Profiling and Debugging Tools

Lars Koesterke

University of Porto, Portugal

May 28-29, 2009

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN
Texas Advanced Computing Center

THE UNIVERSITY OF
TEXAS
AT AUSTIN

Outline

- General (Analysis Tools)
- Listings & Reports
- Timers
- Profilers (gprof, tprof, Tau)
- Hardware performance analysis (PAPI)
- Trace Tools (Paraver, ITC/ITA, KOJAK)
- Debugging (dbx/gdb, DDT)

Analysis Tools

- Determine the TIME spent in each "part" (subroutines, functions or even blocks) of your code.
- Within the most time-consuming sections determine if optimization will improve performance.
- General techniques for analyzing code:
  - Compiler reports and listings
  - Profiling
  - Hardware performance counters

Listings & Reports (Compiler/Loader)

- Compilers will optionally generate **optimization reports** & **listing files**.
- Use the **Loader Map** to determine what libraries you have loaded.

Listings & Reports (Compiler/Loader)

- IA32/EM64T:
  - <compiler> -Minfo=time,loop,inline,sym… {(pgi)}
  - <compiler> -opt-report {optimization, (Intel)}
  - <compiler> -S {listing (Intel)}

Timers: Package

The **`time`** command is available on most Unix systems.
It times the execution of a process and its children.

```
/usr/bin/time -p ./a.out
real      1.54
user      0.51
sys        .73
```

Time for a.out execution.
Output (in seconds).

e.g. for interactive batch, execution time of ibrun (and a.out):

```
Bourne shell: bsub> /usr/bin/time -p ibrun  ./a.out args > out 2>&1
C shell:      bsub> /usr/bin/time -p ibrun  ./a.out args >& out
```

Types of performance analysis information

- Wall clock/CPU time spent on each function
- HW counters (e.g., cache misses, FLOPs)

Tools:

1. profiler
2. profile visualizer
3. API to read/display HW counter info

- trace file (raw)
- timeline
  - ➢ state of thread/process
  - ➢ communication
  - ➢ predefined user events

Tools:

1. trace generator
2. instrumentation API
3. tool for reading/interpreting trace files
4. visualizer

Timers: Code Section

Routine	Type	Resolution (sec)	OS/Compiler
times	user/sys	1000	Linux/AIX/IRIX/ UNICOS
getrusage	wall/user/sys	1000	Linux/AIX/IRIX
gettimeofday	wall clock	1	Linux/AIX/IRIX/ UNICOS
rdtsc	wall clock	0.1	Linux
read_real_time	wall clock	0.001	AIX
system_clock	wall clock	System Dependent	Fortran 90 Intrinsic
MPI_Wtime	wall clock	System dependent	MPI Library (C & Fortran)

Timers: Code Section

The **times**, **getrussage**, **gettimeofday**, **rdtsc**, and **read_real_time** timers have been packaged into a group of C wrapper routines (also callable from Fortran).

```
external   x_timer
real*8   ::  x_timer
real*8   :: sec0, sec1, tseconds
…
sec0      = x_timer()
   …Fortran Code
sec1      = x_timer()
tseconds = sec1-sec0
```

```
double x_timer(void);
              …
double sec0, sec1, tseconds;
…
sec0      = x_timer();
…C Codes
sec1      = x_timer();
tseconds = sec1-sec0
```

X **= {one of rusage, gtod, rdtsc, rrt}**

Profilers: gprof (instrumentation)

<compiler> -g –pg prog.<x>

gprof <executable> gmon.out

E.G.

```
ifort –g –pg prog.f90
./a.out
gprof ./a.out gmon.out    or  gprof
```

-g provides more info
on intrinsics & libs

generates gmon.out

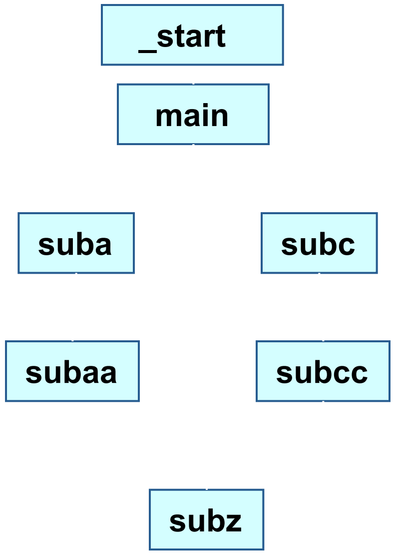
a.out & gmon.out
are defaults

Profilers: Example Code

- Program Structure

```
program prof1
   ...
   do i = 1,2
      call suba(n,a,b,c)
   enddo
   do i = 1,2
      call subc(n,a,b,c)
   enddo
 end

 subroutine suba(n,a,b,c)
   ...
   call subaa(n,a,b,c)
 end

 subroutine subc(n,a,b,c)
   ...
   call subcc(n,a,b,c)
 end
```

```
subroutine subaa(n,a,b,c)
   ...
   do i = 1,20
      ...
      call subz(n,a,b,c)
   end do
end

subroutine subcc(n,a,b,c)
   ...
   call subz(n,a,b,c)
end

subroutine subz(n,a,b,c)
...
end
```



Profiler Example: gprof (output)

- A common Unix profiling tool is **gprof**. Compiler options and libraries provide wrappers for each routine call (mcount), and periodic sampling the program counter (0.01 sec).

% time	**cumulative** secs	self secs	calls	self ms/call	total ms/call	name
86.21	145.6	145.6	42	3468	3468	subz_
8.18	159.4	13.8	2	6910	76262	subaa_
4.10	166.4	6.9	2	3465	6933	subcc_
0.72	167.6	1.2	2	610	76872	suba_
0.52	168.5	0.88	2	440	7372	subc_
0.26	168.9	0.44	2	440	168930	main
0.01	168.9	0.01	1			write

Profiler Example: gprof (output)

```
granularity: each sample hit covers 4 byte(s)
             for 0.01% of 168.94 seconds
index % time  self  children   called  name
              0.44  168.49      1/1        _start [2]
[1]   100     0.44  168.49      1      main       [1]
              1.22  152.52      2/2        suba_  [3]
              0.88   13.87      2/2        subc_  [6]
-----------------------------------------------------
              1.22  152.52      2/2        main   [1]
[3]     91    1.22  152.52      2      suba_      [3]
             13.82  138.70      2/2        subaa_ [4]
-----------------------------------------------------
             13.82  138.70      2/2        suba_  [3]
[4]     90   13.82  138.70      2      subaa_     [4]
            138.70    0.00     40/42       subz_  [5]
-----------------------------------------------------
```

Profiler Example: gprof (output cont.)

```
 6.94     0.00        2/42        subcc_ [7]
             138.70     0.00     40/42        subaa_ [4]
[5]       86  145.64     0.00     42       subz_        [5]
-----------------------------------------------------
                0.88    13.87     2/2          main    [1]
[6]        8    0.88    13.87     2        subc_        [6]
                6.93     6.94     2/2          subcc_ [7]
-----------------------------------------------------
                6.93     6.94     2/2          subc_   [6]
[7]        8    6.93     6.94     2        subcc_       [7]
                6.94     0.00     2/42         subz_   [5]
-----------------------------------------------------
```

Profiling Parallel Programs (gprof)

mpif90 -gp prog.f90	**Instruments code**
setenv GMON_OUT_PREFIX gout.*	**Forces each task to produce a gout.<pid>**
***Submit parallel job for executable* (in this case named a.out)**	**Produces gmon.out trace file**
gprof -s gout.*	**Combines gout.<pid> files into gmon.sum file**
gprof a.out gmon.sum	**Reads executable (a.out) & gmon.sum, report sent to STDOUT**

mpiP: dynamic MPI Profiling

- Scalable Profiling library for MPI applications
- Lightweight
- Collects statistics of MPI functions
  - uses communication only during report generation
  - less overhead & much less data than tracing tools.
- http://mpip.sourceforge.net

