPI Tracer

- Captures “timestamped” data for MPI calls with source traceback
- Provides a color-coded trace of execution
- Very useful to identify load-balancing issues

# MPI Profiler Output

Main Window
File Manual Automatic Windows Tool

DATA VISUALIZATION WINDOW

mpidata

Label	Count	WallClock	Transferred Bytes
- SUMMARY			
MPI_Irecv	1000	0.009671	2.21952e+07
MPI_Isend	1000	0.025689	2.21952e+07
- mhd.F			
- neighbours(mhd.F)			
MPI_Irecv_631	50	0.000579	614400
MPI_Irecv_636	50	0.000394	614400
MPI_Irecv_669	50	0.000426	614400
MPI_Irecv_674	50	0.000448	614400
MPI_Irecv_716	200	0.001829	4.9152e+06
MPI_Irecv_721	200	0.0019	4.9152e+06
MPI_Irecv_771	200	0.001862	4.9536e+06
MPI_Irecv_776	200	0.002233	4.9536e+06
MPI_Isend_629	50	0.00313	614400
MPI_Isend_634	50	0.000426	614400
MPI_Isend_667	50	0.000753	614400
MPI_Isend_672	50	0.000592	614400
MPI_Isend_714	200	0.005746	4.9152e+06
MPI_Isend_719	200	0.002964	4.9152e+06
MPI_Isend_769	200	0.009673	4.9536e+06
MPI_Isend_774	200	0.002405	4.9536e+06

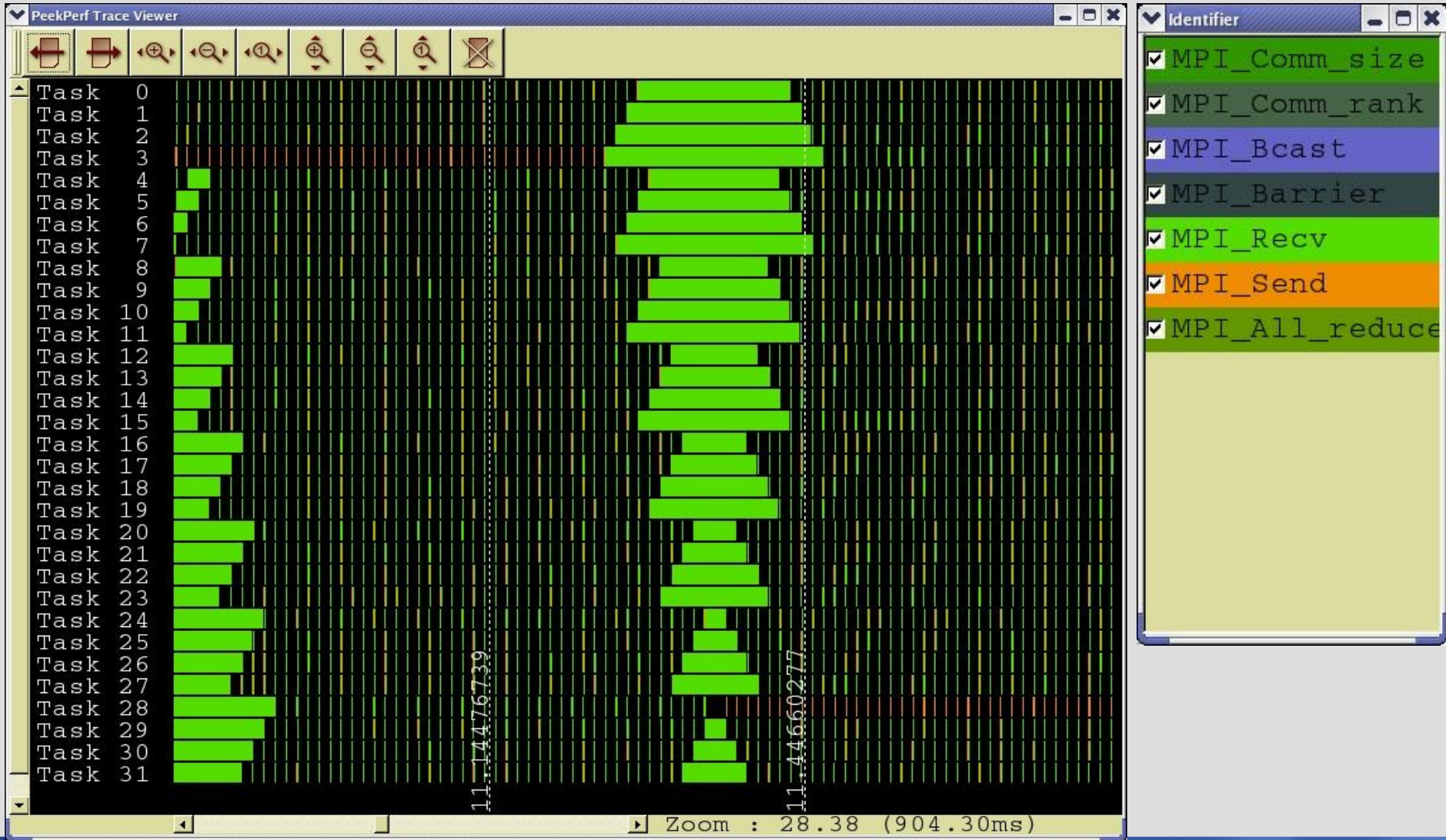
Metric Browser: MPI\_Irecv\_669
Close Metric Options Precision

Task	Count	WallClock	Transferred Bytes
0	50	0.000426	614400
1	50	0.000245	614400
2	50	0.000565	614400
3	50	0.000368	614400

```

sum(te
655
656
657
658
659
660
661
662
663
664   temp2s1(length2+i) = geq(i,jsta,7,2)
665   enddo
666
667   CALL
MPI_ISEND(temp1s2,length3,MPI_DOUBLE_PRECISION,jnext
&
668   1   ,comm2d,isend(3),ierr)
669   CALL
MPI_Irecv(temp1r2,length3,MPI_DOUBLE_PRECISION,jprev
&
670   1   ,comm2d,irecv(3),ierr)
671
672   CALL
MPI_ISEND(temp2s1,length3,MPI_DOUBLE_PRECISION,jprev
&
673   1   ,comm2d,isend(4),ierr)
674   CALL
        
```

# MPI Tracer output



# MPI Message Size Distribution

MPI Function	#Calls	Message Size	#Bytes	Walltime	MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Comm_size	1 (1)	0 ... 4	0	1E-07	MPI_Irecv	2 (1)	0 ... 4	3	4.7E-06
MPI_Comm_rank	1 (1)	0 ... 4	0	1E-07	MPI_Irecv	2 (2)	5 ... 16	12	1.4E-06
MPI_Isend	2 (1)	0 ... 4	3	0.000006	MPI_Irecv	2 (3)	17 ... 64	48	1.5E-06
MPI_Isend	2 (2)	5 ... 16	12	1.4E-06	MPI_Irecv	2 (4)	65 ... 256	192	2.4E-06
MPI_Isend	2 (3)	17 ... 64	48	1.3E-06	MPI_Irecv	2 (5)	257 ... 1K	768	2.6E-06
MPI_Isend	2 (4)	65 ... 256	192	1.3E-06	MPI_Irecv	2 (6)	1K ... 4K	3072	3.4E-06
MPI_Isend	2 (5)	257 ... 1K	768	1.3E-06	MPI_Irecv	2 (7)	4K ... 16K	12288	7.1E-06
MPI_Isend	2 (6)	1K ... 4K	3072	1.3E-06	MPI_Irecv	2 (8)	16K ... 64K	49152	2.23E-05
MPI_Isend	2 (7)	4K ... 16K	12288	1.3E-06	MPI_Irecv	2 (9)	64K ... 256K	196608	9.98E-05
MPI_Isend	2 (8)	16K ... 64K	49152	1.3E-06	MPI_Irecv	2 (A)	256K ... 1M	786432	0.00039
MPI_Isend	2 (9)	64K ... 256K	196608	1.7E-06	MPI_Irecv	1 (B)	1M ... 4M	1048576	0.000517
MPI_Isend	2 (A)	256K ... 1M	786432	1.7E-06	MPI_Waitall	21 (1)	0 ... 4	0	1.98E-05
MPI_Isend	1 (B)	1M ... 4M	1048576	9E-07	MPI_Barrier	5 (1)	0 ... 4	0	7.8E-06

