The IBM High Performance Computing Toolkit on BlueGene/L

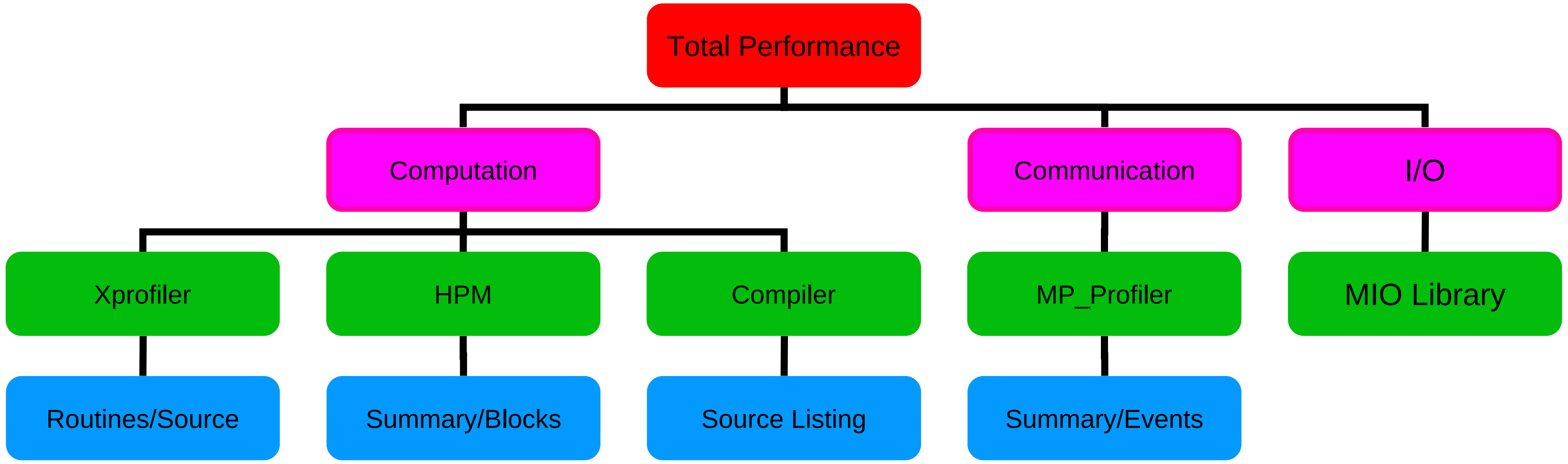
Kirk E Jordan, Ph.D.
Strategic Growth Business/Deep Computing
Systems & Technology Group
kjordan@us.ibm.com

Collaborators:
I-Hsin Chung - ACTC
Bob Walkup – Physical Sciences

Outline

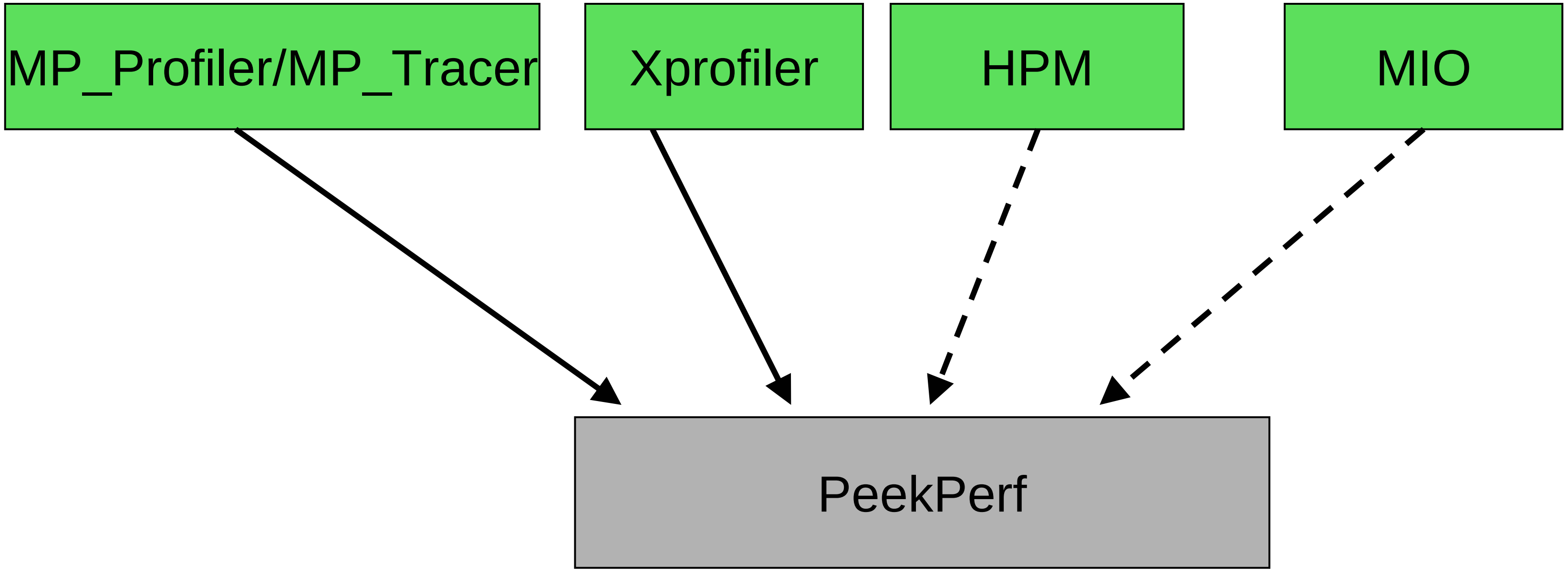
- Performance Decision tree
- IBM High Performance Computing Toolkit
  - MPI performance: MP_Profiler
  - CPU performance: Xprofiler, HPM
  - Modular I/O: MIO
  - Visualization and analysis: PeekPerf
- Challenges
- Future Work