Usage, Instrumentation, Analysis

- How to use
  - No recompiling required
  - Profiling gathered in MPI profiling layer
- Link Static Library before default MPI libraries

  -g -L${TACC_MPIP_LIB} -lmpiP -lbfd -liberty -lintl –lm

  mpicc and mpif90 cmd line libs are loaded first.
- What to analyze
  - Overview of time spent in MPI communication during the application run
  - Aggregate time for individual MPI call

Control

- External Control
  - Set MPIP environment variable (threshold,callsite depth)
    - E.g.   setenv MPIP ‘-t 10 –k 2   export MPIP= ‘-t 10 –k 2’
- Limiting to specific code blocks
  - MPI_Pcontrol(#)

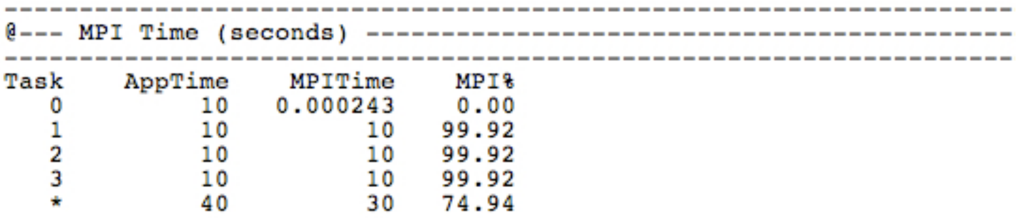
```
MPI_Pcontrol(2);
MPI_Pcontrol(1);
    MPI_Abc(…);
MPI_Pcontrol(0);
```

```
call MPI_Pcontrol(2)
call MPI_Pcontrol(1)
    call MPI_Abc(…)
call MPI_Pcontrol(0)
```

Pcontrol Arg	**Behavior**
0	Disable profiling.
1	Enable Profiling.
2	Reset all callsite data.
3	Generate verbose report.
4	Generate concise report.

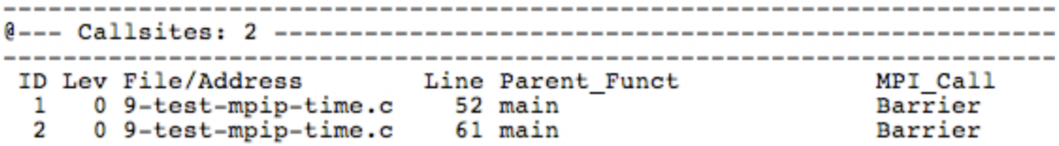
mpiP: output

- MPI-Time: wall-clock time for all MPI calls within application time



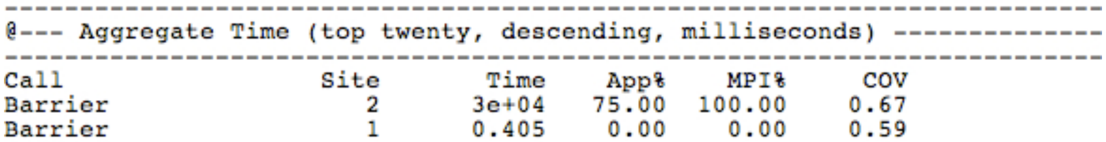
```
---------------------------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0         10   0.000243     0.00
   1         10         10    99.92
   2         10         10    99.92
   3         10         10    99.92
   *         40         30    74.94
```

- MPI callsites within application



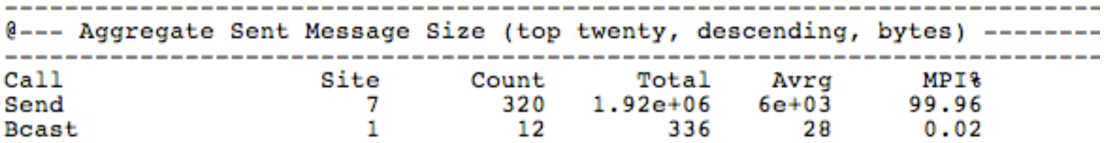
```
---------------------------------------------------------------------------
@--- Callsites: 2 ---------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address         Line Parent_Funct                MPI_Call
  1   0 9-test-mpip-time.c     52 main                        Barrier
  2   0 9-test-mpip-time.c     61 main                        Barrier
```

- Aggregation time (top 20 MPI callsites)



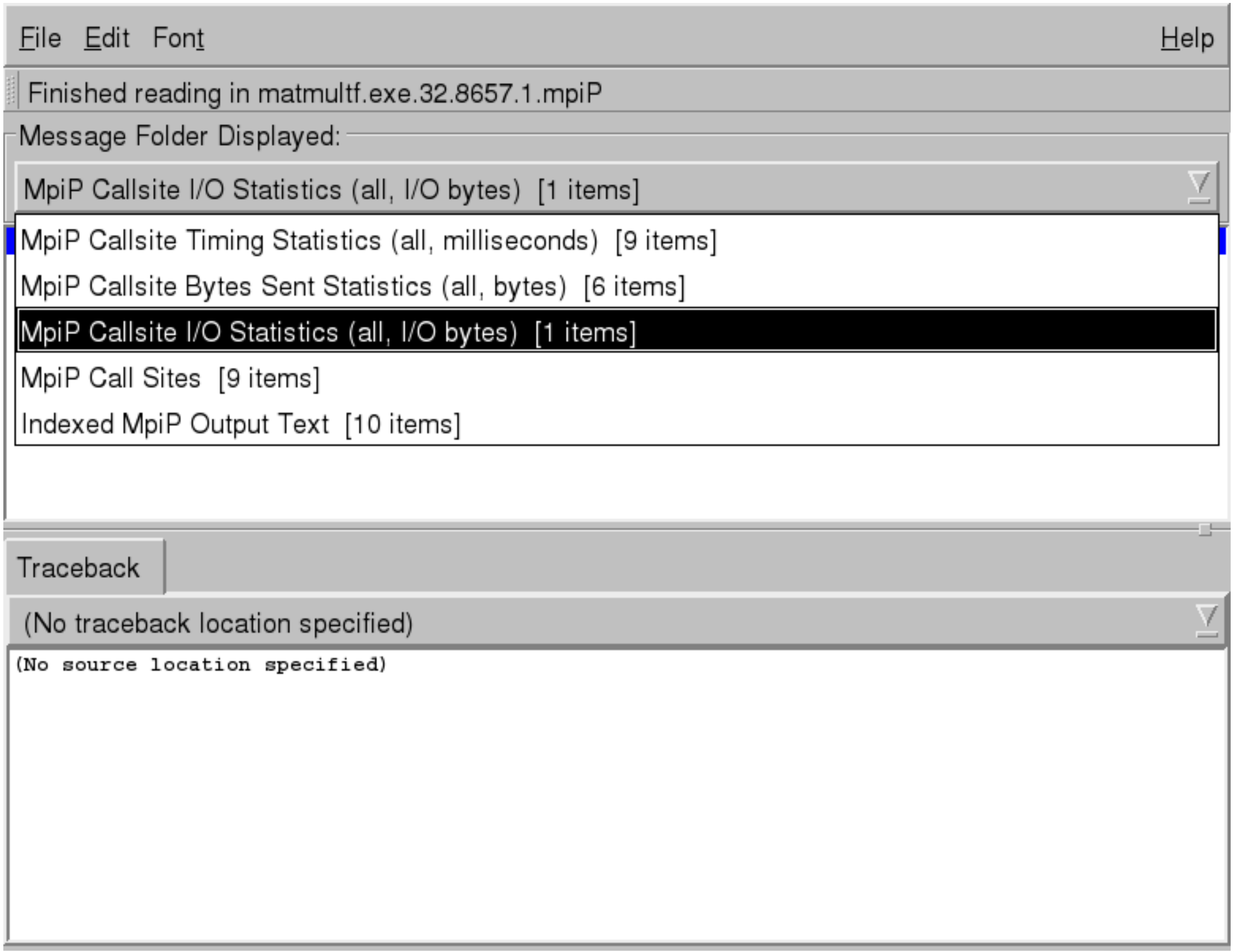
```
---------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
---------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%     COV
Barrier                 2      3e+04   75.00  100.00    0.67
Barrier                 1      0.405    0.00    0.00    0.59
```

