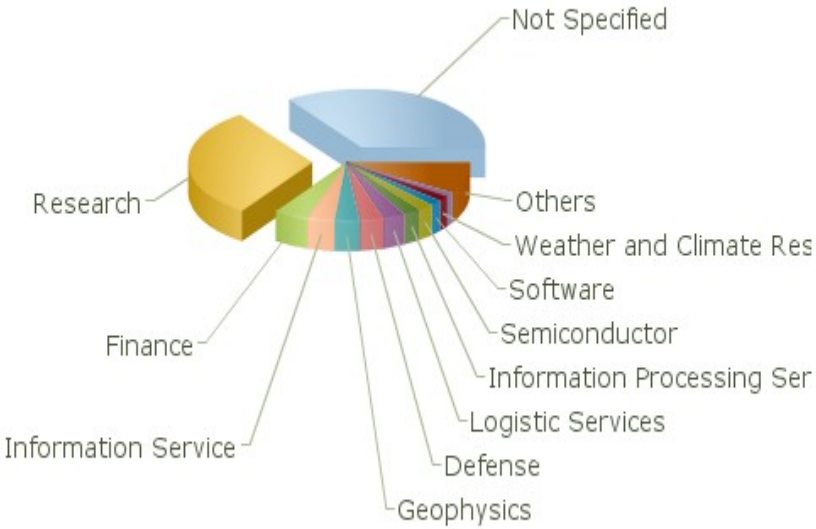
IBM High Performance Computing Toolkit

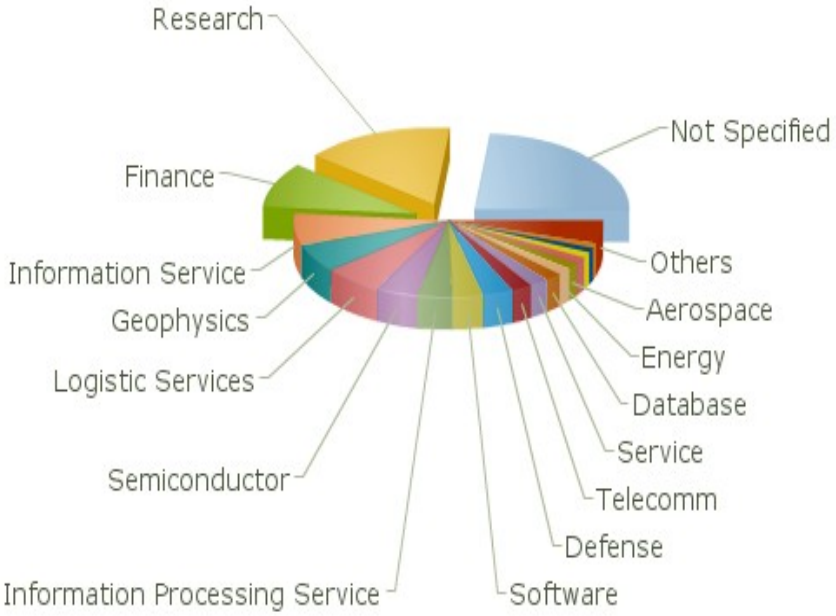
Pidad D'Souza (pidsouza@in.ibm.com)
IBM, India Software Labs

Top 500 : Application areas (November 2011)

Systems



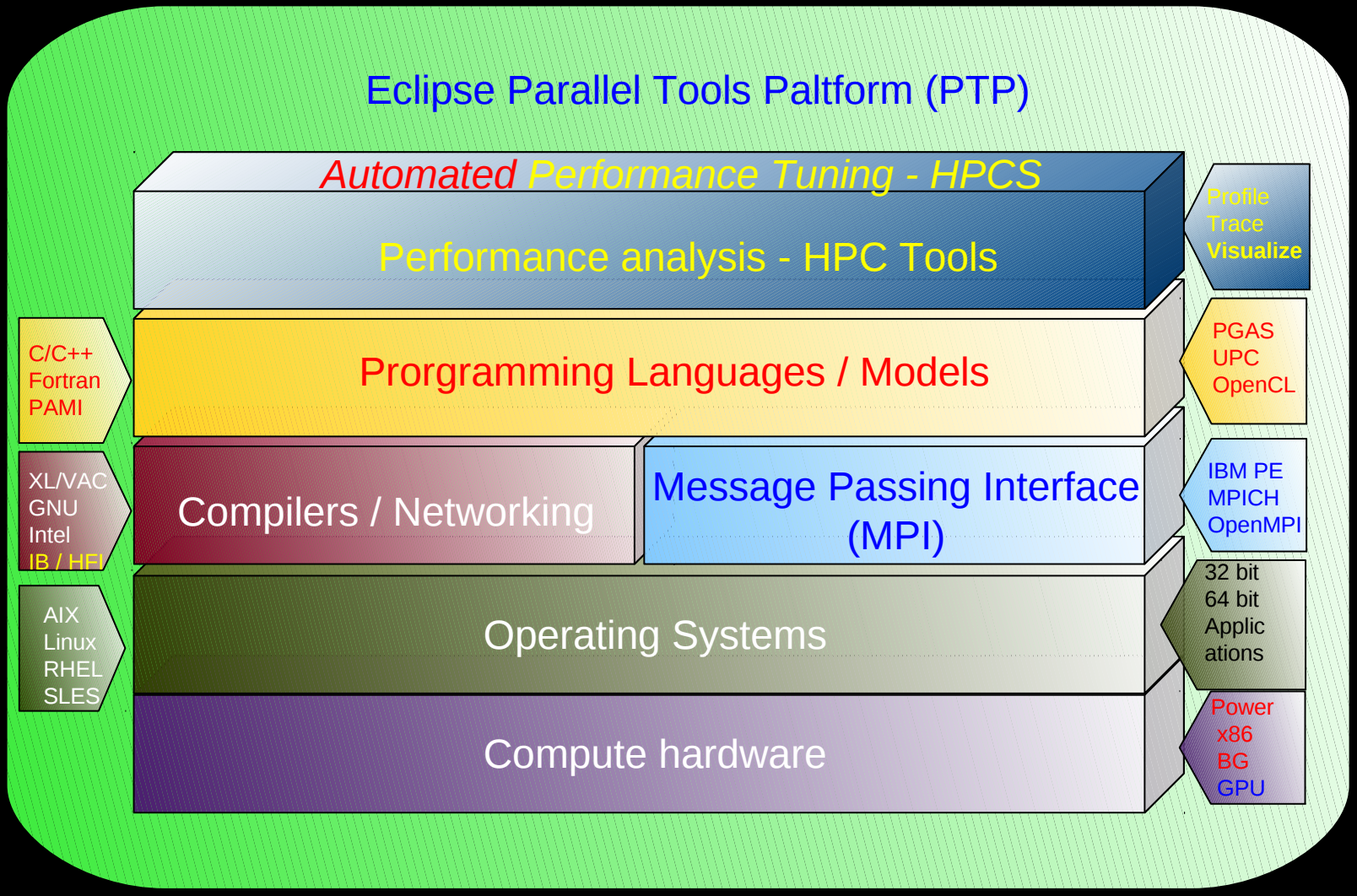
Performance



Source : http://www.top500.org/charts/list/34/apparea

2

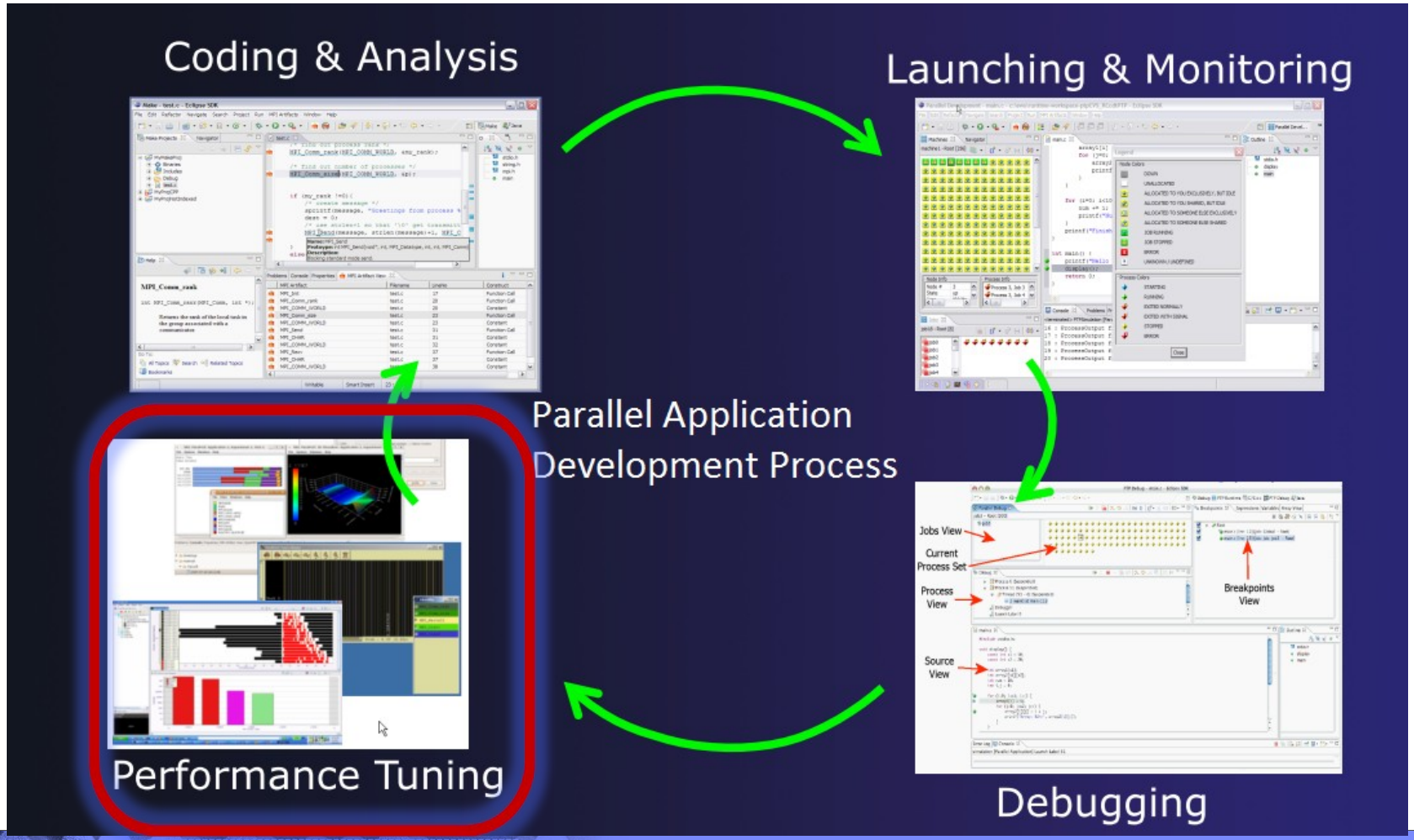
IBM HPC Toolkit – Integrated Picture and dimensions



