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**

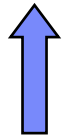
Message-Passing Performance:

- **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

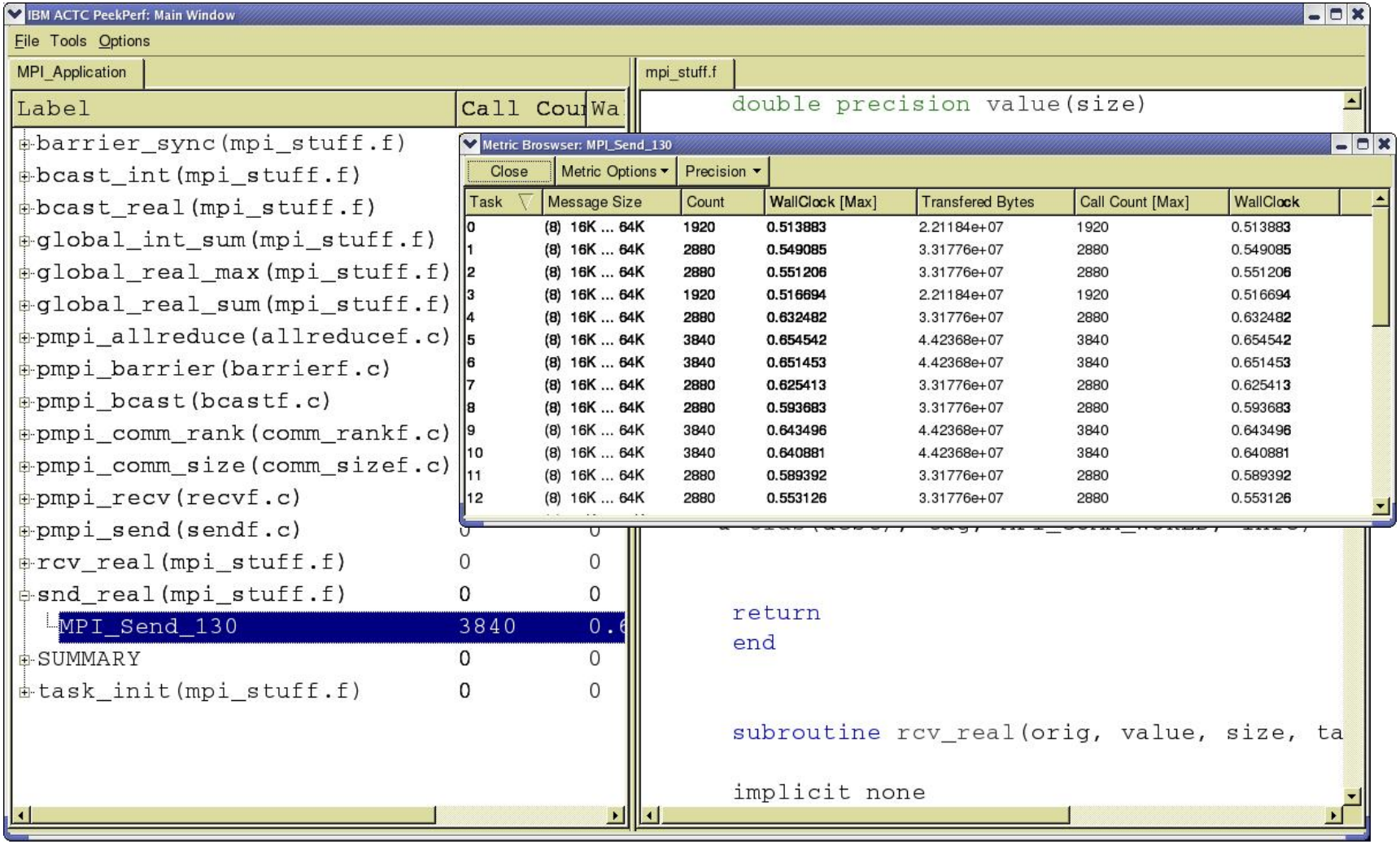
```
BGL=/bgl/BlueLight/ppcfloor
CC=$(BGL)/blrts-gnu/powerpc-bgl-blrts-gnu/bin/gcc
CFLAG= -I $(BGL)/bglsys/include
MPI_LIB= -L $(BGL)/bglsys/lib -lmpich.rts -lmsglayer.rts -lrts.rts -ldevices.rts
TRACE_LIB= -L $(MP_PROFILER) -lmpitrace.rts
BINUTILS_LIB= -L $(BINUTILS) -lbfd -liberty

