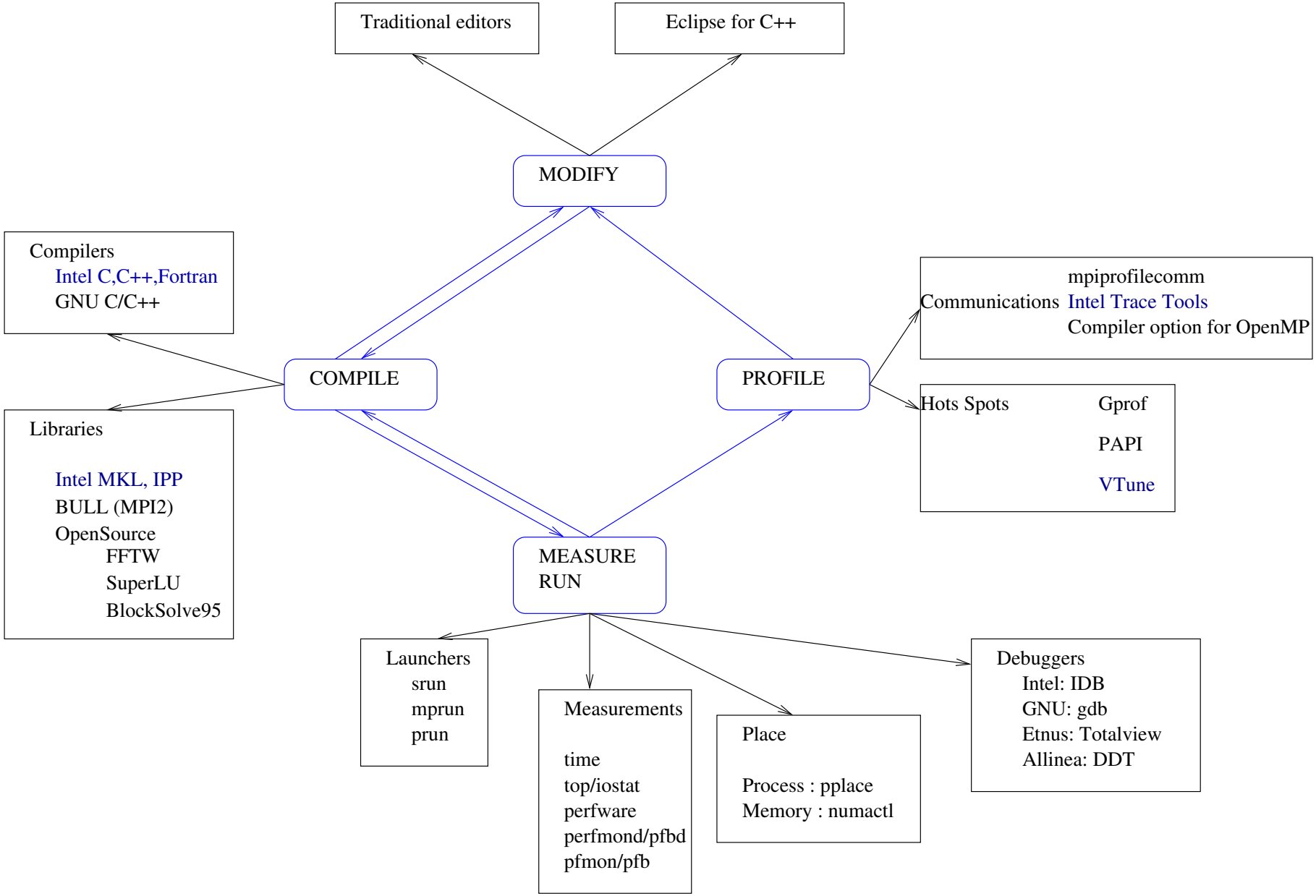
Performance HPC Linux
Bull Echirolles

Software Development Environment

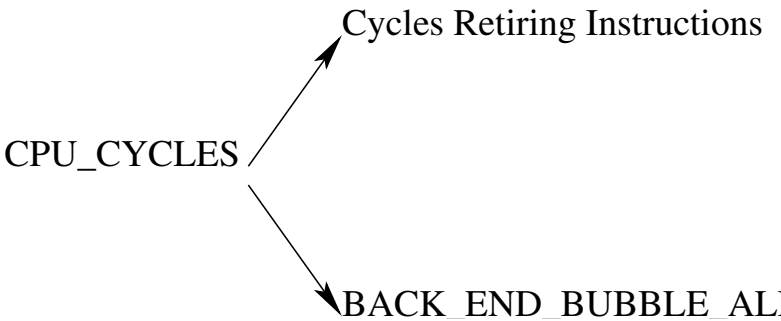
Part Three

C.Berthelot
Christophe.Berthelot@bull.net


Traditional editors
Eclipse for C++
MODIFY
Compilers
Intel C,C++,Fortran
GNU C/C++
COMPILE
PROFILE
Communications
mpiprofilecomm
Intel Trace Tools
Compiler option for OpenMP
Hots Spots
Gprof
PAPI
VTune
Libraries
Intel MKL, IPP
BULL (MPI2)
OpenSource
FFTW
SuperLU
BlockSolve95
MEASURE
RUN
Launchers
srun
mprun
prun
Measurements
time
top/iostat
perfware
perfmond/pfbd
pfmon/pfb
Place
Process : pplace
Memory : numactl
Debuggers
Intel: IDB
GNU: gdb
Etnus: Totalview
Allinea: DDT

- Bubble Analysis
- Latency Analysis
- First dimension: The node
- Second dimension The interconnect
- Questions