3

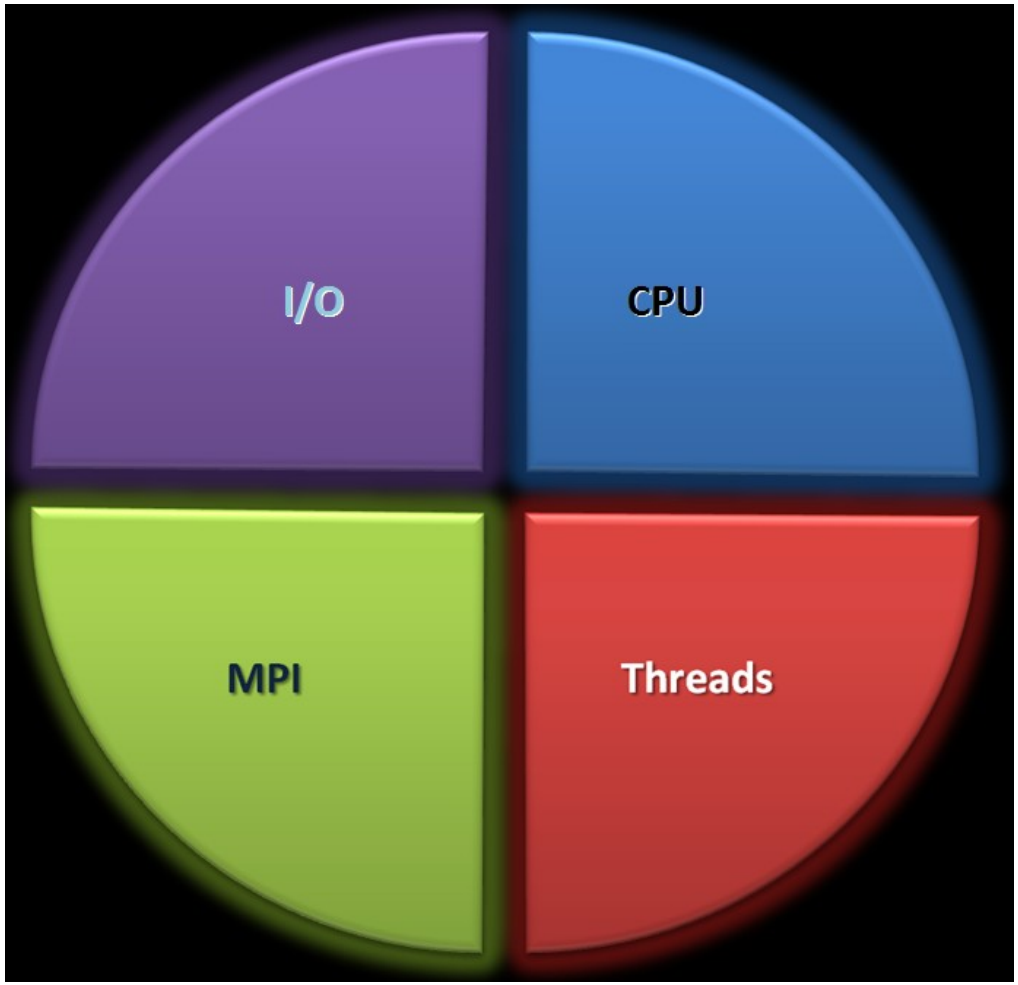
IBM HPC Toolkit

An integrated framework to assist application performance tuning



4

The Performance Pie



Performance Dimensions

CPU Performance

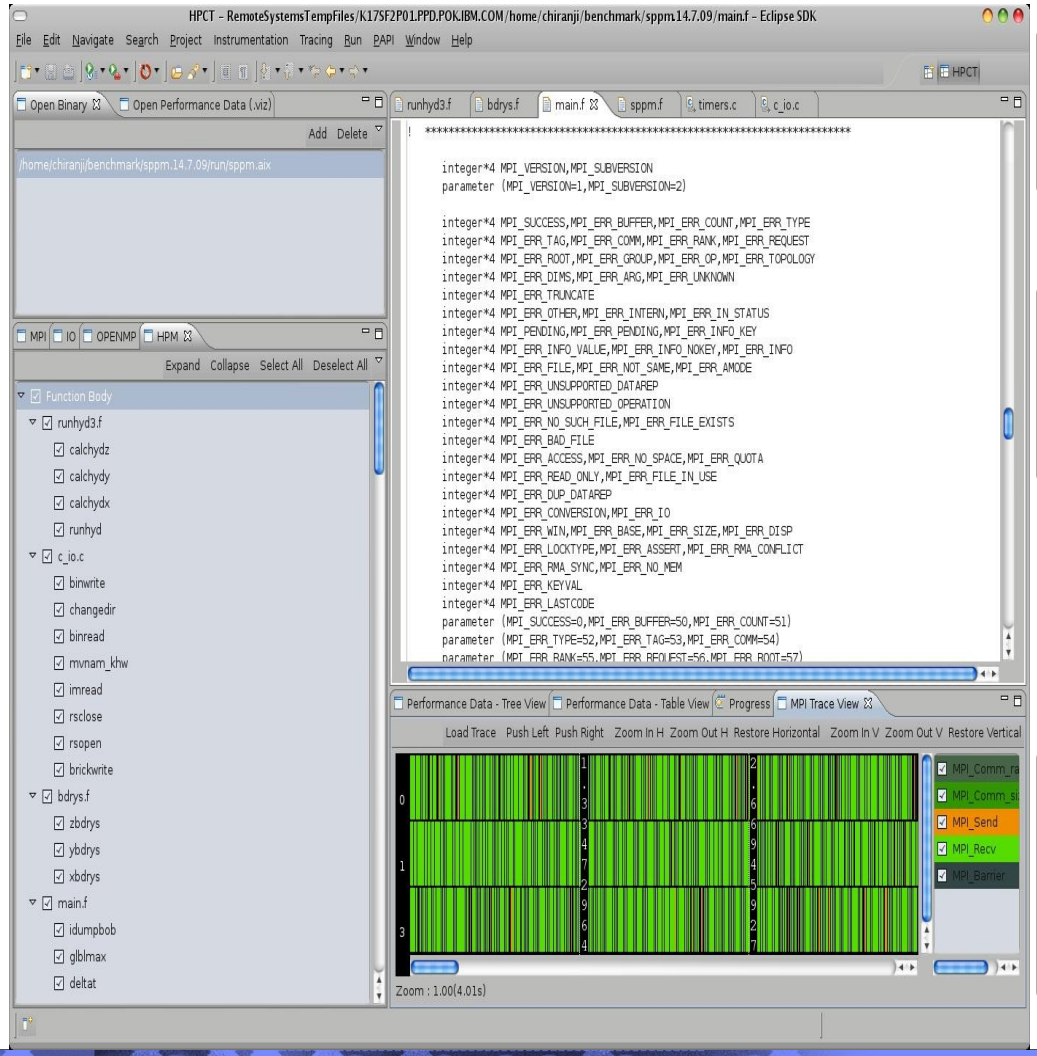
MPI Performance

Threading Performance

I/O Performance

5

IBM HPC Toolkit – usability and features



- IBM System p, Blue Gene L/P, *System x, Hybrid*
- AIX and Linux

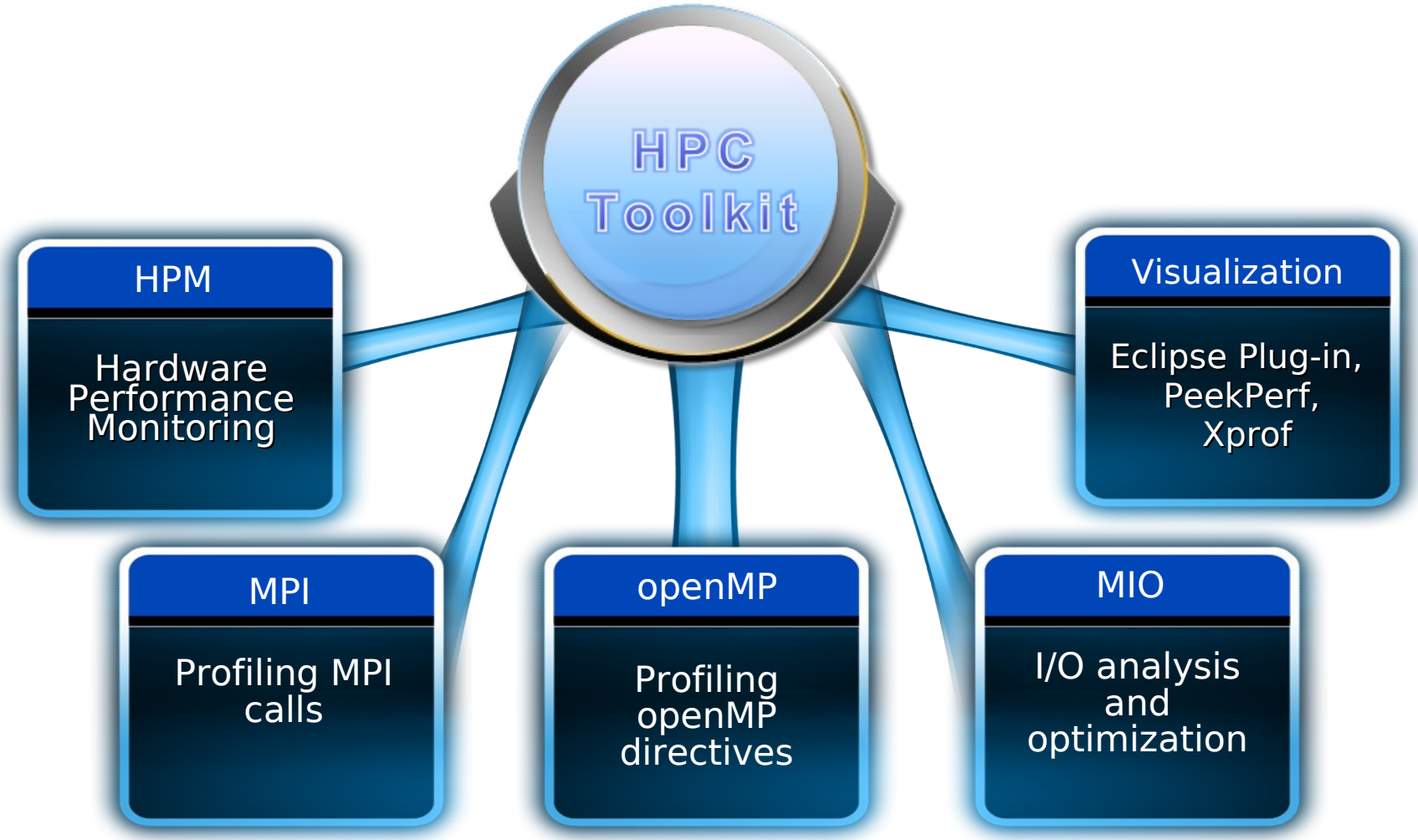
- Operates on the application binary and generate results in terms of source level symbols, with full source code traceback capability

- Presents the results in an easy to use and intuitive GUI

- Can be used as:
  - A collection of command line, visualization utilities
  - A plug-in into the Eclipse IDE
  - As a collection of libraries which are used with the application