Performance Decision Tree



IBM High Performance Computing Toolkit on BG/L

- MPI performance: MP_Profiler, MP_Tracer
- CPU performance: Xprofiler, HPM
- Visualization and analysis: PeekPerf
- Modular I/O: MIO



Message-Passing Performance:

- MP_Profiler Library
  - Captures “summary” data for MPI calls
  - Source code traceback
  - User **MUST** call MPI_Finalize() in order to get output files.
  - No changes to source code
    - MUST compile with –g to obtain source line number information
- MP_Tracer Library
  - Captures “timestamped” data for MPI calls
  - Source traceback

Compiling and Linking Example

```
BGL=/bgl/BlueLight/ppcfloor
CC=$(BGL)/blrts-gnu/powerpc-bgl-blrts-gnu/bin/gcc
CFLAG= -I $(BGL)/bglsys/include
MPI_LIB= -L $(BGL)/bglsys/lib -lmpich.rts -lmsglayer.rts -lrts.rts -ldevices.rts
TRACE_LIB= -L $(MP_PROFILER) -lmpitrace.rts
BINUTILS_LIB= -L $(BINUTILS) -lbfd -liberty

target: source.c
    $(CC) –o $@ $< $(CFLAG) $(TRACE_LIB) $(MPI_LIB) $(BINUTIL_LIB)
```

$(TRACE_LIB) has to precede $(MPI_LIB)

MP_Profiler Summary Output

MPI Routine	#calls	avg. bytes	time(sec)
MPI_Comm_size	3	0.0	0.000
MPI_Comm_rank	12994	0.0	0.016
MPI_Send	19575	11166.9	13.490
MPI_Isend	910791	5804.2	9.216
MPI_Recv	138173	2767.9	73.835
MPI_Irecv	784936	15891.6	2.407
MPI_Sendrecv	894809	352.0	88.705
MPI_Wait	1537375	0.0	288.049
MPI_Waitall	44042	0.0	25.312
MPI_Bcast	464	41936.8	3.272
MPI_Barrier	1312	0.0	34.206
MPI_Gather	68	16399.1	2.680
MPI_Scatter	6	17237.3	0.532

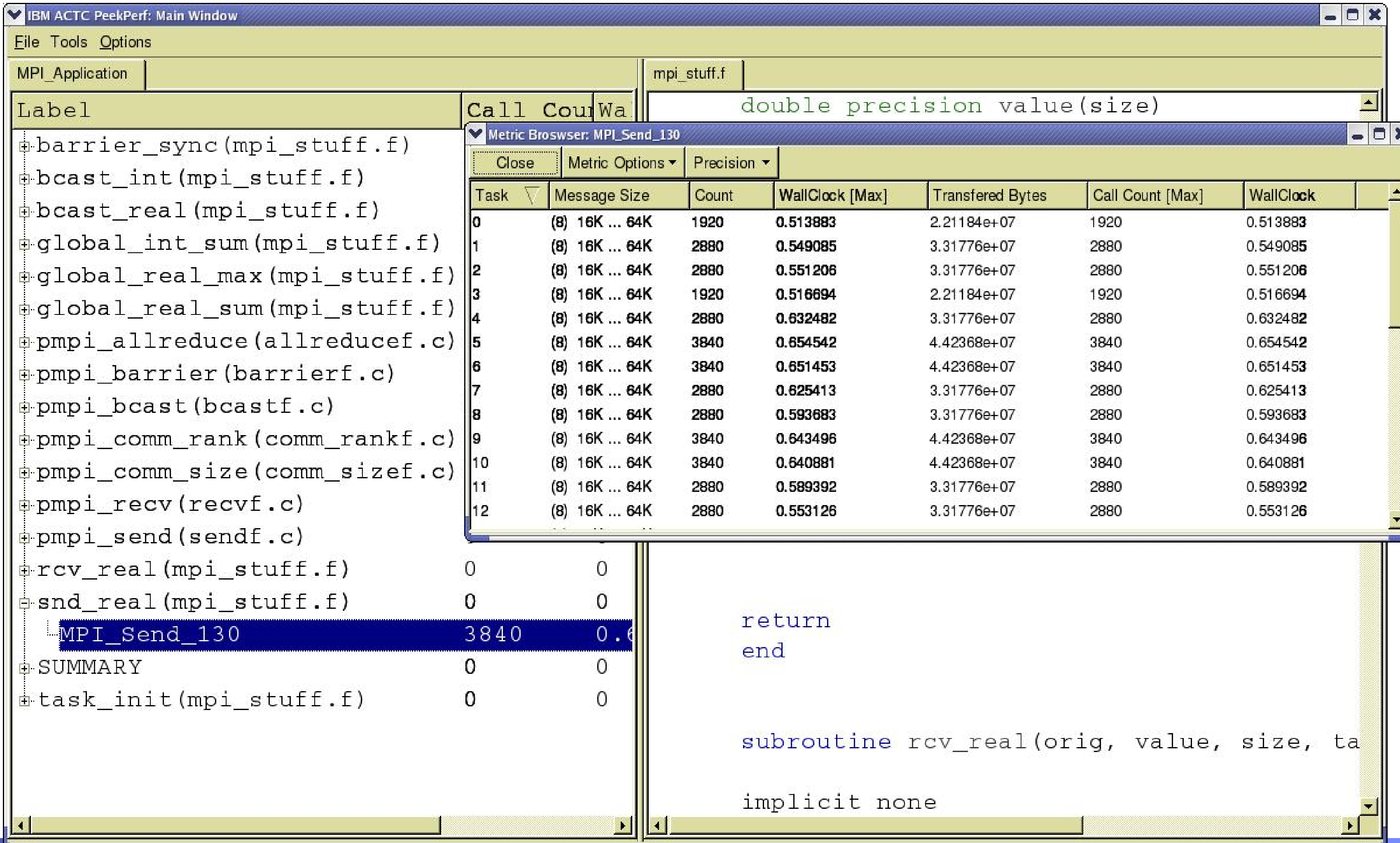
```
total communication time = 770.424 seconds.
total elapsed time       = 1168.662 seconds.
user cpu time            = 1160.960 seconds.
system time              = 0.620 seconds.
maximum memory size      = 68364 KBytes.

To check load balance :  grep "total comm" mpi_profile.*
```