# Xprofiler

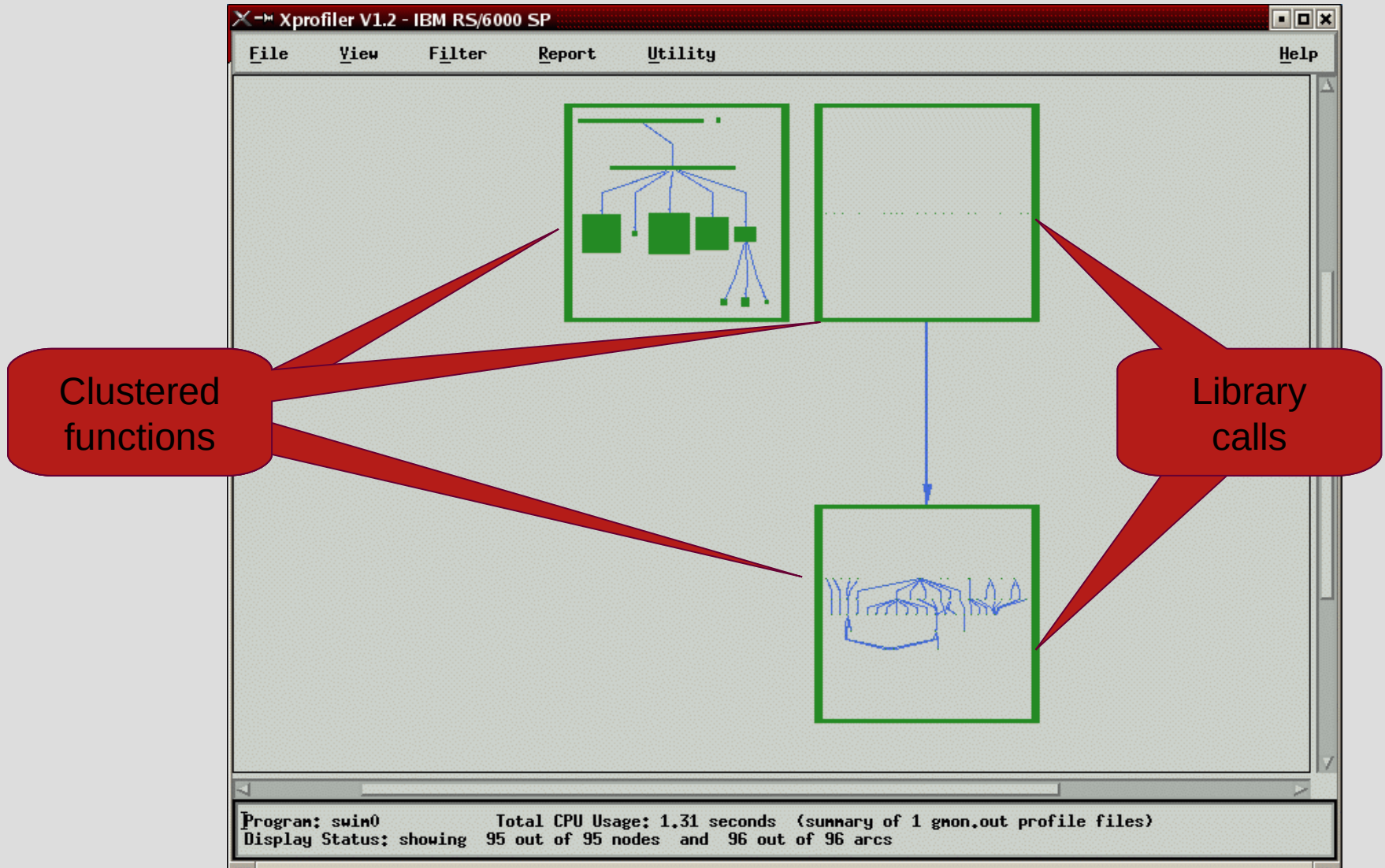
- CPU profiling tool similar to gprof
- Can be used to profile both serial and parallel applications
- Use procedure-profiling information to construct a graphical display of the functions within an application
- Provide quick access to the profiled data and helps users identify functions that are the most CPU-intensive
- Based on sampling (support from both compiler and kernel)
- Charge execution time to source lines and show disassembly code

# CPU Profiling

- **Compile the program with -pg**
- **Run the program**
- **gmon.out file is generated (MPI applications generate gmon.out.1, ..., gmon.out.n)**
- **Run Xprofiler component**

