



IBM Research

The IBM High Performance Computing Toolkit on BlueGene/L

David Klepacki
Advanced Computing Technology
IBM T.J. Watson Research Center

Why Performance Tools are Essential

- **Device Scaling imposing fundamental constraints on system**
 - Power dissipation and energy consumption
 - Physical size / packaging
- **Pressure to re-think system architecture**
 - Blue Gene: low power devices, embedded (small)
 - Cell: Attached (embedded) co-processing engine
- **Systems become inherently more complex**
 - Connectivity / hierarchical topology (torus, intra-cell, DMA)
 - Memory constraints (less per processor)
 - Additional technology “boosts” (hyper-threading, SMT)
- **This poses new challenge to application programming**
 - New programming paradigm? (not on horizon - legacy codes, ISV apps, etc.)
- **Conclusion: New software tools essential to mitigate evolving system complexity**

IBM High Performance Computing Toolkit on BG/L

- Subset of IBM HPC Toolkit on IBM pSeries Servers:

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

<http://www.absoft.com/Products/Tools/hpc-toolkit/>

- **MPI performance: MP_Profiler**
- **CPU performance: Xprofiler, HPM**
- **Visualization and analysis: PeekPerf**

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

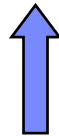
MPI_LIB= -L \$(BGL)/bglsys/lib -lmpich.rts -lmsglayer.rts -lrts.rts -ldevices.rts

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 Output with Peekperf

The screenshot displays the IBM ACTC PeekPerf: Main Window. The main window shows a list of MPI functions and their call counts and wall times. A detailed metric browser for MPI_Send_130 is also visible, showing a table of task metrics.

MPI Application Metrics:

Label	Call Count	Wall Time
barrier_sync(mpi_stuff.f)		
bcast_int(mpi_stuff.f)		
bcast_real(mpi_stuff.f)		
global_int_sum(mpi_stuff.f)		
global_real_max(mpi_stuff.f)		
global_real_sum(mpi_stuff.f)		
pmpi_allreduce(allreducef.c)		
pmpi_barrier(barrierf.c)		
pmpi_bcast(bcastf.c)		
pmpi_comm_rank(comm_rankf.c)		
pmpi_comm_size(comm_sizef.c)		
pmpi_recv(recvf.c)		
pmpi_send(sendf.c)		
rcv_real(mpi_stuff.f)	0	0
snd_real(mpi_stuff.f)	0	0
MPI_Send_130	3840	0.6
SUMMARY	0	0
task_init(mpi_stuff.f)	0	0

Metric Browser: MPI_Send_130

Task	Message Size	Count	WallClock [Max]	Transferred Bytes	Call Count [Max]	WallClock
0	(8) 16K ... 64K	1920	0.513883	2.21184e+07	1920	0.513883
1	(8) 16K ... 64K	2880	0.549085	3.31776e+07	2880	0.549085
2	(8) 16K ... 64K	2880	0.551206	3.31776e+07	2880	0.551206
3	(8) 16K ... 64K	1920	0.516694	2.21184e+07	1920	0.516694
4	(8) 16K ... 64K	2880	0.632482	3.31776e+07	2880	0.632482
5	(8) 16K ... 64K	3840	0.654542	4.42368e+07	3840	0.654542
6	(8) 16K ... 64K	3840	0.651453	4.42368e+07	3840	0.651453
7	(8) 16K ... 64K	2880	0.625413	3.31776e+07	2880	0.625413
8	(8) 16K ... 64K	2880	0.593683	3.31776e+07	2880	0.593683
9	(8) 16K ... 64K	3840	0.643496	4.42368e+07	3840	0.643496
10	(8) 16K ... 64K	3840	0.640881	4.42368e+07	3840	0.640881
11	(8) 16K ... 64K	2880	0.589392	3.31776e+07	2880	0.589392
12	(8) 16K ... 64K	2880	0.553126	3.31776e+07	2880	0.553126

Source Code Snippets:

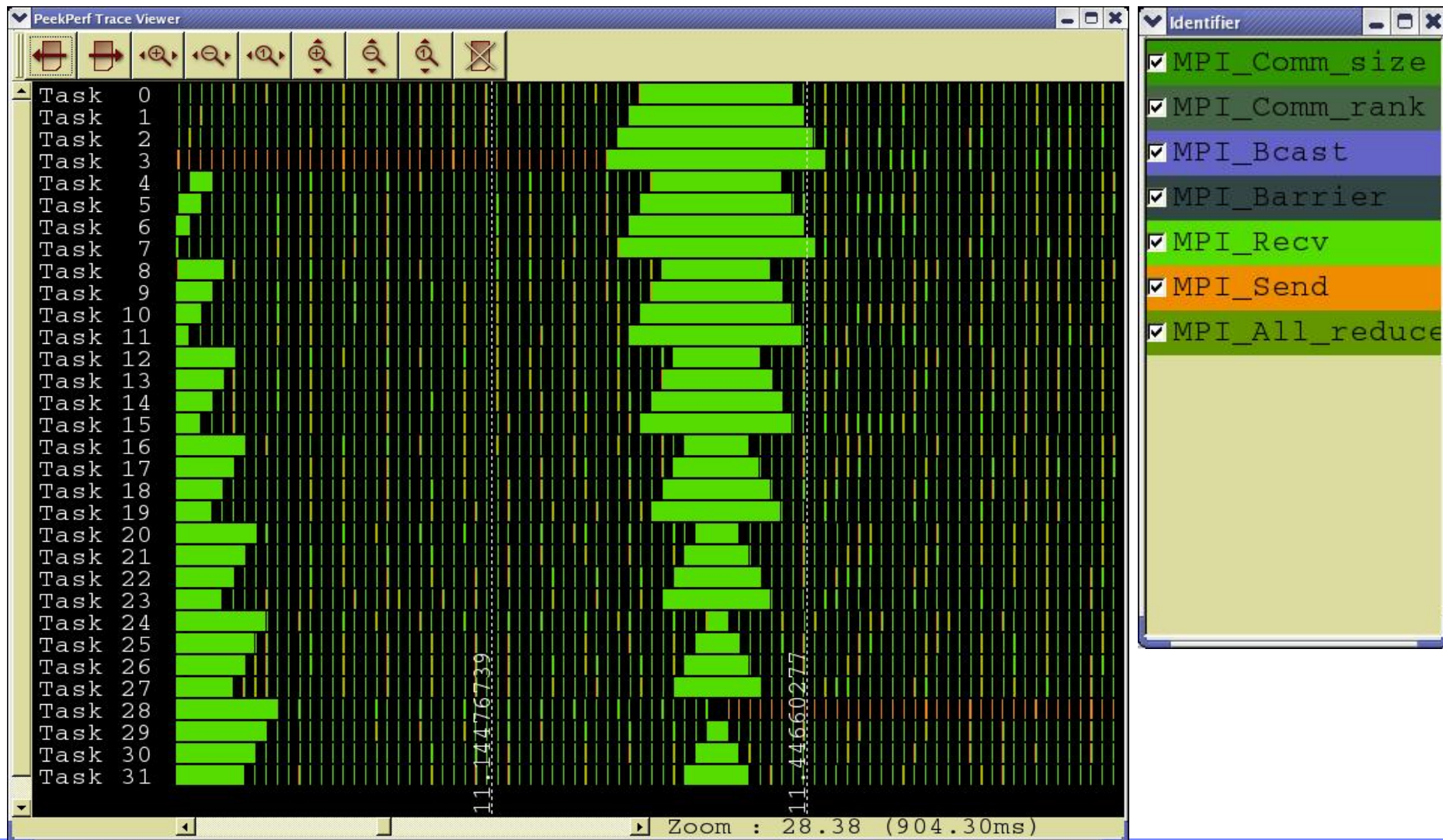
```

return
end

subroutine rcv_real(orig, value, size, ta
implicit none

```


MP_Profiler - Traces



Environment Flags

■ **TRACELEVEL**

- Level of trace back the caller in the stack
- Used to skipped wrappers
- Default: 0

■ **TRACE_TEXTONLY**

- If set to “1”, plain text output is generated
- Otherwise, a viz file is generated

– **TRACE_PROFILE**

- If set to “1”, the output is shown for each source file
- Otherwise, output is a summary of all source files

– **TRACE_PERSIZE**

- If set to “1”, the static for a function is shown for every message size
- Otherwise, summary for all message sizes is given

