

The IBM High Performance Computing Toolkit

PeekPerf Documentation

The peekperf GUI is the control center of the HPC Toolkit. It allows you to control the instrumentation, execute the application and visualize and analyze the collected performance data within the same user interface. The dimensions of performance data provided in our current framework are:

- CPU (HPM)
- Message Passing (MPI)
- Threads (OpenMP)
- Memory
- IO

The collected performance data will be mapped to the source code. So you will easier find bottlenecks and points for optimizations. Peekperf provides filtering and sorting capabilities to help you analyze the data. PeekPerf is available for several UNIX derivations (AIX, Linux) and Microsoft® Windows®. However, the instrumentation only supports for 32-bit applications on AIX Power and 32-bit and 64-bit applications on Linux Power.

Content

The IBM High Performance Computing Toolkit.....	1
How to start PeekPerf.....	3
PeekPerf Main Window (Visualization).....	4
PeekPerf Main Window with Instrumentation (AIX/LINUX Power)	12
PeekPerf Simple IDE	18

How to start PeekPerf

PeekPerf is started from the command line with (where `tc>` is my prompt)

```
tc> peekperf  
or  
tc> peekperf <-num max_src_files> <vizfiles>
```

You can specify more than one `.viz` file. PeekPerf will open all this `.viz` files and combine the data from all the `.viz` files. Peekperf will also try to open the source files if the source files are available. In some applications, there may be hundreds of source files. By default, peekperf will open up to 15 number of source files. If there are more than 15 source files, peekperf will ask for your input to select a list of source files to be open. You can reset the default value by using the `-num` option.

The following is available on AIX Power and Linux Power only:

```
tc> peekperf <-num max_src_files> <binary> <vizfiles>
```

You can specify the binary in the command line. Peekperf will invoke the binary instrumentation engine and obtain the information about the binary. It will then allow you to control the instrumentation from the GUI.

PeekPerf Main Window (Visualization)

PeekPerf will try to find your source files. If peekperf fails to locate the source files, it will pop up a window to ask you to select the top level directory for your source code. If peekperf finds more than one file with the same name, you will be asked to select the right one. You can also open the files manually by using the menu point **File -> Open Sources**. The .viz files can be opened by using the menu point **File->Open Performance Data**.

We will start with the performance data visualization. The instrumentation will be discussed in the next section.

The screenshot displays the PeekPerf Main Window, which is split into two main sections: the Data Visualization Window on the left and the Source Code Window on the right.

Data Visualization Window: This window shows a table of performance data for various source files. The table has four columns: Label, Count, WallClock, and Transferred Bytes. A context menu is visible over the table with options: Set Filter, No Filter, Expand the Tree, Collapse the Tree, Show as Table, and View Tracer.

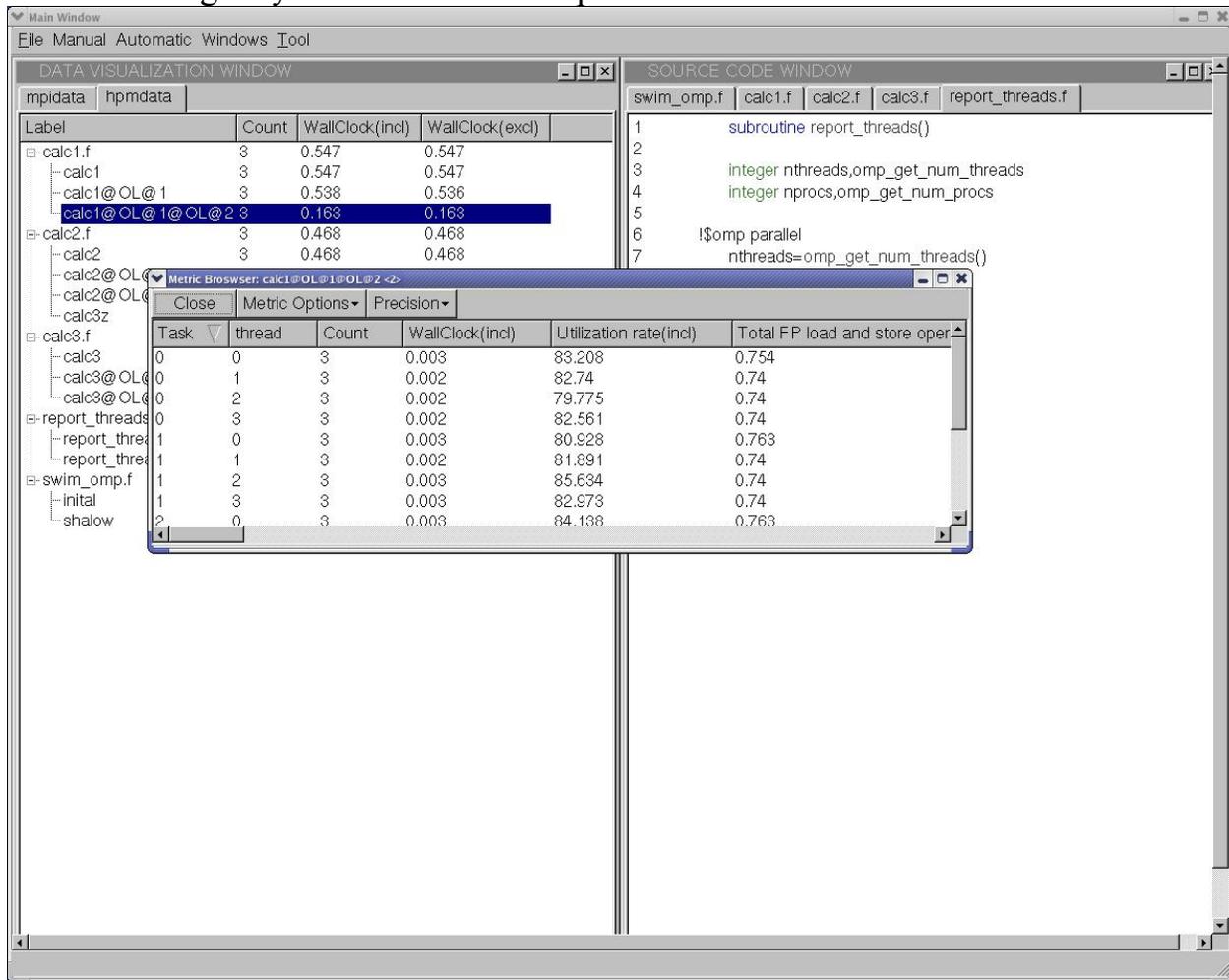
Label	Count	WallClock	Transferred Bytes
SUMMARY	50	0.581199	127376
calc1.f	3	0.147875	12312
calc2.f	3	0.43331	12312
calc3.f	1	0.000157	4104
swim_omp.f	1	0.017161	4104