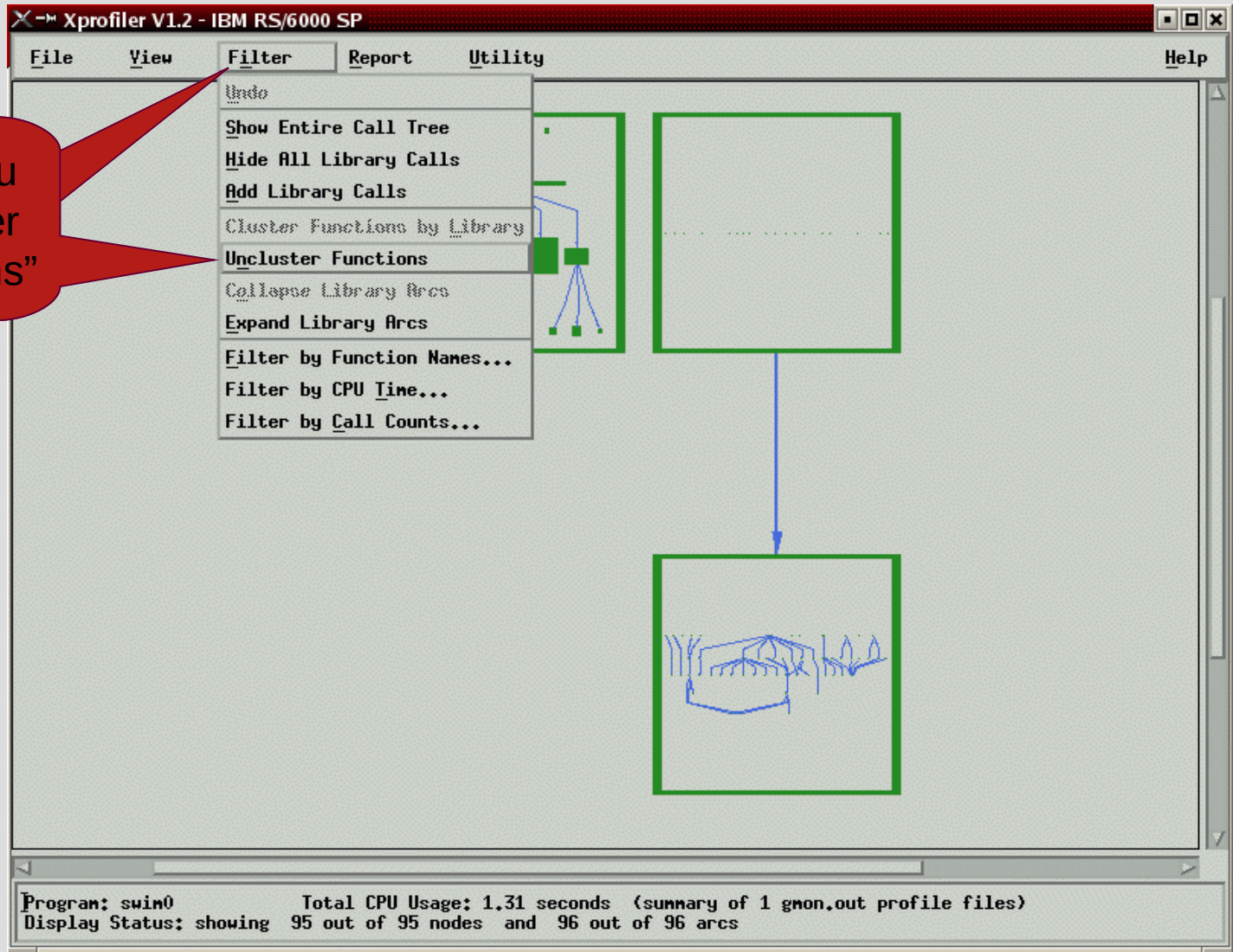


# Xprofiler - Initial View



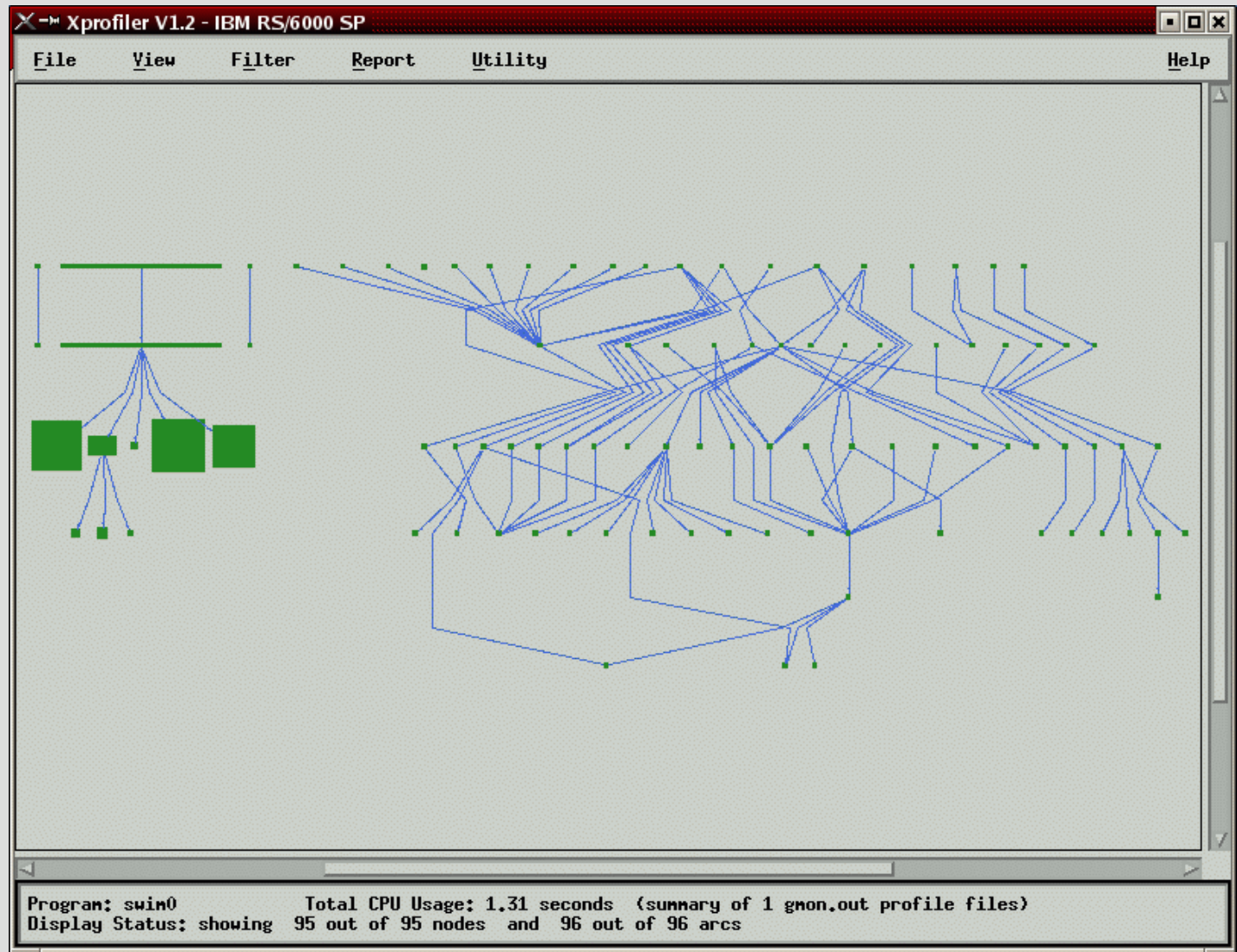


# Xprofiler - Unclustering Functions

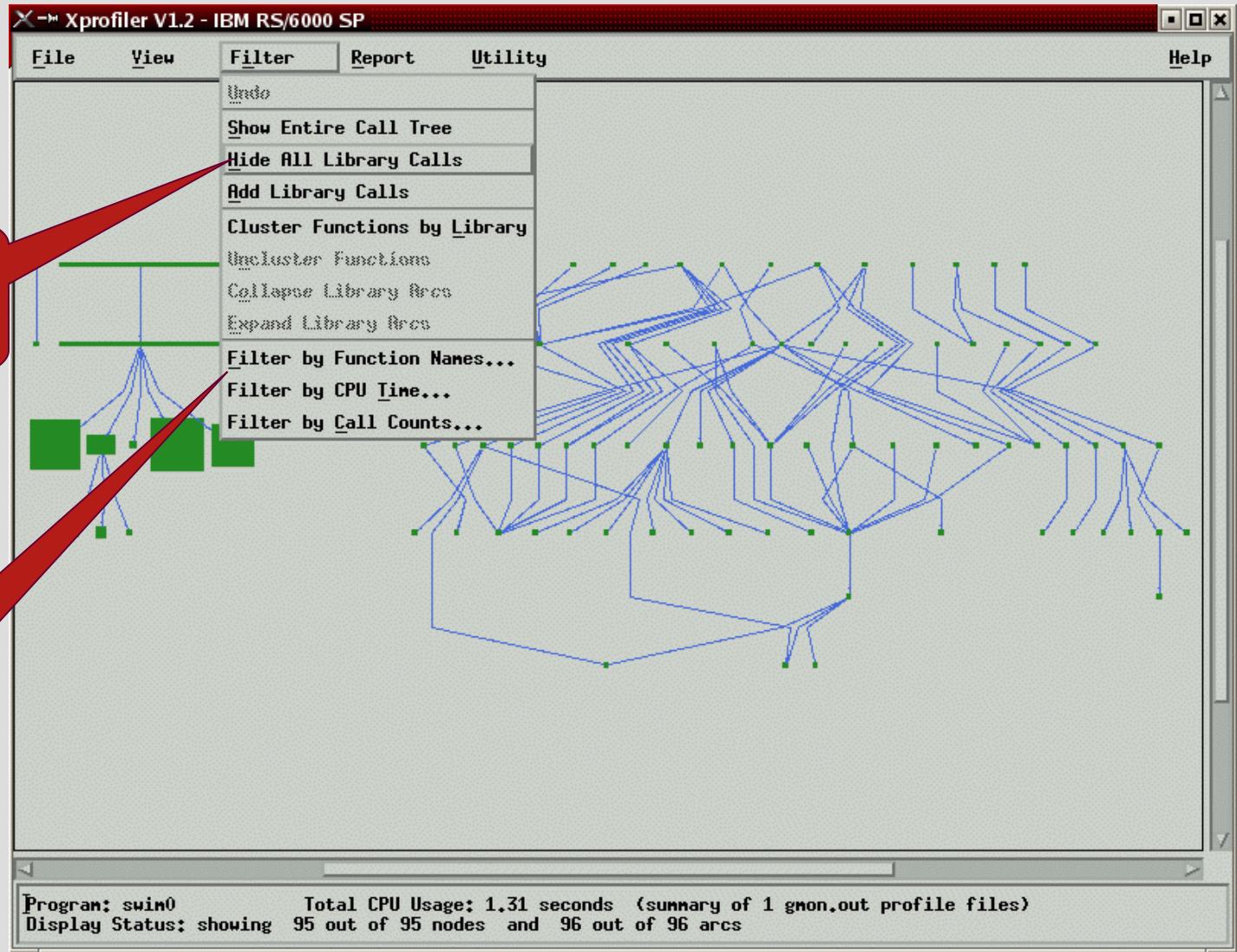


on "Filter" menu  
select "Uncluster  
Functions"