- Message size of top 20 callsites



```
---------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site      Count      Total     Avrg     MPI%
Send                    7        320   1.92e+06    6e+03    99.96
Bcast                   1         12        336       28     0.02
```

Better view with mpipview



Finished reading in matmultf.exe.32.8657.1.mpiP

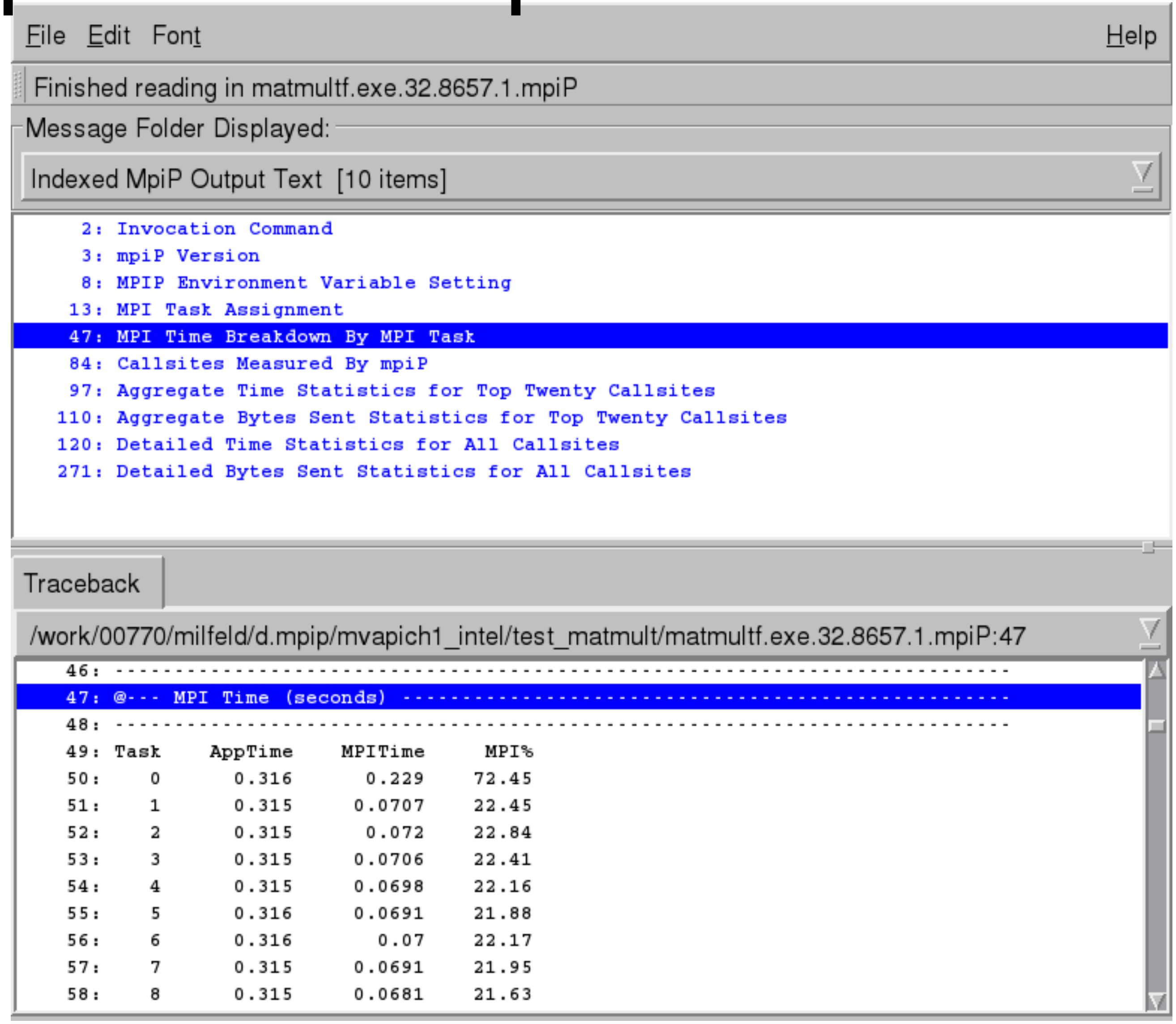
Message Folder Displayed: MpiP Callsite I/O Statistics (all, I/O bytes) [1 items]

- MpiP Callsite Timing Statistics (all, milliseconds) [9 items]
- MpiP Callsite Bytes Sent Statistics (all, bytes) [6 items]
- MpiP Callsite I/O Statistics (all, I/O bytes) [1 items]
- MpiP Call Sites [9 items]
- Indexed MpiP Output Text [10 items]

(No traceback location specified)

```
(No source location specified)
```

mpipview - output



Finished reading in matmultf.exe.32.8657.1.mpiP

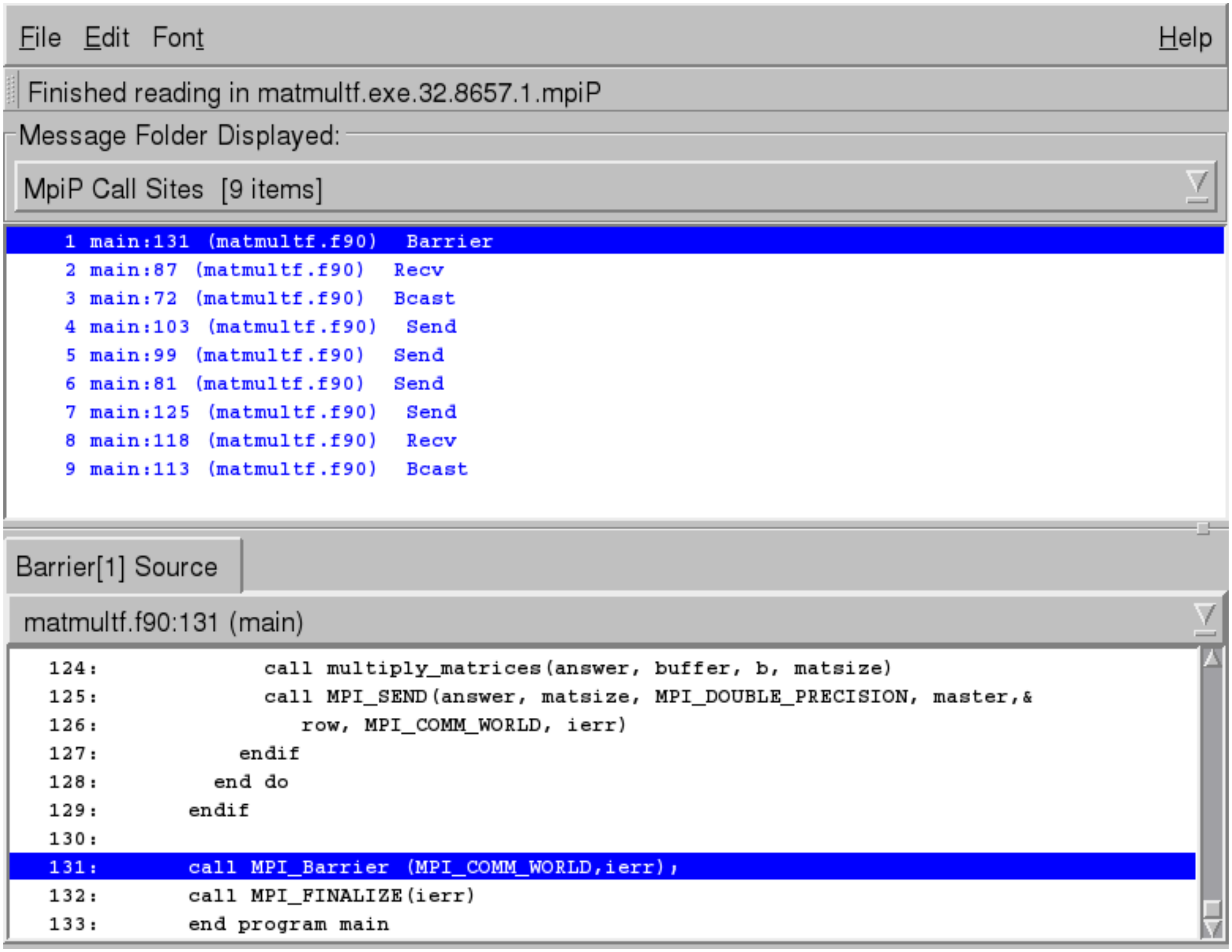
Indexed MpiP Output Text [10 items]

```
   2: Invocation Command
   3: mpiP Version
   8: MPIP Environment Variable Setting
  13: MPI Task Assignment
  47: MPI Time Breakdown By MPI Task
  84: Callsites Measured By mpiP
  97: Aggregate Time Statistics for Top Twenty Callsites
 110: Aggregate Bytes Sent Statistics for Top Twenty Callsites
 120: Detailed Time Statistics for All Callsites
 271: Detailed Bytes Sent Statistics for All Callsites
```