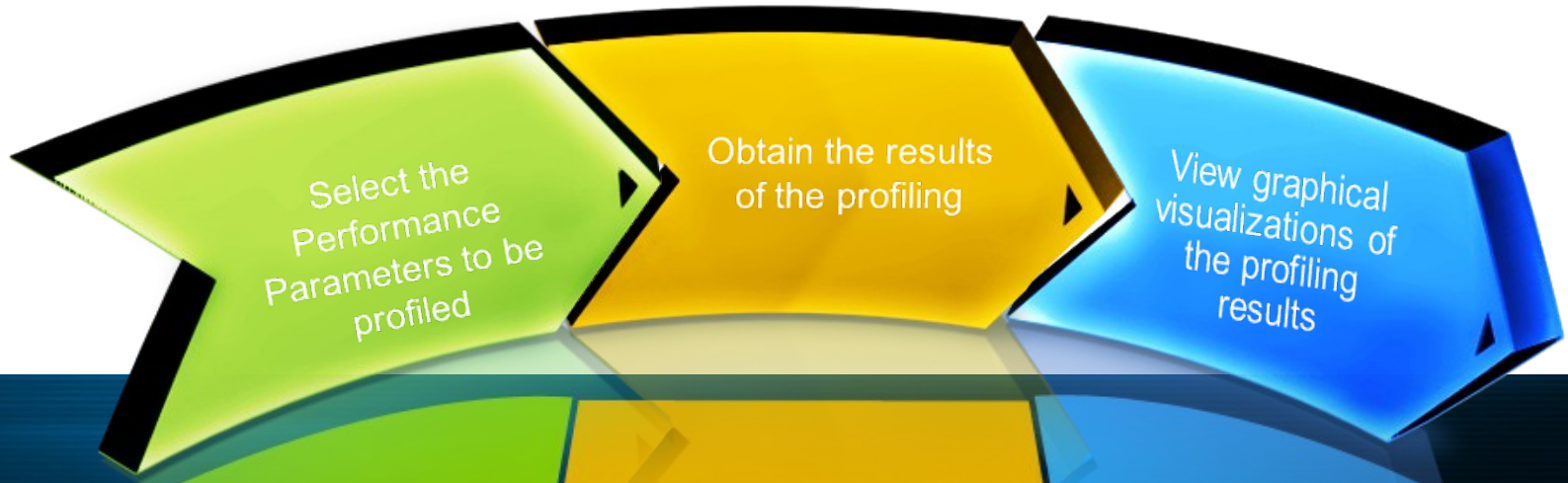
6

IBM HPC Toolkit Components



7

Using the IBM HPC Toolkit



HPM, OpenMP, MPI, MIO

One or more performance metrics can be chosen for a desired region of code

Binary Instrumentation or Profile Library calls

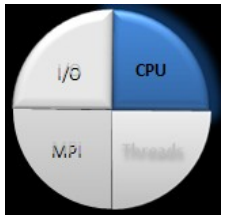
Running the instrumented binary produces profiling results

Eclipse Plug-In, PeekPerf, Xprof

Rendering runtime state within a visualization of the workflow control flow

8

CPU Performance – HPM module



Utilizes Performance Counters – (*special purpose registers built into the processor to store the counts of hardware-related activities*)

Facilitates low-level performance analysis and tuning

Updated at each cycle and low overhead

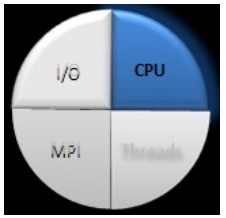
HPM: Hardware Counters Examples

- Cycles
- Instructions
- Floating point instructions
- Integer instructions
- Load/stores
- Cache misses
- TLB misses
- Branch taken / not taken
- Branch mispredictions

• Useful derived metrics

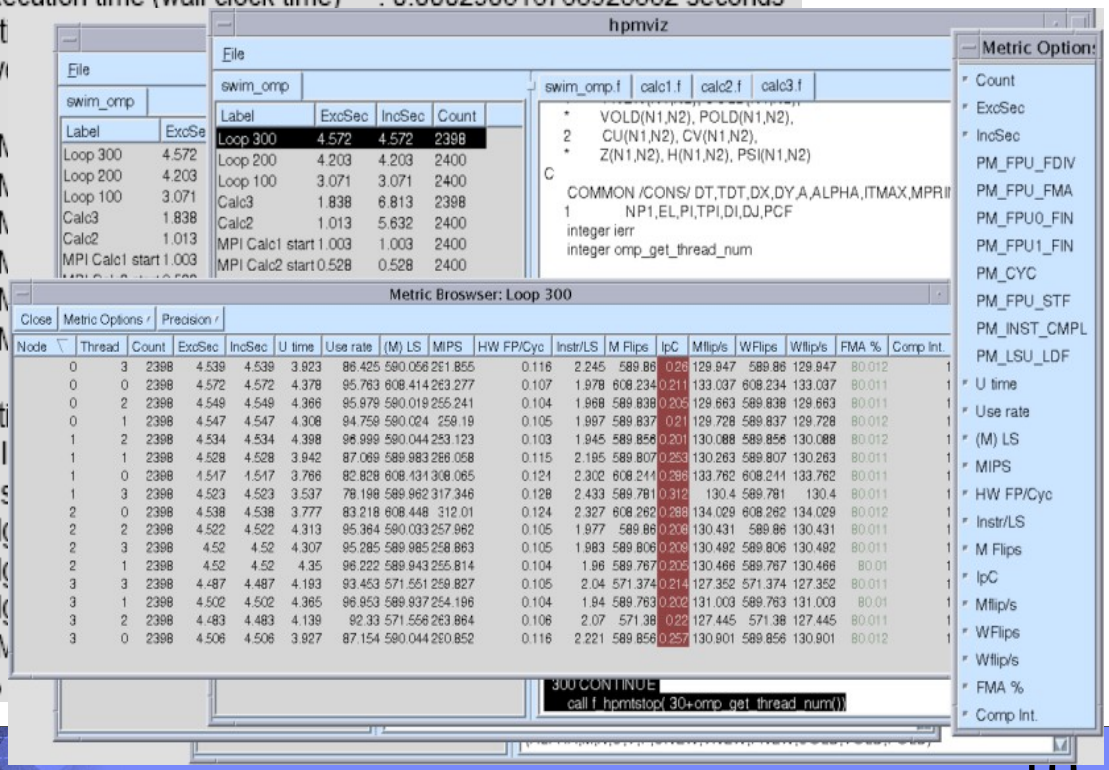
- ✓IPC - instructions per cycle
- ✓Float point rate (Mflop/s)
- ✓Computation intensity
- ✓Instructions per load/store
- ✓Load/stores per cache miss
- ✓Cache hit rate
- ✓Loads per load miss
- ✓Stores per store miss
- ✓Loads per TLB miss
- ✓Branches mispredicted %

CPU Performance



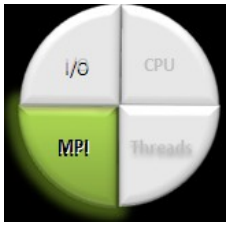

```
libhpm v3.2.1 (IHPCT v2.2.0) summary

######## 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
Init
Ov
PM
PM
PM
PM
PM
PM
Uti
MI
Ins
Alg
Alg
Alg
FM
%
```



10

MPI Profiling



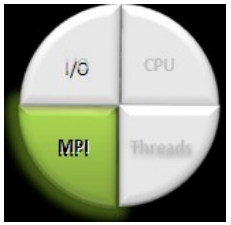
Profiles MPI calls using the PMPI interface

- Start / stop timer
- PMPI equivalent function

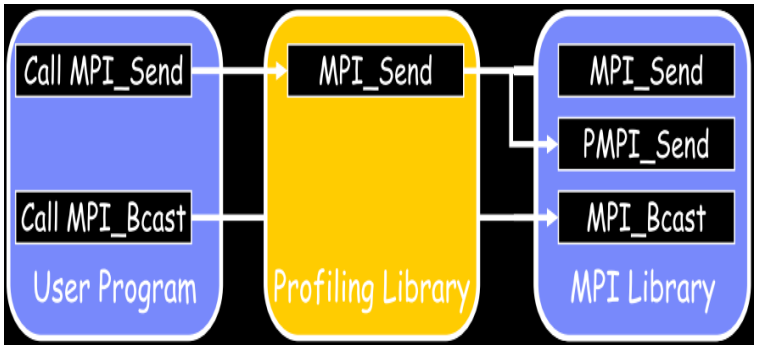
MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Comm_size	1 (1)	0 ... 4	0	1E-07
MPI_Comm_rank	1 (1)	0 ... 4	0	1E-07
MPI_Isend	2 (1)	0 ... 4	3	0.000006
MPI_Isend	2 (2)	5 ... 16	12	1.4E-06
MPI_Isend	2 (3)	17 ... 64	48	1.3E-06
MPI_Isend	2 (4)	65 ... 256	192	1.3E-06
MPI_Isend	2 (5)	257 ... 1K	768	1.3E-06
MPI_Isend	2 (6)	1K ... 4K	3072	1.3E-06
MPI_Isend	2 (7)	4K ... 16K	12288	1.3E-06
MPI_Isend	2 (8)	16K ... 64K	49152	1.3E-06
MPI_Isend	2 (9)	64K ... 256K	196608	1.7E-06
MPI_Isend	2 (A)	256K ... 1M	786432	1.7E-06
MPI_Isend	1 (B)	1M ... 4M	1048576	9E-07

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

MPI Profiling

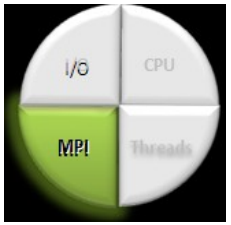



```
-----------------------------------------------------------
MPI Routine            #calls     avg. bytes      time(sec)
-----------------------------------------------------------
MPI_Comm_size               3            0.0         0.000
MPI_Comm_rank           12994            0.0         0.016
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.*
```