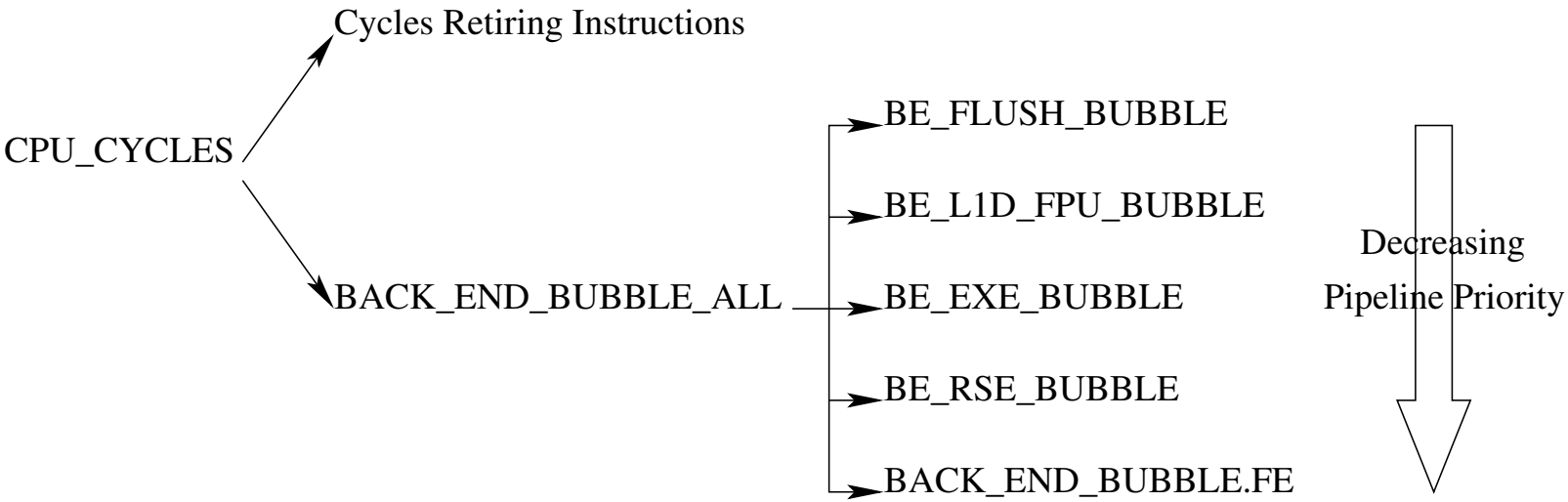
Starting the Tree Structure Analysis



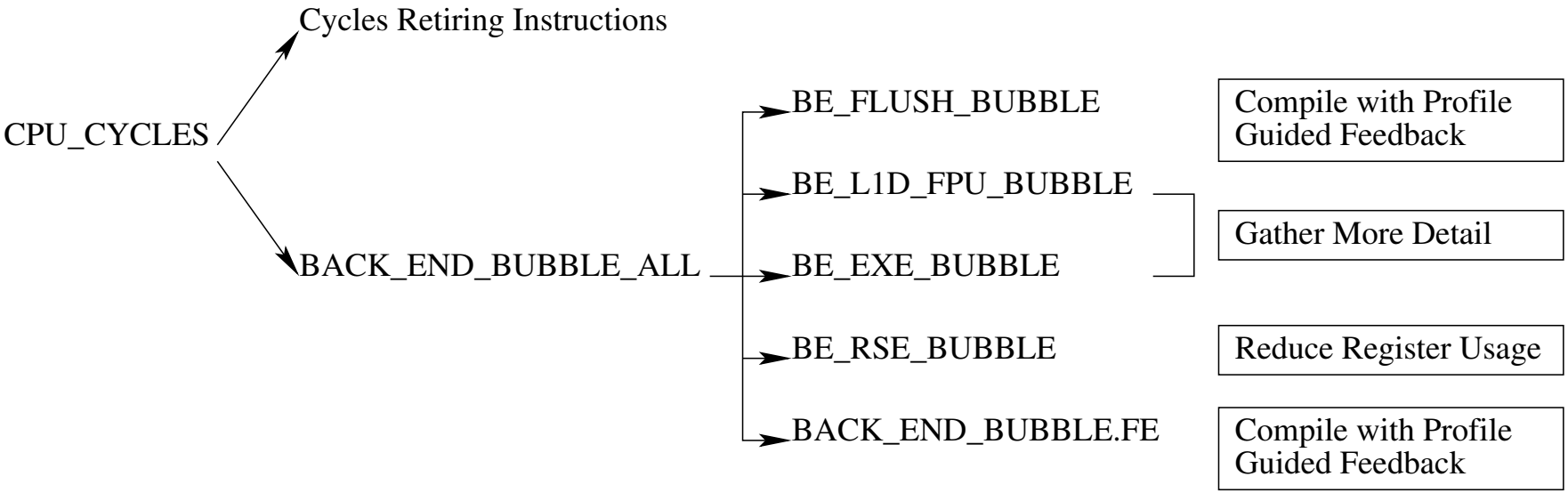
are the stall cycles to be reduced.

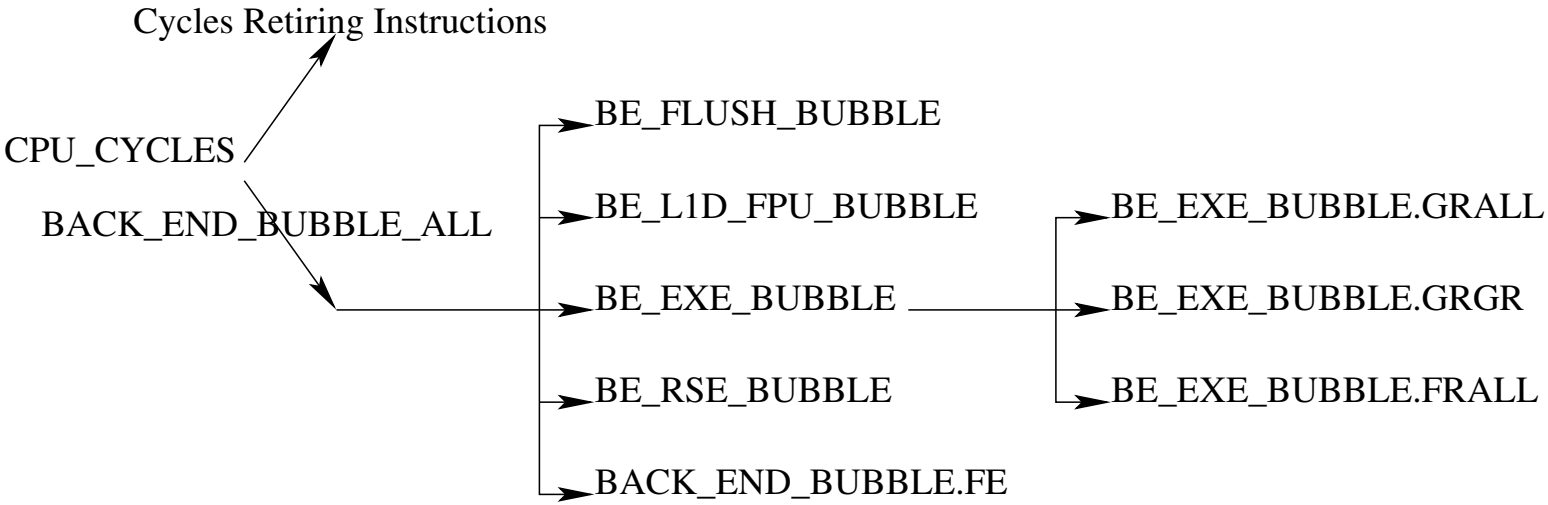
Eliminating the major contributions to this number is the most promising way for optimization

Stall Cycles Have 5 Components



$$
\begin{aligned}
BACK\_END\_BUBBLE \quad = \quad & BE\_FLUSH\_BUBBLE \\
+ \quad & BE\_L1D\_FPU\_BUBBLE \\
+ \quad & BE\_EXE\_BUBBLE \\
+ \quad & BE\_RSE\_BUBBLE \\
+ \quad & BACK\_END\_BUBBLE.FE
\end{aligned}
$$


Cycles Retiring Instructions
CPU_CYCLES
BACK_END_BUBBLE_ALL
BE_FLUSH_BUBBLE
BE_L1D_FPU_BUBBLE
BE_EXE_BUBBLE
BE_RSE_BUBBLE
BACK_END_BUBBLE.FE
Compile with Profile Guided Feedback
Gather More Detail
Reduce Register Usage
Compile with Profile Guided Feedback



- BE_EXE_BUBBLE.GRALL Counts all stall cycles due to waiting for valid data delivery to general register
- BE_EXE_BUBBLE.GRGR Counts all stall cycles due to waiting for valid data delivery to general register from an integer instruction
- BE_EXE_BUBBLE.GRALL-BE_EXE_BUBBLE.GRGR Approximates memory access stall for integer data
- BE_EXE_BUBBLE.FRALL Counts all stall cycles due to waiting for valid data delivery to an FP register

Long Latency Loads

L3 cache misses

L3 cache misses due to loads (and stores) cause hundreds of stalled cycles.
L3_reads.data_reads.miss is caused by loads, stores and prefetch instructions.

Data EAR events

- Allow user to select minimum latency (DATA_EAR_CACHE_LAT 8,16,32,64,...cycles)
- Minimum latency $>$ 32/64 identifies unprefetched data loads

The Big four

Events

- CPU_CYCLES (SAV[a]=1000000)
- BACK_END_BUBBBLE.ALL (SAV=1000000)
- DEAR_LATENCY_GT_64 (SAV=1000)
- BUS_MEMORY.ALL.SELF (SAV=10000)

[a] Sample After Value

Identifies

- Low hanging fruit with BACK_END_BUBBBLE.ALL
- Hotspots (cpu_cycles/back_end_bubble.all)
- Long latency data access/non prefetched data/thread sharing problemes
- Bandwidth limited loops

- Bubble Analysis
- Latency Analysis
- First dimension: The node
  Level 1: subroutines by subroutines
  Level 2: lines by lines
  Level 3: assembly
- Second dimension The interconnect
- Questions

Level 1: subroutines by subroutines

Tool

Classical open source gprof

Advantages

- Standard
- Well known
- Supported by Intel compiler
- Easy to use
- MPI support

Example

Traditional tool: gprof

- Simple to use (compile with -p)
- Use GPROF_PREFIX or GMON_OUT_PREFIX
- gprof -s a.out $GMON_OUT_PREFIX.*
- gprof a.out gmon.sum > summary-profile

Level 2: lines by lines

For tools

- Gprof
- Intel Vtune
- API PAPI
- HPCToolkit

Advantages

- Gprof: simple, standard
- Vtune: time information and hardware (and ratio) lines by lines inside a GUI or with command line interface
- PAPI: You need use api inside your source code

Medium Level: PAPI

PAPI aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware. PAPI is used for the following reasons:

- to provide a solid foundation for cross-platform performance analysis tools
- to present a set of standard definitions for performance metrics on all platforms
- to provide a standard API among users, vendors and academics.

PAPI supplies two interfaces:

- a high-level interface, for simple measures
- a low-level interface, programmable, adapting the specificity of the machine and linking the measures.