MP_Profiler Sample Call Graph Output

```
communication time = 143.940 sec, parent = gwwrloc
   MPI Routine           #calls        time(sec)
   MPI_Barrier             2311          143.734
   MPI_Gatherv             2311            0.206
communication time = 137.823 sec, parent = f2drecv
   MPI Routine           #calls        time(sec)
   MPI_Recv               91959          137.823
communication time = 108.960 sec, parent = puttsf
   MPI Routine           #calls        time(sec)
   MPI_Barrier            23607          106.821
   MPI_Gatherv            23607            2.139
communication time = 94.435 sec, parent = fft_tri_recv
   MPI Routine           #calls        time(sec)
   MPI_Recv                6378           94.435
communication time = 83.836 sec, parent = fft2drecv
   MPI Routine           #calls        time(sec)
   MPI_Recv               93003           83.836
```

MP_Profiler Output with Peekperf



Measuring Communication Performance

The MPI standard provides alternate entry points for timing:

```
MPI_Send(…)  {
    Start_timer();
    PMPI_Send(…);
    Stop_timer();
    Log_the_event();
}
```

Implement a scalable approach that can support >10**5 MPI tasks.

Two modes: summary and event-logging.

MP_Profiler : Summary Mode

- Low overhead : <0.1 microsecond per call
- Low data volume : save just a few small text files
- Make use of the parallelism to do data collection and analysis
  - Save details for MPI ranks with minimum, maximum, and median communication times.
  - Collect total communication time for all ranks in one text file.
  - Collect data for all hardware counter events in one shot.
  - Automatically filter other profile data from gprof etc.

MPI Timing Summary

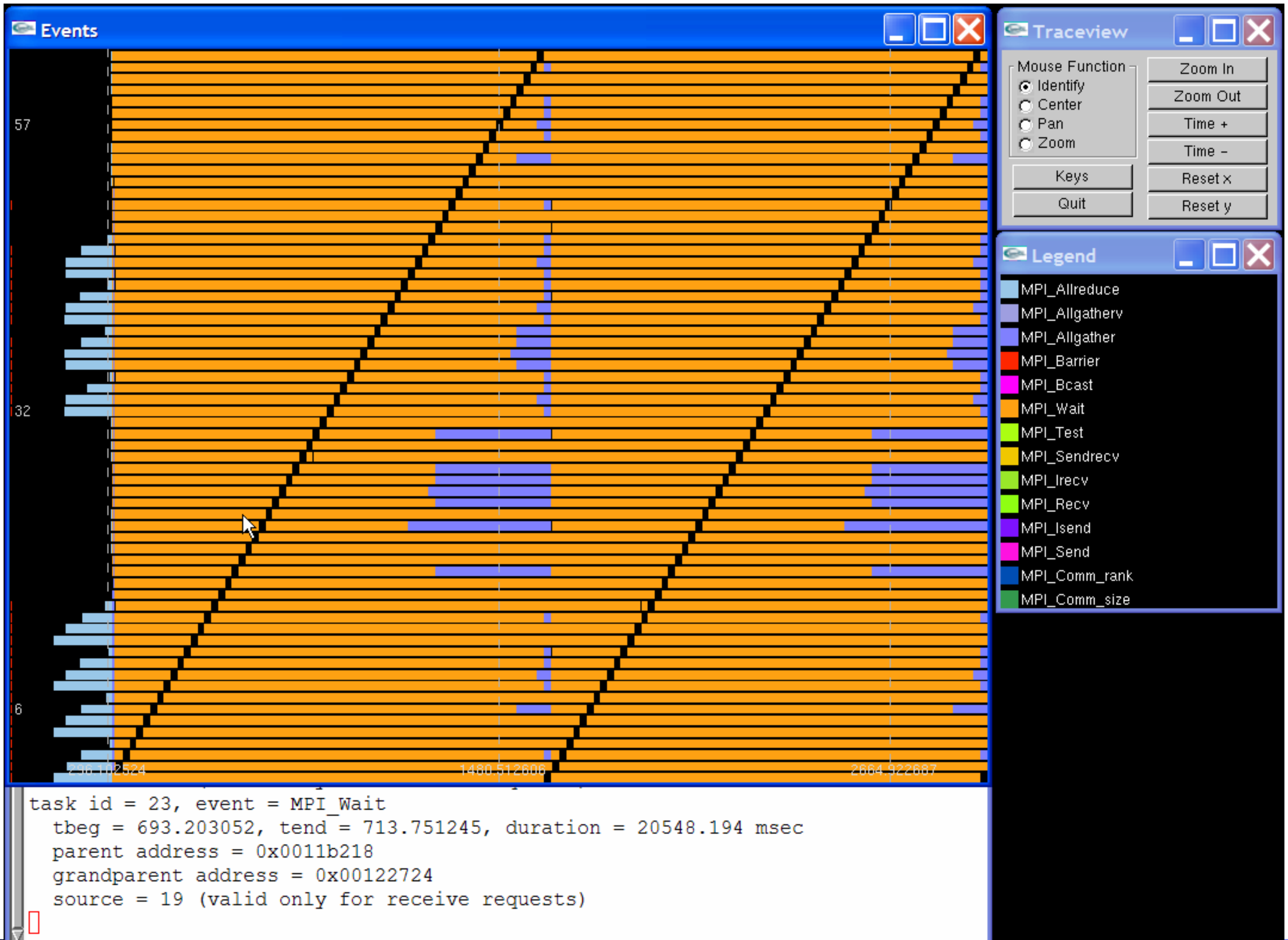
```
MPI Rank 0 out of 1024.
-----------------------------------------------------------------
elapsed time from clock-cycles using freq = 700.0 MHz.
-----------------------------------------------------------------
total communication time = 4.943 seconds.
total elapsed time       = 43.113 seconds.
top of the heap address  = 158.305 MBytes.

-----------------------------------------------------------------
MPI Routine                     #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                        2            0.0          0.000
MPI_Comm_rank                        2            0.0          0.000
MPI_Isend                          171       765163.0          0.001
MPI_Irecv                          174       757120.0          0.000
MPI_Wait                           340            0.0          1.912
MPI_Barrier                         11            0.0          0.038
MPI_Allreduce                       11           16.7          2.992
```