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

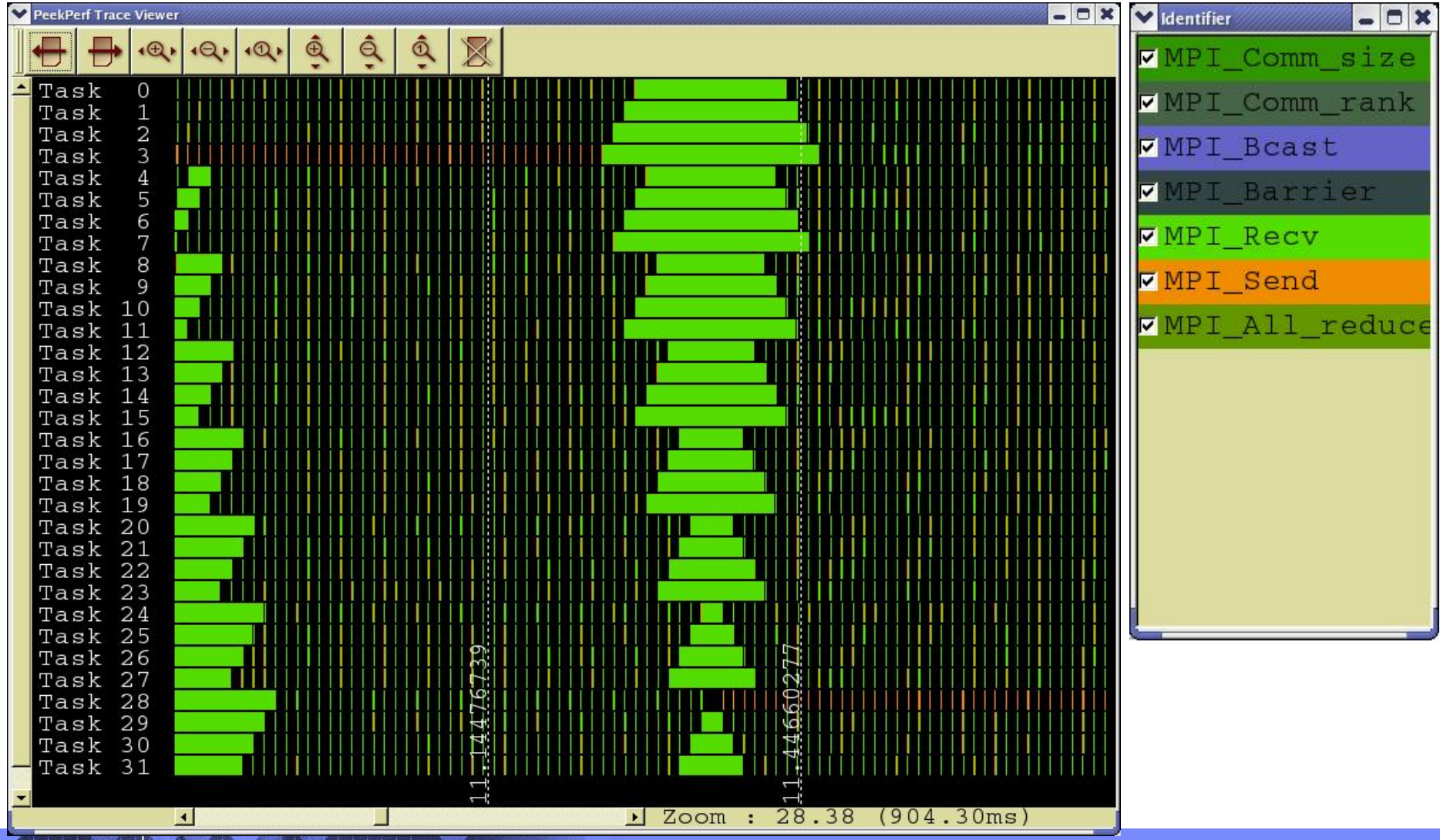


$(TRACE_LIB) has to precede $(MPI_LIB)

MP_Profiler Output with Peekperf



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_PERFILE**
    - 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

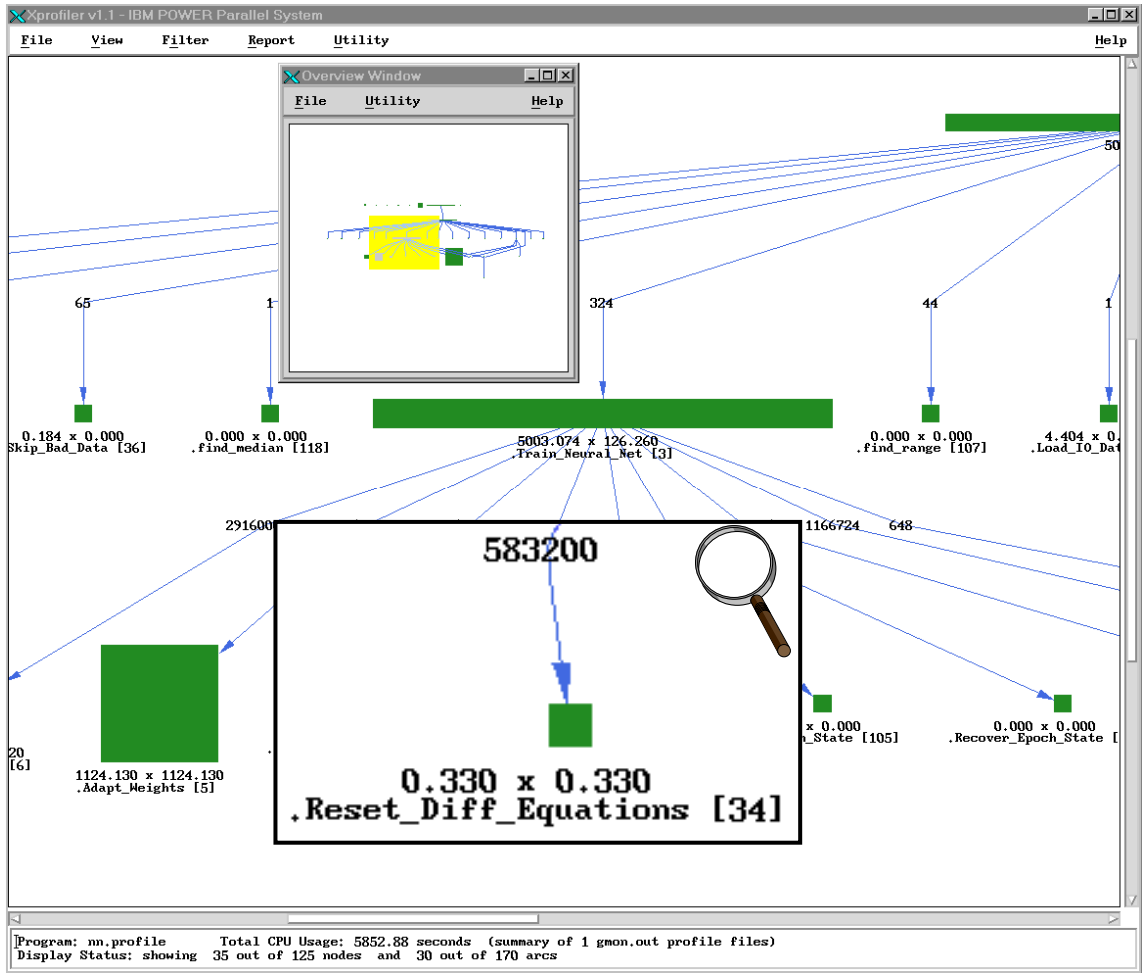
- **CPU profiling tool similar to gprof**
- **Can be used to profile both serial and parallel applications**
- **Use procedure-profiling information to construct a graphical display of the functions within an application**