12

MPI Trace

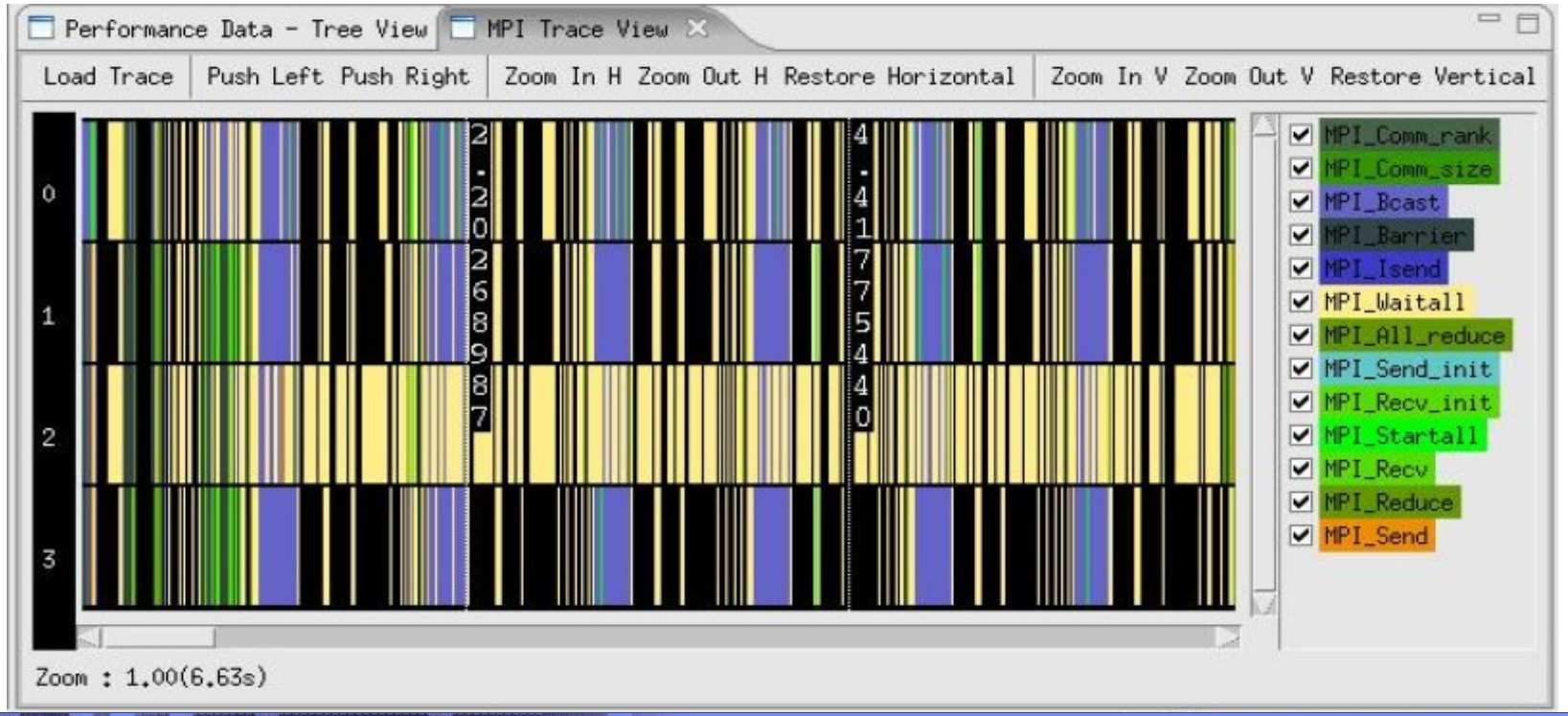


Captures timestamped data for MPI calls

Provides a colour-coded trace of execution

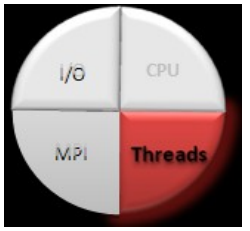
Provides source traceback

Useful in identifying load-balancing issues



13

OpenMP profiling



Uses the POMP OpenMP Monitoring Interface

Groups of events identified for profiling:

- OpenMP constructs and directives/pragmas
- OpenMP API calls
- User functions and regions

Generates profile describing overheads and time spent by each thread in:

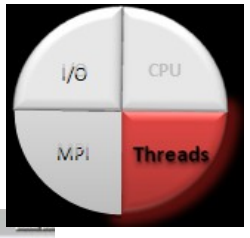
- Parallel regions
- OpenMP loops inside a parallel region
- User defined functions

Profile data is presented in the form of an XML file => can be visualized with visualization utilities

```
 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:  }
```


14

OpenMP profiling



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.(
loop_1379	1	0.031769	0.031769	0.198128	13.846	0.(
func_rdparam_174	1	0.01605	0.01605	0	0	0
loop_855	1	0.013306	0.013306	0.224878	342.237	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 Broswser: loop_1125