MP_Profiler : Event Logging

- Visualize the time-sequence of MPI events.
- Preserve the connection between MPI events and source code.
- Use a lightweight fast viewer based on OpenGL.
- Keep the data volume manageable.
- Limit event logging to a subset of MPI ranks, and/or to a subset of the overall simulation.

MPI Events for Enzo - Original



Enzo : Original Code

```
For each pair of regions
    conditionally call MPI_Test(…) to check for outstanding send requests
    if regions overlap
        call MPI_Isend(send_request,…) to "send" data

For each pair of regions
    if regions overlap
        call MPI_Irecv(recv_request) to receive boundary data
        call MPI_Wait(recv_request,…)
```

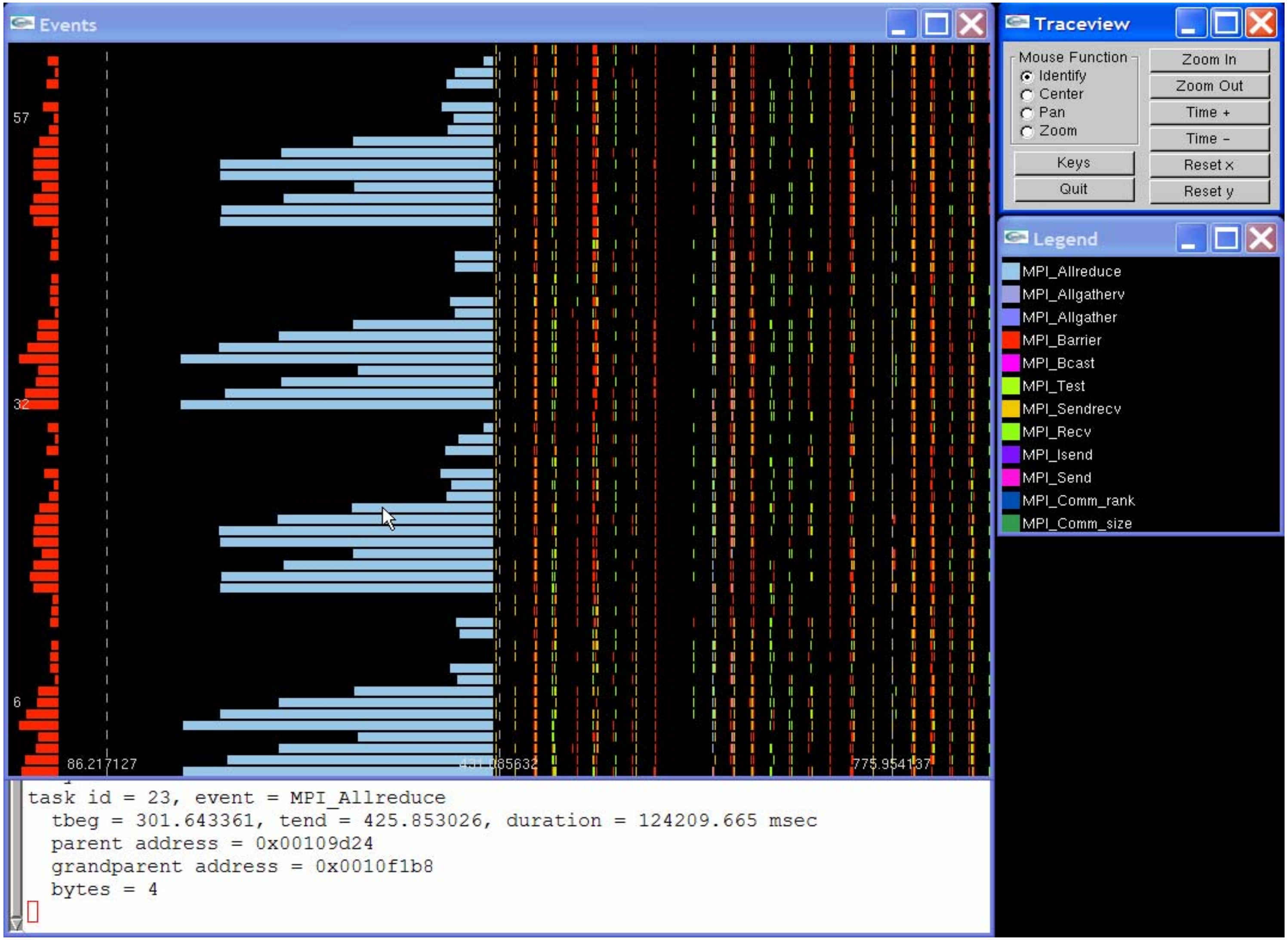
Enzo : Modified Code

```
For each pair of regions
      conditionally call MPI_Test(…) to check for outstanding send requests
      if regions overlap
            call MPI_Isend(send_request,…) to "send" data

call MPI_Barrier(…)

For each pair of regions
      if regions overlap
            call MPI_Irecv(recv_request) to receive boundary data
            call MPI_Wait(recv_request,…)
```