Xprofiler

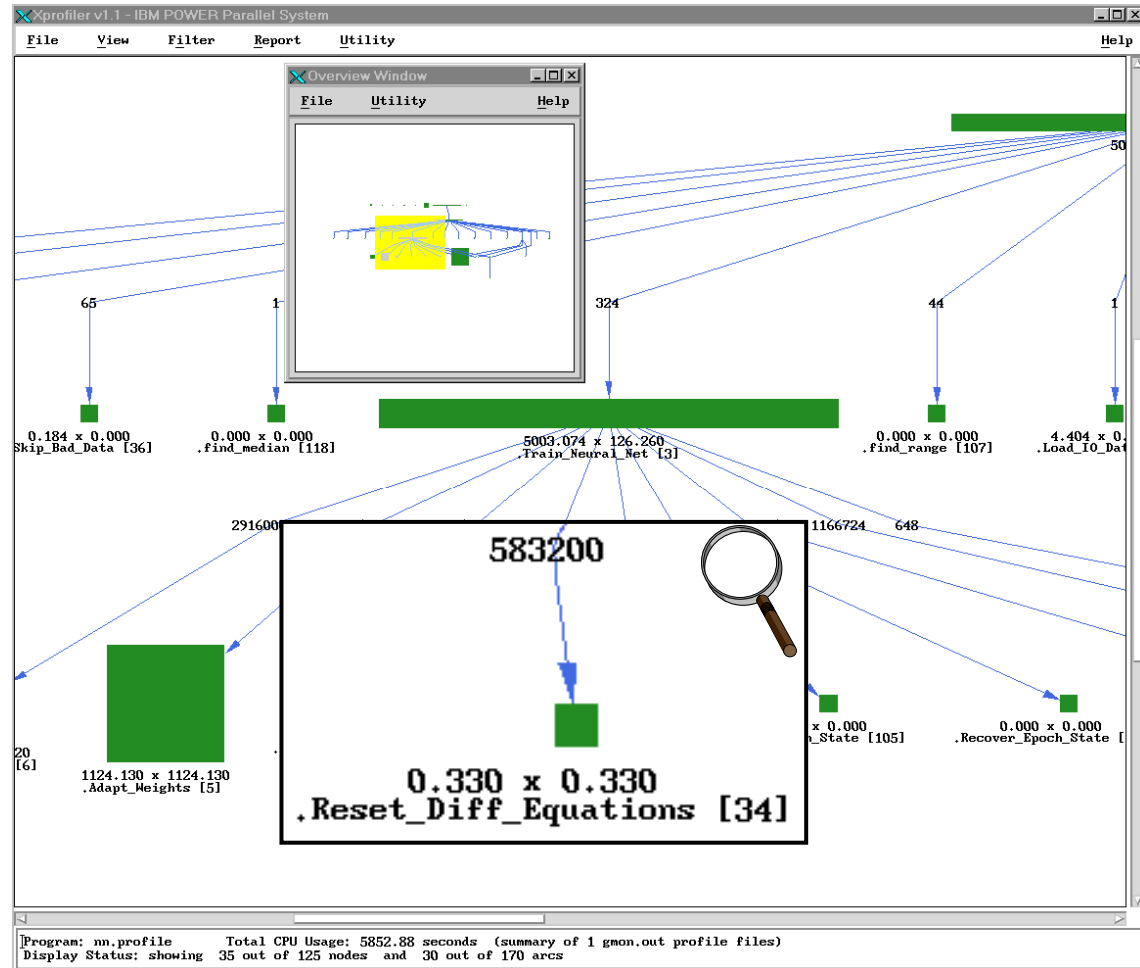
- **CPU profiling tool similar to gprof**
- **Can be used to profile both serial and parallel applications**
- **Use procedure-profiling information to construct a graphical display of the functions within an application**
- **Provide quick access to the profiled data and helps users identify functions 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: 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: 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

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++)
214		{
217	229	t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
218		t11 = t11 + a[i*n+k]*bt[j*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)

thsub

Xprofiler - Disassembler Code

Disassembler Code for .calc3 [3]					
File					
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)					

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 Use

- 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 );
```

- Flags

- Compiling: -I\$(HPM_DIR)/include
- Linking: -L\$(BGL_FLOOR)/bgl/sys/lib -L\$(HPM_DIR)/lib -lhpm.rts -lm -lbgl_perfctr.rts

Example of Use (continued)

- 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 )
```

Output

- **Summary report for each task**

- `perfhpm<taskID>.<pid>`

libhpm (V 2.6.0) summary

Total execution time of instrumented code (wall time): 0.143824 seconds

Instrumented section: 3 - Label: job 1 - process: 1

file: sanity.c, lines: 33 <--> 70

Count: 1

Wall Clock Time: 0.143545 seconds

BGL_FPU_ARITH_MULT_DIV (Multiplication and divisions, fmul, fmuls, fdiv, fdivs (Book E mul, div)) : 0

BGL_FPU_LDST_DBL_ST (...) : 23

...