- **Provide quick access to the profiled data and helps users identify functions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

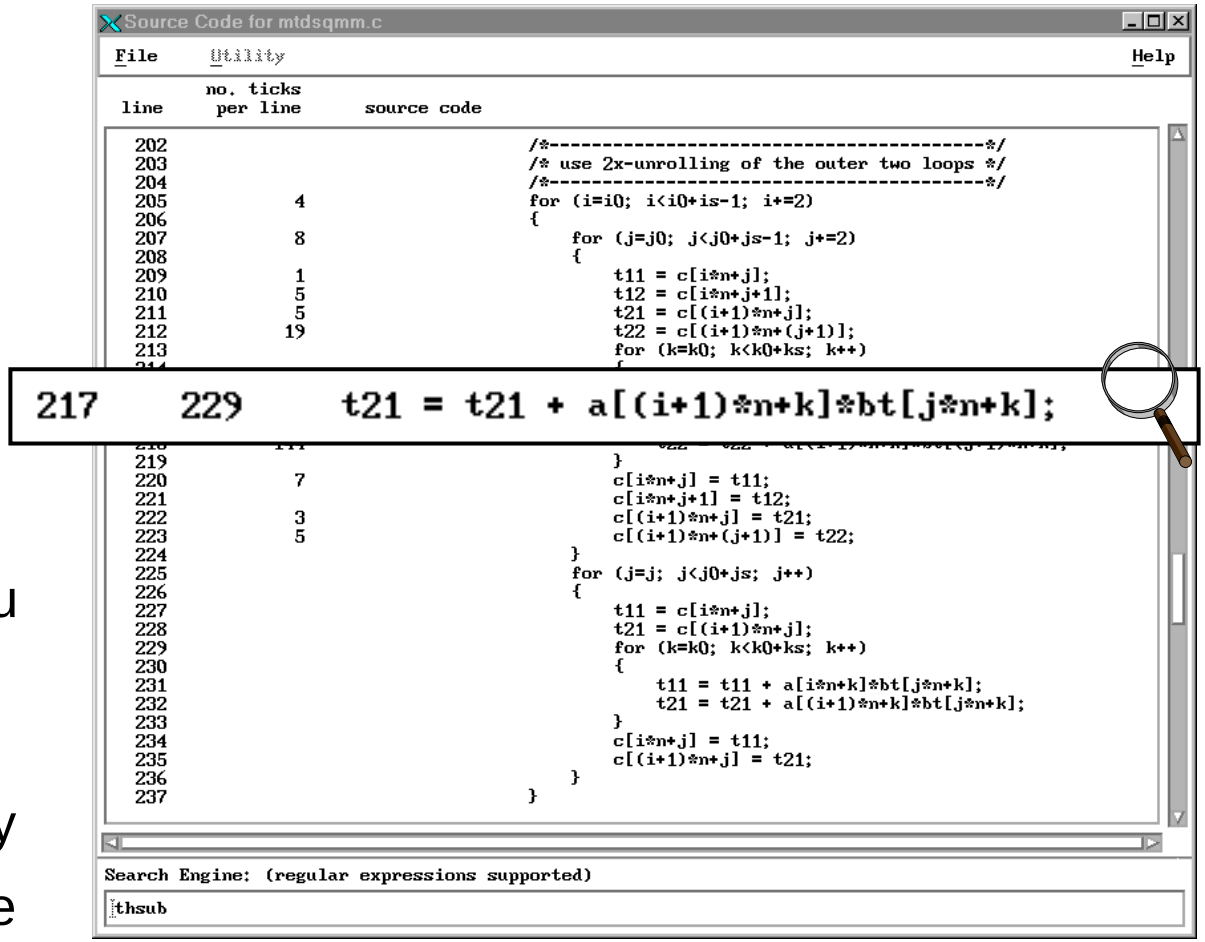
- **Width of a bar: time including called routines**
- **Height of a bar: time excluding called routines**
- **Call arrows labeled with number of calls**
- **Overview window for easy navigation (View → Overview)**



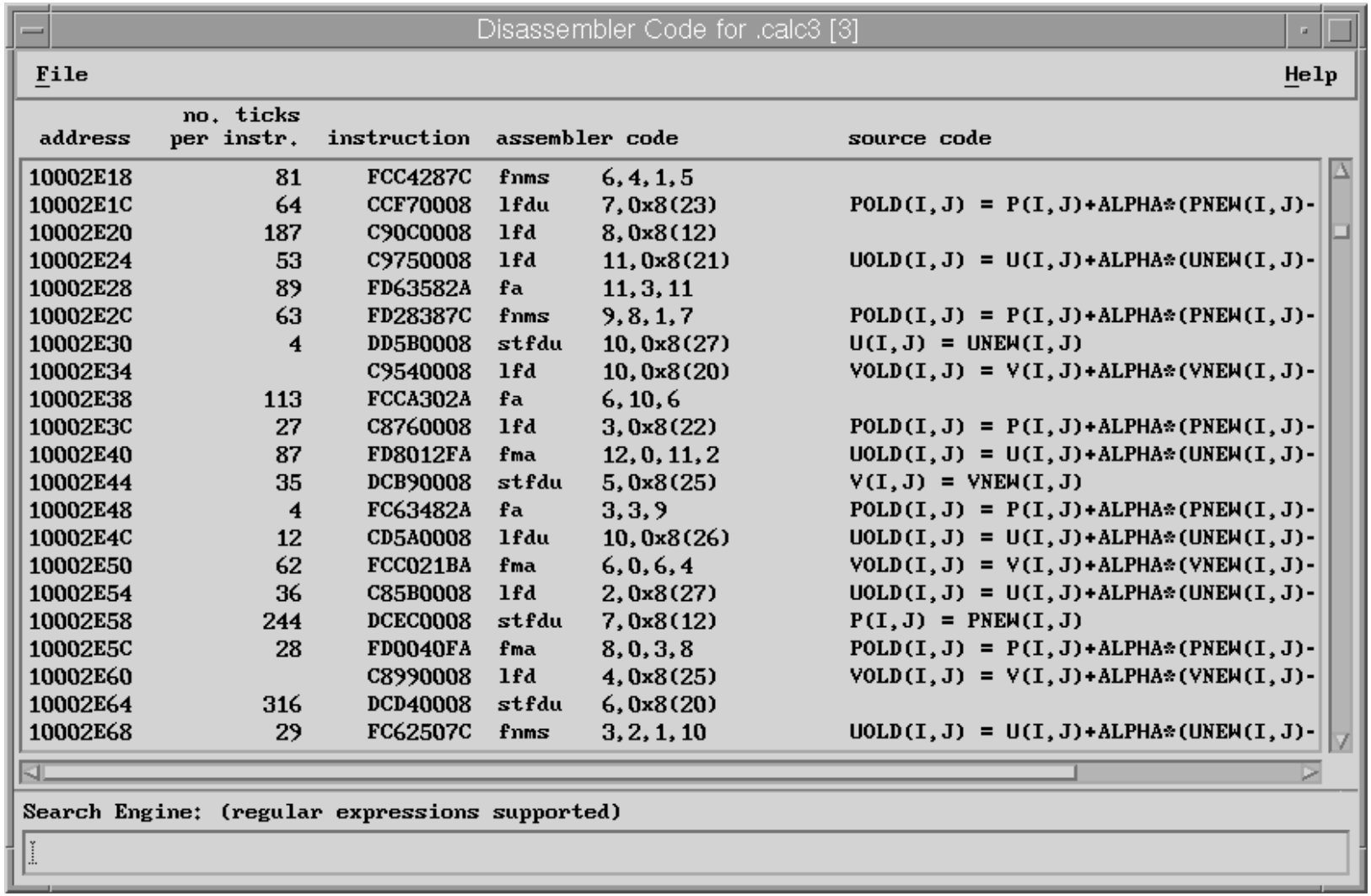
Xprofiler: Source Code Window