Close | Metric Options | Precision

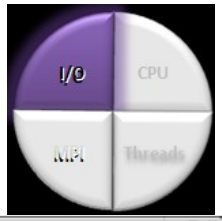
Task	thread	Time in Master	TT: Thread Time	CT: Computation Time	%Imbalance	TO = TT - CT	%TO (Barrier)	%TO (RTL)
0	0	13.3219	13.3219	13.3211	0	0.000723	0.000275	0.005156
0	1	0	15.8615	15.861	19.0664	0.000504	0.000108	0.003679
0	2	0	21.0295	21.029	57.8616	0.000514	0.000103	0.003753
0	3	0	24.4342	24.4338	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,
                                              _mnybdy,
                                              _mxybdy)
                                          _xp, msg_ym, msg_yp,
                                              _mnybdy,
                                              _mxybdy)
                                          (2,4)).and.
                                          0)) then
                                          _xp, msg_ym, msg_yp,
  &        msg_mnzbdy, msg_mnxbdy, msg_mnybdy,
  &        msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
     call bdry3o1r(dddo, nz, nx, ny, 5,
```


15

Modular I/O



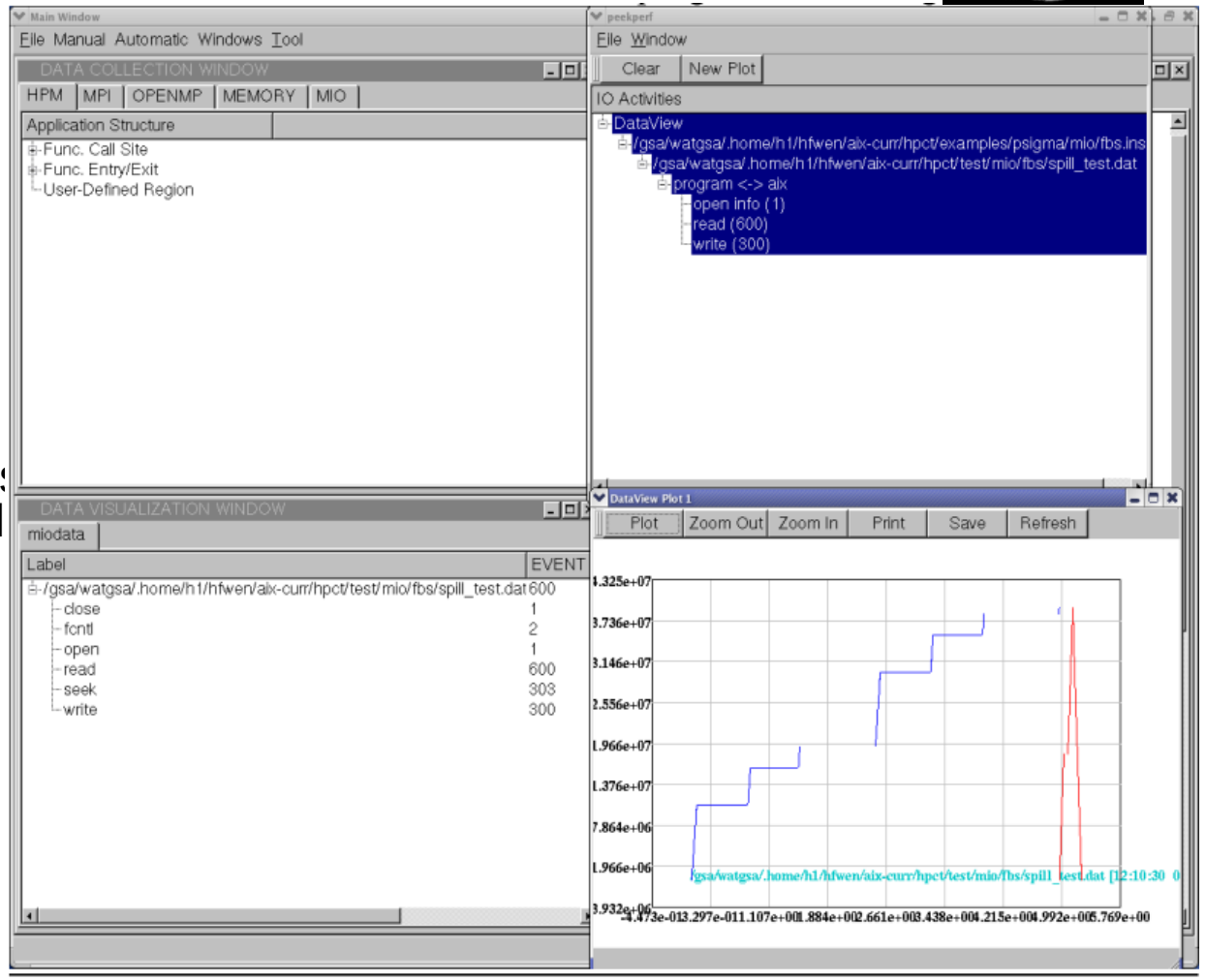
Application level optimization for I/O

Facilitates *analysis* and *tuning* of the I/O at the **application level**

When an application exhibits the I/O pattern of sequential reading of large files, MIO

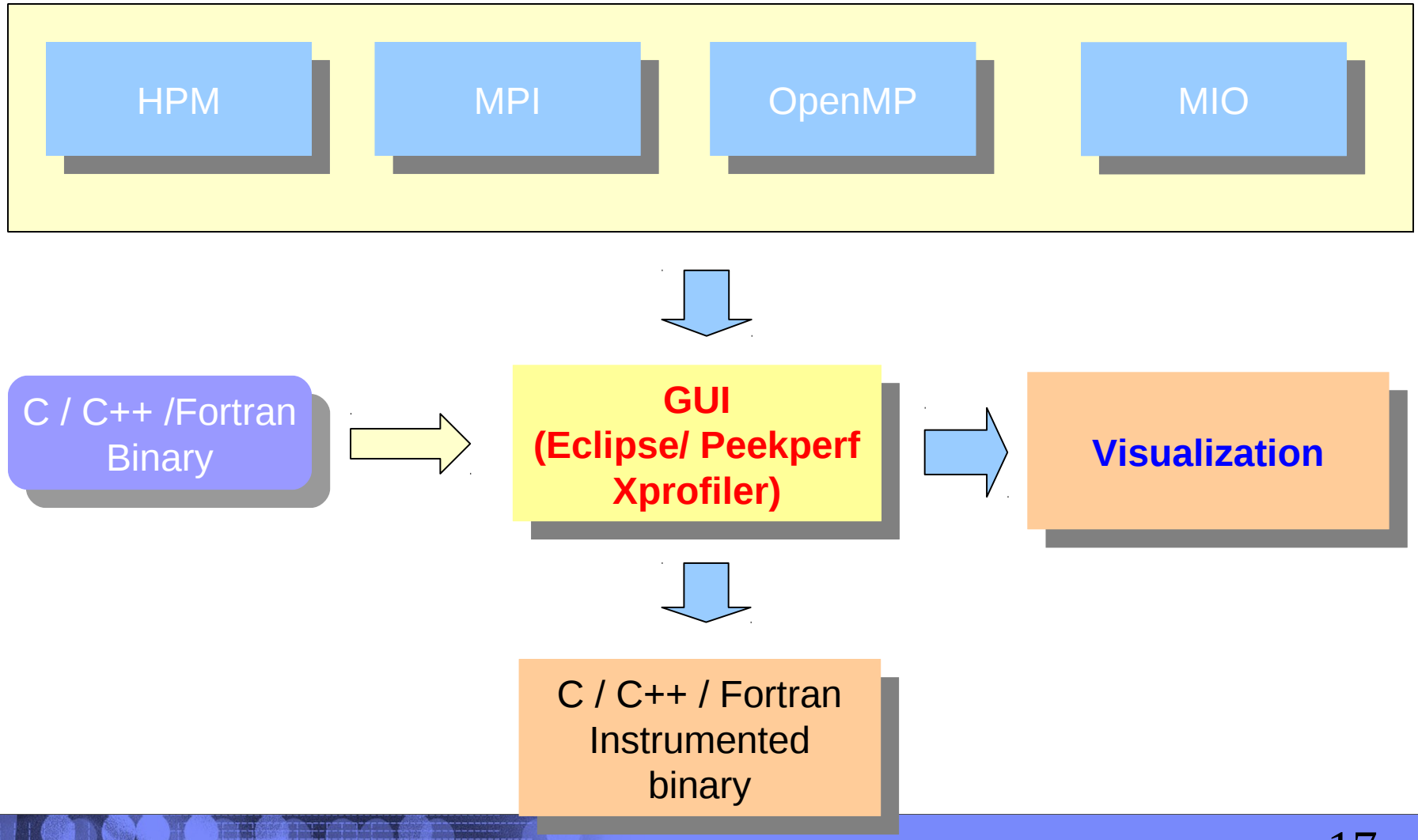
Detects the behavior

Provides source code traceback

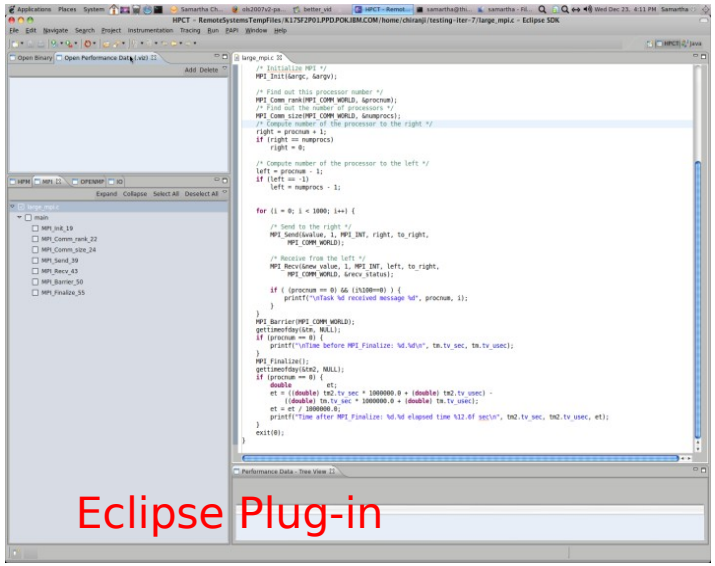


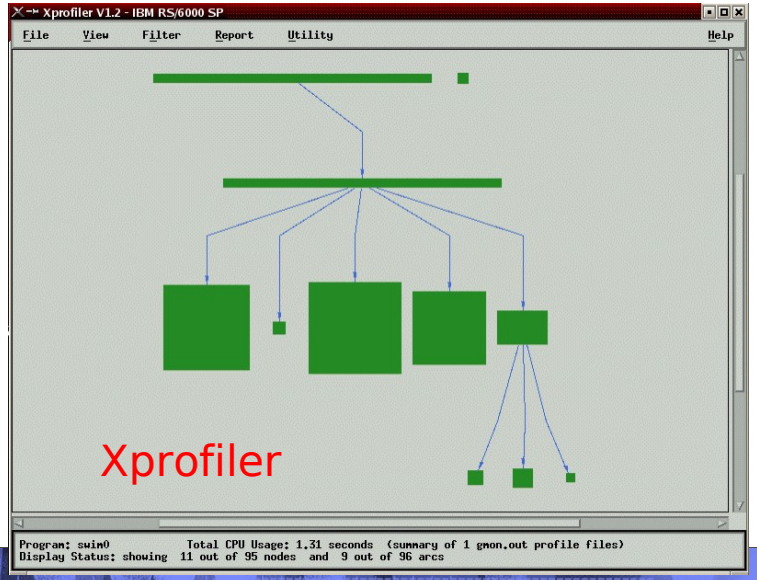
16

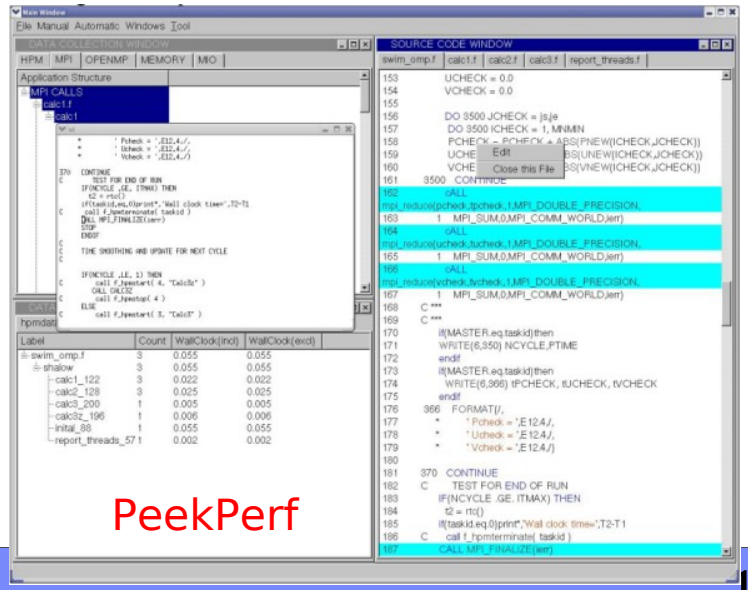
IBM HPC Toolkit – Work flow



Visualization





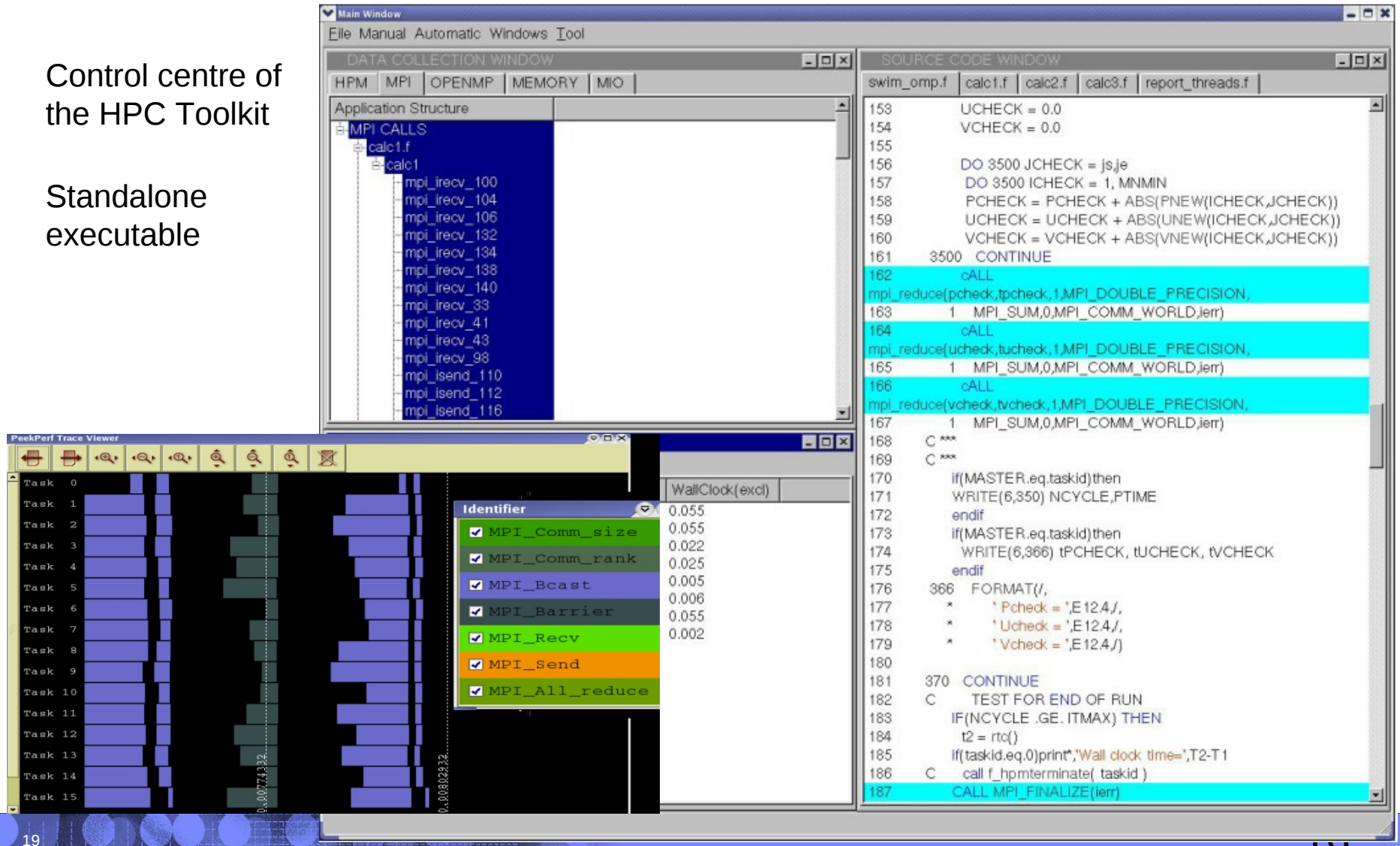


18