Source Code Window: This window displays the source code for the selected file, 'swim_omp.f'. The code is a Fortran subroutine named 'CALCS' that implements a loop with OpenMP parallelization. Key features include time smoothing, parameterization (N1=513, N2=513), and various OpenMP directives like 'COMMON', 'C\$OMP PARALLEL', and 'C\$OMP SHARED'. The code includes headers for 'mpif.h' and 'f_hpm.h'.

```
1 SUBROUTINE CALCS
2 C
3 C TIME SMOOTHER
4 C
5 PARAMETER (N1=513, N2=513)
6
7 include "mpif.h"
8 C #include "f_hpm.h"
9
10 COMMON/decomp/js,je,taskid,numtasks,req(16),
11 1 istat(MPI_STATUS_SIZE,16)
12 integer taskid,req
13 COMMON U(N1,N2), V(N1,N2), P(N1,N2),
14 * UNEW(N1,N2), VNEW(N1,N2),
15 1 PNEW(N1,N2), UOLD(N1,N2),
16 * VOLD(N1,N2), POLD(N1,N2),
17 2 CU(N1,N2), CV(N1,N2),
18 * Z(N1,N2), H(N1,N2), PSI(N1,N2)
19 C
20 COMMON /CONS/
21 DT,TDT,DX,DY,A,ALPHA,ITMAX,MPRINT,M,N,MP1,
22 1 NP1,EL,PI,TP1,DI,DJ,PCF
23 integer ierr, mytID
24 integer omp_get_thread_num
25 C
26 C TIME SMOOTHING AND UPDATE FOR NEXT CYCLE
27 C
28
29 C SPEC removed CCMIC$ DO GLOBAL
30 C$OMP PARALLEL
31 C$OMP&SHARED
32 (ALPHA,M,N,U,V,P,UNEW,VNEW,PNEW,UOLD,VOLD,POLD)
33 C$OMP&SHARED (JS,JE)
34 C$OMP&PRIVATE (I,J)
35 C$OMP&PRIVATE (mytID)
36 mytID = 30+omp_get_thread_num()
37 C call f_hpmstart( mytID, "Loop 300" )
```

There are two types of windows here. The left side of the window is the Data Visualization Window and the right side of the window is the Source Code Window. The Source Code Window is to display the source files. The Data Visualization Window shows the collected performance data. If you click the left

mouse button on a *leaf* node, on the right side the corresponding line of source will be highlighted immediately. If you click the right mouse button on a *leaf* node, you will see the performance data collected for this leaf node in more details. The window will give you all the collected performance metrics.

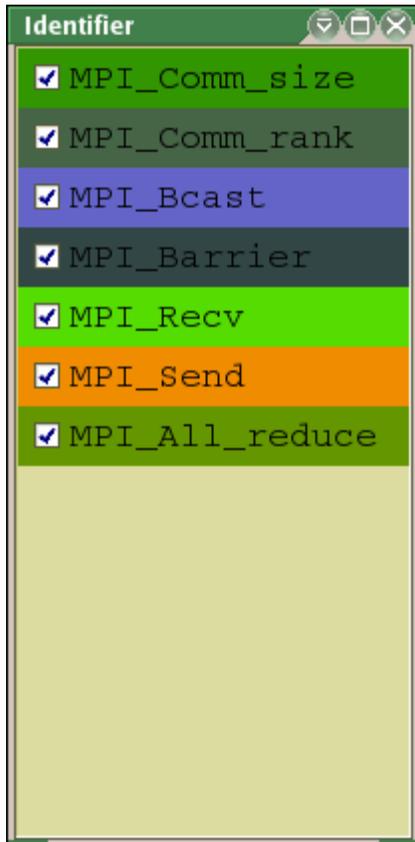


If you click with the right mouse button on a *non-leaf* node or other empty space, you will bring up the *context menu*. You can collapse or expand the tree by selecting the “*Collapse the Tree*” or “*Expand the Tree*”. You can filter the performance data by selecting the “*Set Filter*”. If you want to go back to the original state, you can select “*No Filter*”. In addition to the tree view of performance data, you can also display the performance data as a tabular form by selecting the “*Show as Table*”. If the collected performance data is **MPI**, you will see “*View Tracer*” in the context menu. By clicking the “**View Tracer**” in the context menu, the trace viewer will be presented to you. However, if the collected performance data is **IO**, the “*View Tracer*” will bring up a different viewer for the **IO** trace data. Here are some snapshots for these functionalities:

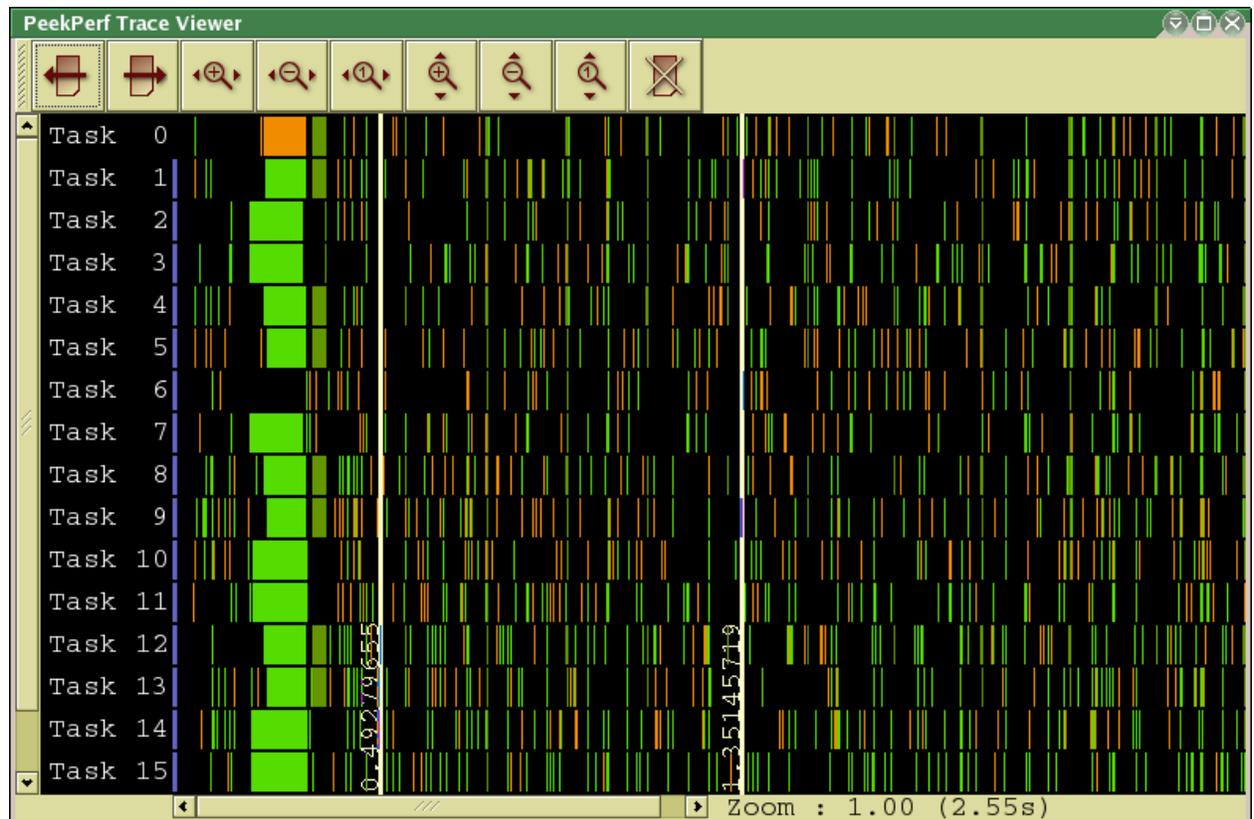
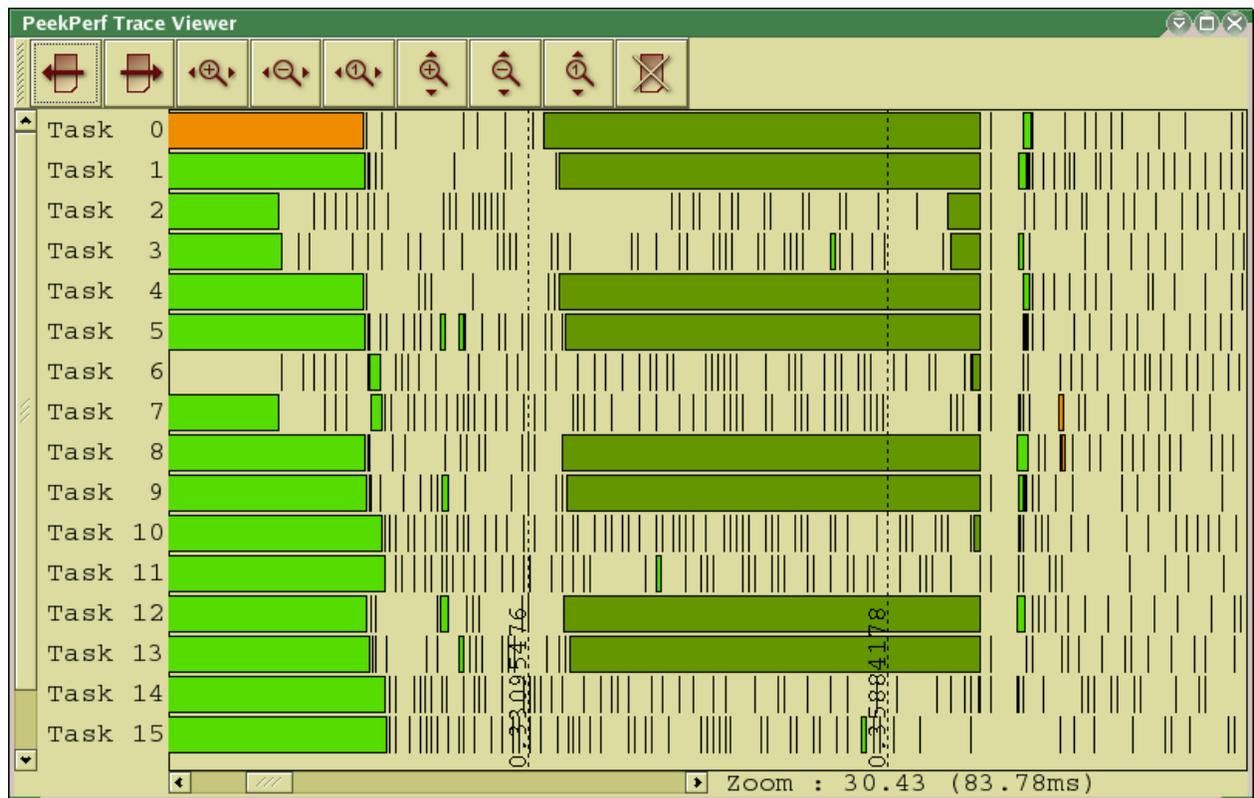
Here is the tabular form of the performance data when you click the “*Show as Table*”. You can *sort* each column by clicking the header of each column.