Example: Simple use

```
#include "fpapi.h"
      real real_time,cpu_time, mflops
      integer*8 fp_ins


      call PAPIf_flops(real_time,cpu_time,fp_ins,mflops,ierr)
c
c     Code to profile
c
      call PAPIf_flops(real_time,cpu_time,fp_ins,mflops,ierr)

       write (6,120) real_time, cpu_time, fp_ins,mflops
  120  format (//'PAPI result'/'real time (secs): ', f15.3,
     $  /'CPU time (secs): ',f15.3,
     $  /'floating point instructions: ', i15,
     $  /'MFLOPS: ', f15.3)
```

Usage haut-niveau

- Include header

```
#include "fpapi.h"
```

- Some declaration

```
 integer events(2),numevents, ierr
 character*PAPI_MAX_STR_LEN errorstring
```

- Array for counters

```
integer*8 values(2)
```

- Counters (cf fpapi.h)

```
numevents= 2
events(1)=PAPI_FP_INS
events(2)=PAPI_TOT_CYC
```

- Start measurements

```
call PAPIf_start_counters(events, numevents, ierr)
if ( ierr .NE. PAPI_OK ) then
 call PAPIF_perror(ierr,errorstring,PAPI_MAX_STR_LEN)
 print *, errorstring
end if
```

- Save counters values and restart counters

```
call PAPIf_read_counters(values,numevents,ierr)
if ( ierr .NE. PAPI_OK ) then
 call PAPIF_perror(ierr,errorstring,PAPI_MAX_STR_LEN)
 print *, errorstring
end if
```

- Accumulate counters

```
PAPIf_accum_counters(values,numevents,ierr)
```

- Stop counters:

```
call PAPIf_stop_counters(values, numevents, ierr)
if ( ierr .NE. PAPI_OK ) then
 call PAPIF_perror(ierr,errorstring,PAPI_MAX_STR_LEN)
 print *,errorstring
end if
```

PAPI with OpenMP

- Include: `use omp_lib #include <f90papi.h>`
- Initialization

```
chkflg = PAPI_VER_CURRENT
call PAPIF_library_init(chkflg)
if (chkflg .ne. PAPI_VER_CURRENT) then
   print *, 'Error initializing the PAPI Library'
   call abort
end if
```

- Threads Initialization

```
call PAPIF_thread_init(omp_get_thread_num,chkflg)
if (chkflg /= PAPI_OK) then
   print *, 'Error in subroutine PAPIF_thread_init'
   call abort
end if
```

Example

```
[berthelc@ns17 (CONF 09) MandelOpenMP_PAPI] ./mandel
-----------------------------
Thread           2 : MFLOPS =    1103.203
Thread           1 : MFLOPS =    1028.133
Thread           0 : MFLOPS =    1009.210
Thread           6 : MFLOPS =    1120.061
Thread           3 : MFLOPS =    1032.386
Thread           5 : MFLOPS =    1009.772
Thread           7 : MFLOPS =    1009.336
Thread           4 : MFLOPS =    1009.654
Loop seconds:   0.8052310
Total MFLOPS:       8321.755
```

HPCToolKit

HPCToolkit is an open-source suite of multi-platform tools for profile-based performance analysis of applications.

- hpcrun: a tool for profiling executions of unmodified application binaries using statistical sampling of hardware performance counters.
- csprof: a tool for collecting call path profiles of optimized application binaries.
- bloop: a tool for analyzing application binaries to recover program structure; namely, to identify where loops are present and what program source lines they contain.
- hpcprof: a tool for interpeting sample-based execution profiles and relating them back to program source lines.
- hpcview: a tool for correlating program structure information, multiple sample-based performance profiles, and program source code to produce a performance database.
- hpcviewer: a java-based GUI for exploring databases consisting of performance information correlated with program source.

First Use

Compile and Run

- Compile with -g
- run with hpcrun ./a.out with default events (PAPI_TOT_CYC)
- Result inside file a.out.PAPI_TOT_CYC.lynx0.25098.0x620a

Analysis 1/2

- Run hpcprof ./a.out a.out.PAPI_TOT_CYC.lynx0.25098.0x620a
- Result: ASCI or HTML

```
Columns correspond to the following events [event:period (events/sample)]
  PAPI_TOT_CYC:32767 - Total cycles (5162 samples)

Load Module Summary:
  99.8% /<path_to_bin>/cg.S
   0.2% /lib/ld-2.3.4.so
[...]
File Summary:
  98.7% <<path_to_bin>/cg.S>><path_to_src>/cg.f
   0.9% <<path_to_bin>/cg.S>><path_to_src>/randi8.f
[...]
```

Analysis 2/2

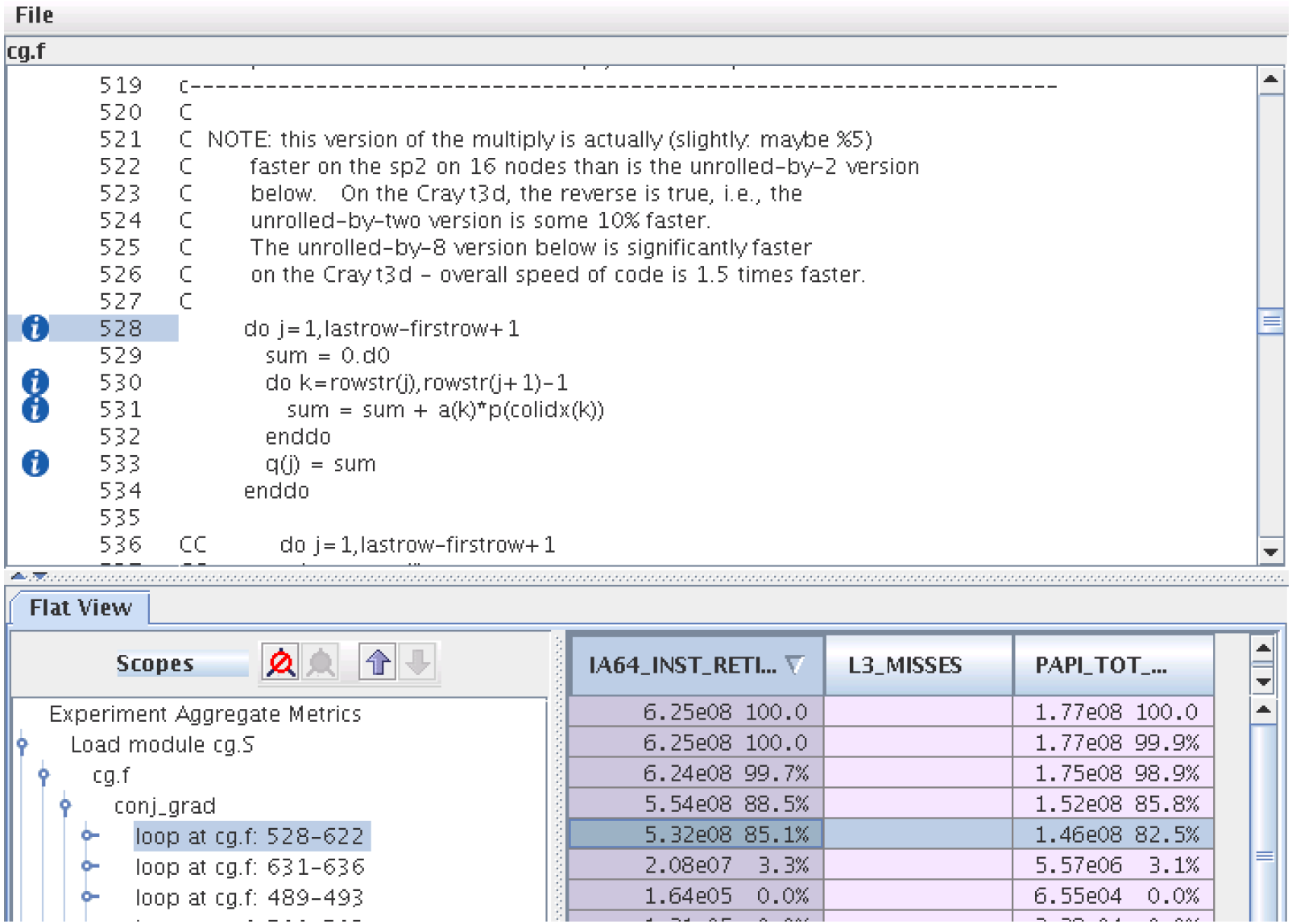
```
Function Summary:
  85.5% <</path_to_src>/cg.S>>conj_grad
  12.5% <<<path_to_bin/cg.S>>sparse
[...]

File <<path_to_bin/cg.S>><path_to_src>/cg.f with profile annotations.

   516         c---------------------------------------------------------------------
   517         c  q = A.p
   518         c  The partition submatrix-vector multiply: use workspace w
   519         c---------------------------------------------------------------------
   520         C
   521         C  NOTE: this version of the multiply is actually (slightly: maybe %5)
   522         C        faster on the sp2 on 16 nodes than is the unrolled-by-2 version
   523         C        below.   On the Cray t3d, the reverse is true, i.e., the
   524         C        unrolled-by-two version is some 10% faster.
   525         C        The unrolled-by-8 version below is significantly faster
   526         C        on the Cray t3d - overall speed of code is 1.5 times faster.
   527         C
   528   0.6%           do j=1,lastrow-firstrow+1
   529                     sum = 0.d0
   530  29.3%              do k=rowstr(j),rowstr(j+1)-1
   531  49.8%                 sum = sum + a(k)*p(colidx(k))
   532                     enddo
   533   1.1%              q(j) = sum
   534                  enddo
[...]
```