MPI Events for Enzo - Tuned



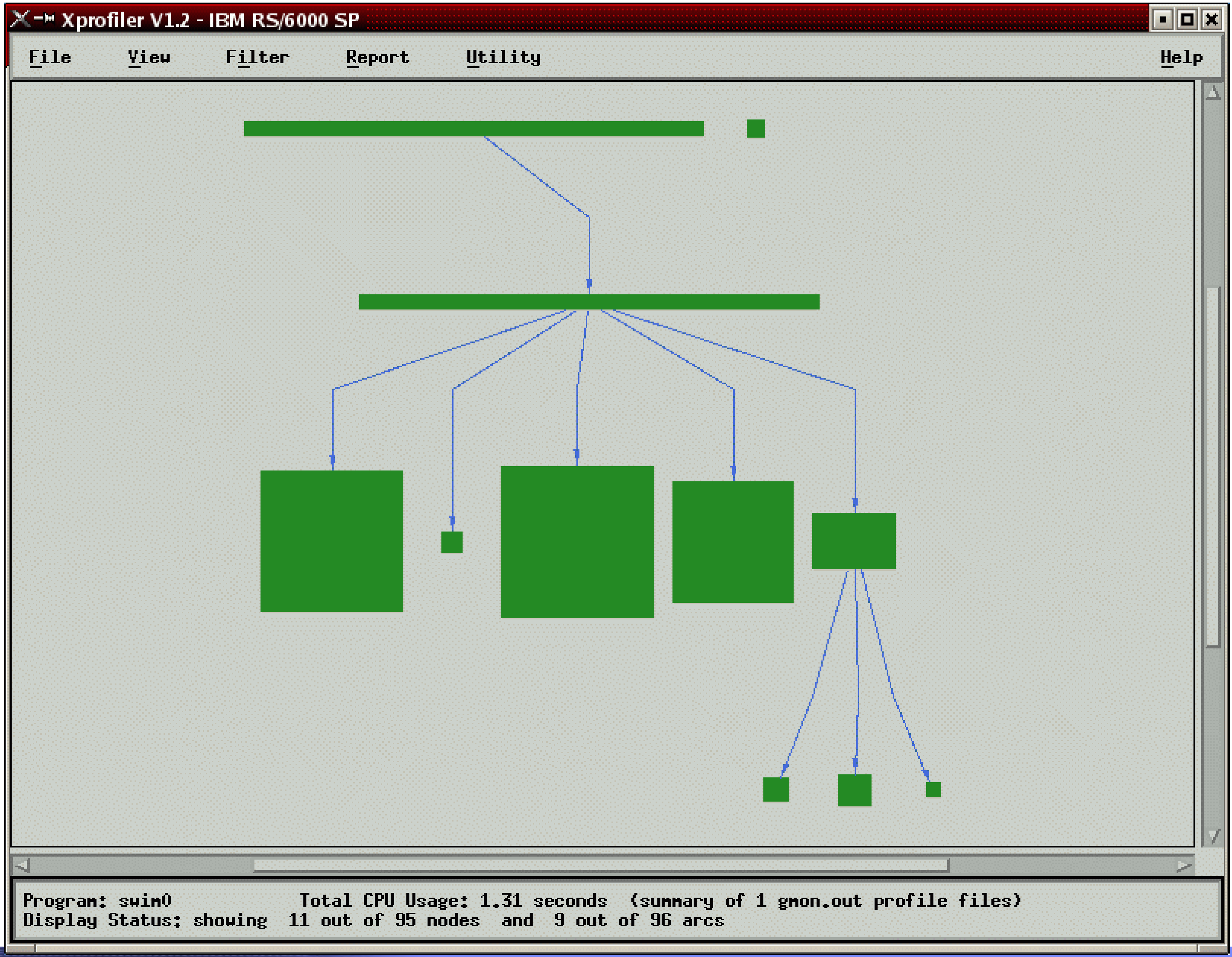
Xprofiler

- 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

Running Xprofiler

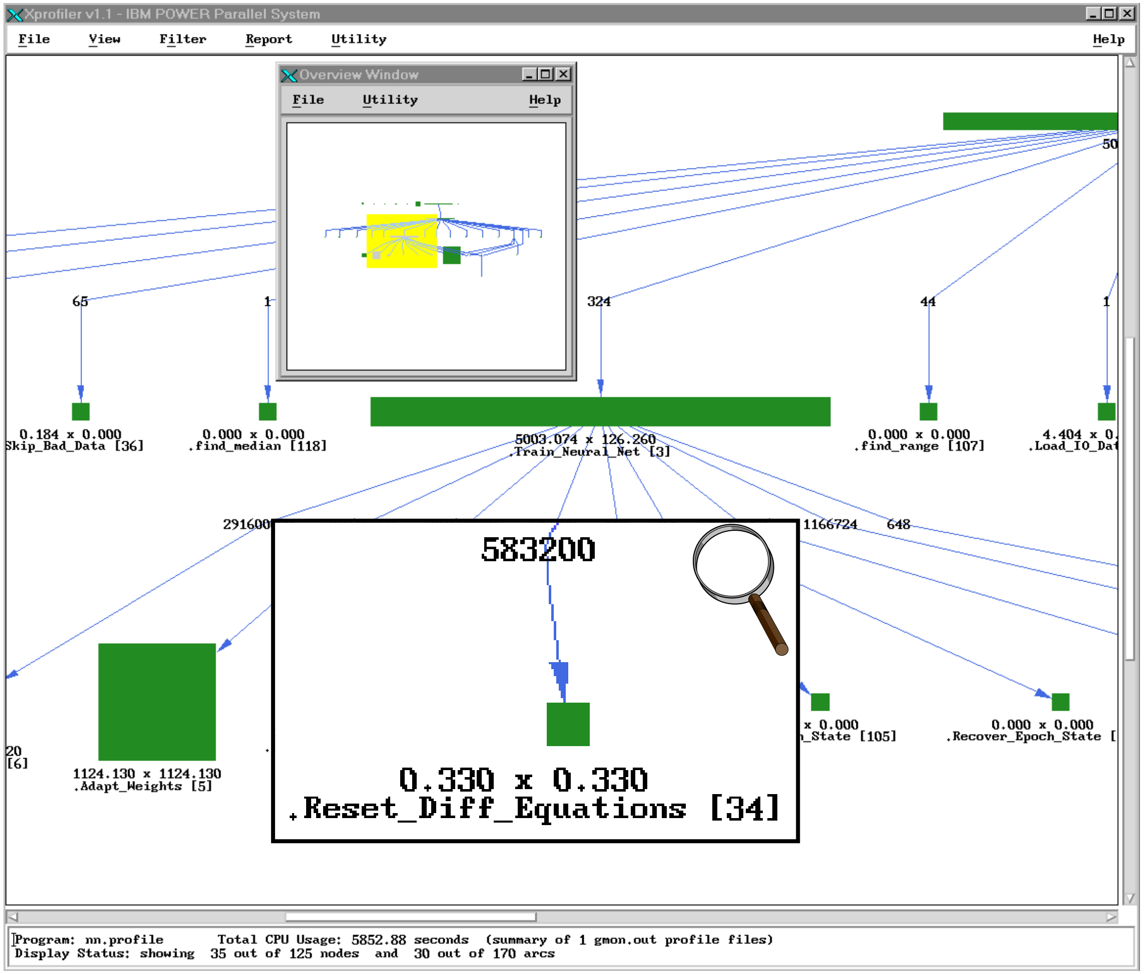
- Compile the program with -pg
- Run the program
- gmon.out file is generated (MPI applications generate gmon.out.1, …, gmon.out.n)
- Run Xprofiler