By bring up the *context menu*, it will allow you to *hide* the column or *save* the data into a plain text file. If you want to move a column to a different place, you can do it by clicking the left mouse button on the column header along with the CTRL key. You can also map the data to the source code by clicking the row number.

	Label	File	Function	Count	WallClock	Transferred B
1	MPI_Comm_rank_65 (0)	swim_omp.f	shalow(swim_omp.f)	1	0.00005	0
2	MPI_Comm_size_66 (0)	swim_omp.f	shalow(swim_omp.f)	1	0.00000	0
3	MPI_Irecv_283 (0)	swim_omp.f	inital(swim_omp.f)	1	0.00008	4104
4	MPI_Waitall_290 (0)	swim_omp.f	inital(swim_omp.f)	1	0.01716	0
5	MPI_Irecv_312 (0)	swim_omp.f	inital(swim_omp.f)	1	0.00004	4104
6	MPI_Isend_324 (0)	swim_omp.f	inital(swim_omp.f)	1	0.00003	4104
7	MPI_Waitall_328 (0)	swim_omp.f	inital(swim_omp.f)	1	0.00002	0
8	MPI_Irecv_336 (0)	swim_omp.f	inital(swim_omp.f)	1	0.00001	8
9	MPI_Isend_348 (0)	swim_omp.f	inital(swim_omp.f)	1	0.00001	8
10	MPI_Waitall_352 (0)	swim_omp.f	inital(swim_omp.f)	1	0.00012	0
11	MPI_Irecv_33 (0)	calc1.f	calc1(calc1.f)	3	0.00005	12312
12	MPI_Isend_47 (0)	calc1.f	calc1(calc1.f)	3	0.00007	12312
13	MPI_Isend_49 (0)	calc1.f	calc1(calc1.f)	3	0.00004	12312
14	MPI_Waitall_53 (0)	calc1.f	calc1(calc1.f)	3	0.00002	0
15	MPI_Irecv_98 (0)	calc1.f	calc1(calc1.f)	3	0.00009	12312
16	MPI_Irecv_100 (0)	calc1.f	calc1(calc1.f)	3	0.00002	12312
17	MPI_Isend_116 (0)	calc1.f	calc1(calc1.f)	3	0.00017	12312
18	MPI_Isend_118 (0)	calc1.f	calc1(calc1.f)	3	0.00003	12312
19	MPI_Waitall_122 (0)	calc1.f	calc1(calc1.f)	3	0.00004	0
20	MPI_Irecv_138 (0)	calc1.f	calc1(calc1.f)	3	0.00001	24
21	MPI_Irecv_140 (0)	calc1.f	calc1(calc1.f)	3	0.00001	24
22	MPI_Isend_151 (0)	calc1.f	calc1(calc1.f)	3	0.00001	24
23	MPI_Isend_153 (0)	calc1.f	calc1(calc1.f)	3	0.00002	24
24	MPI_Waitall_157 (0)	calc1.f	calc1(calc1.f)	3	0.00386	0
25	MPI_Irecv_32 (0)	calc2.f	calc2(calc2.f)	3	0.00060	12312
26	MPI_Irecv_34 (0)	calc2.f	calc2(calc2.f)	3	0.00018	12312
27	MPI_Isend_50 (0)	calc2.f	calc2(calc2.f)	3	0.00050	12312



The *MPI trace viewer* itself shows the tasks at the y axis and the time at the x axis. For every task you can see the timeline. Every MPI call is highlighted with an other color. The MPI traces can be viewed with a black or a bright background. When using the bright background every event is surrounded by a black rectangle which makes it easier to identify very short events. When viewing a lot of events all this black frames will create an very distorted view of the scene, so you are maybe happier with the black background. The picture at the left shows a screenshot of a window which is shown beside the trace viewer. So you are able to map easy from the timeline to a event within the timeline. You can supress some types of events from beeing shown simply be clicking on the event type in the 'Identifier' window.



If you click on an event with in the tracer window you will see when clicking with the:

- Left Mouse Button

That the correspondig line of source within the PeekPerf Main Window is highlighted. If you hold the left mouse button and move the mouse pointer to an other point you will see two lines. The first line is shown at the origin (where you pressed the mouse button). The second line is shown at the current position of the mouse. If you lift the mouse button now the timeline zooms automatically into this selected timeframe.