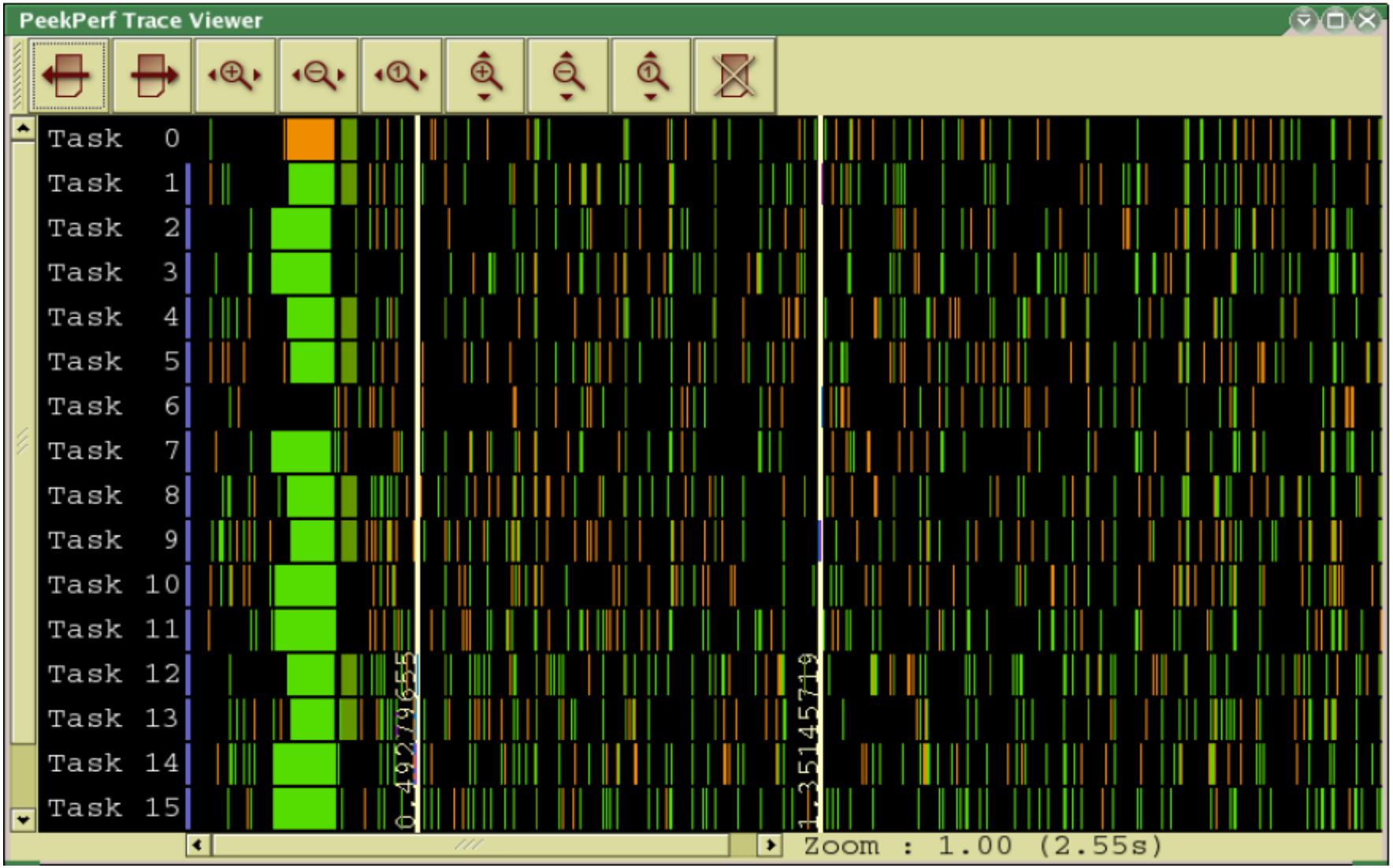
PeekPerf

Control centre of the HPC Toolkit

Standalone executable



PeekPerf


PeekPerf Trace Viewer
Task 0
Task 1
Task 2
Task 3
Task 4
Task 5
Task 6
Task 7
Task 8
Task 9
Task 10
Task 11
Task 12
Task 13
Task 14
Task 15
0.49279655
1.35145719
Zoom : 1.00 (2.55s)

20

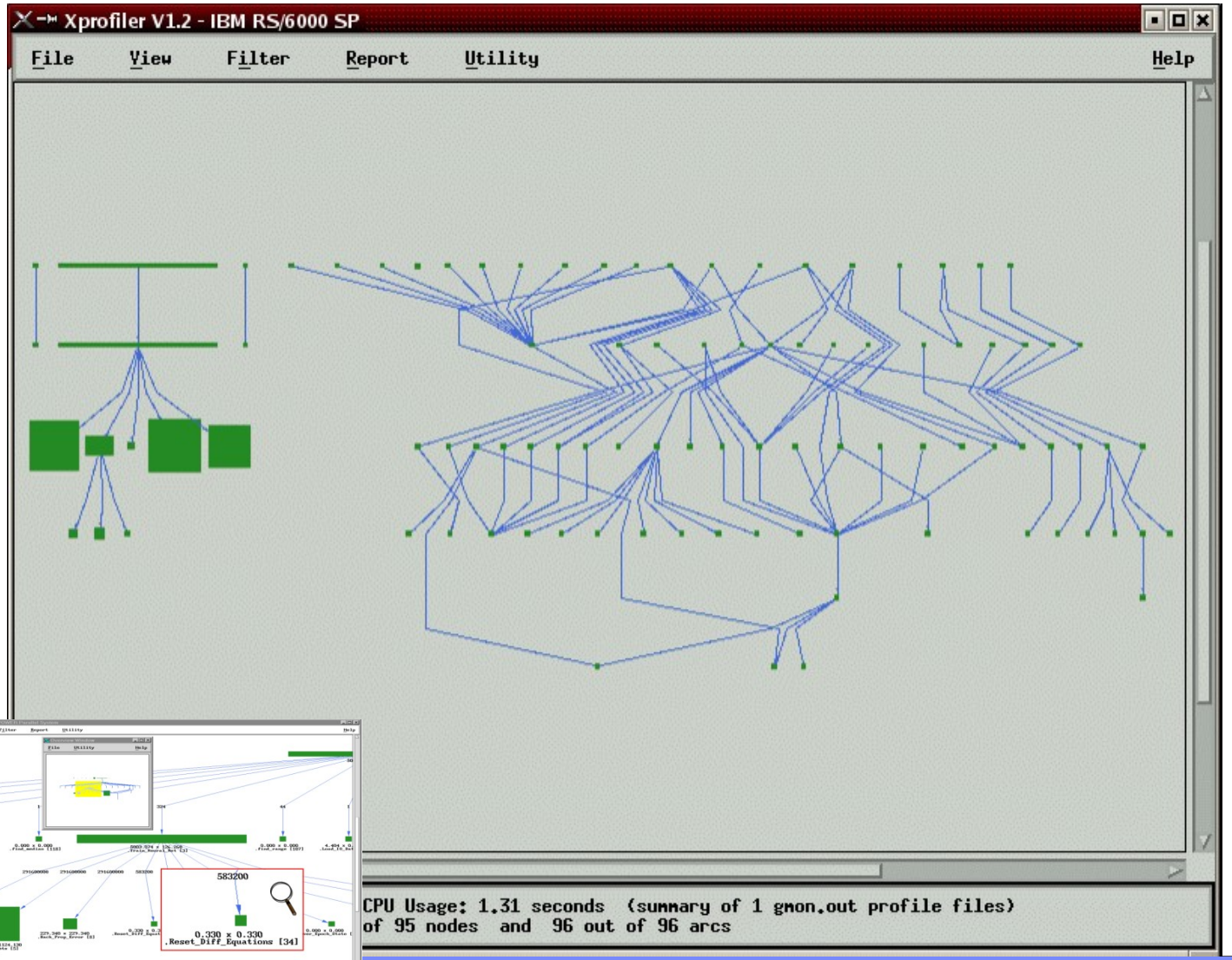
Xprofiler

CPU time profiling data

Visualization of profiled data

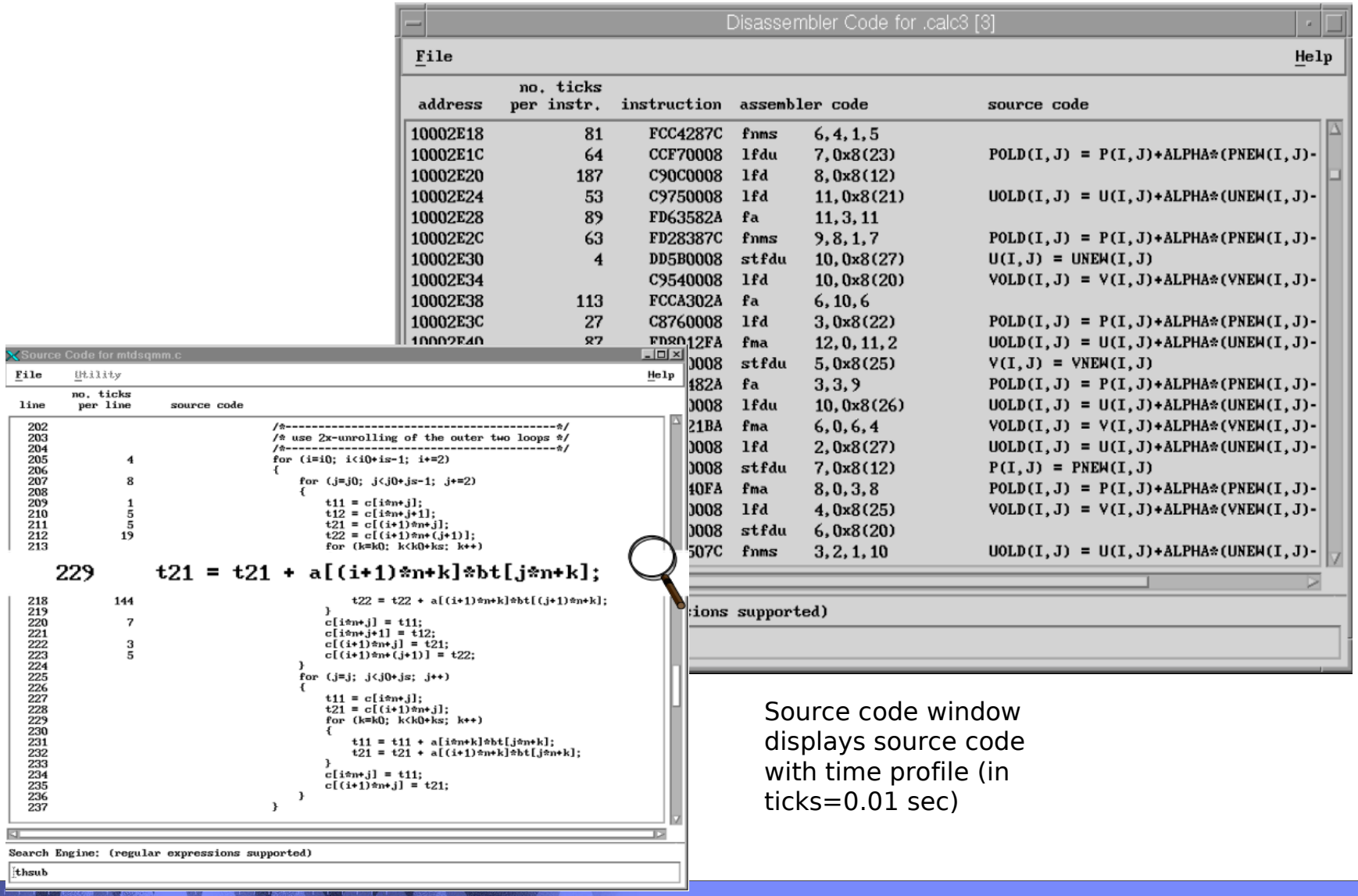
Profile description :

- Flat Profile
- Call Graph
- Profile Function
- Index Function
- Call Summary
- Library
- Statistics



21

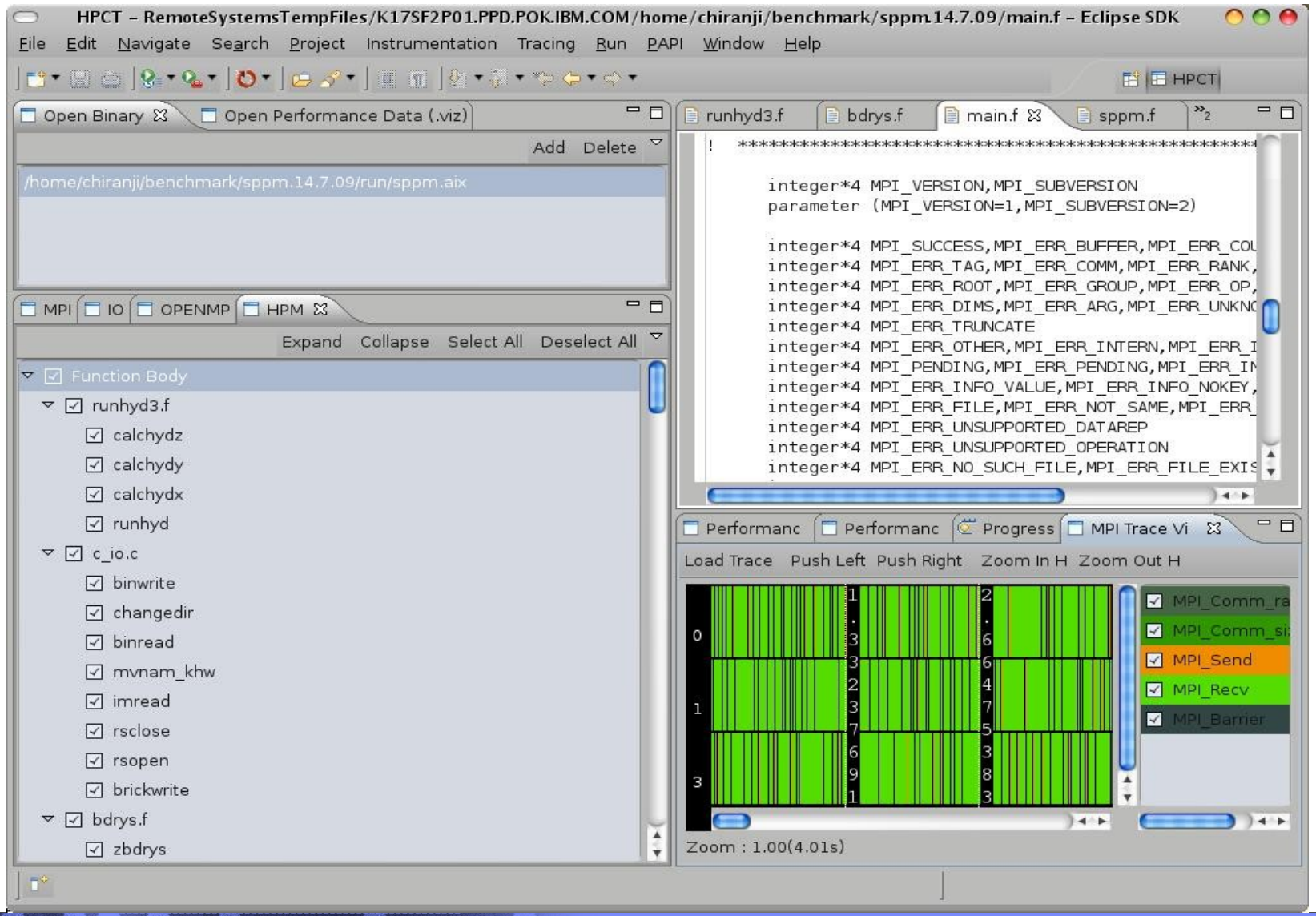
Xprofiler



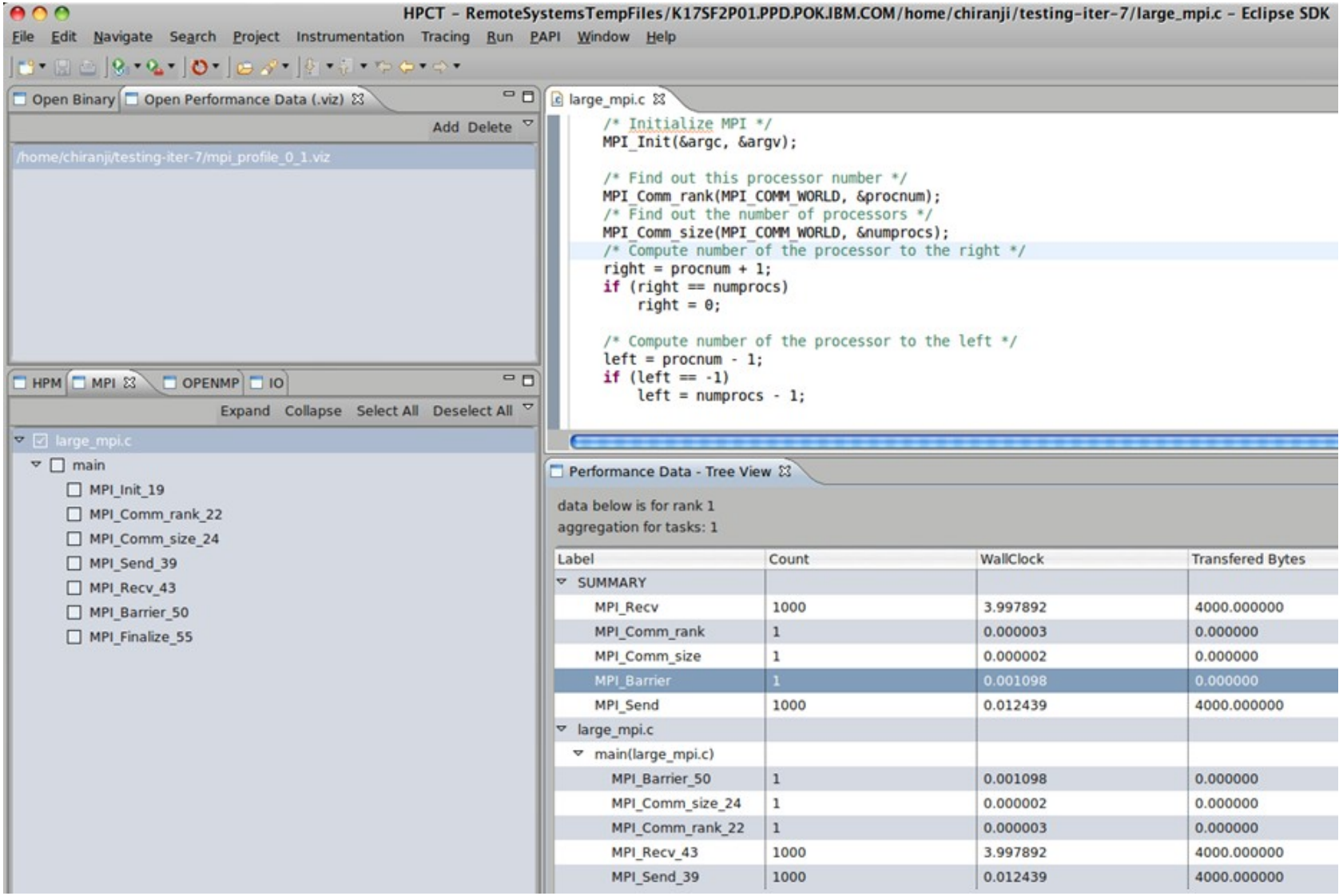
Source code window displays source code with time profile (in ticks=0.01 sec)

22

IBM HPC Toolkit Eclipse plug-in



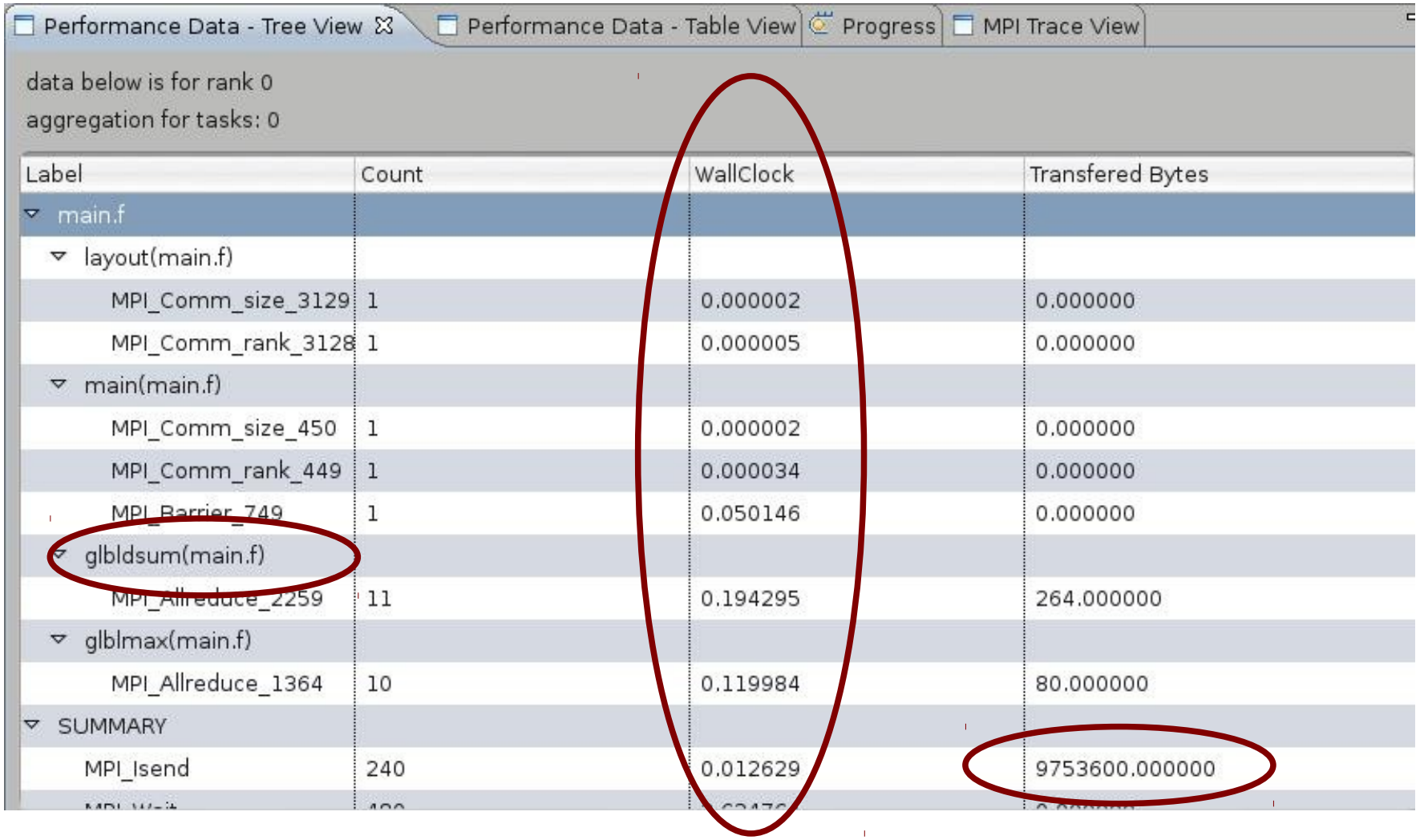