- **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



Xprofiler - Disassembler Code



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)/bglsys/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

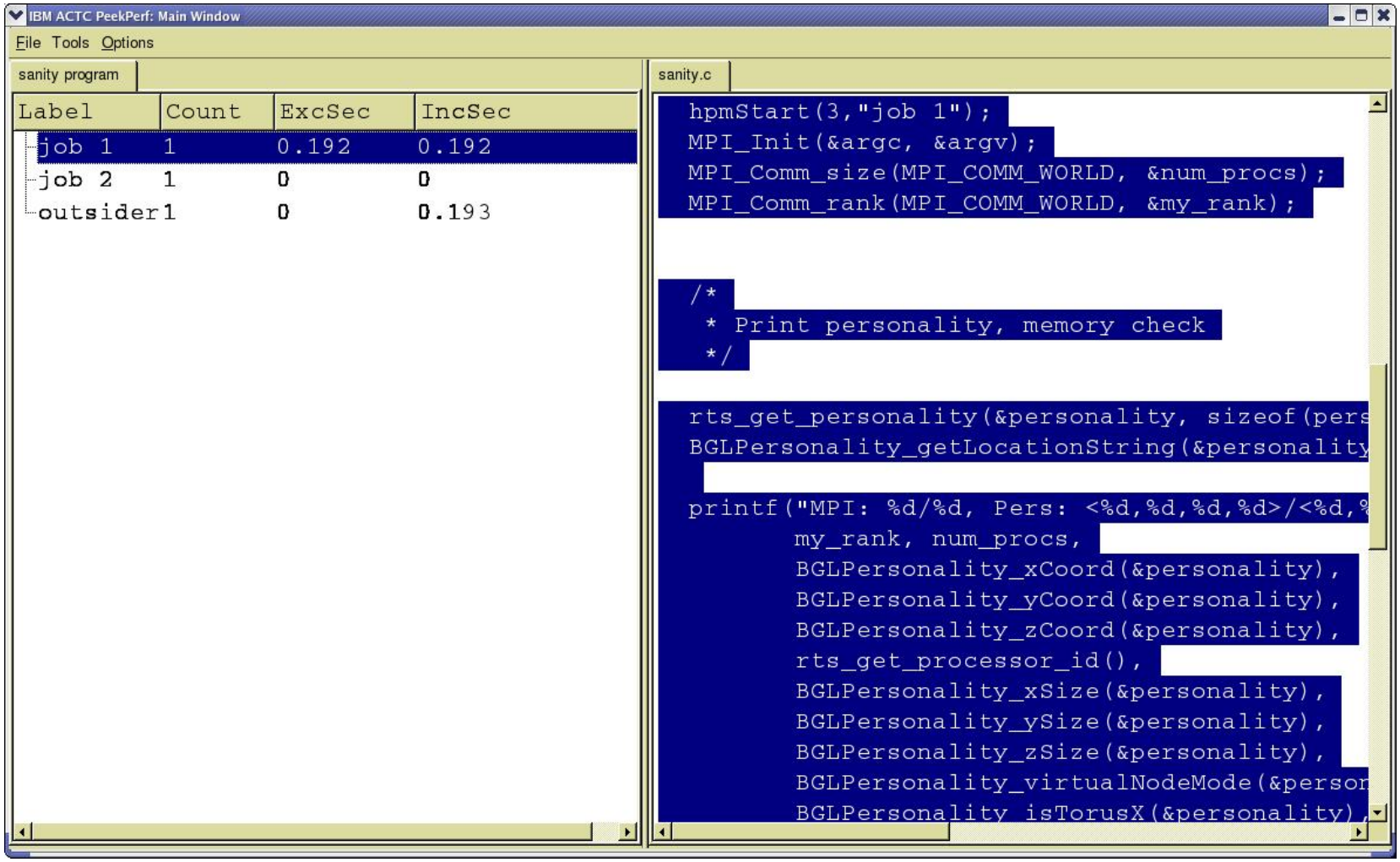
- **HPM_EVENT_SET**
  - Select the event set to be recorded
  - Integer (0 – 15)
- **HPM_NUM_INST_PTS**
  - Overwrite the default of 100 instrumentation sections in the app.
  - Integer value > 0
- **HPM_WITH_MEASUREMENT_ERROR**
  - Deactivate the procedure that removes measurement errors.