- Right Mouse Button

You will see a summary of the collected data for this special event. Please note that when ever an event transmits some bytes we will try to calculate a transfer rate by simply dividing the number of transferred bytes by the elapsed time. In case of non-blocking calls (for example: MPI_Irecv()) these data will not show the associated physical transfer rate.

```
MPI_Recv
Start: 0.31864381
End   : 0.31963038
Bytes: 1152
Rate  : 1.17 MBytes/Sec
```

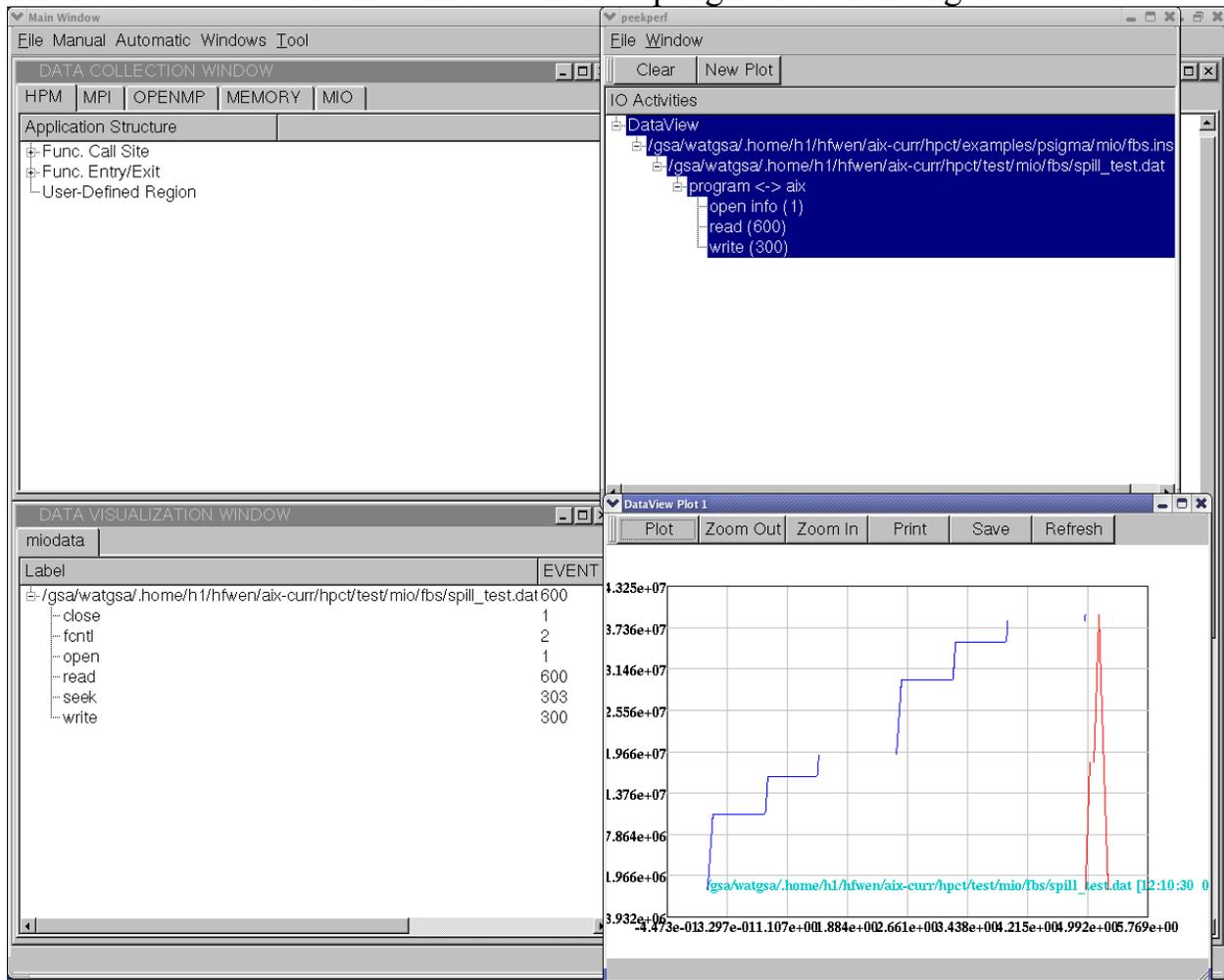
You can move through the trace using the toolbar on top. This toolbar can be undocked from the window. You are able to zoom in and out vertically and horizontally.



An other way to move around within the execution trace are the following keys:

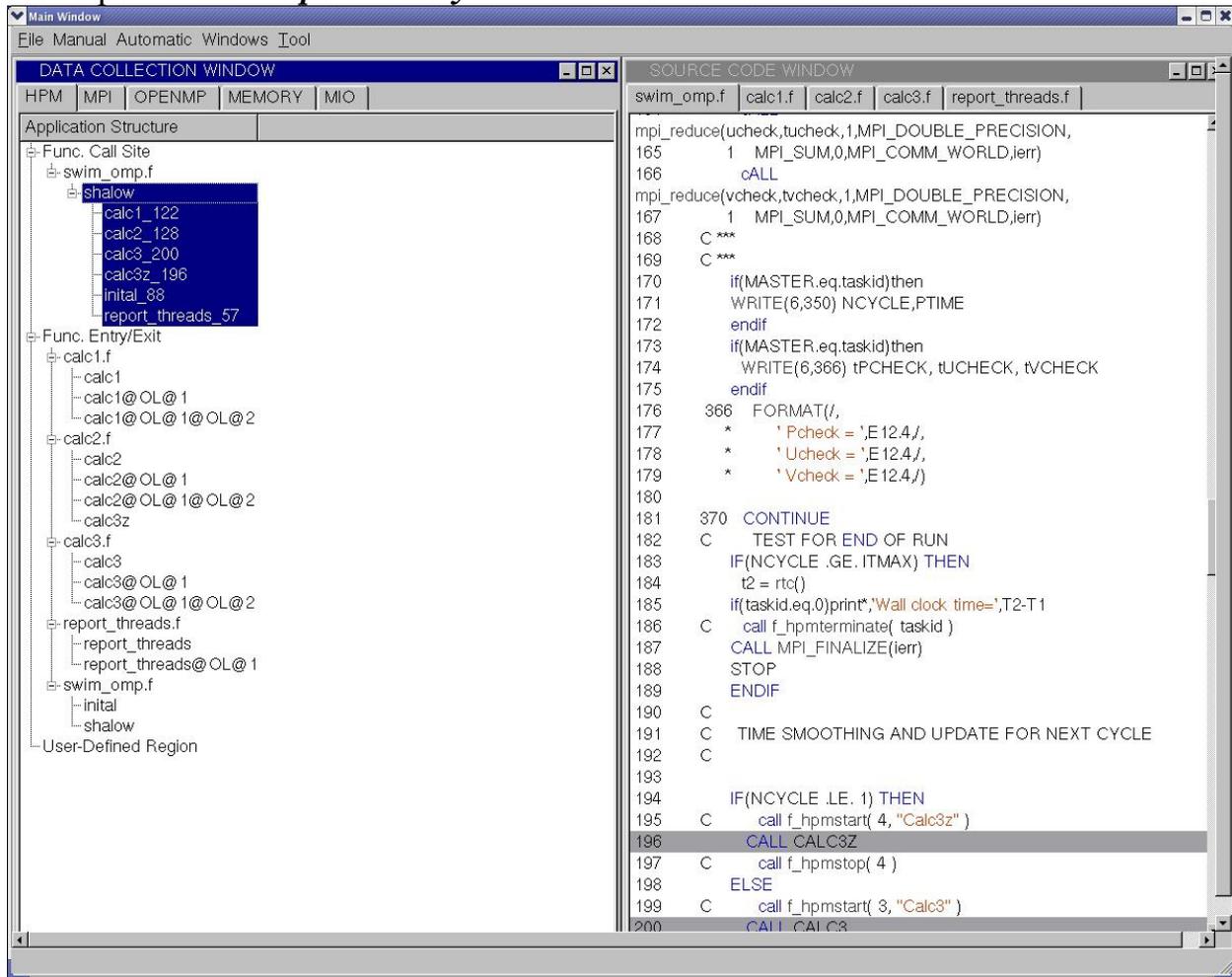
<i>Key</i>	<i>Action</i>
←	move traces to the left
→	move trace to the right
↑	scroll up through tasks
↓	scroll down through tasks
Page Up	scroll trace faster to the left
Page Down	scroll trace faster to the right
'z' or 'y'	zoom time in
'x'	zoom time out
'a'	zoom tasks in
's'	zoom tasks out

The *IO trace viewer* is the window on the top-right side of the figure.



PeekPerf Main Window with Instrumentation (AIX/LINUX Power)

When the binary is given to the peekperf, another type of window called Data Collection Window will be shown. In this window, you are able to control the instrumentation. The binary can be given via the command line or by selecting the menu point **File->Open Binary**.



The tree in each panel will present the program structure and be created based on the type of performance data. For example, the tree in HPM panel contains two subtrees. The "Func. Entry/Exit" subtree will show all the functions. The call site of the functions will be in another "Func. Call Site" subtree.

The **context menu** of Data Collection Window provides you some searching capability in the tree. Select the "**Search**" option in the context menu and input the keyword. Peekperf will search the entire tree and give you the information related to the keyword. Here is an example when the keyword is "calc1".

	Path	File	Function	Call Site
1	/gsa/watgsa/.home	calc1.f	calc1	
2	/gsa/watgsa/.home	calc1.f	calc1@OL@1	
3	/gsa/watgsa/.home	calc1.f	calc1@OL@1@OL@2	
4	/gsa/watgsa/.home	swim_omp.f	shalow	calc1_122

The *instrumentation* is being selected by clicking the *left mouse button* on the nodes in the tree. For example, when you click the shalow node and select the node, all the children in the shalow node will be selected and highlighted. If you want to deselect it, simply click the left mouse button again on the selected node. All the children including this node will be deselected. If you want to clear all your current selected instrumentation, you can do this by selecting the menu point ***Tool->Clear Instrumentation***. Note at this point the instrumented application is not generated yet. You are only selecting the places to put the instrumentation. After you browse through each panel and decide the instrumentation, you will go to the menu point ***Automatic -> Generate an Instrumented Application***. When it is done, it will pop up a window to indicate if the instrumentation is completed. It will also tell you the name of the instrumented application.

After the instrumented application is generated, you can run the instrumented application if the application can be run in the same machine. You can do this by selecting the menu point ***Automatic->Run an Instrumented Application***. Here is an example after the instrumented application is run. The HPM and MPI data are collected in a single run.

Main Window

File Manual Automatic Windows Tool

DATA COLLECTION WINDOW

HPM | MPI | OPENMP | MEMORY | MIO

Application Structure

- [-] MPI CALLS
 - [-] calc1.f
 - [-] calc1
 - [-] mpi_irecv_100
 - [-] mpi_irecv_104
 - [-] mpi_irecv_106
 - [-] mpi_irecv_132
 - [-] mpi_irecv_134
 - [-] mpi_irecv_138
 - [-] mpi_irecv_140
 - [-] mpi_irecv_33
 - [-] mpi_irecv_41
 - [-] mpi_irecv_43
 - [-] mpi_irecv_98
 - [-] mpi_isend_110
 - [-] mpi_isend_112
 - [-] mpi_isend_116