/work/00770/milfeld/d.mpip/mvapich1_intel/test_matmult/matmultf.exe.32.8657.1.mpiP:47

```
46: ---------------------------------------------------------------------------
47: @--- MPI Time (seconds) ---------------------------------------------------
48: ---------------------------------------------------------------------------
49: Task    AppTime    MPITime     MPI%
50:    0      0.316      0.229    72.45
51:    1      0.315     0.0707    22.45
52:    2      0.315      0.072    22.84
53:    3      0.315     0.0706    22.41
54:    4      0.315     0.0698    22.16
55:    5      0.316     0.0691    21.88
56:    6      0.316       0.07    22.17
57:    7      0.315     0.0691    21.95
58:    8      0.315     0.0681    21.63
```

Call “Sites”



Finished reading in matmultf.exe.32.8657.1.mpiP

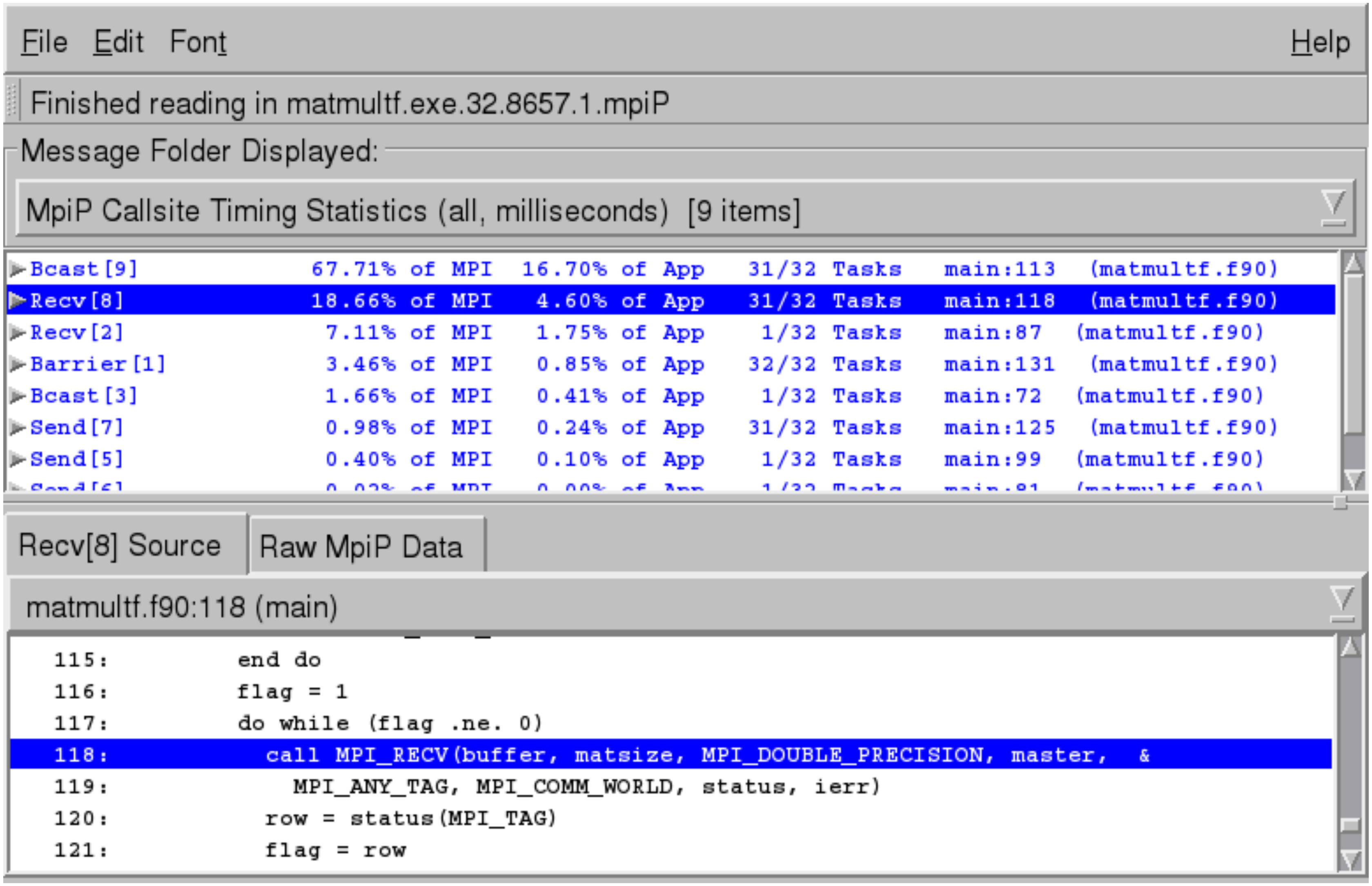
MpiP Call Sites [9 items]

```
1 main:131 (matmultf.f90)  Barrier
2 main:87 (matmultf.f90)  Recv
3 main:72 (matmultf.f90)  Bcast
4 main:103 (matmultf.f90)  Send
5 main:99 (matmultf.f90)  Send
6 main:81 (matmultf.f90)  Send
7 main:125 (matmultf.f90)  Send
8 main:118 (matmultf.f90)  Recv
9 main:113 (matmultf.f90)  Bcast
```

matmultf.f90:131 (main)

```
124:            call multiply_matrices(answer, buffer, b, matsize)
125:            call MPI_SEND(answer, matsize, MPI_DOUBLE_PRECISION, master,&
126:               row, MPI_COMM_WORLD, ierr)
127:          endif
128:        end do
129:      endif
130:
131:      call MPI_Barrier (MPI_COMM_WORLD,ierr);
132:      call MPI_FINALIZE(ierr)
133:      end program main
```