Use GUI

Run

- Analyze application binaries: `bloop ./a.out > a.out.psxml`
- Run: `hpcrun -e IA64_INST_RETIRED -e L3_MISSES -e PAPI_TOT_CYC ./a.out>profile`
- Run: `hpcquick -I<SRC_PATH> -S a.out.psxml -P profile -n`
- Run: `hpcview -o experiment-db hpcquick.xml`
- Run: `hpcviewer`

Hpcviewer



Level 3: assembly

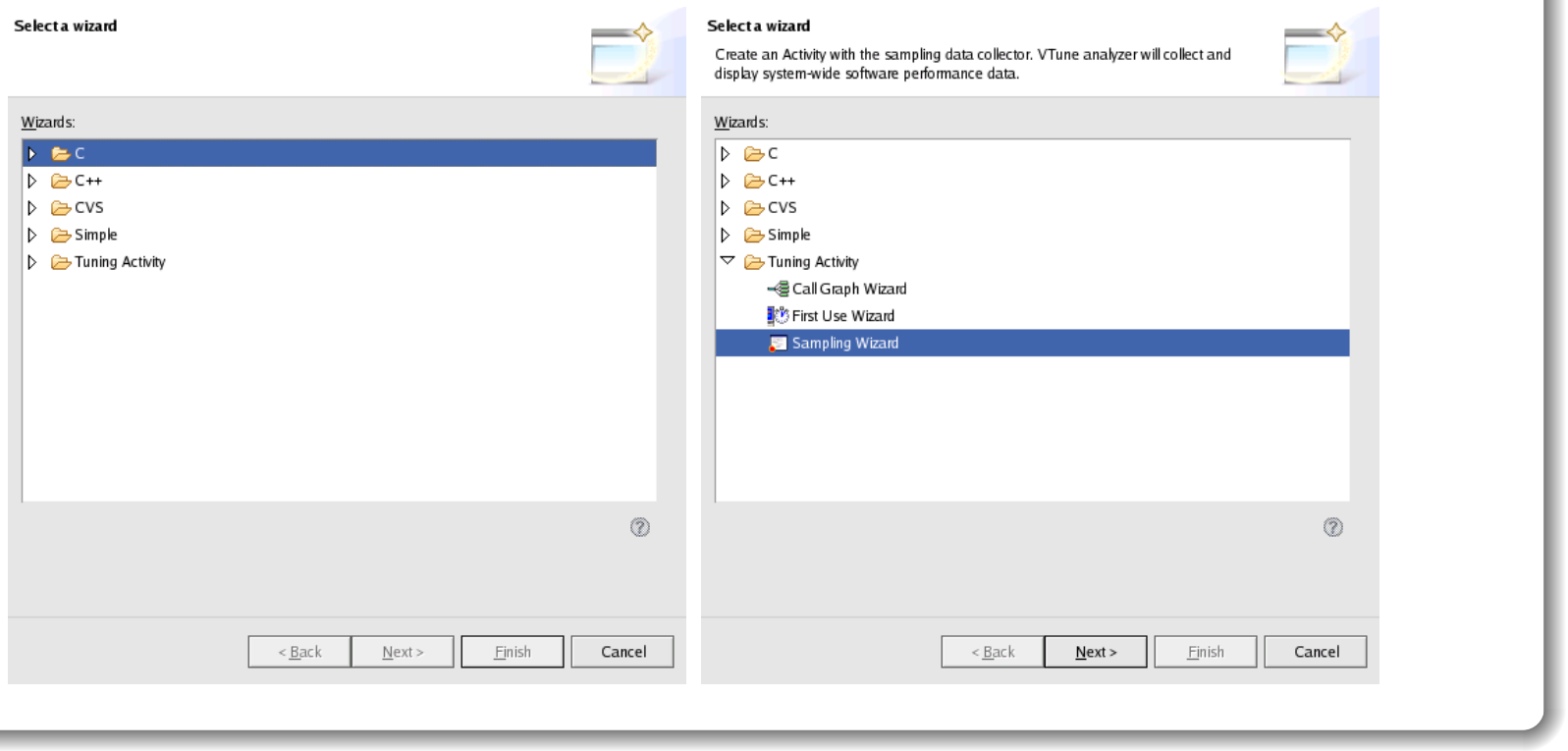
Vtune

Find performance bottlenecks with advanced profiling technologies:

- Event-Based, System-Wide Sampling with little impact on program execution (typically $< 1\%$).
- Call Graph Profiling offers a pictorial view of program flow to help you quickly identify critical functions.

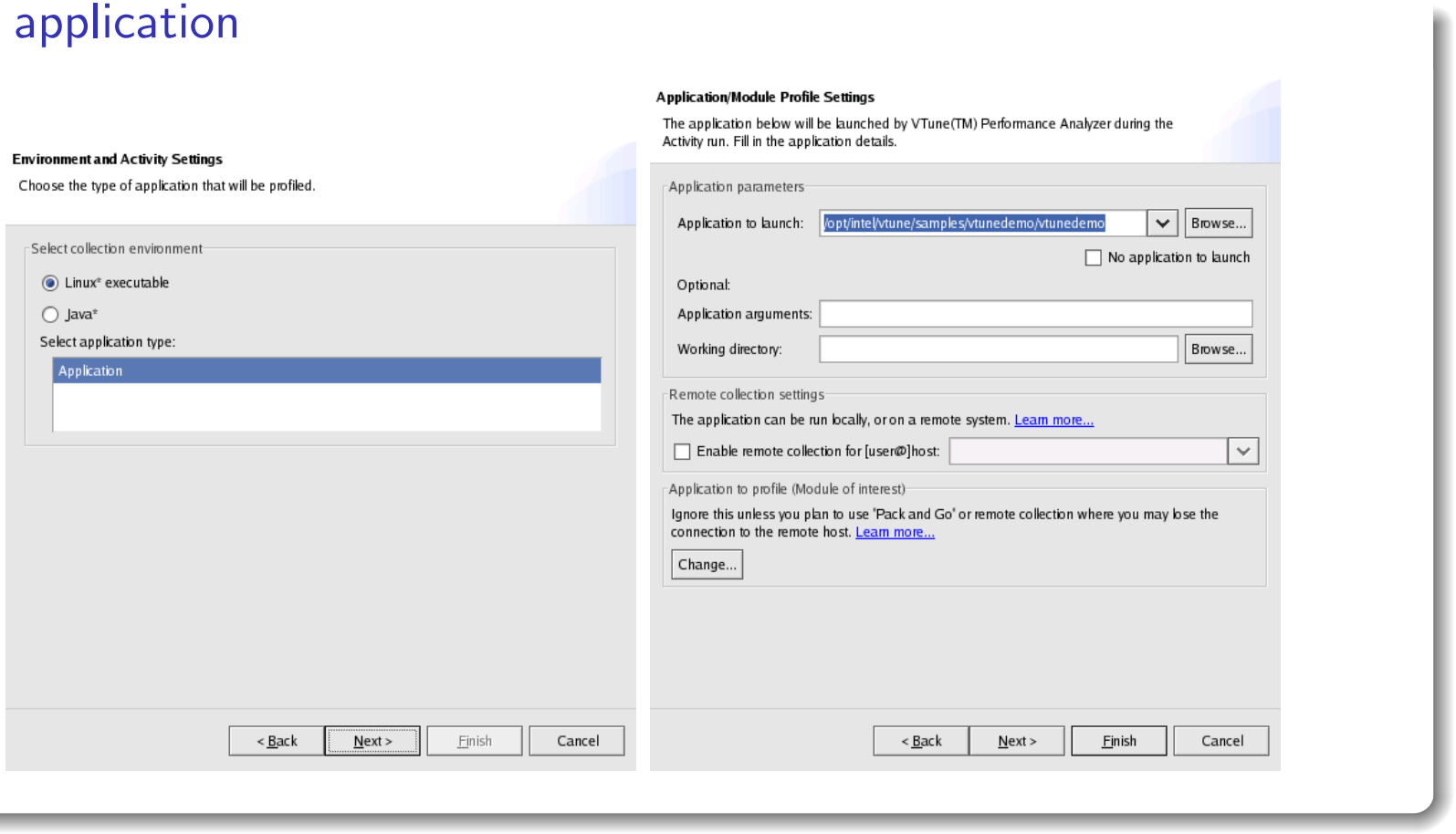
Sampling

File->New->Project...