Xprofiler - Application View



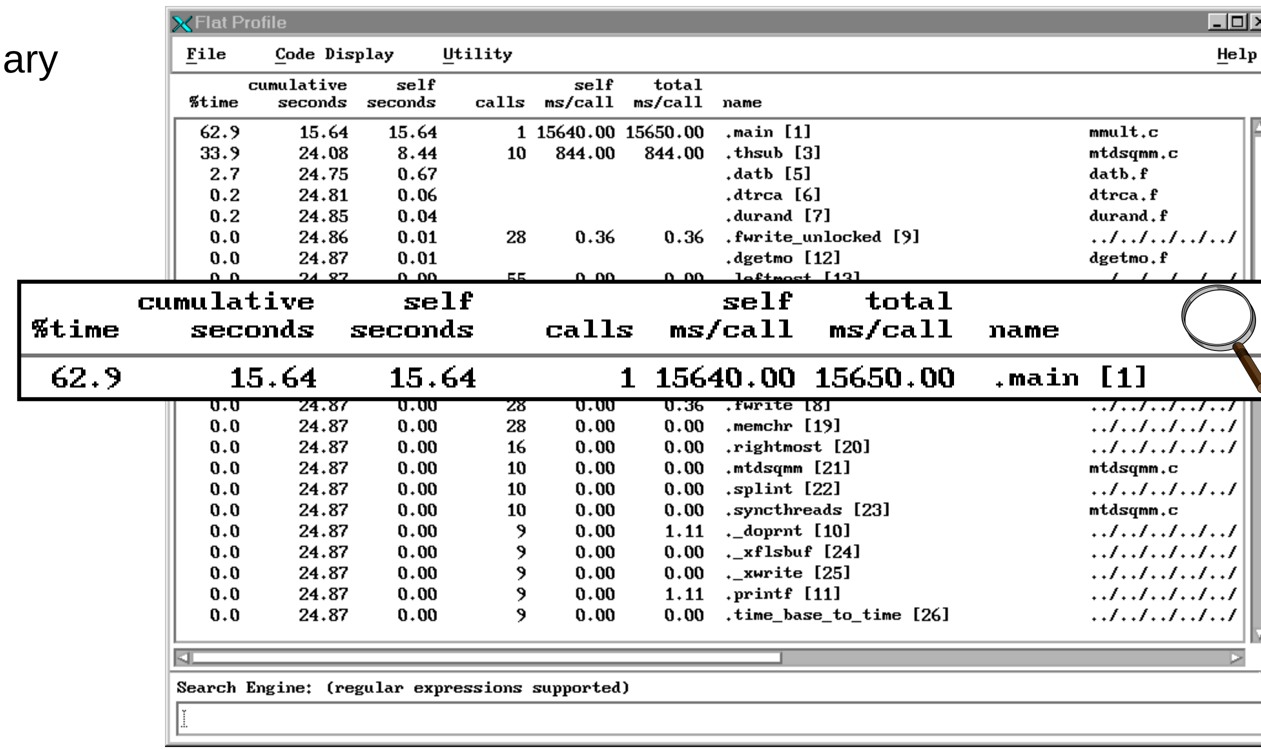
Xprofiler: Main Display

- 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: Flat Profile

- Menu **Report** provides usual gprof reports plus some extra ones
  - Flat Profile
  - Call Graph Profile
  - Function Index
  - Function Call Summary
  - Library Statistics