  - True or 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

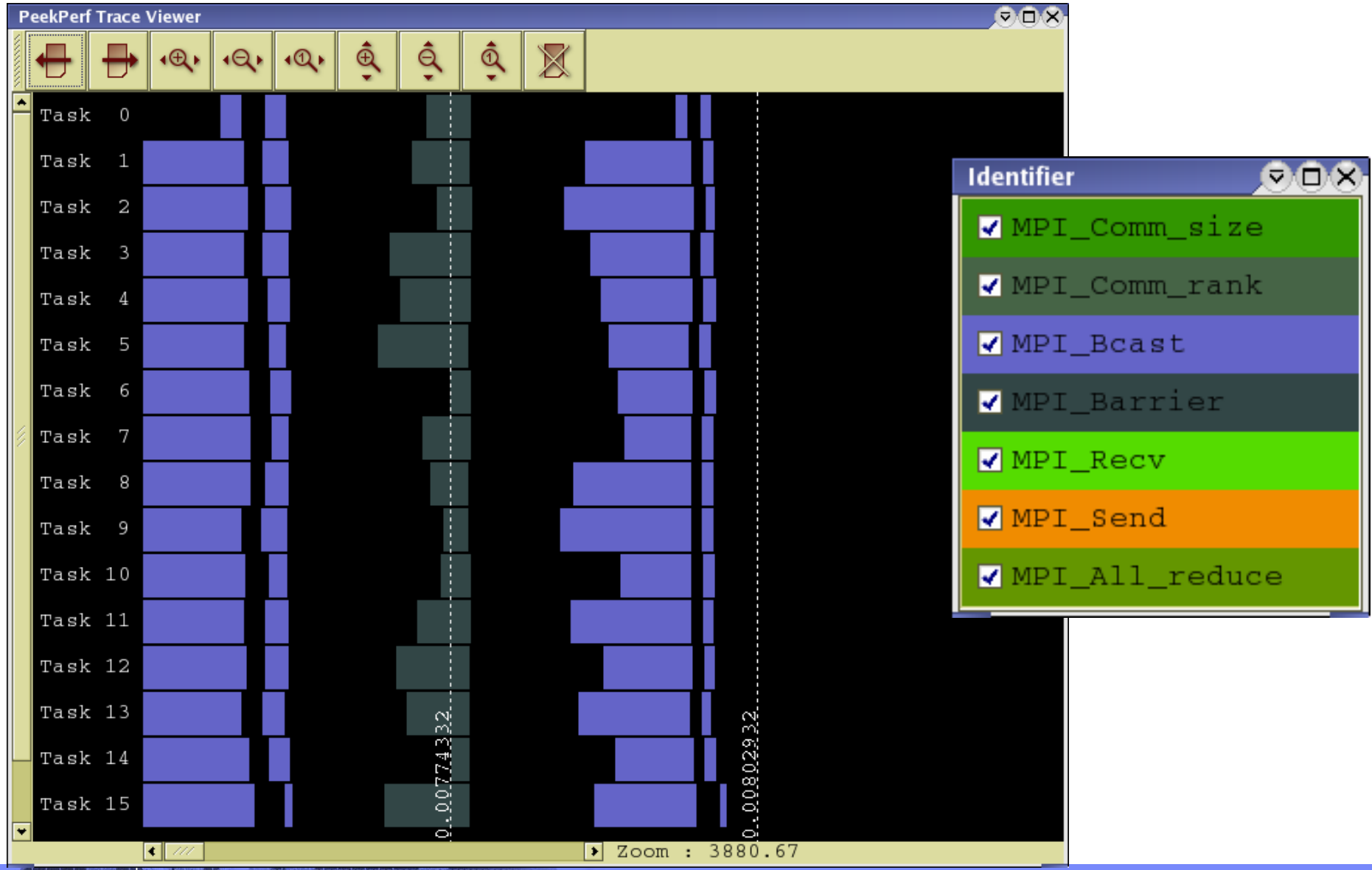
- **Visualization and analysis tool**
- **Offline analysis and viewing capability**
- **Supported platforms**
  - AIX
  - Linux (Power/Intel)
  - Windows (Intel)
  - BlueGene

*The toolkit will be available soon on AMD platform

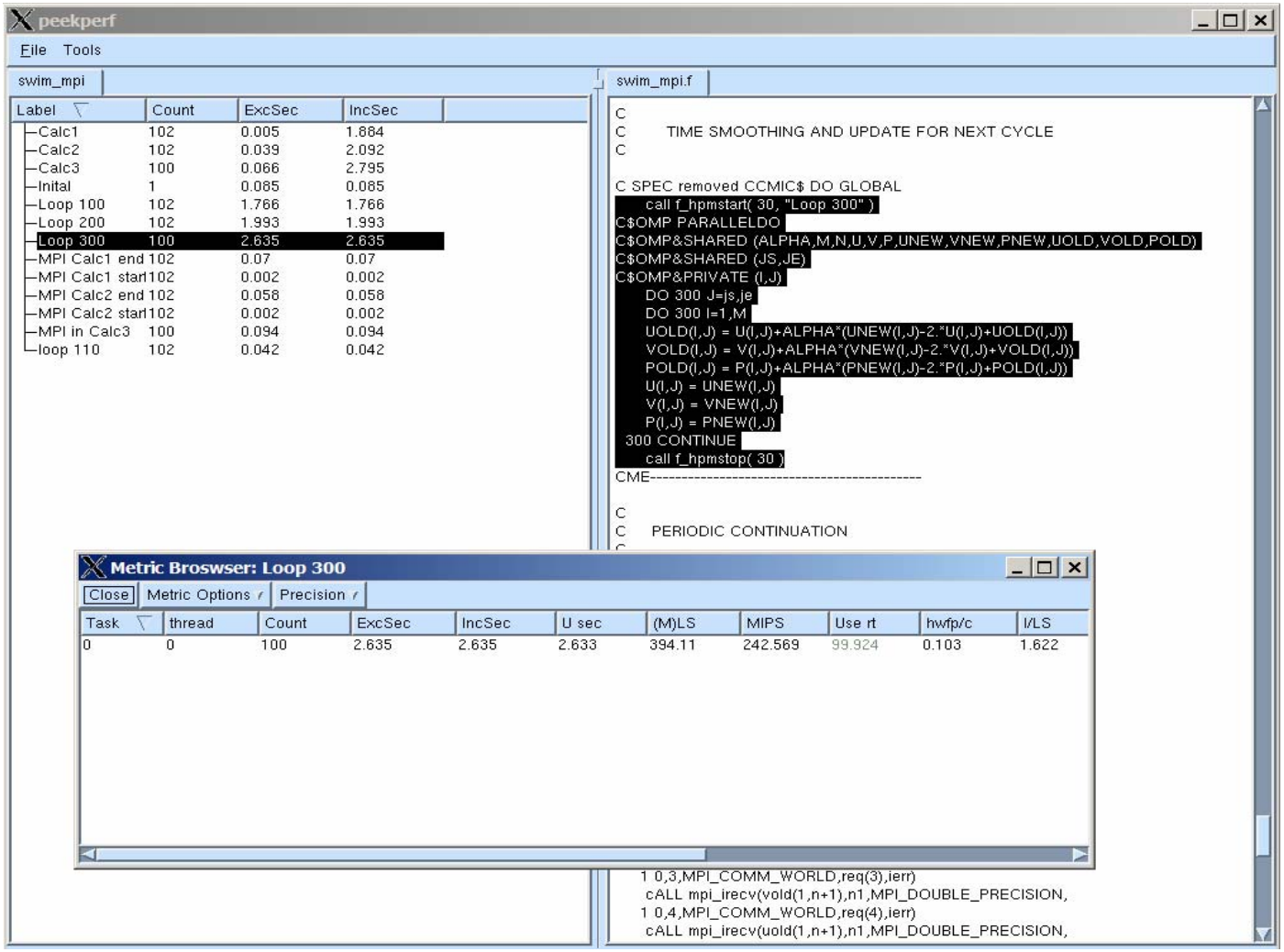