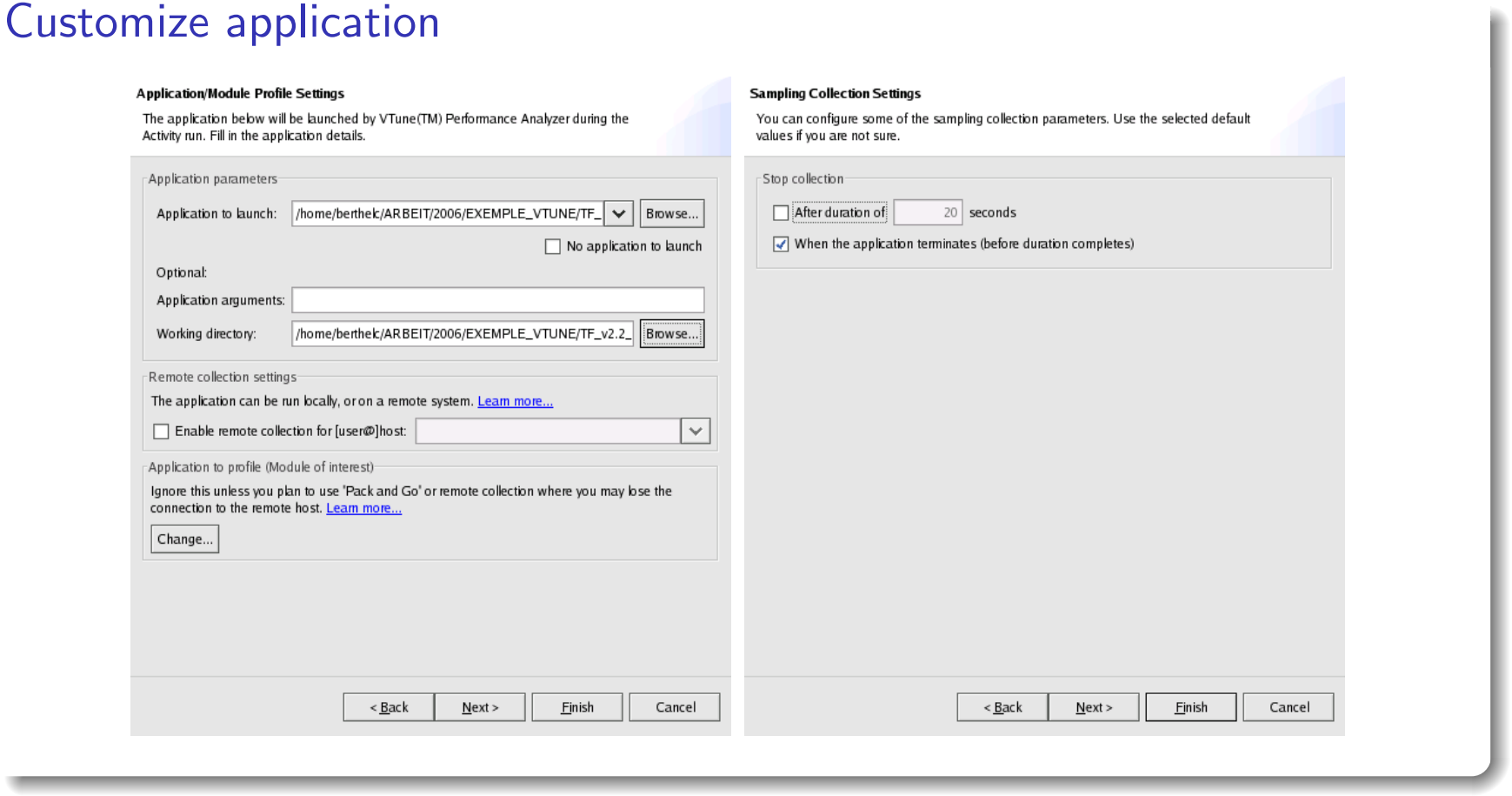
Sampling

Select application

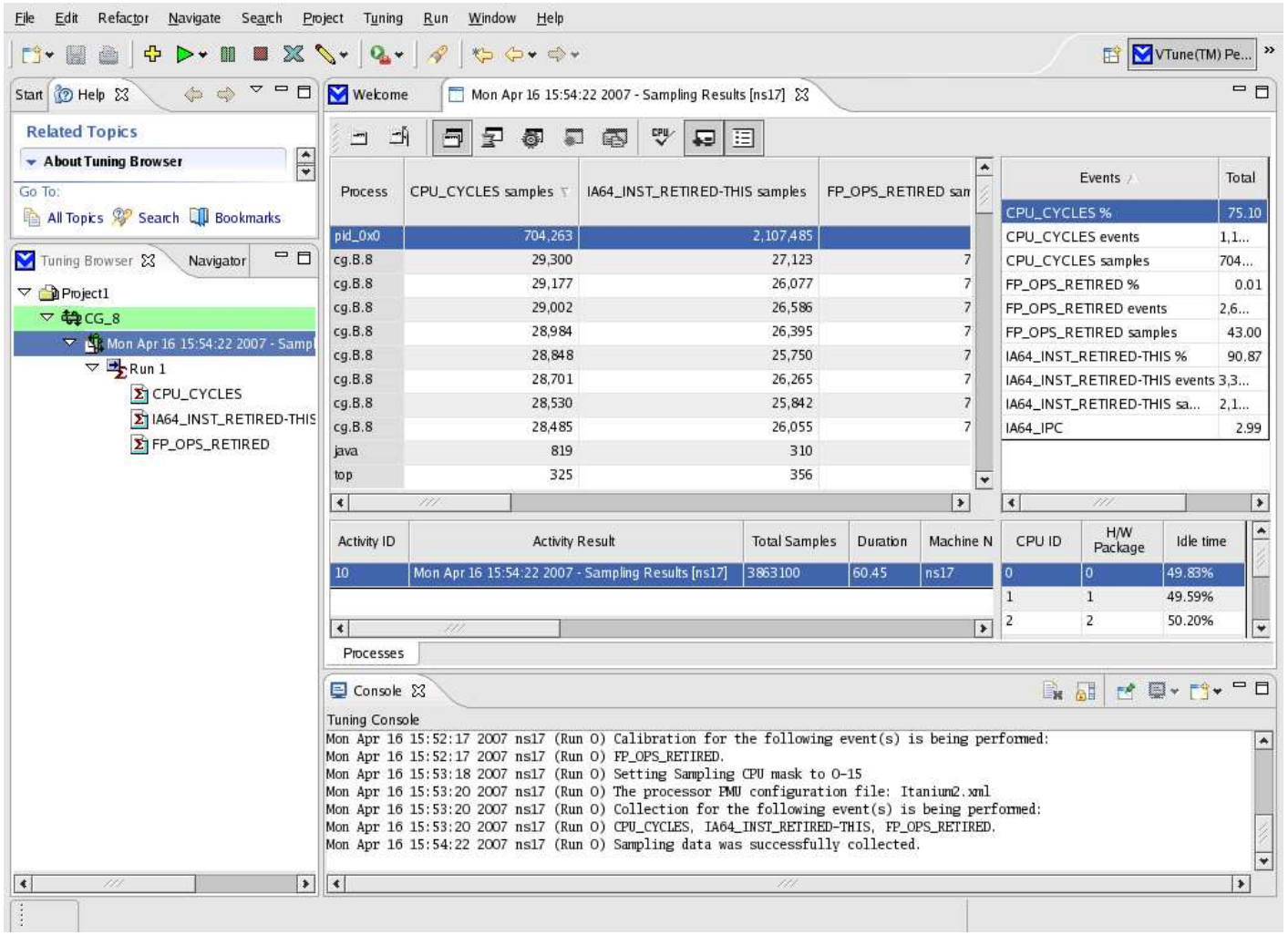


Sampling

Customize application



Sampling First windows



Sampling First windows

Commands line

- The default project file

```
>/opt/intel/vtune/bin/vtl  query -project
Project file: /tmp/vtune_berthelot/Projects/vtldefault.vpj
```

- Import data base inside

```
>/opt/intel/vtune/bin/vtl import tbs7b16.tb5
File tbs7b16.tb5 successfully imported...Use "vtl view <activity result name>"
    to view results
```