SOURCE CODE WINDOW

swim_omp.f | calc1.f | calc2.f | calc3.f | report_threads.f

```

153      UCHECK = 0.0
154      VCHECK = 0.0
155
156      DO 3500 JCHECK = js,je
157      DO 3500 ICHECK = 1, MNMIN
158      PCHECK = PCHECK + ABS(PNEW(ICHECK,JCHECK))
159      UCHECK = UCHECK + ABS(UNEW(ICHECK,JCHECK))
160      VCHECK = VCHECK + ABS(VNEW(ICHECK,JCHECK))
161      3500 CONTINUE
162      cALL
163      mpi_reduce(pcheck,tpcheck,1,MPI_DOUBLE_PRECISION,
164      1 MPI_SUM,0,MPI_COMM_WORLD,ierr)
165      cALL
166      mpi_reduce(ucheck,tucheck,1,MPI_DOUBLE_PRECISION,
167      1 MPI_SUM,0,MPI_COMM_WORLD,ierr)
168      c***
169      c***
170      if(MASTER.eq.taskid)then
171      WRITE(6,350) NCYCLE,PTIME
172      endif
173      if(MASTER.eq.taskid)then
174      WRITE(6,366) tPCHECK, tUCHECK, tVCHECK
175      endif
176      366 FORMAT(/,
177      * ' Pcheck = ',E12.4/,
178      * ' Ucheck = ',E12.4/,
179      * ' Vcheck = ',E12.4/)
180
181      370 CONTINUE
182      C TEST FOR END OF RUN
183      IF(NCYCLE .GE. ITMAX) THEN
184      t2 = rtc()
185      if(taskid.eq.0)print*,'Wall clock time=',T2-T1
186      C call f_hpnterminate( taskid )
187      CALL MPI_FINALIZE(ierr)

```

DATA VISUALIZATION WINDOW

hpmdata | mpidata

Label	Count	WallClock(incl)	WallClock(excl)
[-] swim_omp.f	3	0.055	0.055
[-] shalow	3	0.055	0.055
[-] calc1_122	3	0.022	0.022
[-] calc2_128	3	0.025	0.025
[-] calc3_200	1	0.005	0.005
[-] calc3z_196	1	0.006	0.006
[-] inital_88	1	0.055	0.055
[-] report_threads_57	1	0.002	0.002

For MPI, the tree will be displayed in two different ways, one is grouped by the **MPI function** classes and the other one is grouped by the **File and Function**. The display for MPI can be switched by the option in the context menu of the MPI panel.

Grouped by MPI functions:

The screenshot displays a software interface with two main windows: 'DATA COLLECTION WINDOW' and 'SOURCE CODE WINDOW'.

DATA COLLECTION WINDOW: The 'Application Structure' panel shows a tree view of MPI calls. The root is 'MPI CALLS', which branches into 'mpi_comm_rank', 'mpi_comm_size', 'mpi_finalize', 'mpi_init', and 'mpi_irecv'. Under 'mpi_irecv', there are sub-entries for 'calc1.f' and 'calc2.f', each containing a list of MPI_IRECV_* function calls (e.g., mpi_irecv_100, mpi_irecv_104, etc.).

SOURCE CODE WINDOW: This window shows the source code for 'swim_omp.f'. The code includes MPI-related functions such as 'mpi_reduce' and 'CALL MPI_FINALIZE(ierr)'. Line 196 is highlighted, showing 'CALL CALC3Z'. Other lines include 'CALL CALC3' and various conditional statements and comments.

Grouped by File/Function:

The screenshot displays a software development environment with two main windows. The left window, titled "DATA COLLECTION WINDOW", shows a tree view of MPI calls grouped by file and function. The right window, titled "SOURCE CODE WINDOW", shows the corresponding Fortran source code for the selected function.