MP_Profiler Visualization Using PeekPerf



MP_Tracer Visualization Using PeekPerf

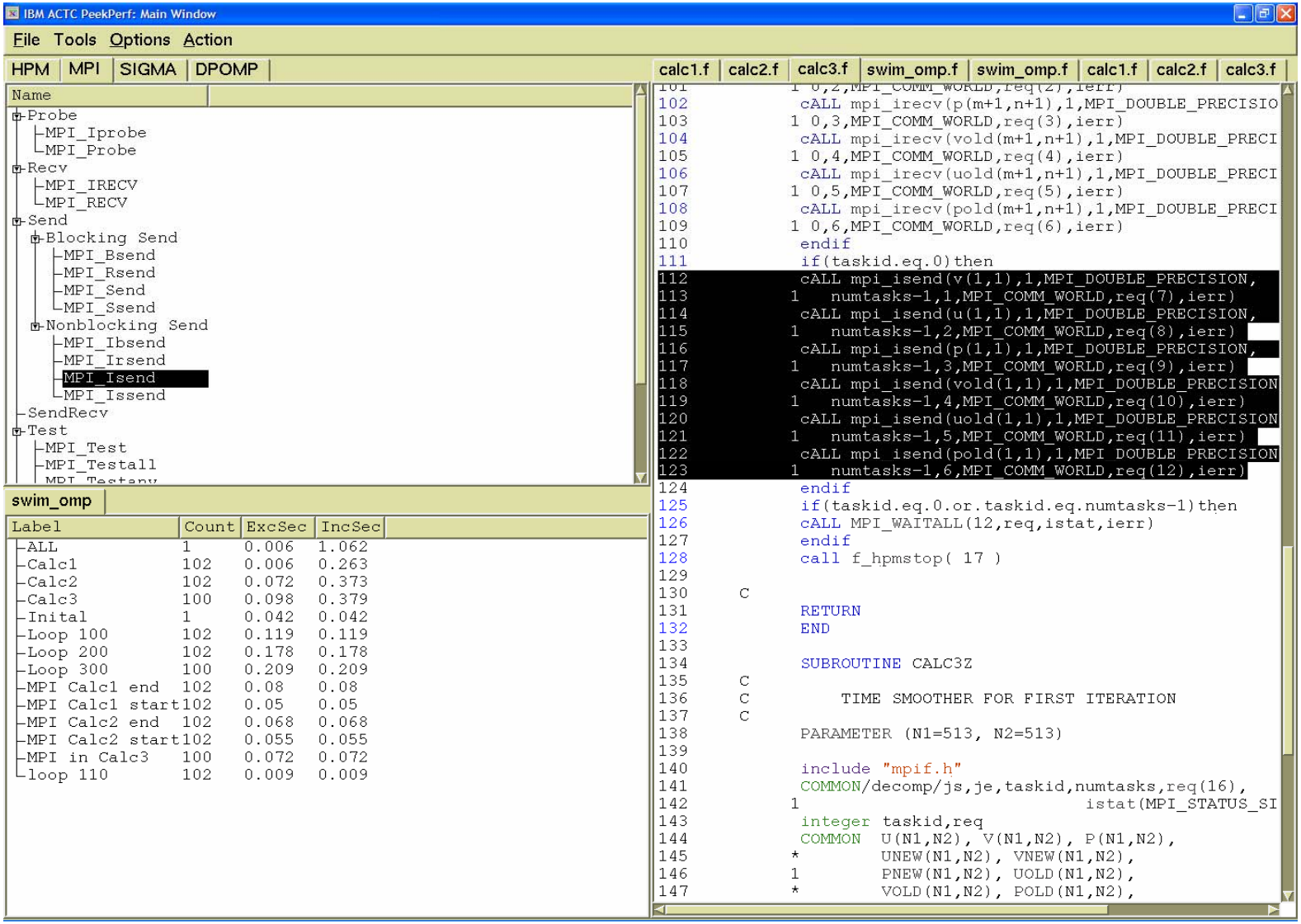


HPM Visualization Using PeekPerf



Future Directions

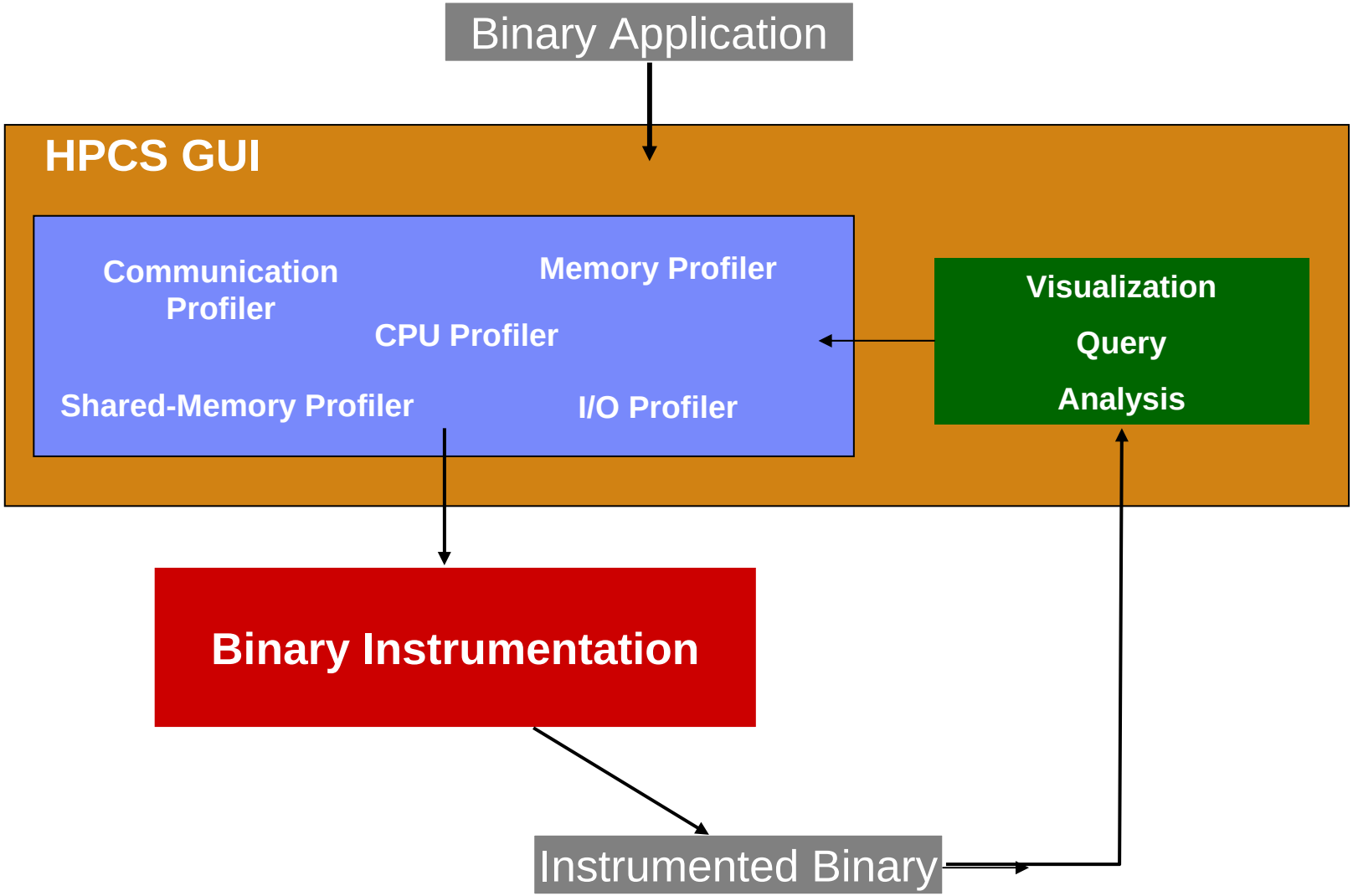
Unified Framework (Instrumentation and Analysis)

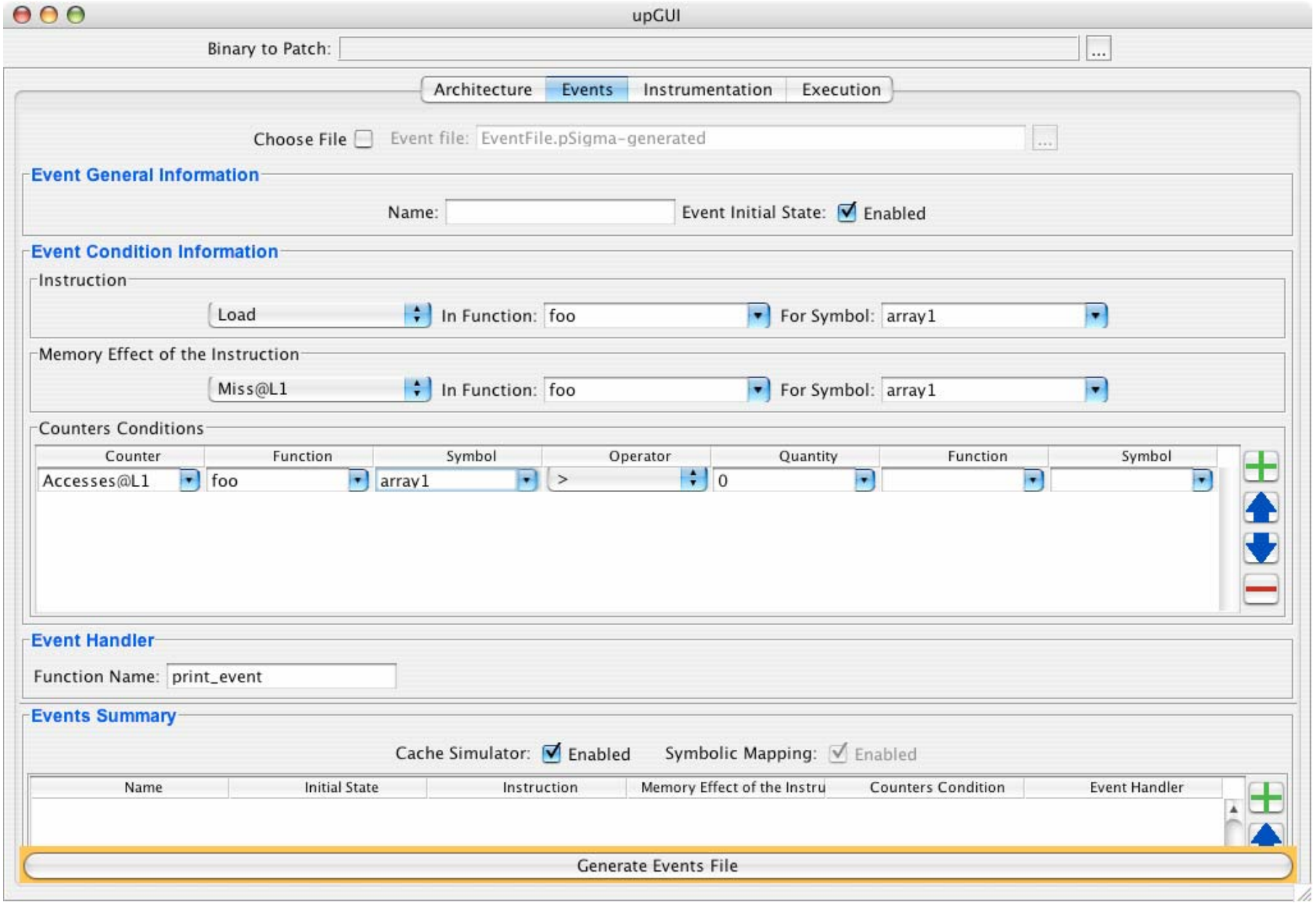


Proposed Technologies for DARPA HPCS (PERCS)