- Look activity name

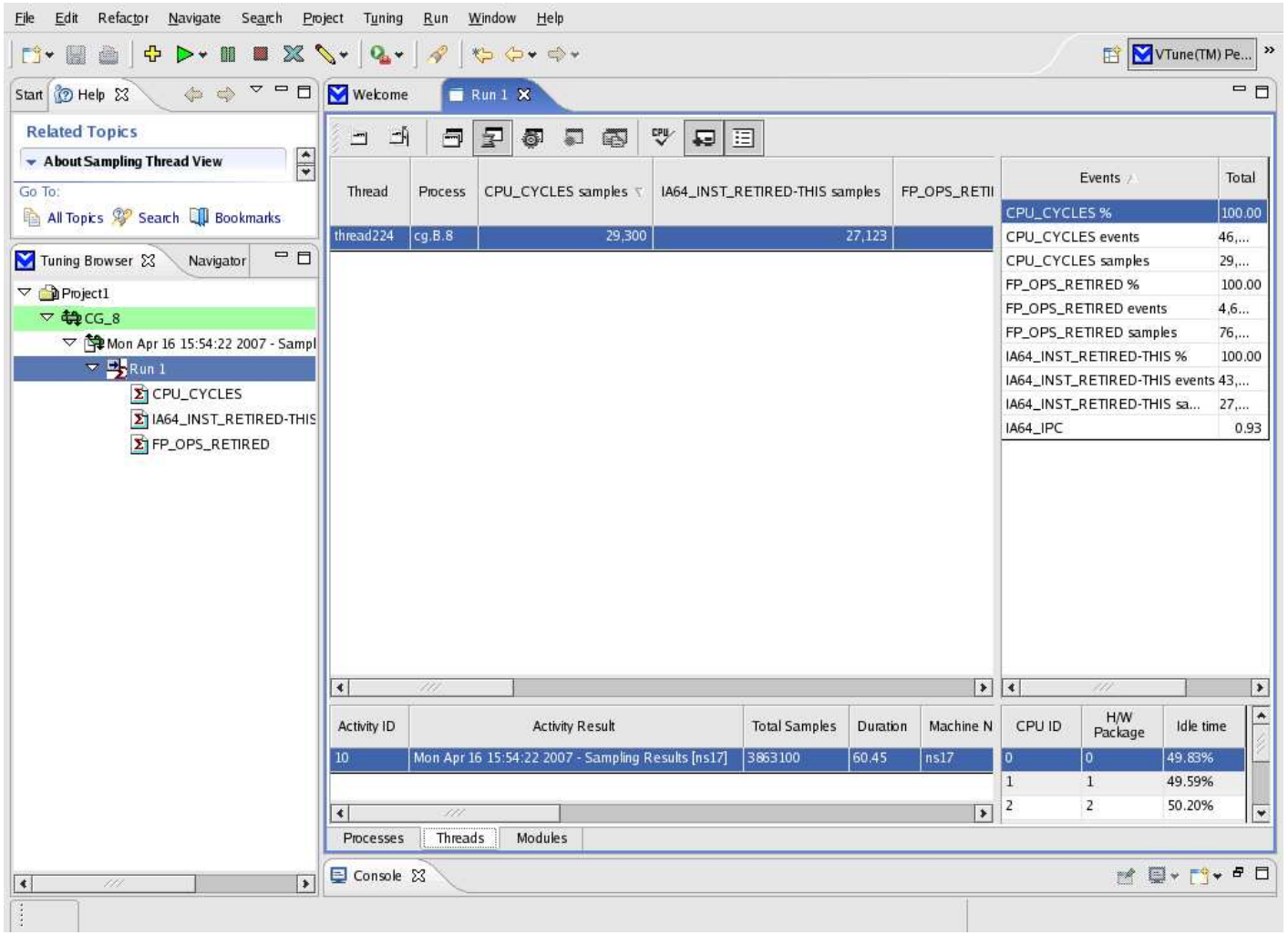
```
>/opt/intel/vtune/bin/vtl show
f1__ImportedResults
    r1___Imported Sampling Results
```

Command line view processes

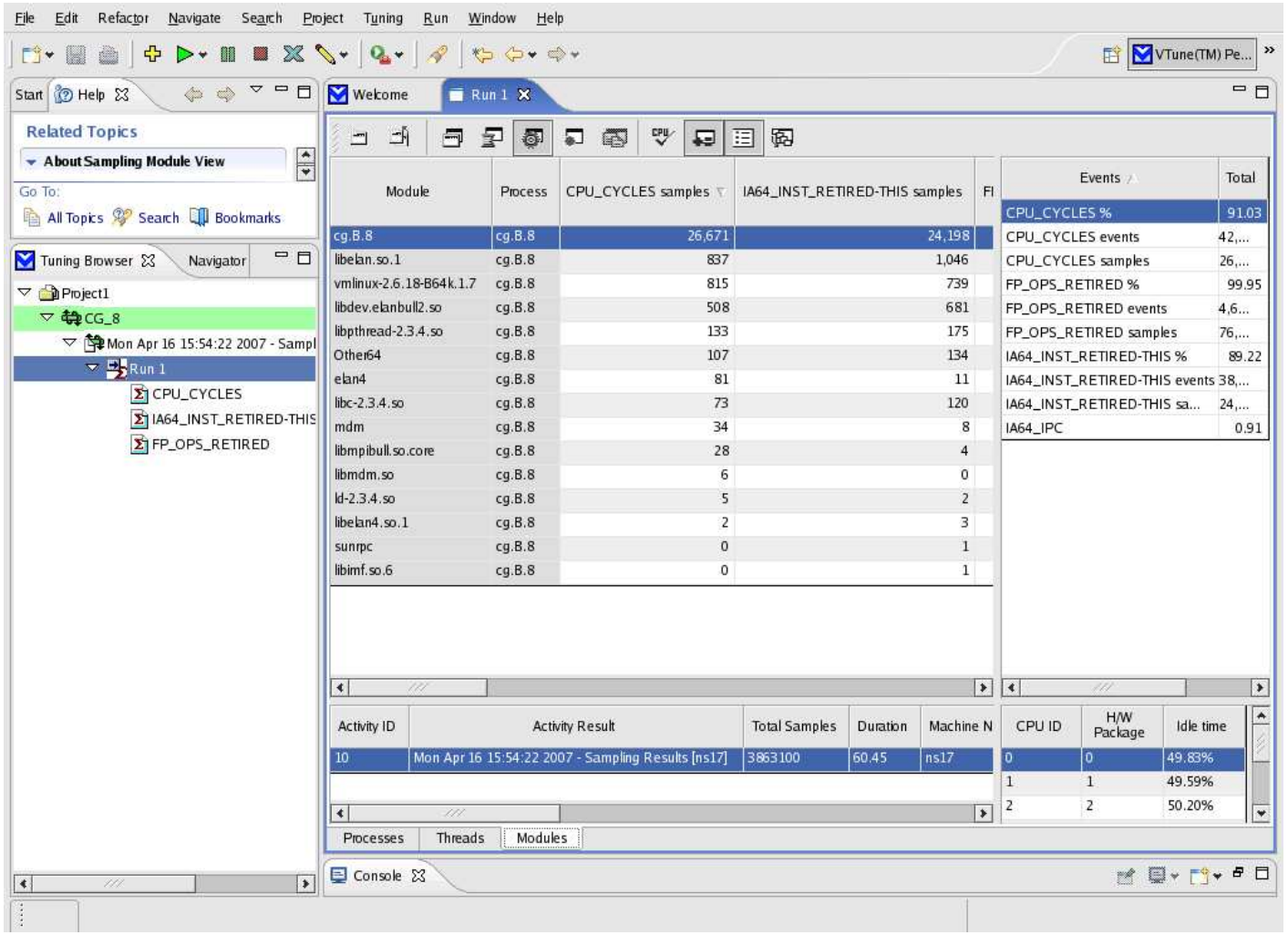
vtl view f1::r1 -processes

Process ID	CPU_CYCLES	IA64_INST_RETIRED-THIS	FP_OPS_RETIRED	IA64_IPC	
pid_0x0	704263	2107485	43	2.992	[...]
cg.B.8	29300	27123	76274	0.926	[...]
cg.B.8	29177	26077	75207	0.894	[...]
cg.B.8	29002	26586	74955	0.917	[...]
cg.B.8	28984	26395	75421	0.911	[...]
cg.B.8	28848	25750	75686	0.893	[...]
cg.B.8	28701	26265	76132	0.915	[...]
cg.B.8	28530	25842	75532	0.906	[...]
cg.B.8	28485	26055	75955	0.915	[...]
java	819	310	88	0.379	[...]
top	325	356	529	1.095	[...]
pid_0x43	234	127	7	0.543	[...]