DATA COLLECTION WINDOW

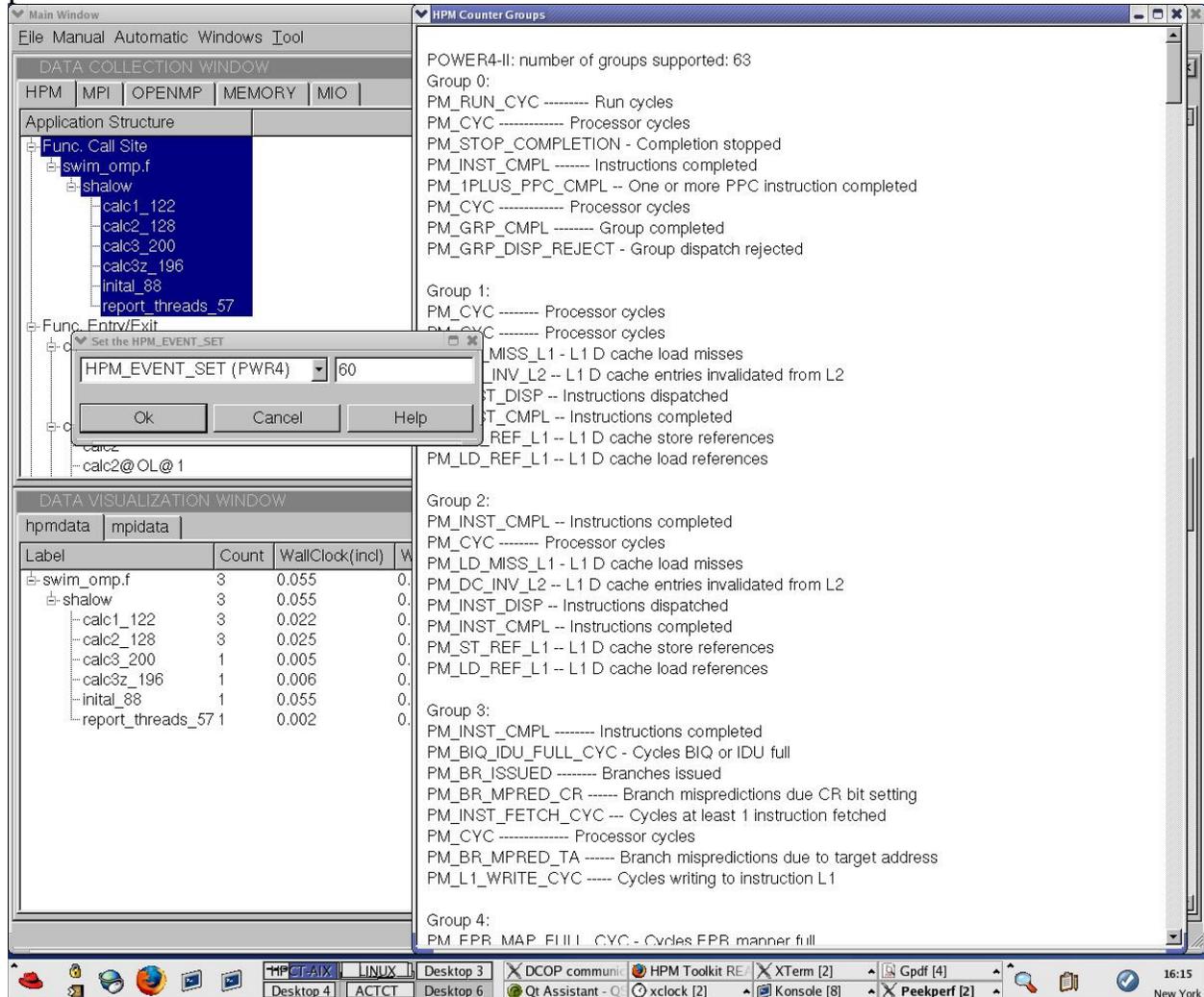
Application Structure

- MPI CALLS
 - calc1.f
 - calc1
 - mpi_irecv_100
 - mpi_irecv_104
 - mpi_irecv_106
 - mpi_irecv_132
 - mpi_irecv_134
 - mpi_irecv_138
 - mpi_irecv_140
 - mpi_irecv_33
 - mpi_irecv_41
 - mpi_irecv_43
 - mpi_irecv_98
 - mpi_isend_110
 - mpi_isend_112
 - mpi_isend_116
 - mpi_isend_118
 - mpi_isend_145
 - mpi_isend_147
 - mpi_isend_151
 - mpi_isend_153
 - mpi_isend_37
 - mpi_isend_47
 - mpi_isend_49
 - mpi_waitall_122
 - mpi_waitall_157
 - mpi_waitall_53
 - calc2.f
 - calc2
 - mpi_irecv_102
 - mpi_irecv_106
 - mpi_irecv_108
 - mpi_irecv_133
 - mpi_irecv_135
 - mpi_irecv_139
 - mpi_irecv_32
 - mpi_irecv_34
 - mpi_irecv_44

SOURCE CODE WINDOW

```
swim_omp.f | calc1.f | calc2.f | calc3.f | report_threads.f
mpi_reduce(ucheck,tucheck,1,MPI_DOUBLE_PRECISION,
165     1 MPI_SUM,0,MPI_COMM_WORLD,ierr)
166     cALL
mpi_reduce(vcheck,tvcheck,1,MPI_DOUBLE_PRECISION,
167     1 MPI_SUM,0,MPI_COMM_WORLD,ierr)
168     C ***
169     C ***
170     if(MASTER.eq.taskid)then
171     WRITE(6,350) NCYCLE,PTIME
172     endif
173     if(MASTER.eq.taskid)then
174     WRITE(6,366) tPCHECK, tUCHECK, tvCHECK
175     endif
176     366  FORMAT(/,
177     *      ' Pcheck = ',E12.4/,
178     *      ' Ucheck = ',E12.4/,
179     *      ' Vcheck = ',E12.4/)
180
181     370  CONTINUE
182     C    TEST FOR END OF RUN
183     IF(NCYCLE .GE. ITMAX) THEN
184     t2 = rtc()
185     if(taskid.eq.0)print*,'Wall clock time=',T2-T1
186     C    call f_hpmterminate( taskid )
187     CALL MPI_FINALIZE(ierr)
188     STOP
189     ENDIF
190     C
191     C    TIME SMOOTHING AND UPDATE FOR NEXT CYCLE
192     C
193
194     IF(NCYCLE .LE. 1) THEN
195     C    call f_hpmstart( 4, "Calc3z" )
196     CALL CALC3Z
197     C    call f_hpmstop( 4 )
198     ELSE
199     C    call f_hpmstart( 3, "Calc3" )
200     CALL CALC3
```

If you want to set up some *environment variables*, you can do so by selecting the menu point *Automatic->Set the Environment Variable*. For HPM, if you want to change the event group, you can bring up the context menu in the HPM panel in the Tool. Select the “*Set the Counter Group*” option. The popup window will allow you to change the event group. You can view the counter group information by clicking the *Help* button. It will show you the counter information in the current platform.



Peekperf Simple IDE

In the Source Code Window, the context menu of the file allows you to open it and edit using the vi editor. After the file is changed and saved, you can reload it back to the Source Code Window. Peekperf will detect if the change has occurred and ask you if you want to reload it. You can also edit any file by selecting the menu point **Manual->Edit**. If you want to compile your application without switching to other terminal, you can do so by selecting the menu point **Manual->Compile**. It will pop up a window to ask you the command. You can run any executable by selecting the menu point **Manual->Run**.

The screenshot displays the Peekperf Simple IDE interface. The main window is titled "Main Window" and contains several panes:

- DATA COLLECTION WINDOW:** Shows the "Application Structure" tree with "MPI CALLS" expanded to "calc1.f". Below it is a table of performance data.
- SOURCE CODE WINDOW:** Displays the source code for "swim_omp.f", "calc1.f", "calc2.f", "calc3.f", and "report_threads.f". The code is highlighted in cyan, and a context menu is visible over it with options like "Edit" and "Close this File".
- DATA:** A table showing performance metrics for various components.

Label	Count	WallClock(incl)	WallClock(excl)
swim_omp.f	3	0.055	0.055
shallow	3	0.055	0.055
calc1_122	3	0.022	0.022
calc2_128	3	0.025	0.025
calc3_200	1	0.005	0.005
calc3z_196	1	0.006	0.006
initial_88	1	0.055	0.055
report_threads_57	1	0.002	0.002