BGL_UPC_L3_WRBUF_LINE_ALLOC (Write buffer line was allocated) :1702

...

- **Peekperf performance file**

- `hpm<taskID>_<progName>_<pid>.viz`

- **Table performance file**

- `tb_hpm<taskID>.<pid>`

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).

Peekperf

- **Visualization and analysis tool**
- **Offline analysis and viewing capability**
- **Supported platforms**
 - AIX
 - Linux (Power/Intel)
 - Windows (Intel)
 - BlueGene

*The toolkit will be available soon on AMD platform

MP_Profiler Visualization Using PeekPerf

The screenshot displays the IBM ACTC PeekPerf Main Window. The window is divided into two main sections. The left section, titled 'sanity program', contains a table with performance data. The right section, titled 'sanity.c', contains a code editor with C code.

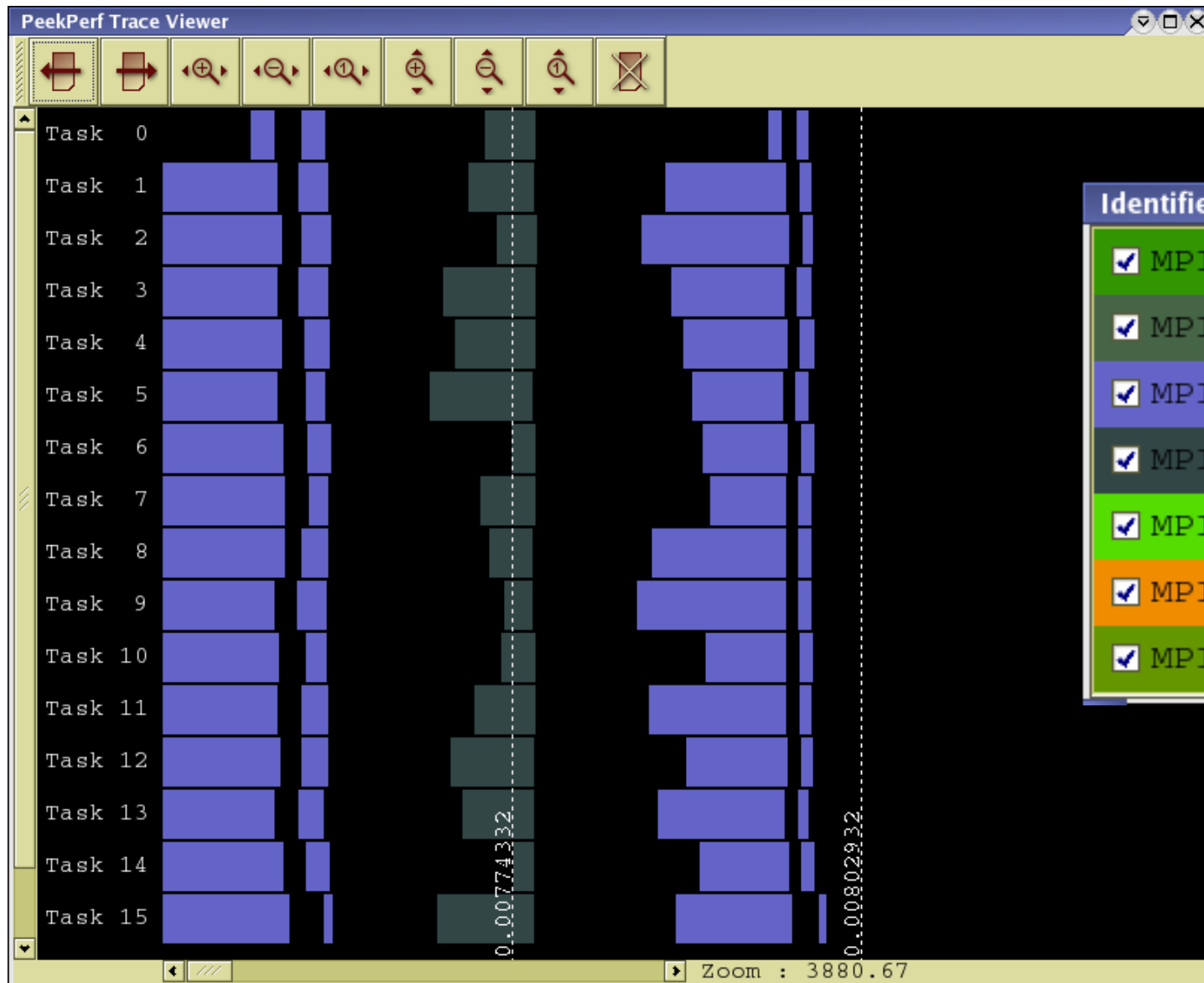
Label	Count	ExcSec	IncSec
job 1	1	0.192	0.192
job 2	1	0	0
outsider1	0	0	0.193

```
hpmStart(3,"job 1");
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

/*
 * Print personality, memory check
 */

rts_get_personality(&personality, sizeof(personality));
BGLPersonality_getLocationString(&personality);
printf("MPI: %d/%d, Pers: <%d,%d,%d,%d>/<%d,%d,%d,%d>\n",
    my_rank, num_procs,
    BGLPersonality_xCoord(&personality),
    BGLPersonality_yCoord(&personality),
    BGLPersonality_zCoord(&personality),
    rts_get_processor_id(),
    BGLPersonality_xSize(&personality),
    BGLPersonality_ySize(&personality),
    BGLPersonality_zSize(&personality),
    BGLPersonality_virtualNodeMode(&personality),
    BGLPersonality_isTorusX(&personality));
```