# Xprofiler - Full View - Application and Library Calls



# Xprofiler - Hide Lib Calls Menu



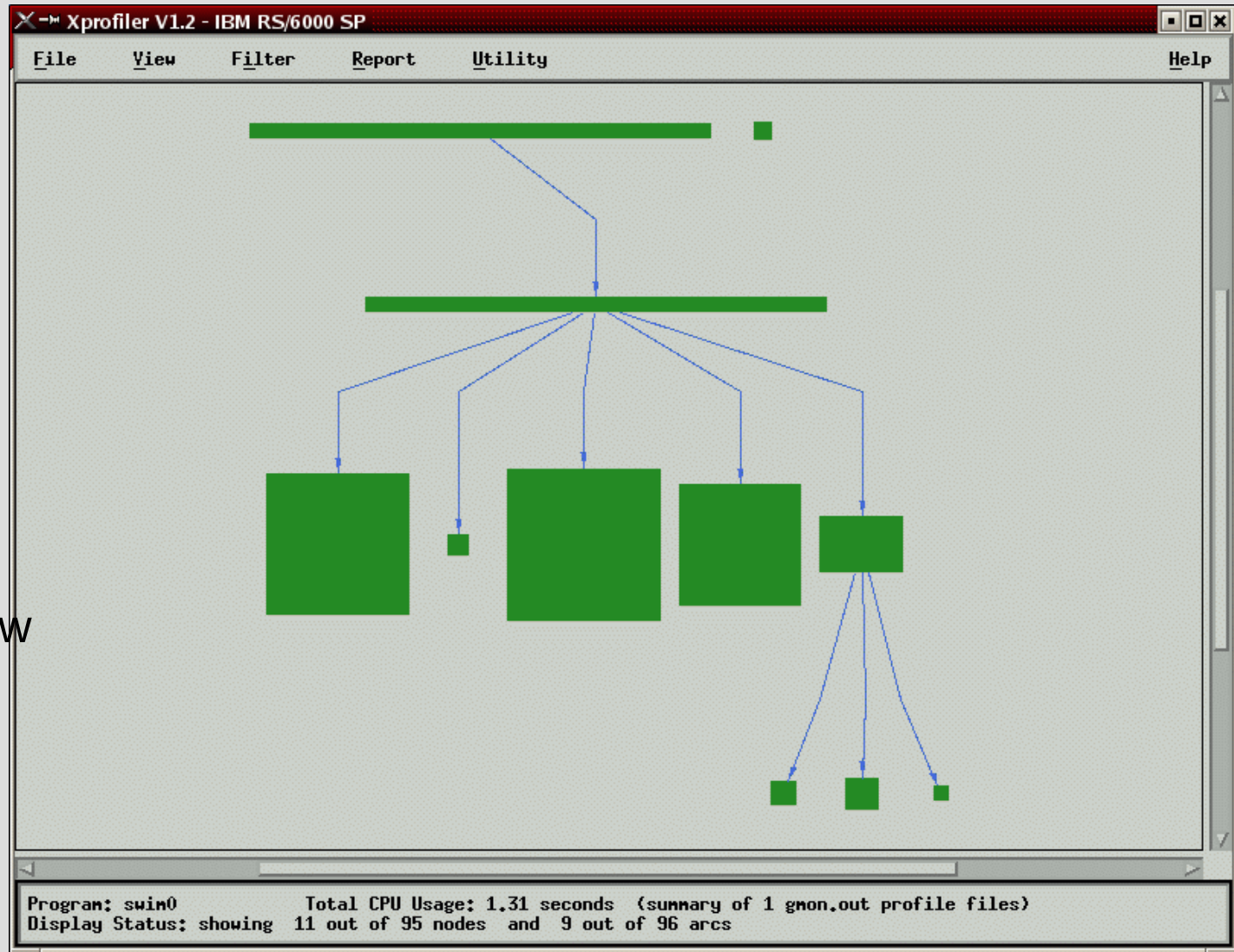
Now select  
"Hide All  
Library Calls"

Can also filter by:  
Function Names,  
CPU Time,  
Call Counts

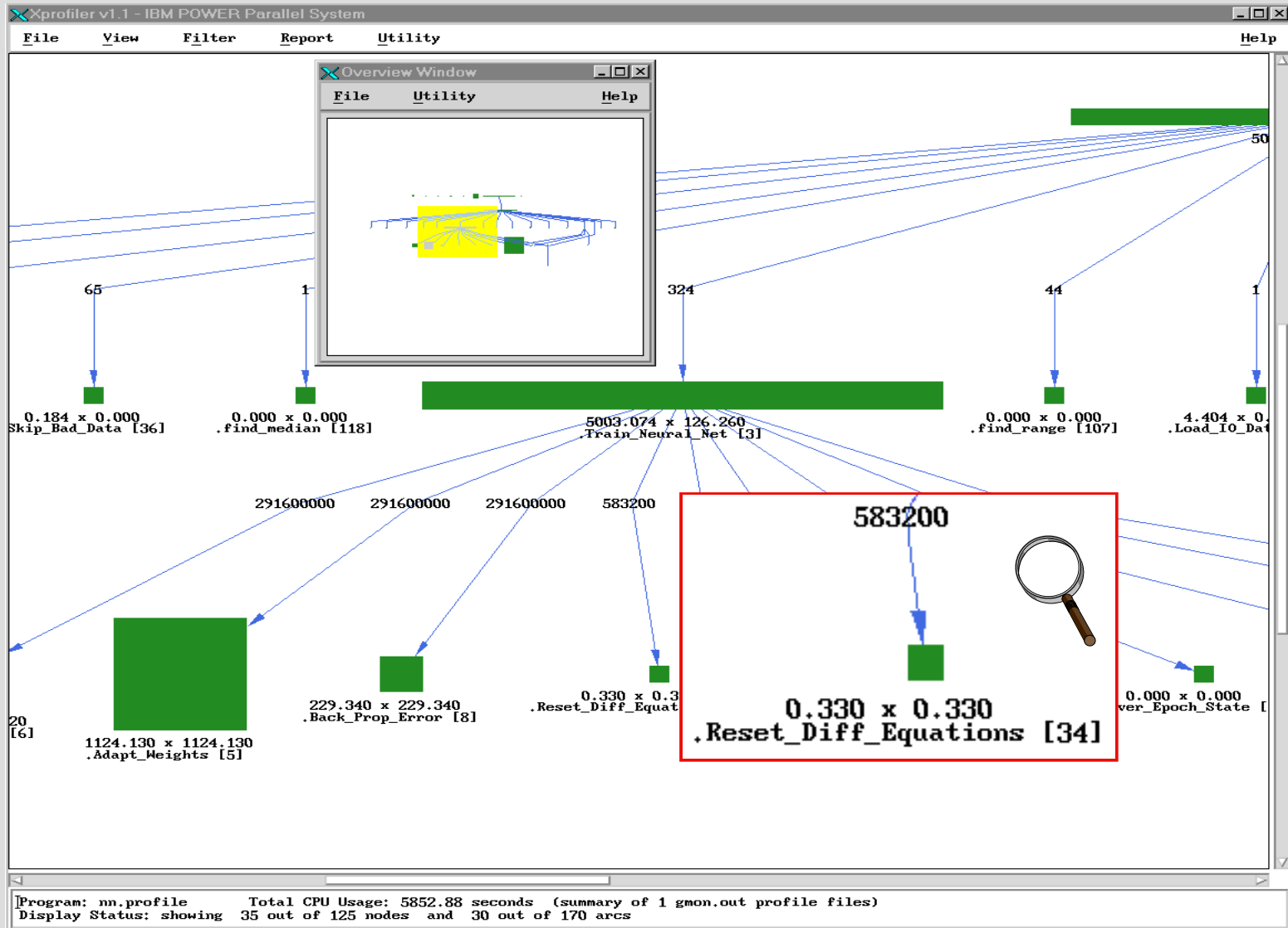


# Xprofiler - Application View

- Width of a bar: time including called routines
- Height of a bar: time excluding called routines
- Call arrows labeled with number of calls
- Overview window for easy navigation (View → Overview)