Sampling: Process view



Sampling: Module view

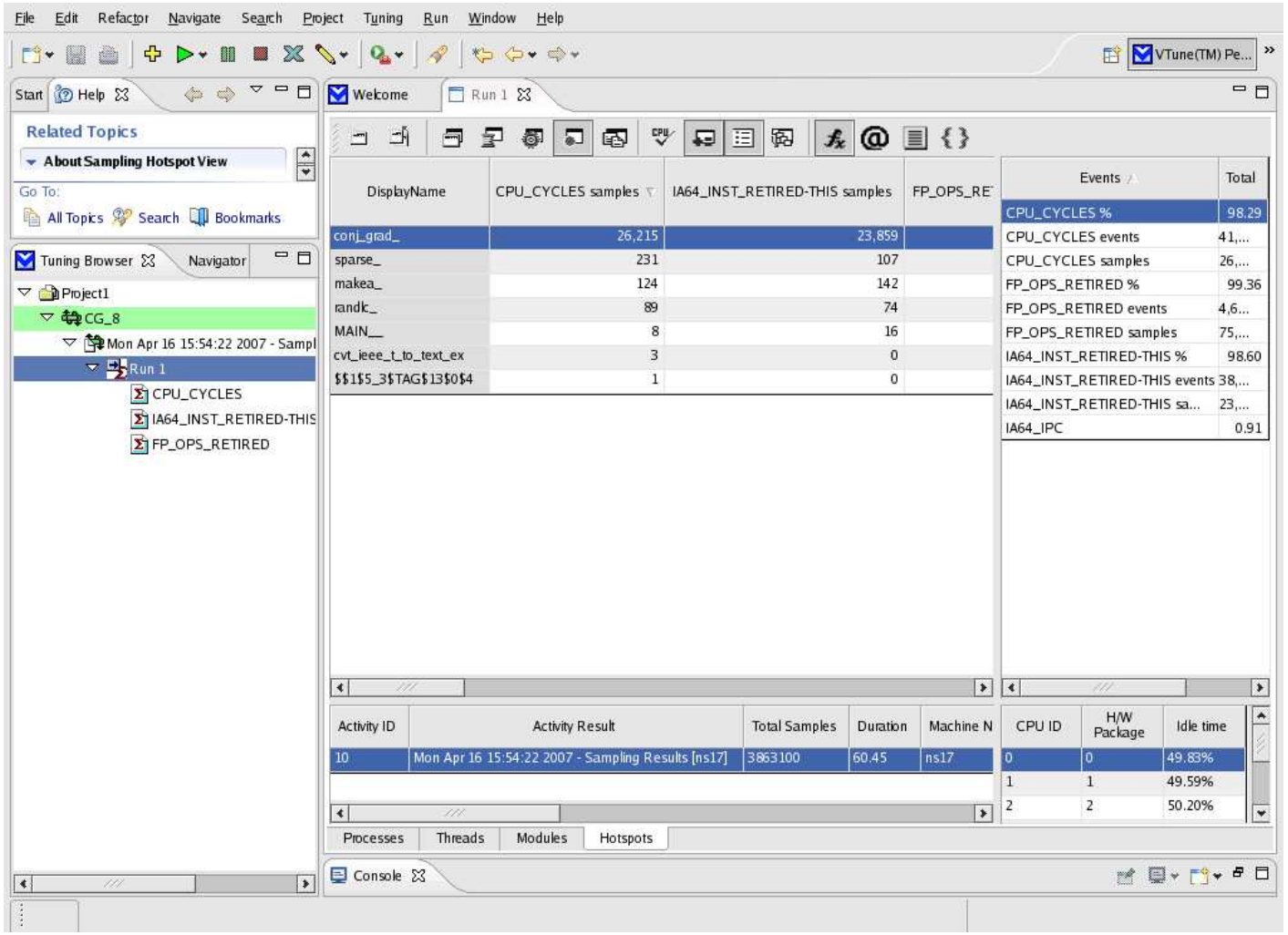


Command line view modules

vtl view f1::r1 -pid 0x6be4

Module	Process	CPU_CYCLES	IA64_INST_RETIRED-THIS	FP_OPS_RETIRED	IA64_IPC
cg.B.8	cg.B.8	26671	24198	76234	0.907
libelan.so.1	cg.B.8	837	1046	0	1.250
vmlinux-2.6.18-B64k.1.7	cg.B.8	815	739	37	0.907
libdev.elanbull2.so	cg.B.8	508	681	0	1.341
libpthread-2.3.4.so	cg.B.8	133	175	0	1.316
Other64	cg.B.8	107	134	0	1.252
elan4	cg.B.8	81	11	0	0.136
libc-2.3.4.so	cg.B.8	73	120	0	1.644
mdm	cg.B.8	34	8	0	0.235
libmpibull.so.core	cg.B.8	28	4	0	0.143
libmdm.so	cg.B.8	6	0	0	0.000
ld-2.3.4.so	cg.B.8	5	2	3	0.400
libelan4.so.1	cg.B.8	2	3	0	1.500
sunrpc	cg.B.8	0	1	0	0.000
libimf.so.6	cg.B.8	0	1	0	0.000

Sampling: Fonction view

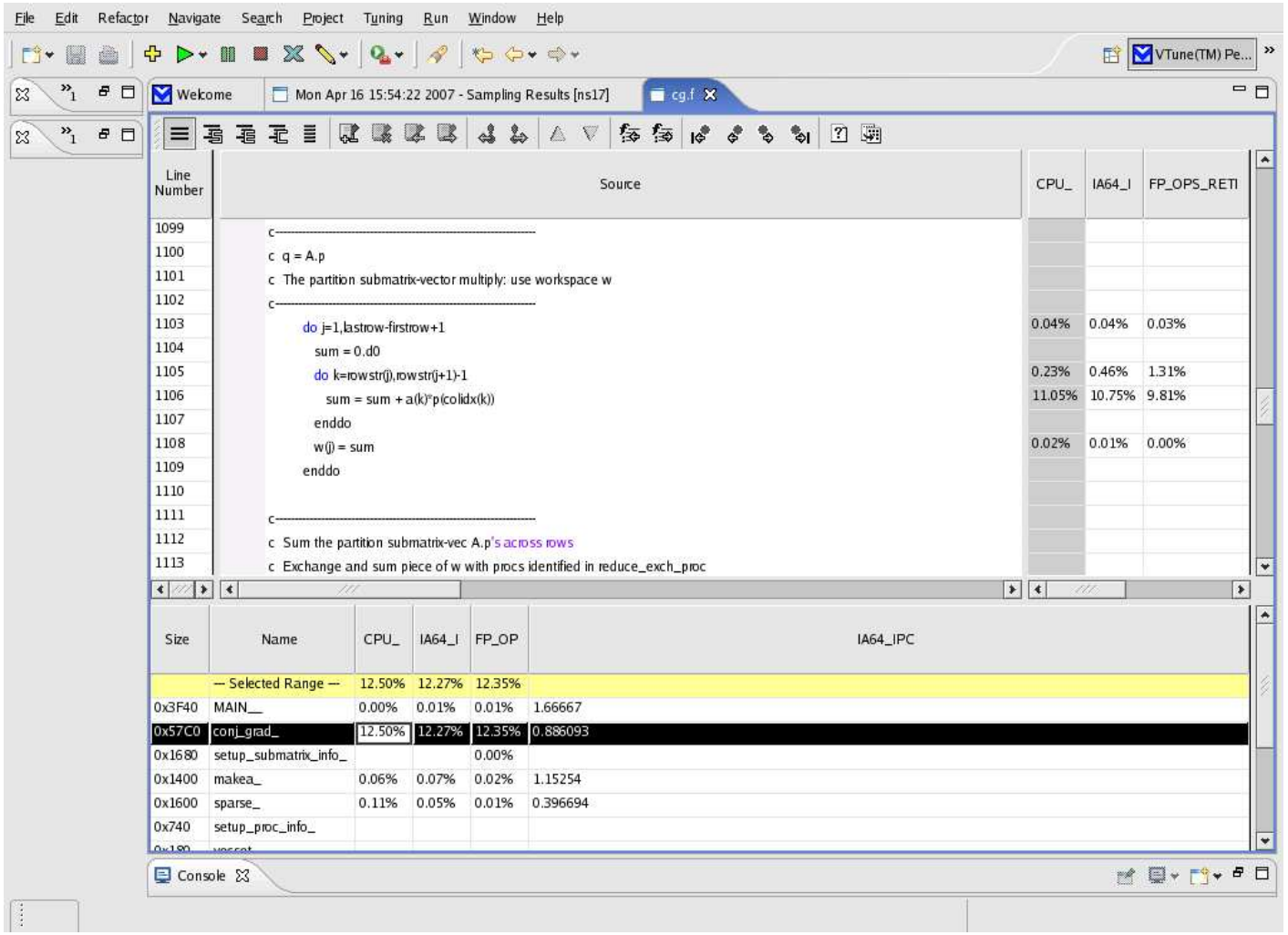


Command line fonction view

vtl view f1::r1 -hf -mn cg.B.8

Name	ModuleName	CPU_CYCLES	IA64_INST_RETIRED-THIS	FP_OPS_RETIRED	IA64_IPC
conj_grad_	cg.B.8	209425	189597	600975	0.905
sparse_	cg.B.8	1820	810	696	0.445
makea_	cg.B.8	928	1091	1079	1.176
randlc_	cg.B.8	739	609	1457	0.824
MAIN__	cg.B.8	69	121	589	1.754
cvt_ieee_t_to_text_ex	cg.B.8	3	0	0	0.000
$$1$5_3$TAG$13$0$4	cg.B.8	1	0	0	0.000
setup_submatrix_info_	cg.B.8	0	0	1	0.000
cvtas_t_to_a	cg.B.8	0	1	0	0.000