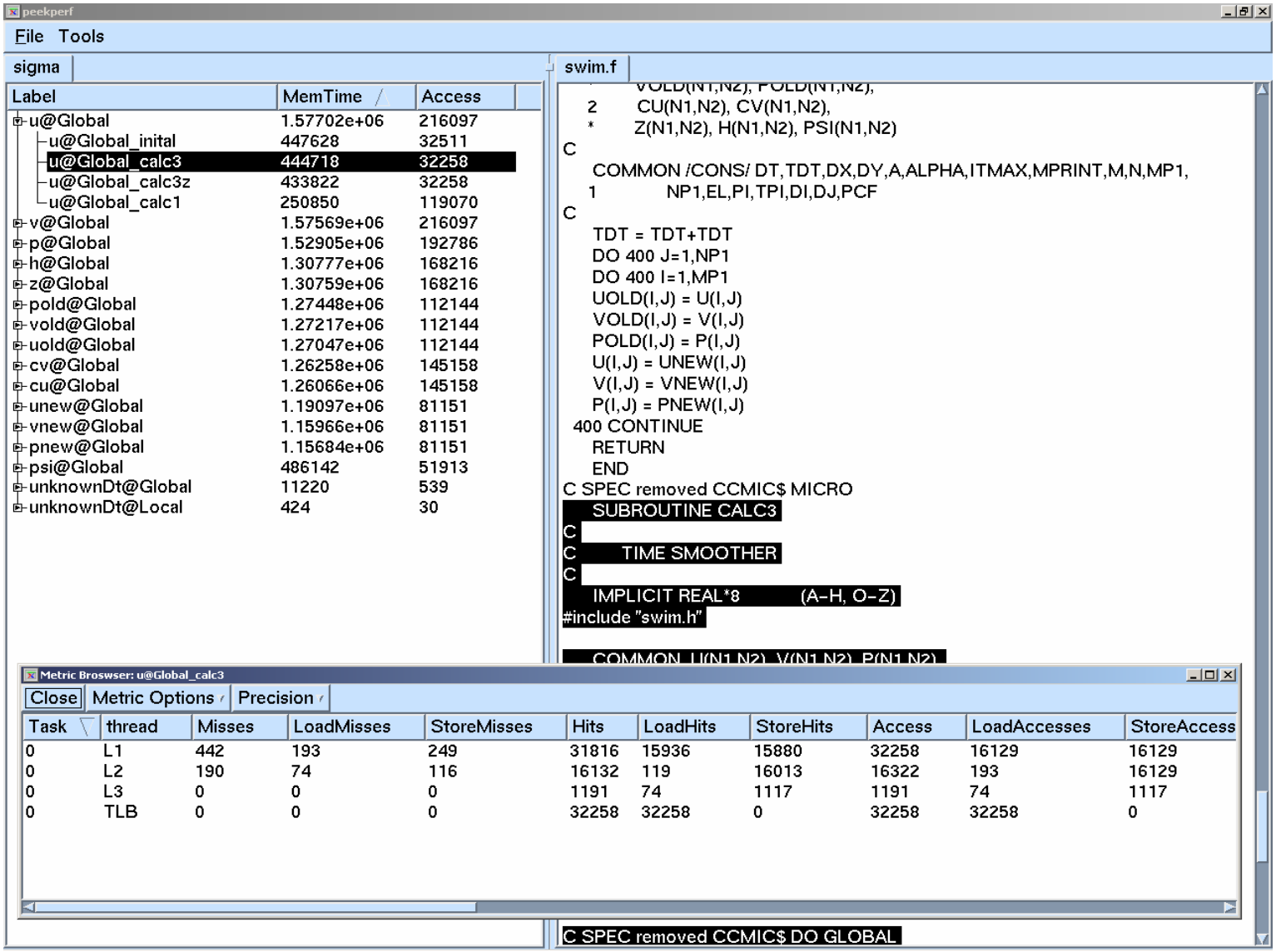
- **Completely Binary Approach (pSigma)**
  - Programmable and dynamic, yet without the need for source code modification.
- **Data-Centric Analysis (DCA)**
  - For HPCS systems, new tools are needed to provide detailed information on the impact of an **application's data structures** in relation to 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