Statistics



Finished reading in matmultf.exe.32.8657.1.mpiP

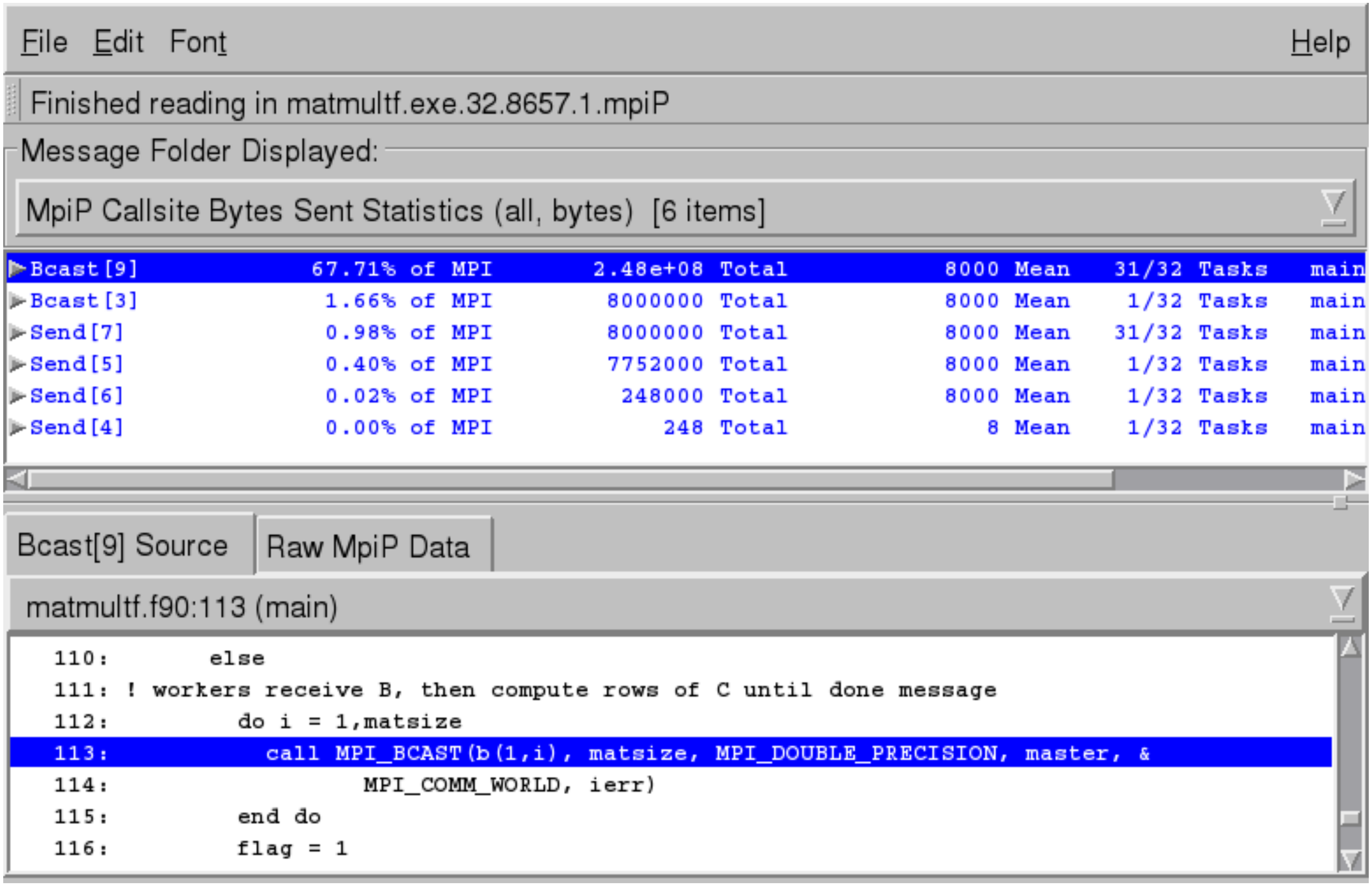
MpiP Callsite Timing Statistics (all, milliseconds) [9 items]

```
Bcast[9]       67.71% of MPI  16.70% of App  31/32 Tasks  main:113  (matmultf.f90)
Recv[8]        18.66% of MPI   4.60% of App  31/32 Tasks  main:118  (matmultf.f90)
Recv[2]         7.11% of MPI   1.75% of App   1/32 Tasks  main:87  (matmultf.f90)
Barrier[1]      3.46% of MPI   0.85% of App  32/32 Tasks  main:131  (matmultf.f90)
Bcast[3]        1.66% of MPI   0.41% of App   1/32 Tasks  main:72  (matmultf.f90)
Send[7]         0.98% of MPI   0.24% of App  31/32 Tasks  main:125  (matmultf.f90)
Send[5]         0.40% of MPI   0.10% of App   1/32 Tasks  main:99  (matmultf.f90)
```

matmultf.f90:118 (main)

```
  115:         end do
  116:         flag = 1
  117:         do while (flag .ne. 0)
  118:           call MPI_RECV(buffer, matsize, MPI_DOUBLE_PRECISION, master,  &
  119:             MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)
  120:           row = status(MPI_TAG)
  121:           flag = row
```

Message Size



Finished reading in matmultf.exe.32.8657.1.mpiP

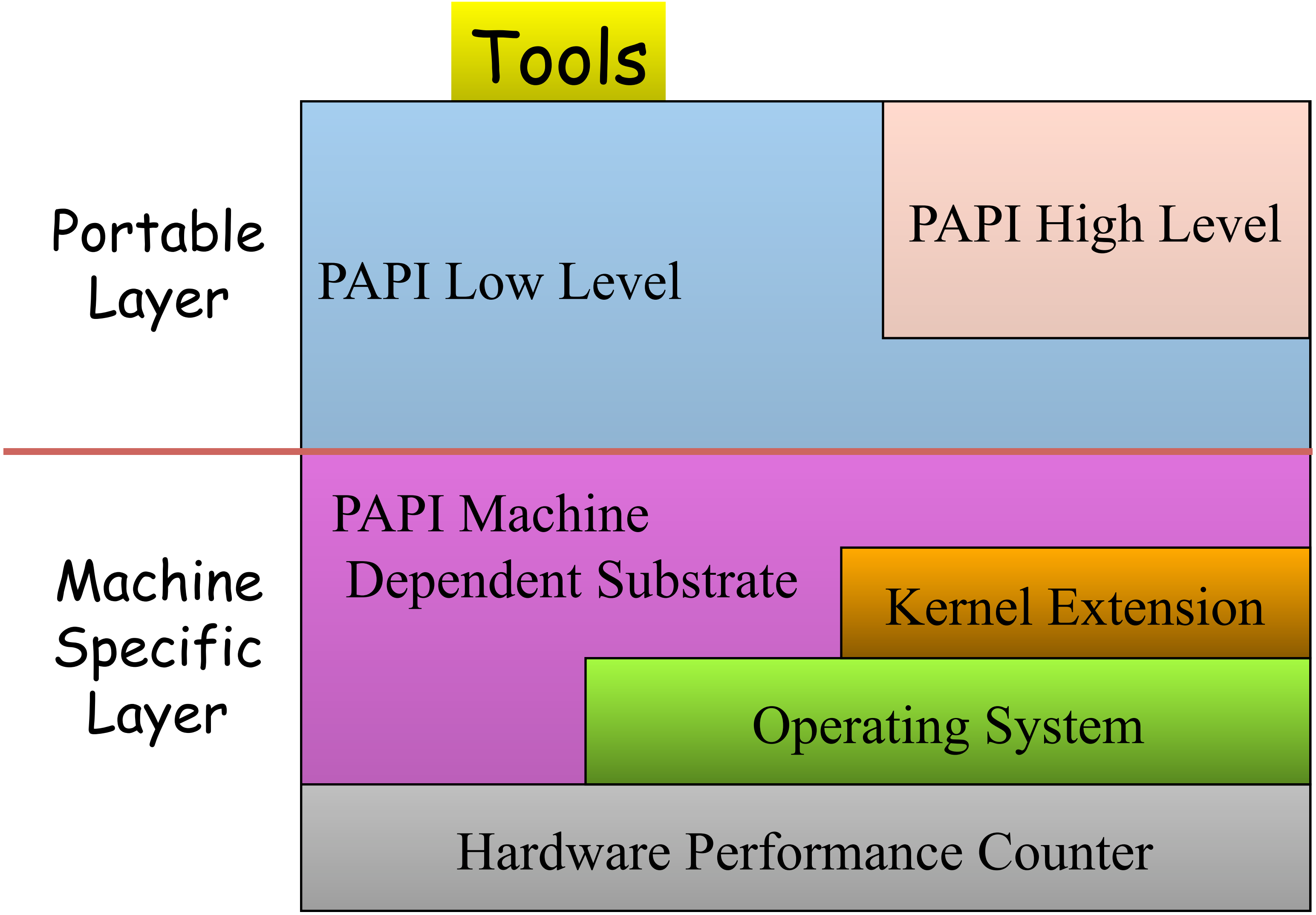
MpiP Callsite Bytes Sent Statistics (all, bytes) [6 items]

```
Bcast[9]          67.71% of MPI        2.48e+08 Total        8000 Mean   31/32 Tasks   main
Bcast[3]           1.66% of MPI         8000000 Total        8000 Mean    1/32 Tasks   main
Send[7]            0.98% of MPI         8000000 Total        8000 Mean   31/32 Tasks   main
Send[5]            0.40% of MPI         7752000 Total        8000 Mean    1/32 Tasks   main
Send[6]            0.02% of MPI          248000 Total        8000 Mean    1/32 Tasks   main
Send[4]            0.00% of MPI             248 Total           8 Mean    1/32 Tasks   main
```