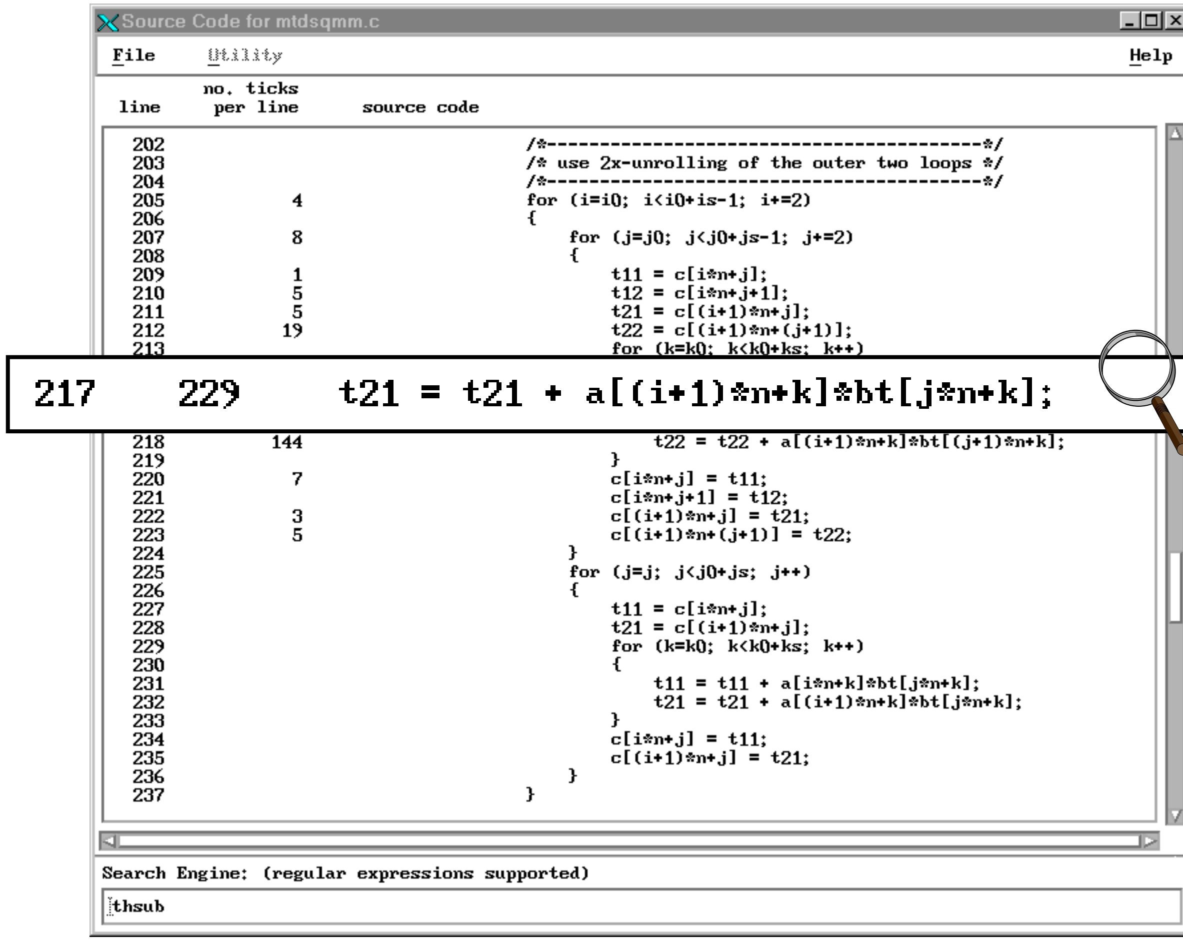


%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name	
62.9	15.64	15.64	1	15640.00	15650.00	.main [1]	mmult.c
33.9	24.08	8.44	10	844.00	844.00	.thsub [3]	mtdsqmm.c
2.7	24.75	0.67				.datb [5]	datb.f
0.2	24.81	0.06				.dtrca [6]	dtrca.f
0.2	24.85	0.04				.durand [7]	durand.f
0.0	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	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	.syncthreads [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]	../../../../../

%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]

Xprofiler: Source Code Window

- Source code window displays source code with time profile (in ticks=.01 sec)
- Access
  - Select function in main display
  - → context menu
  - Select function in flat profile
  - → Code Display
  - → Show Source Code



Xprofiler - Disassembler Code

Disassembler Code for .calc3 [3]

File Help

address	no. ticks per instr.	instruction	assembler code		source code
10002E18	81	FCC4287C	fnms	6,4,1,5	
10002E1C	64	CCF70008	lfdu	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	lfdu	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)

Hardware Performance Monitor (HPM)

- Provides comprehensive reports of events that are critical to performance on IBM systems.
- Gather critical hardware performance metrics, e.g.
  - Number of misses on all cache levels
  - Number of floating point instructions executed
  - Number of instruction loads that cause TLB misses
- Helps to identify and eliminate performance bottlenecks.

LIBHPM

- Instrumentation library
- Provides performance information for instrumented program sections
- Supports multiple (nested) instrumentation sections
- Multiple sections may have the same ID
- Run-time performance information collection
- Based on bgl_perfctr layer – can be eliminated in BG/P

Event Sets

- 16 sets (0-15); 328 events
- Information for
  - Time
  - FPU (0,1)
  - L3 memory
  - Processing Unit (0,1)
  - Tree network
  - Torus network
- For detailed names and descriptions: event_sets.txt

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 <= 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.

Functions (continued)

- hpmGetTimeAndCounters(numCounters, time, values) / f_GetTimeAndCounters (numCounters, time, values)
  - returns the time in seconds and counts since the call to hpmInit.
  - numCounters: integer indicating the number of counters to be accessed.
  - time: double precision float
  - values: “long long” vector of size “numCounters”.
- hpmGetCounters(values) / f_hpmGetCounters (values)
  - Similar to hpmGetTimeAndCounters
  - only returns the total counts since the call to hpmInit

Example of Instrumented Code

- C / C++

declaration:

```
    #include "libhpm.h"
  use:
    hpmInit( taskID, "my program" );
    hpmStart( 1, "outer call" );
    do_work();
    hpmStart( 2, "computing meaning of
                life" );
    do_more_work();
    hpmStop( 2 );
    hpmStop( 1 );
    hpmTerminate( taskID );
```