# Xprofiler: Zoom In



# Xprofiler: Flat Profile

- Menu **Report** provides usual gprof reports plus some extra ones

- Flat Profile

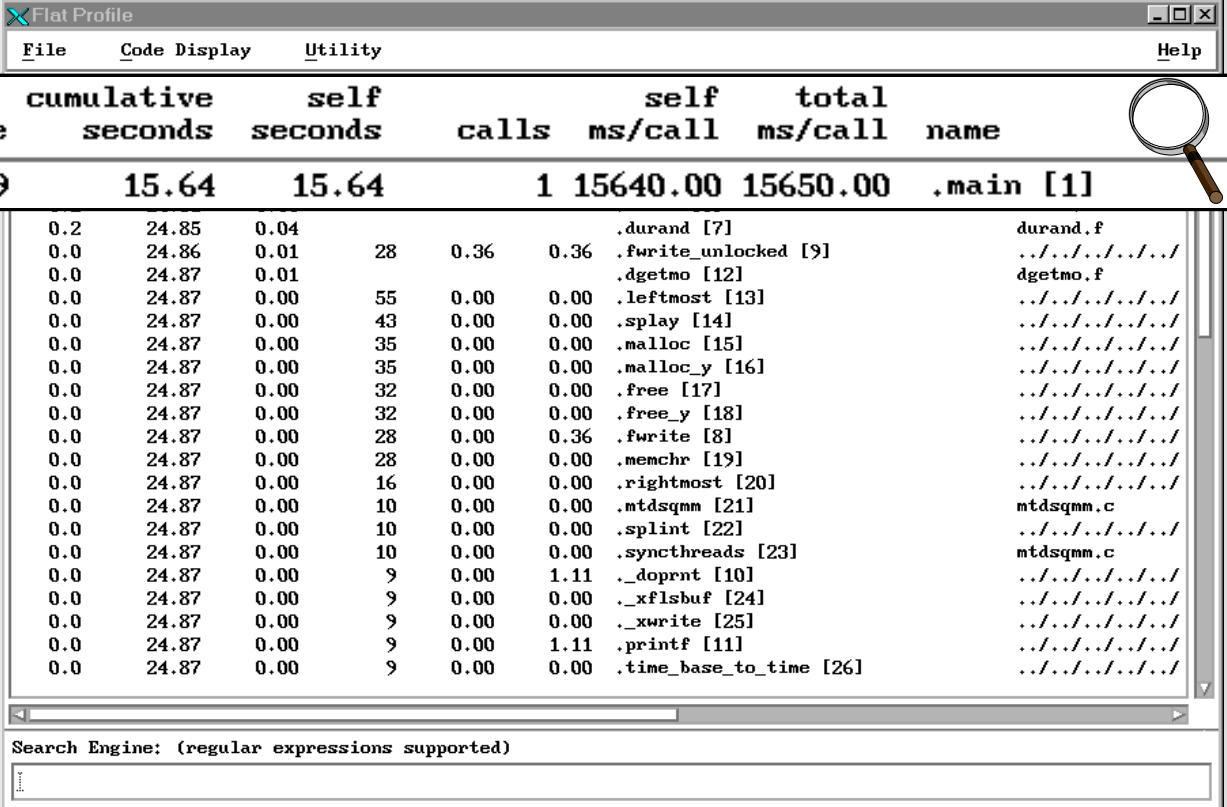
- Call Graph Profile

- Function Index

- Function Call

- Summary

- Library Statistics



The screenshot shows a window titled "Flat Profile" with a menu bar containing "File", "Code Display", "Utility", and "Help". The main content is a table with the following columns: "%time", "cumulative seconds", "self seconds", "calls", "self ms/call", "total ms/call", and "name". The table lists various functions and their associated metrics. A magnifying glass icon is visible on the right side of the table.

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
62.9	15.64	15.64	1	15640.00	15650.00	.main [1]
0.2	24.85	0.04				.durand [7] durand.f
0.0	24.86	0.01	28	0.36	0.36	.fwrite_unlocked [9] .....
0.0	24.87	0.01				.dgetmo [12] dgetmo.f
0.0	24.87	0.00	55	0.00	0.00	.leftmost [13] .....
0.0	24.87	0.00	43	0.00	0.00	.splay [14] .....
0.0	24.87	0.00	35	0.00	0.00	.malloc [15] .....
0.0	24.87	0.00	35	0.00	0.00	.malloc_y [16] .....
0.0	24.87	0.00	32	0.00	0.00	.free [17] .....
0.0	24.87	0.00	32	0.00	0.00	.free_y [18] .....
0.0	24.87	0.00	28	0.00	0.36	.fwrite [8] .....
0.0	24.87	0.00	28	0.00	0.00	.memchr [19] .....
0.0	24.87	0.00	16	0.00	0.00	.rightmost [20] .....
0.0	24.87	0.00	10	0.00	0.00	.mtdsqmm [21] mtdsqmm.c
0.0	24.87	0.00	10	0.00	0.00	.splint [22] .....
0.0	24.87	0.00	10	0.00	0.00	.syncthread [23] mtdsqmm.c
0.0	24.87	0.00	9	0.00	1.11	._doprnt [10] .....
0.0	24.87	0.00	9	0.00	0.00	._xflsbuf [24] .....
0.0	24.87	0.00	9	0.00	0.00	._xwrite [25] .....
0.0	24.87	0.00	9	0.00	1.11	._printf [11] .....
0.0	24.87	0.00	9	0.00	0.00	._time_base_to_time [26] .....

Search Engine: (regular expressions supported)

# Xprofiler: Source Code Window

- Source code window displays source code with time profile (in ticks=0.01 sec)
- Access
  - Select function in main display
  - → context menu
  - Select function in flat profile
  - → Code Display
  - → Show Source Code

line	no. ticks per line	source code
202		/*-----*/
203		/* use 2x-unrolling of the outer two loops */
204		/*-----*/
205	4	for (i=i0; i<i0+is-1; i+=2)
206		{
207	8	for (j=j0; j<j0+js-1; j+=2)
208		{
209	1	t11 = c[i*n+j];
210	5	t12 = c[i*n+j+1];
211	5	t21 = c[(i+1)*n+j];
212	19	t22 = c[(i+1)*n+(j+1)];
213		for (k=k0; k<k0+ks; k++)
217	229	t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
218	144	t22 = t22 + a[(i+1)*n+k]*bt[(j+1)*n+k];
219		}
220	7	c[i*n+j] = t11;
221		c[i*n+j+1] = t12;
222	3	c[(i+1)*n+j] = t21;
223	5	c[(i+1)*n+(j+1)] = t22;
224		}
225		for (j=j; j<j0+js; j++)
226		{
227		t11 = c[i*n+j];
228		t21 = c[(i+1)*n+j];
229		for (k=k0; k<k0+ks; k++)
230		{
231		t11 = t11 + a[i*n+k]*bt[j*n+k];
232		t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
233		}
234		c[i*n+j] = t11;
235		c[(i+1)*n+j] = t21;
236		}
237		}

Search Engine: (regular expressions supported)

# Xprofiler - Disassembler Code

Disassembler Code for .calc3 [3]

address	no. ticks per instr.	instruction	assembler code	source code
10002E18	81	FCC4287C	fnms 6, 4, 1, 5	
10002E1C	64	CCF70008	lfd 7, 0x8(23)	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E20	187	C90C0008	lfd 8, 0x8(12)	
10002E24	53	C9750008	lfd 11, 0x8(21)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E28	89	FD63582A	fa 11, 3, 11	
10002E2C	63	FD28387C	fnms 9, 8, 1, 7	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E30	4	DD5B0008	stfdu 10, 0x8(27)	U(I, J) = UNEW(I, J)
10002E34		C9540008	lfd 10, 0x8(20)	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E38	113	FCCA302A	fa 6, 10, 6	
10002E3C	27	C8760008	lfd 3, 0x8(22)	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E40	87	FD8012FA	fma 12, 0, 11, 2	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E44	35	DCB90008	stfdu 5, 0x8(25)	V(I, J) = VNEW(I, J)
10002E48	4	FC63482A	fa 3, 3, 9	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E4C	12	CD5A0008	lfd 10, 0x8(26)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E50	62	FCC021BA	fma 6, 0, 6, 4	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E54	36	C85B0008	lfd 2, 0x8(27)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E58	244	DCEC0008	stfdu 7, 0x8(12)	P(I, J) = PNEW(I, J)
10002E5C	28	FD0040FA	fma 8, 0, 3, 8	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E60		C8990008	lfd 4, 0x8(25)	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E64	316	DCD40008	stfdu 6, 0x8(20)	
10002E68	29	FC62507C	fnms 3, 2, 1, 10	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-