IBM HPC Toolkit Eclipse plug-in



Source level traceback of performance information

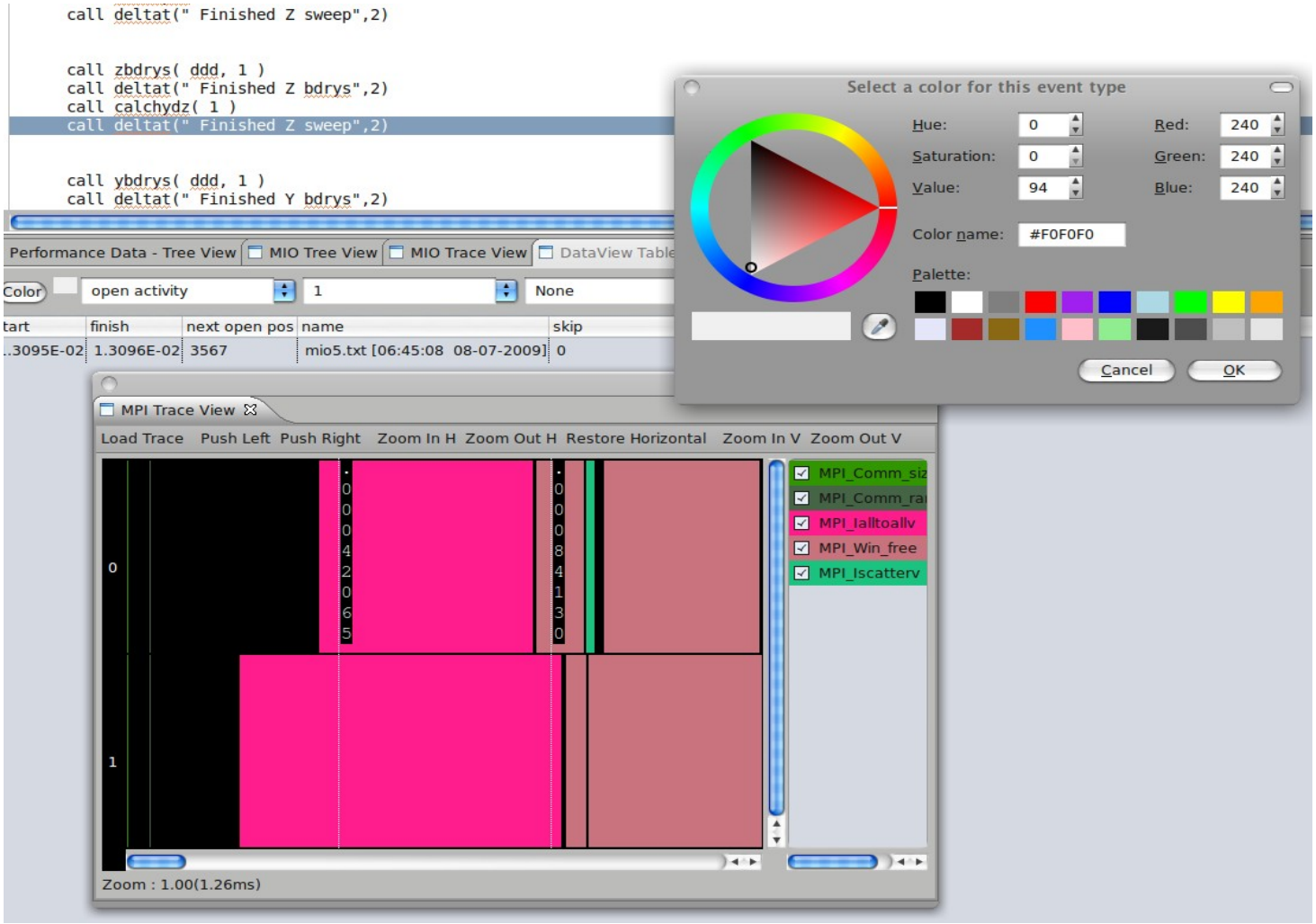
24

IBM HPC Toolkit Eclipse plug-in



25

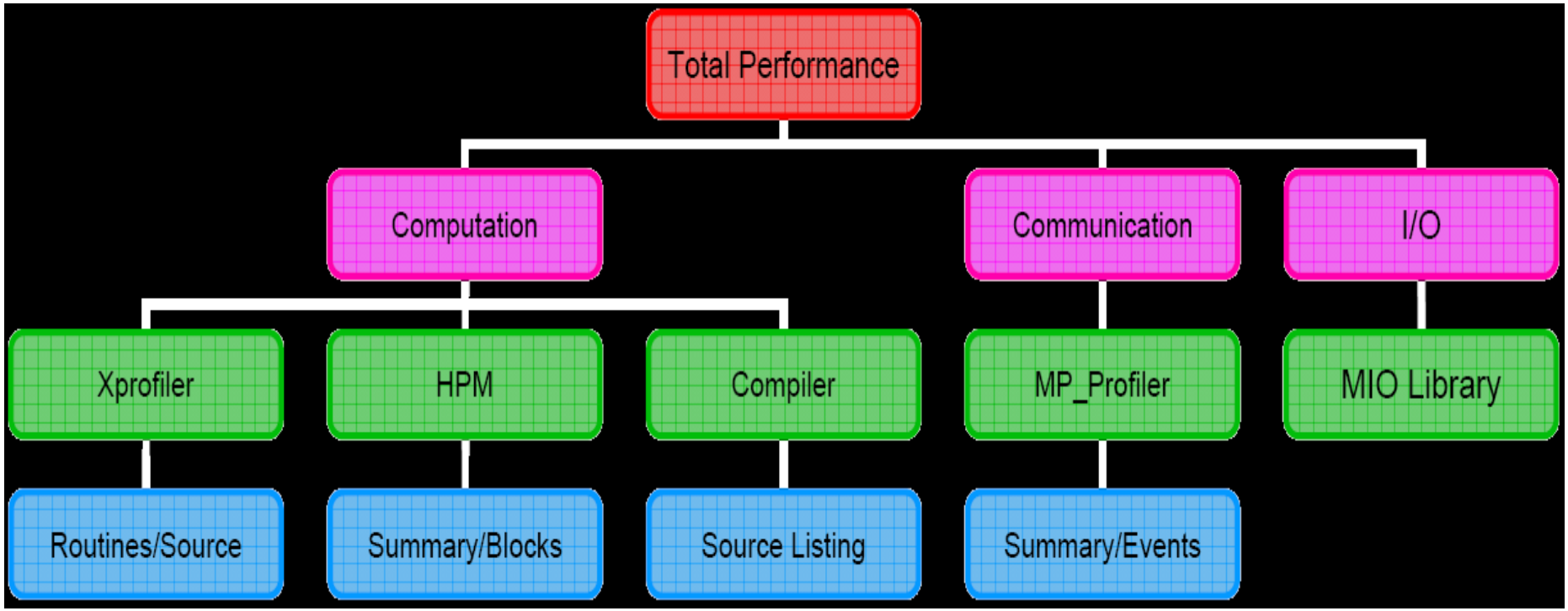
IBM HPC Toolkit Eclipse plug-in



Visualization of IBM HPCT in Eclipse Plug-in

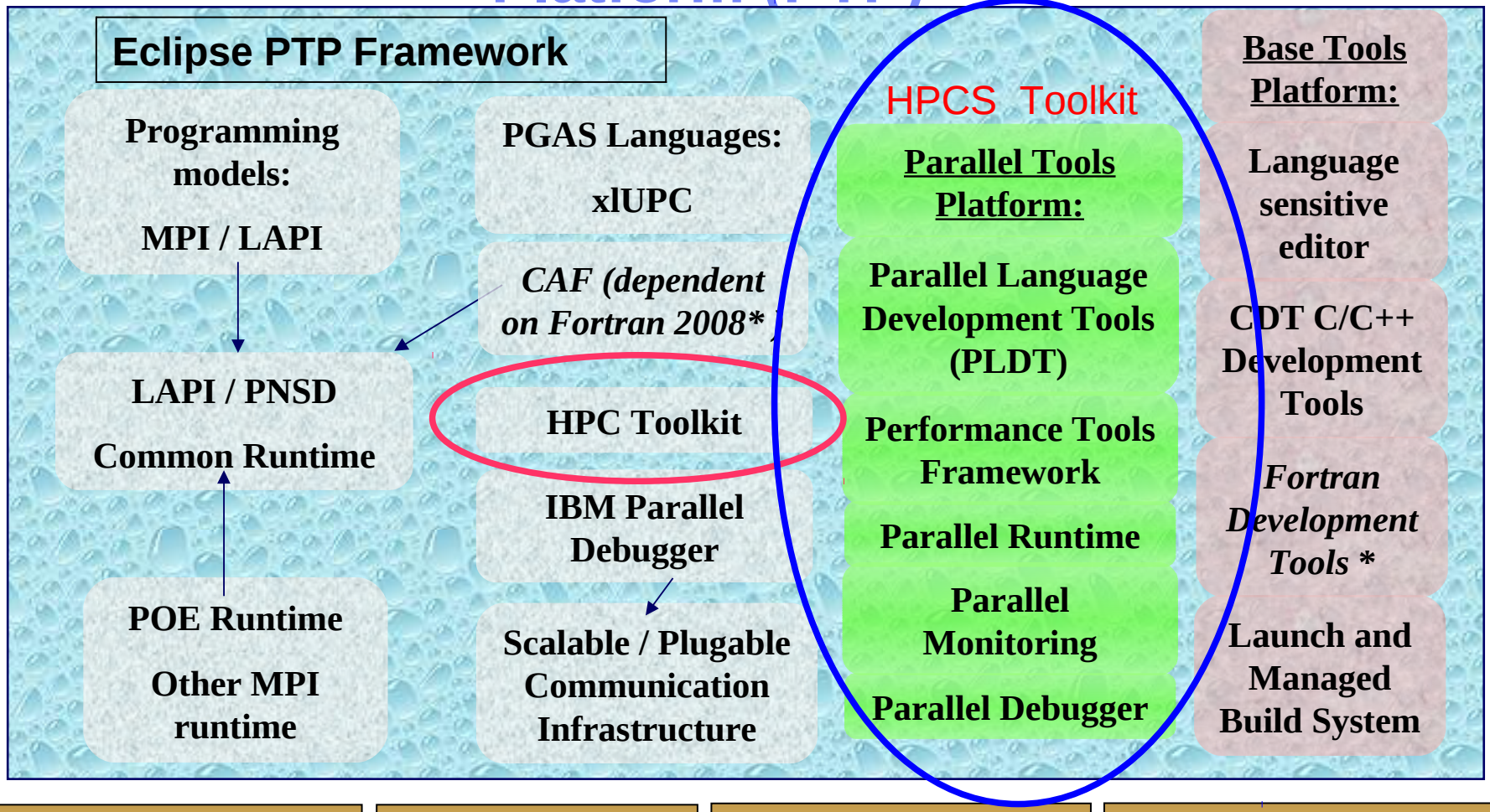
26

IBM HPC Toolkit in a nutshell



27

Direction of Migration – Eclipse Parallel Tools Platform (PTP)



Power Clusters: P5, P6, P7	x86 Clusters: Intel/AMD	Power Clusters: BlueGene	Hybrid Clusters: Accelerator

28

Summary

- 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) – for ***beginners*** to ***black-belts***
- Full source code traceback capability
- Dynamically activate/deactivate data collection and change what information to collect

29

Thank You