MP_Tracer Visualization Using PeekPerf



Identifier

- ☒ MPI_Comm_size
- ☒ MPI_Comm_rank
- ☒ MPI_Bcast
- ☒ MPI_Barrier
- ☒ MPI_Recv
- ☒ MPI_Send
- ☒ MPI_All_reduce

HPM Visualization Using PeekPerf

The screenshot displays the PeekPerf application interface, which is used for visualizing HPM (High Performance Metrics) data. The main window shows a table of performance metrics for the program 'swim_mpi.f'. The table includes columns for Label, Count, ExcSec, and IncSec. The 'Loop 300' section is highlighted, showing a Count of 100, ExcSec of 2.635, and IncSec of 2.635.

Label	Count	ExcSec	IncSec
Calc1	102	0.005	1.884
Calc2	102	0.039	2.092
Calc3	100	0.066	2.795
Initial	1	0.085	0.085
Loop 100	102	1.766	1.766
Loop 200	102	1.993	1.993
Loop 300	100	2.635	2.635
MPI Calc1 end	102	0.07	0.07
MPI Calc1 start	102	0.002	0.002
MPI Calc2 end	102	0.058	0.058
MPI Calc2 start	102	0.002	0.002
MPI in Calc3	100	0.094	0.094
loop 110	102	0.042	0.042

The 'Metric Browser: Loop 300' window provides a detailed view of the metrics for this specific loop. It includes columns for Task, thread, Count, ExcSec, IncSec, U sec, (M)LS, MIPS, Use rt, hwrp/c, and I/LS. The data for task 0, thread 0 is as follows:

Task	thread	Count	ExcSec	IncSec	U sec	(M)LS	MIPS	Use rt	hwrp/c	I/LS
0	0	100	2.635	2.635	2.633	394.11	242.569	99.924	0.103	1.622

The code editor on the right shows the Fortran source code for the loop, including MPI calls and calculations. The code is as follows:

```

C
C   TIME SMOOTHING AND UPDATE FOR NEXT CYCLE
C
C SPEC removed CCMIC$ DO GLOBAL
  call f_hpmstart( 30, "Loop 300" )
C$OMP PARALLELDO
C$OMP&SHARED (ALPHA,M,N,U,V,P,UNEW,VNEW,PNEW,UOLD,VOLD,POLD)
C$OMP&SHARED (JS,JE)
C$OMP&PRIVATE (I,J)
  DO 300 J=js,je
    DO 300 I=1,M
      UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
      VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
      POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
    300 CONTINUE
    call f_hpmstop( 30 )
CME-----
C
C   PERIODIC CONTINUATION
C
  1 0,3,MPI_COMM_WORLD,req(3),ierr)
  call mpi_irecv(vold(1,n+1),n1,MPI_DOUBLE_PRECISION,
  1 0,4,MPI_COMM_WORLD,req(4),ierr)
  call mpi_irecv(uold(1,n+1),n1,MPI_DOUBLE_PRECISION,