matmultf.f90:113 (main)

```
110:       else
111: ! workers receive B, then compute rows of C until done message
112:          do i = 1,matsize
113:             call MPI_BCAST(b(1,i), matsize, MPI_DOUBLE_PRECISION, master, &
114:                            MPI_COMM_WORLD, ierr)
115:          end do
116:          flag = 1
```

PAPI Implementation



Tools

Portable Layer

PAPI Low Level

PAPI High Level

Machine Specific Layer

PAPI Machine Dependent Substrate

Kernel Extension

Operating System

Hardware Performance Counter

PAPI Performance Monitor

- Provides high level counters for events:
  - Floating point instructions/operations,
  - Total instructions and cycles
  - Cache accesses and misses
  - Translation Lookaside Buffer (TLB) counts
  - Branch instructions taken, predicted, mispredicted
- PAPI_flops routine for basic performance analysis
  - Wall and processor times
  - Total floating point operations and MFLOPS

    http://icl.cs.utk.edu/projects/papi
- Low level functions are thread-safe, high level are not

High-level Interface

- Meant for application programmers wanting coarse-grained measurements
- Not thread safe
- Calls the lower level API
- Allows only PAPI preset events
- Easier to use and less setup (additional code) than low-level

High-level API

- C interface
  PAPI_start_counters
  PAPI_read_counters
  PAPI_stop_counters
  PAPI_accum_counters
  PAPI_num_counters
  PAPI_flips
  PAPI_ipc

- Fortran interface
  PAPIF_start_counters
  PAPIF_read_counters
  PAPIF_stop_counters
  PAPIF_accum_counters
  PAPIF_num_counters
  PAPIF_flips
  PAPIF_ipc

TAU Instrumentation

- Manually using TAU instrumentation API
- Automatically using
  - Program Database Toolkit (PDT)
  - MPI profiling library
  - Opari OpenMP rewriting tool
- Uses PAPI to access hardware counter data

Program Database Toolkit (PDT)

- Program code analysis framework for developing source-based tools
- High-level interface to source code information
- Integrated toolkit for source code parsing, database creation, and database query
  - commercial grade front end parsers
  - portable IL analyzer, database format, and access API
  - open software approach for tool development
- Target and integrate multiple source languages
- Use in TAU to build automated performance instrumentation tools

OPARI: Basic Usage (f90)

- Reset OPARI state information
  - `rm -f opari.rc`
- Call OPARI for each input source file

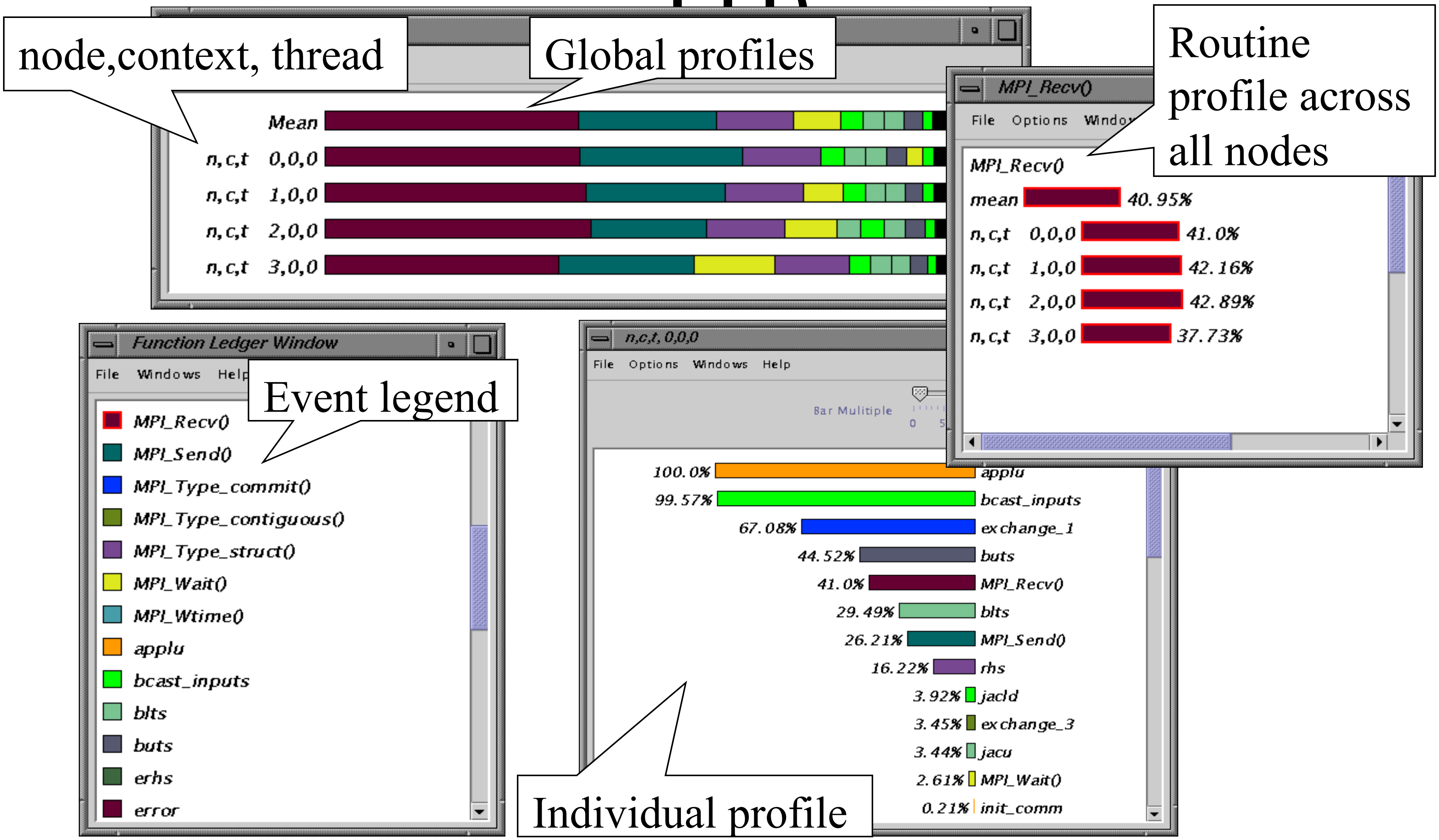
  -
    ```
    opari file1.f90
    ...
    opari fileN.f90
    ```
- Generate OPARI runtime table, compile it with ANSI C