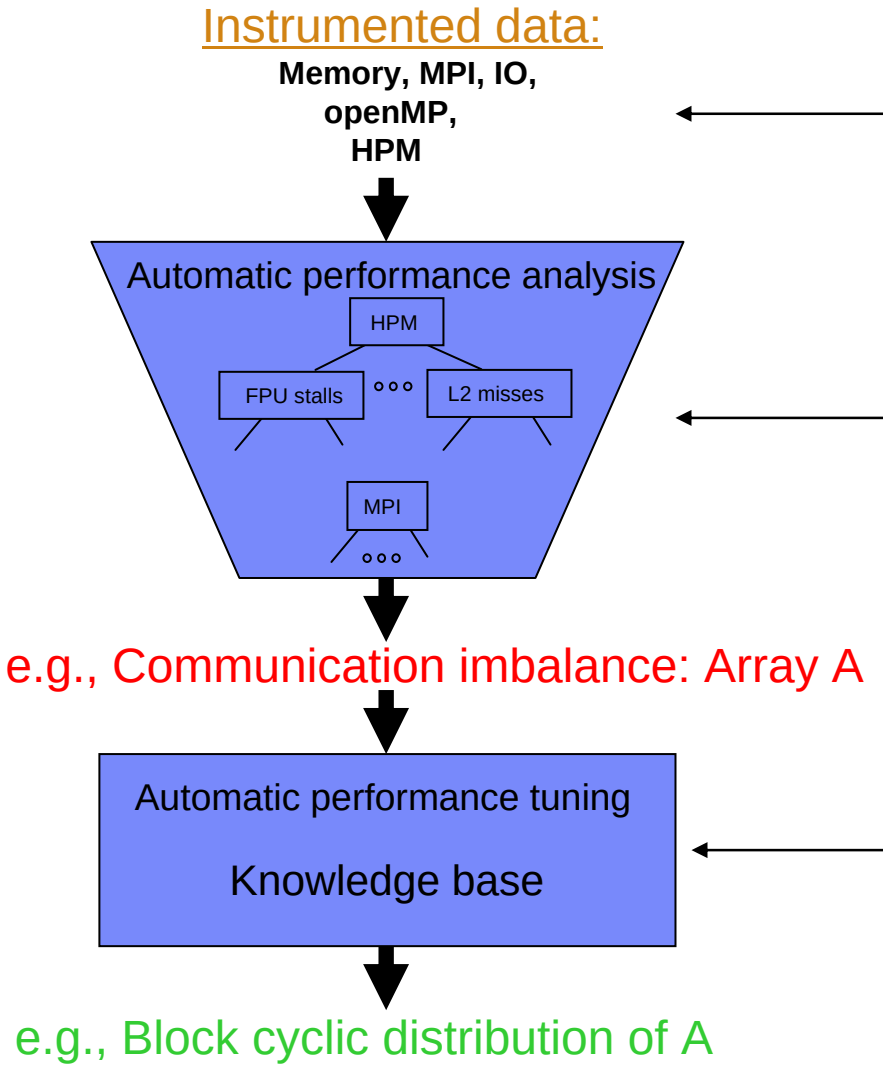




Data-Centric Analysis Technology (DCA)



IBM Vision for DARPA HPCS (PERCS)



1. Performance Data Collection Using Binary Executable Only.
2. Intelligent Mining of Data to Determine Bottlenecks.
3. Characterization of Bottlenecks and Solution Determination.
4. Apply Solution to Source Code.