```

Future Directions

Unified Framework (Instrumentation and Analysis)

IBM ACTC PeekPerf: Main Window

File Tools Options Action

HPM MPI SIGMA DPOMP

Name

- Probe
 - MPI_Iprobe
 - MPI_Probe
- Recv
 - MPI_Irecv
 - MPI_RECV
- Send
 - Blocking Send
 - MPI_Bsend
 - MPI_Rsend
 - MPI_Send
 - MPI_Ssend
 - Nonblocking Send
 - MPI_Ibsend
 - MPI_Irsend
 - MPI_Isend
 - MPI_Issend
- SendRecv
- Test
 - MPI_Test
 - MPI_Testall
 - MPI_Testany

swim_omp

Label	Count	ExcSec	IncSec
-ALL	1	0.006	1.062
-Calc1	102	0.006	0.263
-Calc2	102	0.072	0.373
-Calc3	100	0.098	0.379
-Initial	1	0.042	0.042
-Loop 100	102	0.119	0.119
-Loop 200	102	0.178	0.178
-Loop 300	100	0.209	0.209
-MPI Calc1 end	102	0.08	0.08
-MPI Calc1 start	102	0.05	0.05
-MPI Calc2 end	102	0.068	0.068
-MPI Calc2 start	102	0.055	0.055
-MPI in Calc3	100	0.072	0.072
-loop 110	102	0.009	0.009