  -
    ```
    opari -table opari.tab.c
    cc -c opari.tab.c
    ```
- Compile modified files `*.mod.f90` using OpenMP
- Link the resulting object files, the OPARI runtime table `opari.tab.o` and the TAU POMP RTL

TAU Analysis

- Parallel profile analysis
  - *Pprof*
    - parallel profiler with text-based display
  - *ParaProf*
    - Graphical, scalable, parallel profile analysis and display
- Trace analysis and visualization
  - Trace merging and clock adjustment (if necessary)
  - Trace format conversion (SDDF, VTF, Paraver)
  - Trace visualization using Intel Trace Analyzer (Pallas VAMPIR)

ParaProf (NAS Parallel Benchmark LU)



node,context, thread

Global profiles

Routine profile across all nodes

Event legend

Individual profile

Mean
n,c,t 0,0,0
n,c,t 1,0,0
n,c,t 2,0,0
n,c,t 3,0,0

MPI_Recv()

MPI_Recv()

mean	40.95%
n,c,t 0,0,0	41.0%
n,c,t 1,0,0	42.16%
n,c,t 2,0,0	42.89%
n,c,t 3,0,0	37.73%

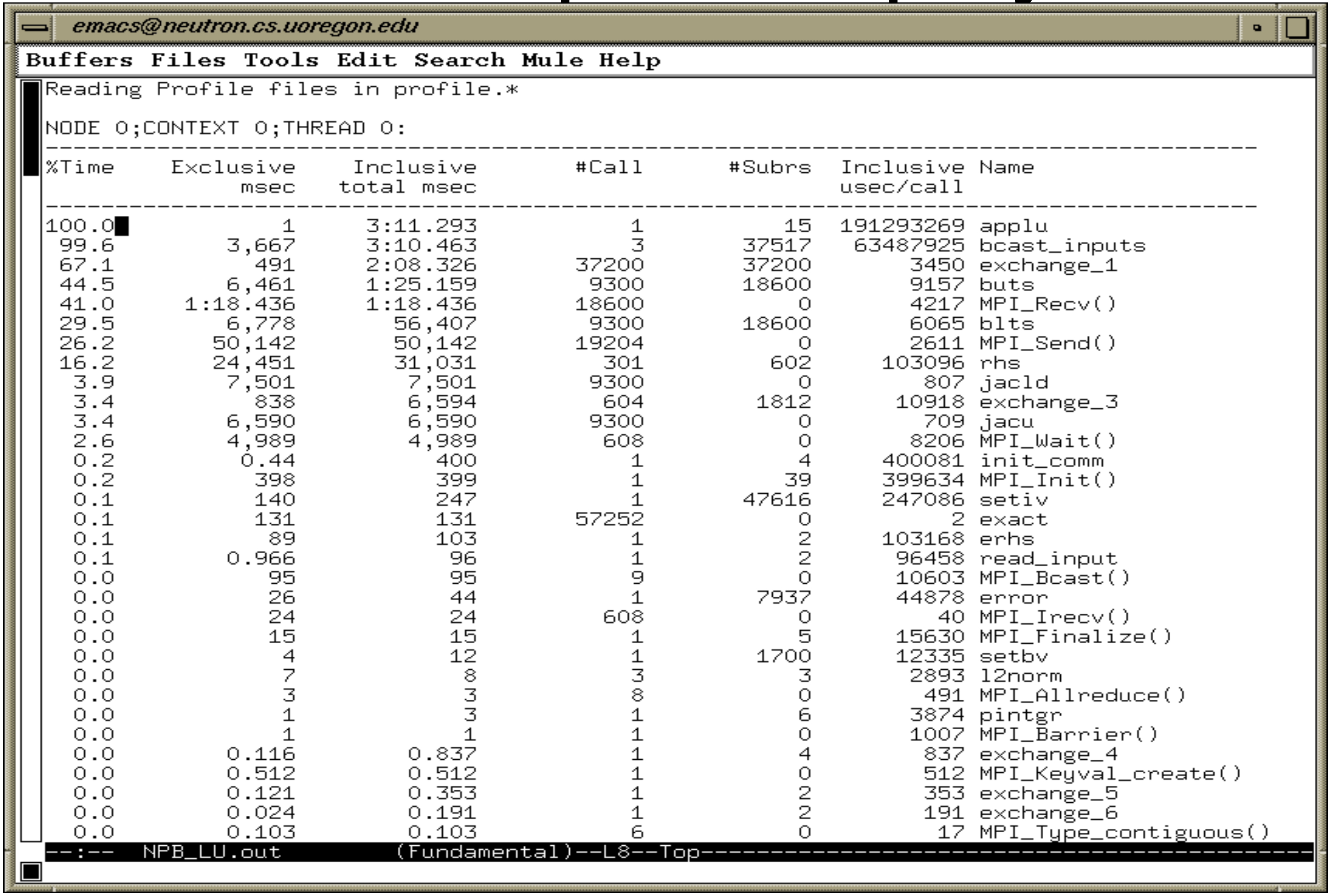
Function Ledger Window

- MPI_Recv()
- MPI_Send()
- MPI_Type_commit()
- MPI_Type_contiguous()
- MPI_Type_struct()
- MPI_Wait()
- MPI_Wtime()
- applu
- bcast_inputs
- blts
- buts
- erhs
- error

n,c,t,0,0,0

100.0%	applu
99.57%	bcast_inputs
67.08%	exchange_1
44.52%	buts
41.0%	MPI_Recv()
29.49%	blts
26.21%	MPI_Send()
16.22%	rhs
3.92%	jacld
3.45%	exchange_3
3.44%	jacu
2.61%	MPI_Wait()
0.21%	init_comm

TAU Pprof Display



```
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            1     3:11.293           1          15  191293269 applu
 99.6        3,667     3:10.463           3       37517   63487925 bcast_inputs
 67.1          491     2:08.326       37200       37200       3450 exchange_1
 44.5        6,461     1:25.159        9300       18600       9157 buts
 41.0     1:18.436     1:18.436       18600           0       4217 MPI_Recv()
 29.5        6,778       56,407        9300       18600       6065 blts
 26.2       50,142       50,142       19204           0       2611 MPI_Send()
 16.2       24,451       31,031         301         602     103096 rhs
  3.9        7,501        7,501        9300           0        807 jacld
  3.4          838        6,594         604        1812      10918 exchange_3
  3.4        6,590        6,590        9300           0        709 jacu
  2.6        4,989        4,989         608           0       8206 MPI_Wait()
  0.2         0.44          400           1           4     400081 init_comm
  0.2          398          399           1          39     399634 MPI_Init()
  0.1          140          247           1       47616     247086 setiv
  0.1          131          131       57252           0          2 exact
  0.1           89          103           1           2     103168 erhs
  0.1        0.966           96           1           2      96458 read_input
  0.0           95           95           9           0      10603 MPI_Bcast()
  0.0           26           44           1        7937      44878 error
  0.0           24           24         608           0         40 MPI_Irecv()
  0.0           15           15           1           5      15630 MPI_Finalize()
  0.0            4           12           1        1700      12335 setbv
  0.0            7            8           3           3       2893 l2norm
  0.0            3            3           8           0        491 MPI_Allreduce()
  0.0            1            3           1           6       3874 pintgr
  0.0            1            1           1           0       1007 MPI_Barrier()
  0.0        0.116        0.837           1           4        837 exchange_4
  0.0        0.512        0.512           1           0        512 MPI_Keyval_create()
  0.0        0.121        0.353           1           2        353 exchange_5
  0.0        0.024        0.191           1           2        191 exchange_6
  0.0        0.103        0.103           6           0         17 MPI_Type_contiguous()
--:--  NPB_LU.out        (Fundamental)--L8--Top-------------------------------------------
```