Search Engine: (regular expressions supported)



## Xprofiler: Tips and Hints

- Simplest when gmon.out.\*, executable, and source code are in one directory
  - Select “**Set File Search Path**” on “**File**” menu to set source directory when source, and executable are not in the same directory
  - Can use **-qfullpath** to encode the path of the source files into the binary
- By default, call tree in main display is “clustered”
  - Menu Filter → Uncluster Functions
  - Menu Filter → Hide All Library Calls
- Libraries must match across systems!
  - on measurement nodes
  - on workstation used for display!
- Must sample realistic problem (sampling rate is 1/100 sec)

# HPM: What Are Performance Counters

- Extra logic inserted in the processor to count specific events
- Updated at every cycle
- Strengths:
  - Non-intrusive
  - Accurate
  - Low overhead
- Weaknesses:
  - Specific for each processor
  - Access is not well documented
  - Lack of standard and documentation on what is counted

## HPM: Hardware Counters Examples

- Cycles
  - Instructions
  - Floating point instructions
  - Integer instructions
  - Load/stores
  - Cache misses
  - TLB misses
  - Branch taken / not taken
  - Branch mispredictions
- Useful derived metrics
    - ✓ IPC - instructions per cycle
    - ✓ Float point rate (Mflop/s)
    - ✓ Computation intensity
    - ✓ Instructions per load/store
    - ✓ Load/stores per cache miss
    - ✓ Cache hit rate
    - ✓ Loads per load miss
    - ✓ Stores per store miss
    - ✓ Loads per TLB miss
    - ✓ Branches mispredicted %

# Event Sets

- **4 sets (0-3); ~1000 events**
- **Information for**
  - Time
  - FPU
  - L3 memory
  - Processing Unit
  - Tree network
  - Torus network

# Functions

- **hpmInit( taskID, progName ) / f\_hpminit( taskID, progName )**
  - taskID is an integer value indicating the node ID.
  - progName is a string with the program name.
- **hpmStart( instID, label ) / f\_hpmstart( instID, label )**
  - instID is the instrumented section ID. It should be  $> 0$  and  $\leq 100$  ( can be overridden)
  - Label is a string containing a label, which is displayed by PeekPerf.
- **hpmStop( instID ) / f\_hpmstop( instID )**
  - For each call to hpmStart, there should be a corresponding call to hpmStop with matching instID
- **hpmTerminate( taskID ) / f\_hpmterminate( taskID )**
  - This function will generate the output. If the program exits without calling hpmTerminate, no performance information will be generated.

# LIBHPM

Go in the source code and instrument different sections independently

- Supports MPI (OpenMP, threads on other PowerPC platforms)
- Multiple instrumentation points
- Nested sections
- Supports Fortran, C, C++

- Declaration:
  - #include f\_hpm.h
- Use:
  - call `f_hpminit( 0, "prog" )`
  - call `f_hpmstart( 1, "work" )`
  - do
    - call `do_work()`
    - call `f_hpmstart( 22, "more work" )`
    - call `compute_meaning_of_life()`
    - call `f_hpmstop( 22 )`
  - end do
  - call `f_hpmstop( 1 )`
  - call `f_hpmterminate( 0 )`

Use MPI  
taskID with  
MPI programs

# HPM Data Visualization

File

swim\_omp

Label	ExcSec
Loop 300	4.572
Loop 200	4.203
Loop 100	3.071
Calc3	1.838
Calc2	1.013
MPI Calc1 start 1.003	
MPI Calc2 start 0.528	

hpmviz

File

swim\_omp

Label	ExcSec	IncSec	Count
Loop 300	4.572	4.572	2398
Loop 200	4.203	4.203	2400
Loop 100	3.071	3.071	2400
Calc3	1.838	6.813	2398
Calc2	1.013	5.632	2400
MPI Calc1 start 1.003	1.003	2400	
MPI Calc2 start 0.528	0.528	2400	

```

swim_omp.f  calc1.f  calc2.f  calc3.f
*  VOLD(N1,N2), POLD(N1,N2),
2  CU(N1,N2), CV(N1,N2),
*  Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
COMMON /CONS/ DT,TDT,DX,DY,A,ALPHA,ITMAX,MPIR
1      NP1,EL,PI,TPI,DI,DJ,PCF
integer ierr
integer omp_get_thread_num
                    
```

Metric Options:

- Count
- ExcSec
- IncSec
- PM\_FPU\_FDIV
- PM\_FPU\_FMA
- PM\_FPU0\_FIN
- PM\_FPU1\_FIN
- PM\_CYC
- PM\_FPU\_STF
- PM\_INST\_CMPL
- PM\_LSU\_LDF
- U time
- Use rate
- (M) LS
- MIPS
- HW FP/Cyc
- Instr/LS
- M Flips
- IpC
- Mflip/s
- WFlips
- Wflip/s
- FMA %
- Comp Int.

Metric Browser: Loop 300

Node	Thread	Count	ExcSec	IncSec	U time	Use rate	(M) LS	MIPS	HW FP/Cyc	Instr/LS	M Flips	IpC	Mflip/s	WFlips	Wflip/s	FMA %	Comp Int.
0	3	2398	4.539	4.539	3.923	86.425	590.056	291.855	0.116	2.245	589.86	0.26	129.947	589.86	129.947	80.012	1
0	0	2398	4.572	4.572	4.378	95.763	608.414	263.277	0.107	1.978	608.234	0.211	133.037	608.234	133.037	80.011	1
0	2	2398	4.549	4.549	4.366	95.979	590.019	255.241	0.104	1.968	589.838	0.205	129.663	589.838	129.663	80.011	1
0	1	2398	4.547	4.547	4.308	94.759	590.024	259.19	0.105	1.997	589.837	0.21	129.728	589.837	129.728	80.012	1
1	2	2398	4.534	4.534	4.398	96.999	590.044	253.123	0.103	1.945	589.856	0.201	130.088	589.856	130.088	80.012	1
1	1	2398	4.528	4.528	3.942	87.069	589.983	286.058	0.115	2.195	589.807	0.253	130.263	589.807	130.263	80.011	1
1	0	2398	4.547	4.547	3.766	82.828	608.434	308.065	0.124	2.302	608.244	0.286	133.762	608.244	133.762	80.011	1
1	3	2398	4.523	4.523	3.537	78.198	589.962	317.346	0.128	2.433	589.781	0.312	130.4	589.781	130.4	80.011	1
2	0	2398	4.538	4.538	3.777	83.218	608.448	312.01	0.124	2.327	608.262	0.288	134.029	608.262	134.029	80.012	1
2	2	2398	4.522	4.522	4.313	95.364	590.033	257.962	0.105	1.977	589.86	0.208	130.431	589.86	130.431	80.011	1
2	3	2398	4.52	4.52	4.307	95.285	589.985	258.863	0.105	1.983	589.806	0.209	130.492	589.806	130.492	80.011	1
2	1	2398	4.52	4.52	4.35	96.222	589.943	255.814	0.104	1.96	589.767	0.205	130.466	589.767	130.466	80.01	1
3	3	2398	4.487	4.487	4.193	93.453	571.551	259.827	0.105	2.04	571.374	0.214	127.352	571.374	127.352	80.011	1
3	1	2398	4.502	4.502	4.365	96.953	589.937	254.196	0.104	1.94	589.763	0.202	131.003	589.763	131.003	80.01	1
3	2	2398	4.483	4.483	4.139	92.33	571.556	263.864	0.106	2.07	571.38	0.22	127.445	571.38	127.445	80.011	1
3	0	2398	4.506	4.506	3.927	87.154	590.044	290.852	0.116	2.221	589.856	0.257	130.901	589.856	130.901	80.012	1

```

300 CONTINUE
call f_hpmstop( 30+omp_get_thread_num())
                    
```