calc1.f calc2.f calc3.f swim_omp.f swim_omp.f calc1.f calc2.f calc3.f

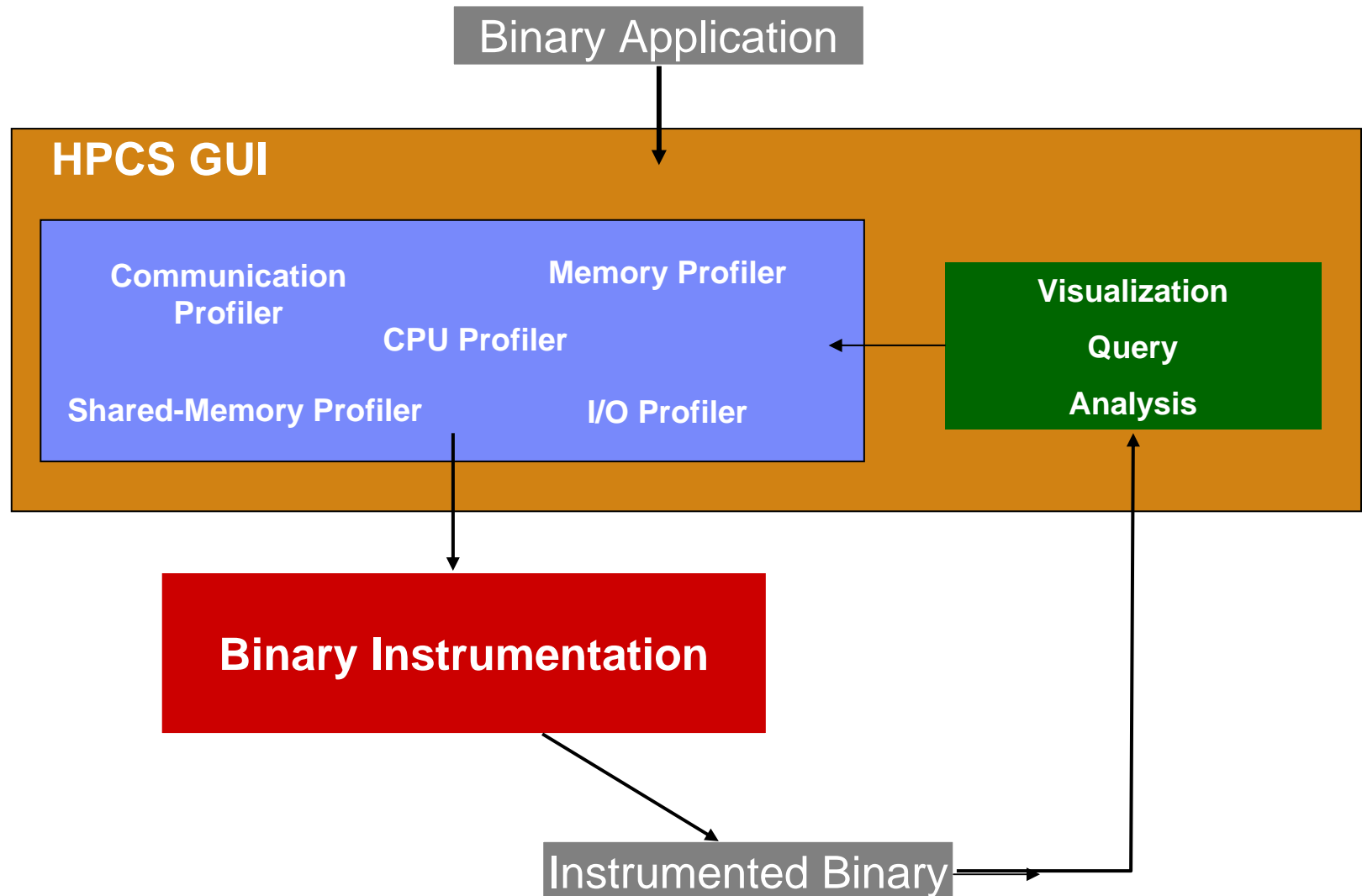
```

101 1 0,2,MPI_COMM_WORLD,req(2),ierr)
102 CALL mpi_irecv(p(m+1,n+1),1,MPI_DOUBLE_PRECISIO
103 1 0,3,MPI_COMM_WORLD,req(3),ierr)
104 CALL mpi_irecv(vold(m+1,n+1),1,MPI_DOUBLE_PRECI
105 1 0,4,MPI_COMM_WORLD,req(4),ierr)
106 CALL mpi_irecv(uold(m+1,n+1),1,MPI_DOUBLE_PRECI
107 1 0,5,MPI_COMM_WORLD,req(5),ierr)
108 CALL mpi_irecv(pold(m+1,n+1),1,MPI_DOUBLE_PRECI
109 1 0,6,MPI_COMM_WORLD,req(6),ierr)
110 endif
111 if(taskid.eq.0)then
112 CALL mpi_isend(v(1,1),1,MPI_DOUBLE_PRECISION,
113 1 numtasks-1,1,MPI_COMM_WORLD,req(7),ierr)
114 CALL mpi_isend(u(1,1),1,MPI_DOUBLE_PRECISION,
115 1 numtasks-1,2,MPI_COMM_WORLD,req(8),ierr)
116 CALL mpi_isend(p(1,1),1,MPI_DOUBLE_PRECISION,
117 1 numtasks-1,3,MPI_COMM_WORLD,req(9),ierr)
118 CALL mpi_isend(vold(1,1),1,MPI_DOUBLE_PRECISION
119 1 numtasks-1,4,MPI_COMM_WORLD,req(10),ierr)
120 CALL mpi_isend(uold(1,1),1,MPI_DOUBLE_PRECISION
121 1 numtasks-1,5,MPI_COMM_WORLD,req(11),ierr)
122 CALL mpi_isend(pold(1,1),1,MPI_DOUBLE_PRECISION
123 1 numtasks-1,6,MPI_COMM_WORLD,req(12),ierr)
124 endif
125 if(taskid.eq.0.or.taskid.eq.numtasks-1)then
126 CALL MPI_WAITALL(12,req,istat,ierr)
127 endif
128 call f_hpmstop( 17 )
129
130 C
131 RETURN
132 END
133
134 SUBROUTINE CALC3Z
135 C
136 C TIME SMOOTHER FOR FIRST ITERATION
137 C
138
139 PARAMETER (N1=513, N2=513)
140
141 include "mpif.h"
142 COMMON/decomp/js,je,taskid,numtasks,req(16),
143 1 istat(MPI_STATUS_SI
144
145 integer taskid,req
146 COMMON U(N1,N2), V(N1,N2), P(N1,N2),
147 * UNEW(N1,N2), VNEW(N1,N2),
148 * PNEW(N1,N2), UOLD(N1,N2),
149 * VOLD(N1,N2), POLD(N1,N2),
  