- Fortran

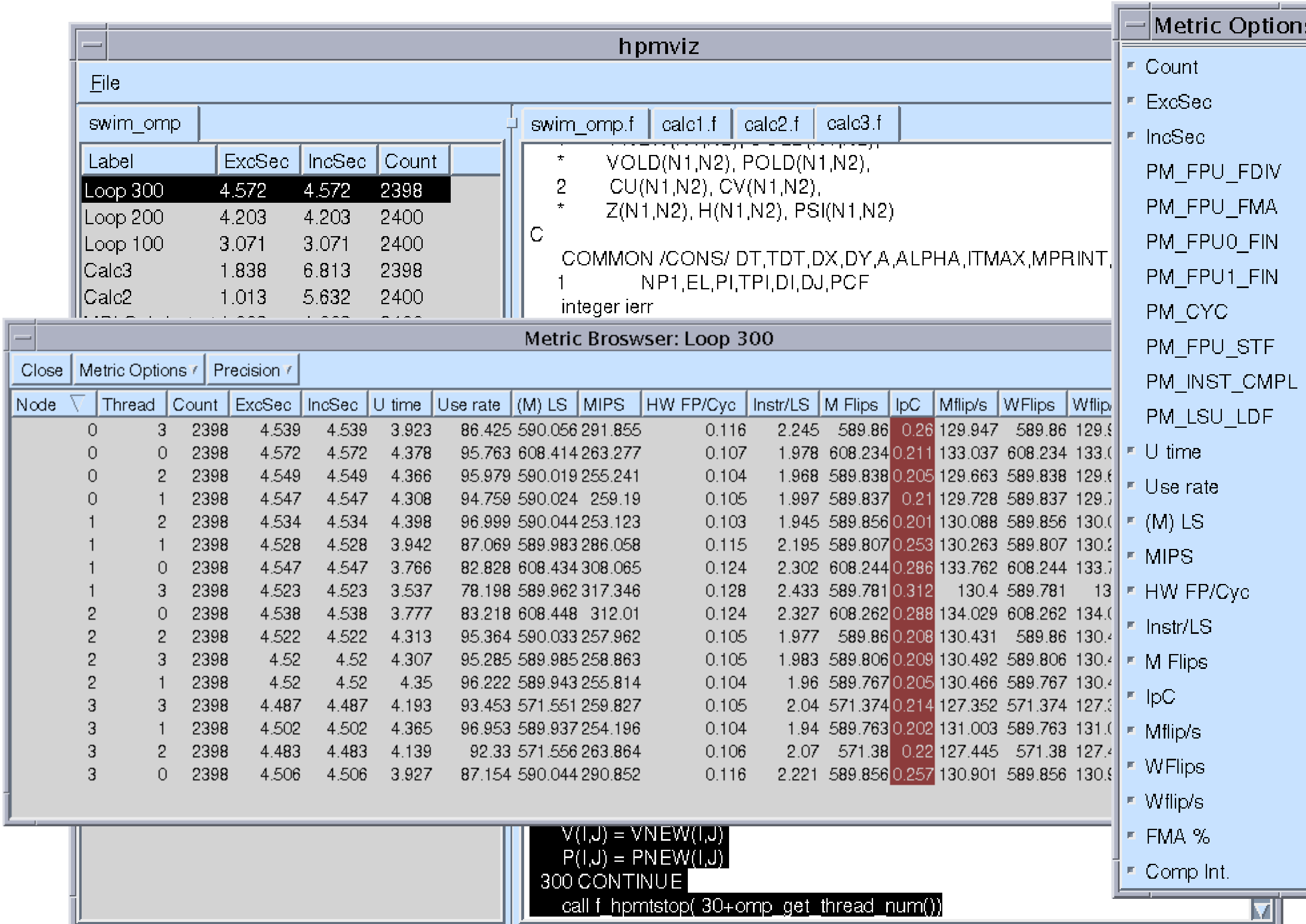
declaration:

```
    #include "f_hpm.h"
  use:
    call f_hpminit( taskID, "my program" )
    call f_hpmstart( 1, "Do Loop"  )
    do …
      call do_work()
      call f_hpmstart( 5, "computing meaning
                of life" );
      call do_more_work();
      call f_hpmstop( 5 );
    end do
    call f_hpmstop( 1 )
    call f_hpmterminate( taskID )
```

Hpmcount Example

bash-2.05a$ hpm_count swim	
…… // program output	
hpmcount (V 2.5.4) summary	
Execution time (wall clock time)	: 7.378159 seconds
######## Resource Usage Statistics ########	
Total amount of time in user mode	: 0.010000 seconds
Average shared memory use in text segment	: 672 Kbytes*sec
………	
PM_FPU_FDIV (FPU executed FDIV instruction)	: 0
PM_FPU_FMA (FPU executed multiply-add instruction)	: 0
PM_CYC (Processor cycles)	: 54331072
PM_FPU_STF (FPU executed store instruction)	: 2172
PM_INST_CMPL (Instructions completed)	: 17928229
Utilization rate	: 0.446 %
Total load and store operations	: 0.004 M
MIPS	: 2.140
Instructions per cycle	: 0.330

HPM Visualization Using PeekPerf



Environment Flags

- HPM_EVENT_SET
  - Select the event set to be recorded
  - Integer (0 – 15)
- HPM_NUM_INST_PTS
  - Overwrite the default of 100 instrumentation sections in the app.
  - Integer value > 0
- HPM_WITH_MEASUREMENT_ERROR
  - Deactivate the procedure that removes measurement errors.
  - True or False (0 or 1).
- HPM_OUTPUT_NAME
  - Define an output file name different from the default.
  - String
- HPM_VIZ_OUTPUT
  - Indicate if ".viz" file (for input to PeekPerf) should be generated or not.
  - True or False (0 or 1).
- HPM_TABLE_OUTPUT
  - Indicate table text file should be generated or not.
  - True or False (0 or 1).

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.
- Planned Integration into HPC Toolkit with PeekPerf capabilities
  - Source code traceback
  - Future capability for dynamic I/O instrumentation

Challenges

- Hardware
  - Different performance counter set
- Scalability
  - 64k nodes
  - Tracking communications for each pair:
    - $(65{,}536)^2$x48 bytes = 200 GB
  - Number of processes and events vs. number of screen pixels

Future work

- MP_Profiler
  - Scalability
- Xprofiler
  - Improved GUI
  - Integration with other platforms
- HPM
  - Useful derived metrics and verifications
  - Hpmcount / Hpmstat ?
  - Integration with other platforms

- Next target
  - Modular I/O: MIO