Debug Compile Options

- **Intel**
  - **For use with a debugger**

    -g –O0
  - **Full options to catch code failures(very slow)**

```
-g -traceback -CB -check uninit -fpe0 \
-check arg_temp_created -check pointers
```

- **PGI**
  - **For use with a debugger**

```
-g -O0
```

  - **Full options to catch code failures(very slow)**

```
-O0 -g -C -Mchkfpstk -Mchkptr -Mchkstk \
 -Ktrap=fp -traceback
```

Standard Debuggers

- The standard command line debugging tool is **gdb** in Linux. You can use these debuggers for programs written in C, C++ and Fortran.
- For effective debugging a couple of commands need to be mastered – set breakpoints, display the value of variables, set new values, and single step through a program. Less used commands can be learned as they become necessary.
- A High Level interface allows users to start, stop and record events. (Provides a “standard” set of controls)
- A Low Level interface allows developers to manipulate events and variables.

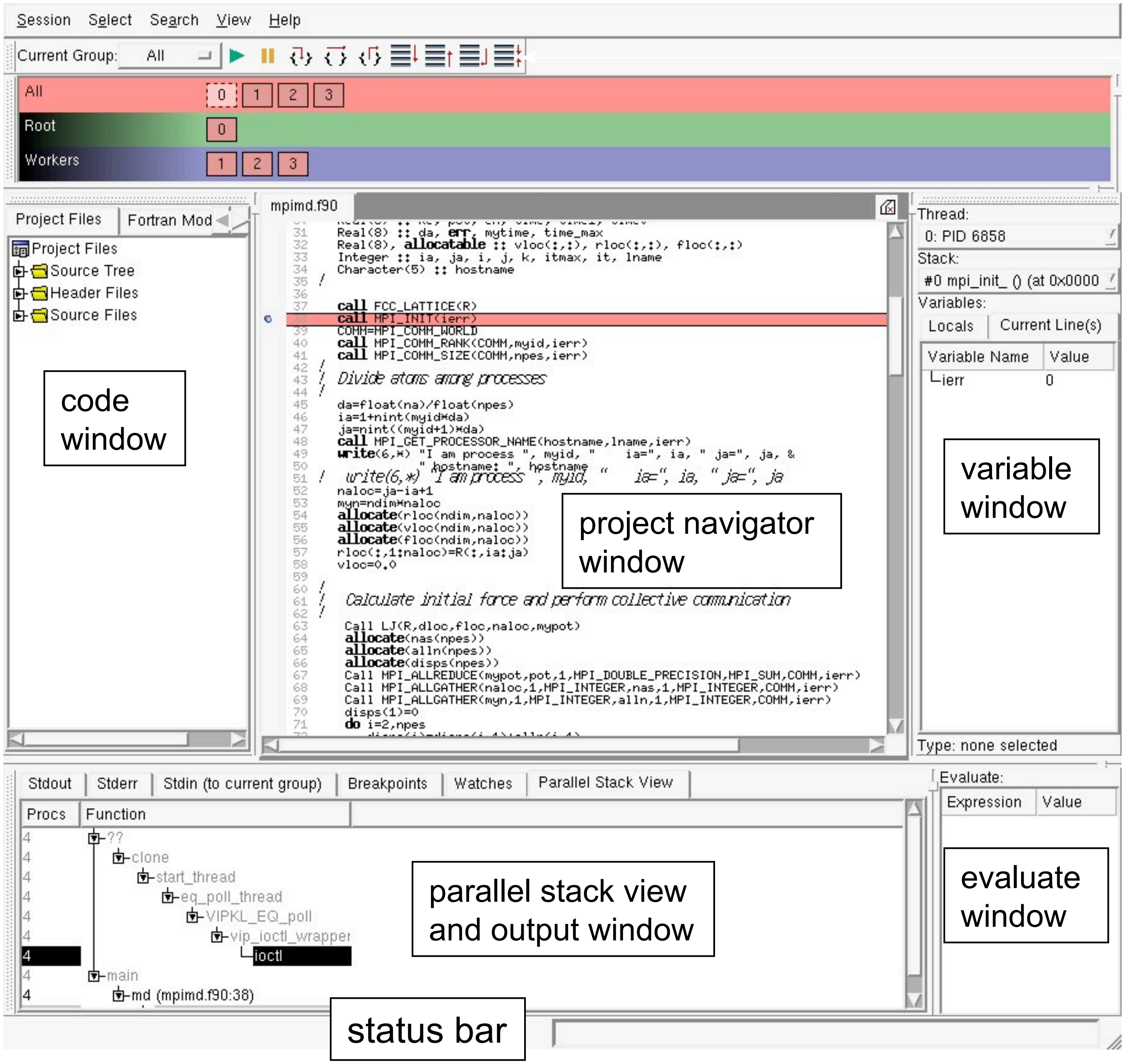
Parallel Debugging: DDT

Interactive, parallel, symbolic debuggers with GUI interface

- Works with C, C++ and Fortran Compilers
- Available on my different platforms.
  (IBM, CRAY, AMD, INTEL, SUN, SGI, …)
- Supports OpenMP & MPI
  (and hybrid paradigm)
- Support 32- and 64-bit architectures
- Simple to use (intuitive)

Instrumenting Code and Running TotalView

```
% module load ddt    {sets environment variables}
% module help ddt    {follow instructions}
% ifort -g prog.f90
% ddt &
```



```
      Real(8) :: da, err, mytime, time_max
      Real(8), allocatable :: vloc(:,:), rloc(:,:), floc(:,:)
      Integer :: ia, ja, i, j, k, itmax, it, lname
      Character(5) :: hostname
!

      call FCC_LATTICE(R)
      call MPI_INIT(ierr)
      COMM=MPI_COMM_WORLD
      call MPI_COMM_RANK(COMM,myid,ierr)
      call MPI_COMM_SIZE(COMM,npes,ierr)
!
!     Divide atoms among processes
!
      da=float(na)/float(npes)
      ia=1+nint(myid*da)
      ja=nint((myid+1)*da)
      call MPI_GET_PROCESSOR_NAME(hostname,lname,ierr)
      write(6,*) "I am process ", myid, "    ia=", ia, " ja=", ja, &
                 " hostname: ", hostname
!      write(6,*) "I am process ", myid, "    ia=", ia, " ja=", ja
      naloc=ja-ia+1
      myn=ndim*naloc
      allocate(rloc(ndim,naloc))
      allocate(vloc(ndim,naloc))
      allocate(floc(ndim,naloc))
      rloc(:,1:naloc)=R(:,ia:ja)
      vloc=0.0

!
!      Calculate initial force and perform collective communication
!
       Call LJ(R,dloc,floc,naloc,mypot)
       allocate(nas(npes))
       allocate(alln(npes))
       allocate(disps(npes))
       Call MPI_ALLREDUCE(mypot,pot,1,MPI_DOUBLE_PRECISION,MPI_SUM,COMM,ierr)
       Call MPI_ALLGATHER(naloc,1,MPI_INTEGER,nas,1,MPI_INTEGER,COMM,ierr)
       Call MPI_ALLGATHER(myn,1,MPI_INTEGER,alln,1,MPI_INTEGER,COMM,ierr)
       disps(1)=0
       do i=2,npes
```

```
4   ??
4     clone
4       start_thread
4         eq_poll_thread
4           VIPKL_EQ_poll
4             vip_ioctl_wrapper
4               ioctl
4   main
4     md (mpimd.f90:38)
```

code window

project navigator window

variable window

parallel stack view and output window

evaluate window

status bar