```

Proposed Technologies for DARPA HPCS (PERCS)

- **Completely Binary Approach (pSigma)**
 - Programmable and dynamic, yet without the need for source code modification.
- **Data-Centric Analysis (DCA)**
 - For HPCS systems, new tools are needed to provide detailed information on the impact of an **application's data structures** in relation to the underlying hardware.
- **Alternate Scenario Prediction (ASP)**
 - Data structure layout
 - “what if my array A was dimensioned like ...”
 - Order of a parallel computation, scheduling of threads, etc.
- **User-Controlled Automation (autoPerf)**
 - Productivity is controlled by degree of automation chosen by programmer.
 - Can be fully automated if desired.

The IBM HPC Toolkit for DARPA HPCS



upGUI

Binary to Patch:

Architecture Events Instrumentation Execution

Choose File ☐ Event file:

Event General Information

Name: Event Initial State: ☒ Enabled

Event Condition Information

Instruction

In Function: For Symbol:

Memory Effect of the Instruction

In Function: For Symbol:

Counters Conditions

Counter	Function	Symbol	Operator	Quantity	Function	Symbol
<input type="text" value="Accesses@L1"/>	<input type="text" value="foo"/>	<input type="text" value="array1"/>	<input type="text" value=">"/>	<input type="text" value="0"/>	<input type="text"/>	<input type="text"/>

Event Handler

Function Name:

Events Summary

Cache Simulator: ☒ Enabled Symbolic Mapping: ☒ Enabled

Name	Initial State	Instruction	Memory Effect of the Instru	Counters Condition	Event Handler
------	---------------	-------------	-----------------------------	--------------------	---------------

Generate Events File

Data-Centric Analysis Technology (DCA)

peekperf

File Tools

sigma

Label	MemTime /	Access
u@Global	1.57702e+06	216097
u@Global_initial	447628	32511
u@Global_calc3	444718	32258
u@Global_calc3z	433822	32258
u@Global_calc1	250850	119070
v@Global	1.57569e+06	216097
p@Global	1.52905e+06	192786
h@Global	1.30777e+06	168216
z@Global	1.30759e+06	168216
pold@Global	1.27448e+06	112144
vold@Global	1.27217e+06	112144
uold@Global	1.27047e+06	112144
cv@Global	1.26258e+06	145158
cu@Global	1.26066e+06	145158
unew@Global	1.19097e+06	81151
vnew@Global	1.15966e+06	81151
pnew@Global	1.15684e+06	81151
psi@Global	486142	51913
unknownDt@Global	11220	539
unknownDt@Local	424	30

swim.f

```

VOLD(N1,N2), FOLD(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, MPRINT, M, N, MP1,
1  NP1, EL, PI, TPI, DI, DJ, PCF
C
TDT = TDT+TDT
DO 400 J=1, NP1
DO 400 I=1, MP1
UOLD(I,J) = U(I,J)
VOLD(I,J) = V(I,J)
POLD(I,J) = P(I,J)
U(I,J) = UNEW(I,J)
V(I,J) = VNEW(I,J)
P(I,J) = PNEW(I,J)
400 CONTINUE
RETURN
END
C SPEC removed CCMIC$ MICRO
SUBROUTINE CALC3
C
C  TIME SMOOTHER
C
IMPLICIT REAL*8      (A-H, O-Z)
#include "swim.h"

COMMON LI(N1,N2) V(N1,N2) P(N1,N2)
  
```

Metric Browser: u@Global_calc3

Close Metric Options Precision

Task	thread	Misses	LoadMisses	StoreMisses	Hits	LoadHits	StoreHits	Access	LoadAccesses	StoreAccess
0	L1	442	193	249	31816	15936	15880	32258	16129	16129
0	L2	190	74	116	16132	119	16013	16322	193	16129
0	L3	0	0	0	1191	74	1117	1191	74	1117
0	TLB	0	0	0	32258	32258	0	32258	32258	0

C SPEC removed CCMIC\$ DO GLOBAL

IBM Vision for DARPA HPCS (PERCS)