## HPM component

### Plain Text File Output

libhpm v3.2.1 (IHPCT v2.2.0) summary

##### Resource Usage Statistics #####

```
Total amount of time in user mode      :
6.732208 seconds
Total amount of time in system mode    :
5.174914 seconds
Maximum resident set size              : 12184
Kbytes
Average shared memory use in text segment :
17712 Kbytes*sec
Average unshared memory use in data segment :
61598 Kbytes*sec
Number of page faults without I/O activity : 13829
Number of page faults with I/O activity   : 0
Number of times process was swapped out   : 0
Number of times file system performed INPUT : 0
Number of times file system performed OUTPUT : 0
Number of IPC messages sent               : 0
Number of IPC messages received           : 0
Number of signals delivered               : 0
Number of voluntary context switches      : 233
Number of involuntary context switches     : 684
```

##### End of Resource Statistics #####

Instrumented section: 7 - Label: find\_my\_seed

process: 274706, thread: 1  
file: is.c, lines: 412 <--> 441

Context is process context.

No parent for instrumented section.

Inclusive timings and counter values:

```
Execution time (wall clock time) : 0.000290516763925552 seconds
Initialization time (wall clock time): 1.15633010864258e-05 seconds
Overhead time (wall clock time)   : 1.44504010677338e-05 seconds
```

```
PM_FPU_1FLOP (FPU executed one flop instruction) :      1259
PM_FPU_FMA (FPU executed multiply-add instruction) :       247
PM_ST_REF_L1 (L1 D cache store references)       :      20933
PM_LD_REF_L1 (L1 D cache load references)        :      38672
PM_INST_CMPL (Instructions completed)            :     157151
PM_RUN_CYC (Run cycles)                          :     254222
```

```
Utilization rate           :      52.895 %
MIPS                       :     540.936
Instructions per load/store :       2.637
Algebraic floating point operations :      0.002 M
Algebraic flop rate (flops / WCT) :      6.034 Mflop/s
Algebraic flops / user time :     11.408 Mflop/s
FMA percentage             :     28.180 %
% of peak performance      :      0.172 %
```

## PomProf - “Standard” OpenMP Monitoring API?

- **Problem:**

- OpenMP (unlike MPI) does not define standard monitoring interface (at SC06 they accepted a proposal from SUN and others)
- OpenMP is defined mainly by directives/pragmas

- **Solution:**

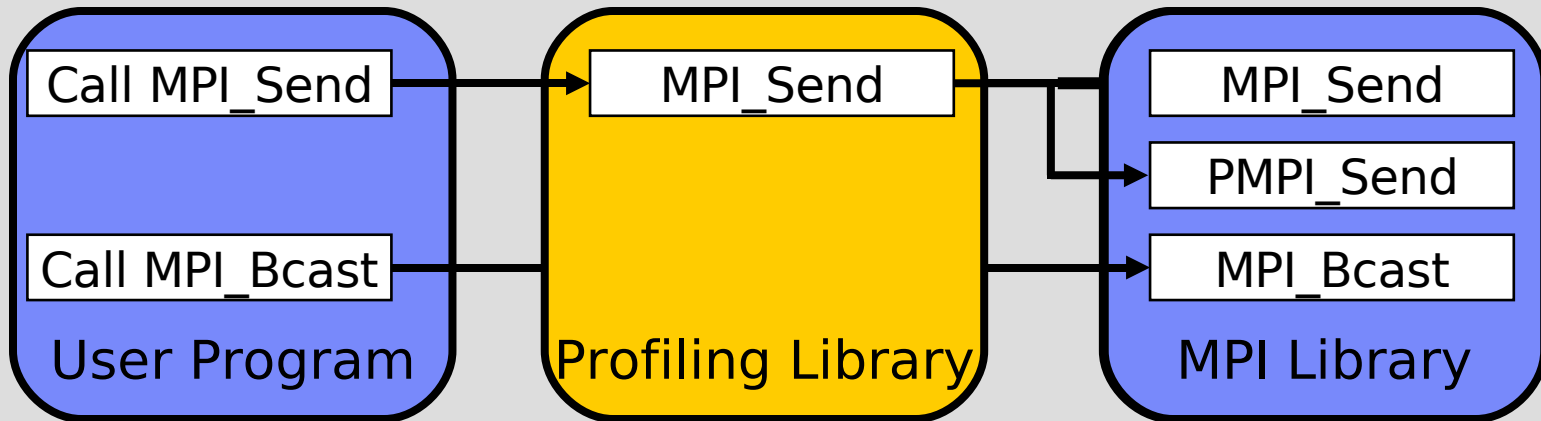
- **POMP**: OpenMP Monitoring Interface
- Joint Development
  - Forschungszentrum Jülich
  - University of Oregon
- Presented at EWOMP’01, LACSI’01 and SC’01
  - “The Journal of Supercomputing”, 23, Aug. 2002.





## Profiling of OpenMP Applications: POMP

- Portable cross-platform/cross-language API to simplify the design and implementation of OpenMP tools
- POMP was motivated by the MPI profiling interface (PMPI)
  - PMPI allows selective replacement of MPI routines at link time
  - Used by most MPI performance tools (including MPI Profiler/Tracer)



# POMP Proposal

- **Three groups of events**
  - **OpenMP constructs and directives/pragmas**
    - Enter/Exit around each OpenMP construct
      - Begin/End around associated body
    - Special case for parallel loops:
      - ChunkBegin/End, IterBegin/End, or IterEvent instead of Begin/End
    - “Single” events for small constructs like atomic or flush
  - **OpenMP API calls**
    - Enter/Exit events around `omp_set_*_lock()` functions
    - “single” events for all API functions
  - **User functions and regions**
- **Allows application programmers to specify and control amount of instrumentation**

## Example: POMP Instrumentation

```
1:  int main() {
2:      int id;
***  POMP_Init();
3:
***  { POMP_handle_t pomp_hd1 = 0;
***      int32 pomp_tid = omp_get_thread_num();
***      POMP_Parallel_enter(&pomp_hd1, pomp_tid, -1, 1,
***          "49*type=pregion*file=demo.c*slines=4,4*elines=8,8**");
4:      #pragma omp parallel private(id)
5:      {
***          int32 pomp_tid = omp_get_thread_num();
***          POMP_Parallel_begin(pomp_hd1, pomp_tid);
6:          id = omp_get_thread_num();
7:          printf("hello from %d\n", id);
***          POMP_Parallel_end(pomp_hd1, pomp_tid);
8:      }
***          POMP_Parallel_exit(pomp_hd1, pomp_tid);
***      }
***      POMP_Finalize();
9:  }
```

# POMP Profiler (PompProf)

**Generates a detailed profile describing overheads and time spent by each thread in three key regions of the parallel application:**

- Parallel regions
- OpenMP loops inside a parallel region
- User defined functions
- 
- **Profile data is presented in the form of an XML file that can be visualized with PeekPerf**

peekperf
\_ □ ×

File Tools

main.f

Label	Count	Excl. Time	Incl. Time	%Total Overhead	%Imbalance	A
pregion_324	1	54.4638	128.508	4.1e-05	6.3e-05	12
loop_1125	10	13.3219	13.3219	0.005431	88.4638	21
loop_852	10	12.854	12.854	0.005178	52.5563	17
loop_549	10	12.3907	12.3907	0.005676	98.8048	21
loop_1685	10	12.3036	12.3036	0.005682	62.1319	16
loop_1408	10	11.8975	11.8975	0.005578	96.0548	19
loop_1934	10	10.1843	10.1843	0.006926	41.4999	13
loop_790	1	0.522151	0.522151	0.013839	37.576	0.4
loop_1668	1	0.485633	0.485633	0.00896	36.8549	0.4
loop_1623	1	0.039318	0.039318	0.596515	2.79627	0.0
loop_1379	1	0.031769	0.031769	0.198128	13.846	0.0
func_rddparam_174	1	0.01605	0.01605	0	0	0
loop_855	1	0.013306	0.013306	0.224878	342.237	0.0
func_main_155	1	0.000771	128.525	0	0	0
func_deltat_550	1	2.2e-05	2.2e-05	0	0	0
func_layout_296	1	6e-06	6e-06	0	0	0
func_changedir_1841	1	3e-06	3e-06	0	0	0

Metric Browser: loop\_1125
\_ □ ×

Close
Metric Options ▾
Precision ▾

Task	thread	Time in Master	TT: Thread Time	CT: Computation Time	%Imbalance	TO = TT - CT	%TO (Barrier)	%TO (RTL)
0	0	13.3219	13.3219	13.3211	0	0.000723	0.000275	0.005156
0	1	0	15.8615	15.861	19.0664	0.000504	0.000108	0.003679
0	2	0	21.0295	21.029	57.8616	0.000514	0.000103	0.003753
0	3	0	24.4342	24.4338	83.4208	0.000416	4e-06	0.003119
0	4	0	24.1806	24.1802	81.5175	0.000423	0	0.003173
0	5	0	25.0957	25.0952	88.3864	0.00047	1.4e-05	0.00351
0	6	0	25.1062	25.1055	88.4638	0.00062	0.000157	0.004495
0	7	0	23.9859	23.9854	80.0548	0.000556	0.000112	0.004063

main.f
runhyd3.f

```

&      msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
endif

c$omp do schedule(static)
do ip=1,nchunk

    if ((ithread.eq.1).and.(ip.ge.icomm_point(1,4)).and.
&      (irec1br.eq.0) .and.      (iflag.eq.0)) then
        irec1br = 1

        call bdrys1br(dddo, nz, nx, ny, 5,
&      msg_zm, msg_zp, msg_xm, msg_xp, msg_ym, msg_yp,
&      msg_mnzbdy, msg_mnxbdy, msg_mnybdy,
&      msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
        call bdry2o1s(dddo, nz, nx, ny, 5,
&      msg_zm, msg_zp, msg_xm, msg_xp, msg_ym, msg_yp,
&      msg_mnzbdy, msg_mnxbdy, msg_mnybdy,
&      msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
        call bdry3o1r(dddo, nz, nx, ny, 5,
&      msg_zm, msg_zp, msg_xm, msg_xp, msg_ym, msg_yp,
&      msg_mnzbdy, msg_mnxbdy, msg_mnybdy,
&      msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
    
```

# Modular I/O (MIO)

- **Addresses the need of application-level optimization for I/O.**
- **Analyze and tune I/O at the application level**
  - For example, when an application exhibits the I/O pattern of sequential reading of large files
  - MIO
    - Detects the behavior
    - Invokes its asynchronous prefetching module to prefetch user data.
- **Source code traceback**
- **Future capability for dynamic I/O instrumentation**

# Modular I/O Performance Tool (MIO)

- **I/O Analysis**
  - Trace module
  - Summary of File I/O Activity + Binary Events File
  - Low CPU overhead
- **I/O Performance Enhancement Library**
  - Prefetch module (optimizes asynchronous prefetch and write-behind)
  - System Buffer Bypass capability
  - User controlled pages (size and number)





# Eclipse Integration - Instrumentation

The screenshot shows the Eclipse IDE interface for instrumenting a Fortran application. The main editor window displays the source code for `mod_xc.F.f`, with the following code snippet highlighted:

```

c
c   remote sums.
c
do mp= mpe_1(nproc)+1, ipr
  l = iisum(mp,nproc)*jj
  if (l.gt.0) then
    call MPI_RECV(sum8t,l,mpi_real8,
&                idproc(mp,nproc), 9900,
&                mpi_comm_world, mpiestat, mpierr)
  do j=1,jj
    do l= 1,iisum(mp,nproc)
      + sum8t(l + (j-1)*iisum(mp,
&                2.2)') 'xcsum: np, j, sum = ',
      mp, j, sum8j(j)
    enddo
  enddo
enddo
if (l.gt.0) then
  call mpi_send(sum8t,l,mpi_real8,

```

An **Instrumentation Completed** dialog box is displayed in the center, containing the following text:

```

The instrumented binary is successfully generated!
The instrumented application can be found at
/home/hfwen/tmp/hycom.mpi4.inst.

```

The dialog box has an **OK** button.

The left sidebar shows a tree view of the project structure, with the following items checked:

- mod\_xc.F.f
  - \_\_mod\_xc\_NMOD\_xcgetc
    - mpi\_bcast\_456
  - \_\_mod\_xc\_NMOD\_xcmaxr\_1
    - mpi\_allreduce\_839
  - \_\_mod\_xc\_NMOD\_xcsumj
    - mpi\_recv\_1707
    - mpi\_recv\_1737
    - mpi\_send\_1745
    - mpi\_send\_1722
  - \_\_mod\_xc\_NMOD\_xcstop
    - mpi\_finalize\_1464
    - mpi\_finalize\_1464
  - \_\_mod\_xc\_NMOD\_xcaput
    - mpi\_barrier\_242
    - mpi\_waitall\_273
    - mpi\_isend\_284
    - mpi\_waitall\_288
    - mpi\_recv\_290
    - mpi\_isend\_267
    - mpi\_recv\_275
  - IPRA,\$\_\_mod\_xc\_NMOD\_xctilr
    - mpi\_send\_init\_2004
    - mpi\_send\_init\_2007
    - mpi\_recv\_init\_2010

The bottom status bar shows **Writable**, **Insert**, and **1708 : 1**.

# Performance Data Visualization

The screenshot displays the Eclipse IDE interface for performance analysis. The top toolbar shows standard IDE functions. The main workspace is divided into several panes:

- Source Editor:** Shows the code for `mod_xc.F.f`, including an `mpi_allreduce` call and a loop for calculating `a(is0+i) = c(i)`.
- Performance Data - Tree View:** A hierarchical tree showing performance events. The `MPI_Allreduce_922` event is highlighted with a count of 465.
- MPI\_Allreduce\_922 Metric Browser:** A table showing metrics for rank 0 across 4 threads.

Task	Thread	Count	WallClock	Transferred Bytes
0		465	0.089211	8872.000000
1		465	0.067923	8872.000000
2		465	0.234179	8872.000000
3		465	0.019482	8872.000000

# MPI Trace Visualization

The screenshot shows the Eclipse IDE interface for HPCT. The main editor displays the following Fortran code:

```

b(i) = a(is0+i)
enddo
c
  call mpi_allreduce(b,c,nn,mpi_real8,mpi_min,
&
  mpi_comm_world,mpierr)
do i= 1,nn
  a(is0+i) = c(i)
enddo
enddo
c
if (nxc.eq.10) then

```

The Performance Data - Tree View window shows a detailed tree of MPI operations for process `mod_xc.F.f`. The MPI Trace View window displays a Gantt-style visualization of MPI events for 4 processes (rank 0-3) over time. The legend on the right includes the following checked items:

- MPI\_Comm\_rank
- MPI\_Comm\_size
- MPI\_Bcast
- MPI\_Barrier
- MPI\_Isend
- MPI\_Waitall
- MPI\_Allreduce
- MPI\_Send\_init
- MPI\_Recv\_init
- MPI\_Startall
- MPI\_Recv
- MPI\_Reduce
- MPI\_Send

# Summary Remarks

- The IBM HPC Toolkit provides an **integrated framework** for performance analysis
- Support **iterative analysis** and **automation** of the performance tuning process
- The standardized software layers make it easy to **plug in** new performance analysis tools
- Operates on the **binary** and yet provide reports in terms of source-level symbols
- Provides **multiple layers** that the user can exploit (from low-level instrumentations to high-level performance analysis)
- Full source code **traceback** capability
- **Dynamically** activate/deactivate data collection and change what information to collect
- IBM Redbook: IBM System Blue Gene Solution: High Performance Computing Toolkit for Blue Gene/P