Sampling: line view



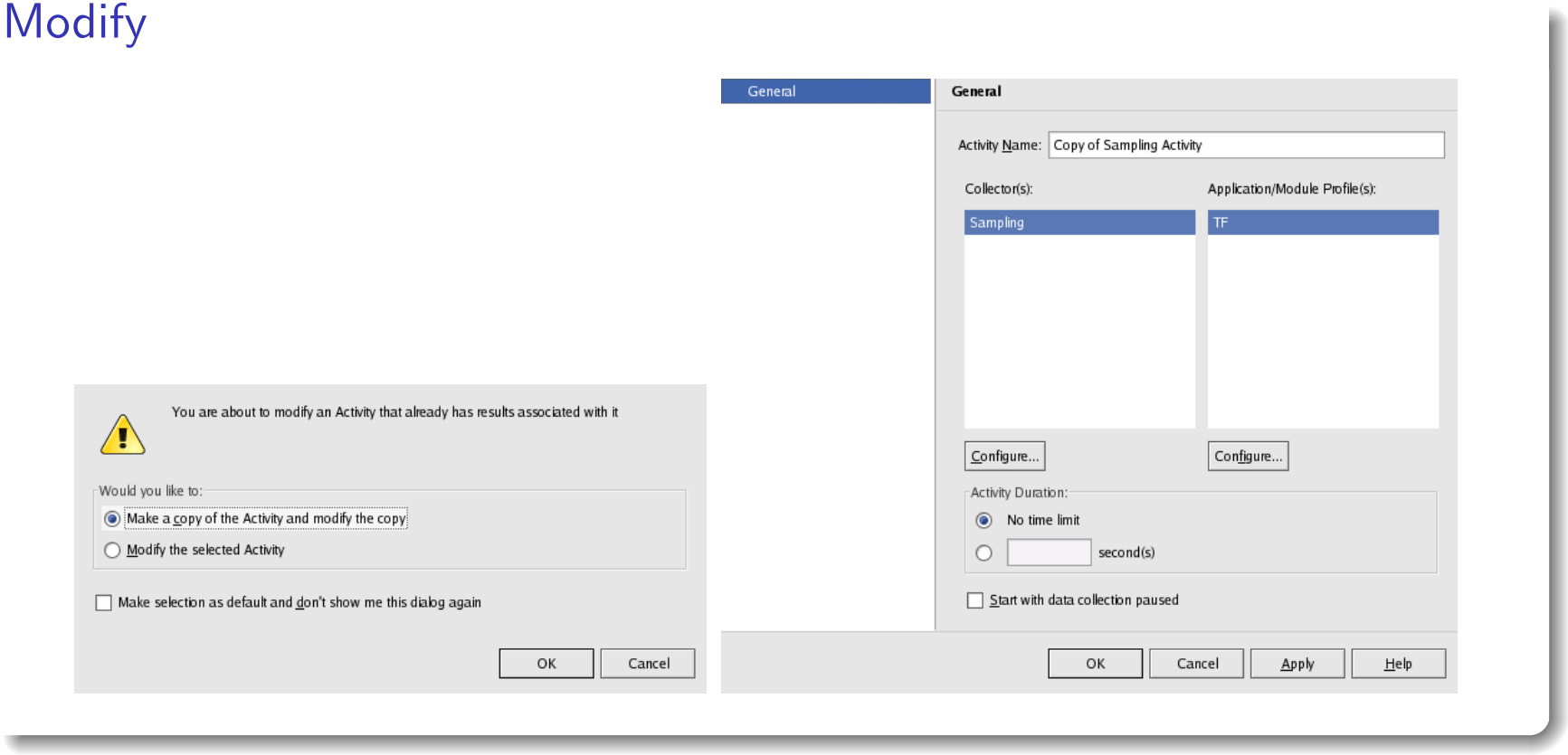
Command line line view

vtl view f1::r1 -code -mn cg.B.8 -fn conj_grad_

```
Legend
Ev1     =       CPU_CYCLES samples
Ev2     =       IA64_INST_RETIRED-THIS samples
Ev3     =       FP_OPS_RETIRED samples
Address Line Number Ev1    Ev2    Ev3    Source
0x7d90  1103        682    549    1529           do j=1,lastrow-firstrow+1
0x8121  1104        0      0      0                 sum = 0.d0
0x8120  1105        3903   6796   63078             do k=rowstr(j),rowstr(j+1)-1
0x8260  1106        185751 166594 478451               sum = sum + a(k)*p(colidx(k))
        1107        0      0      0                 enddo
0x8310  1108        276    85     93                w(j) = sum
        1109        0      0      0              enddo
```

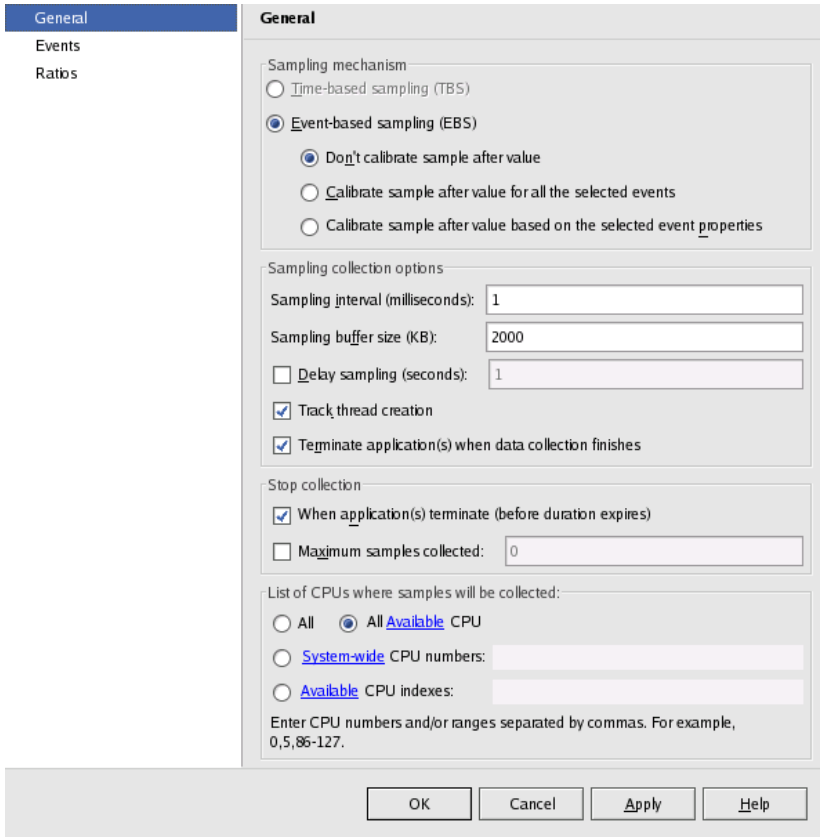
Sampling

Modify



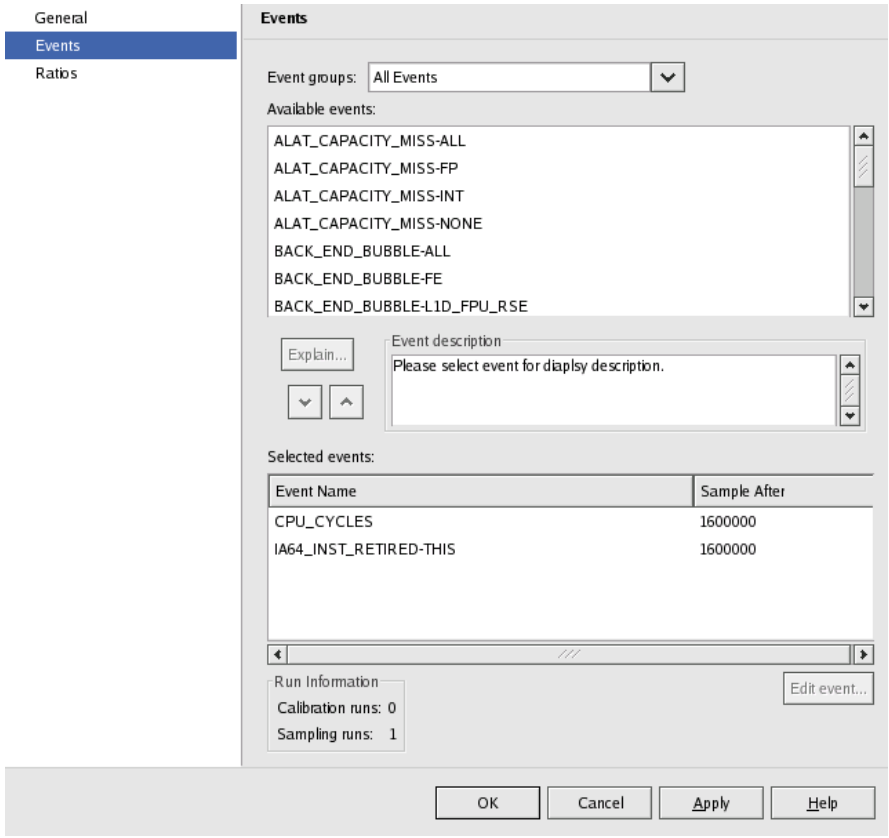
Sampling

Properties



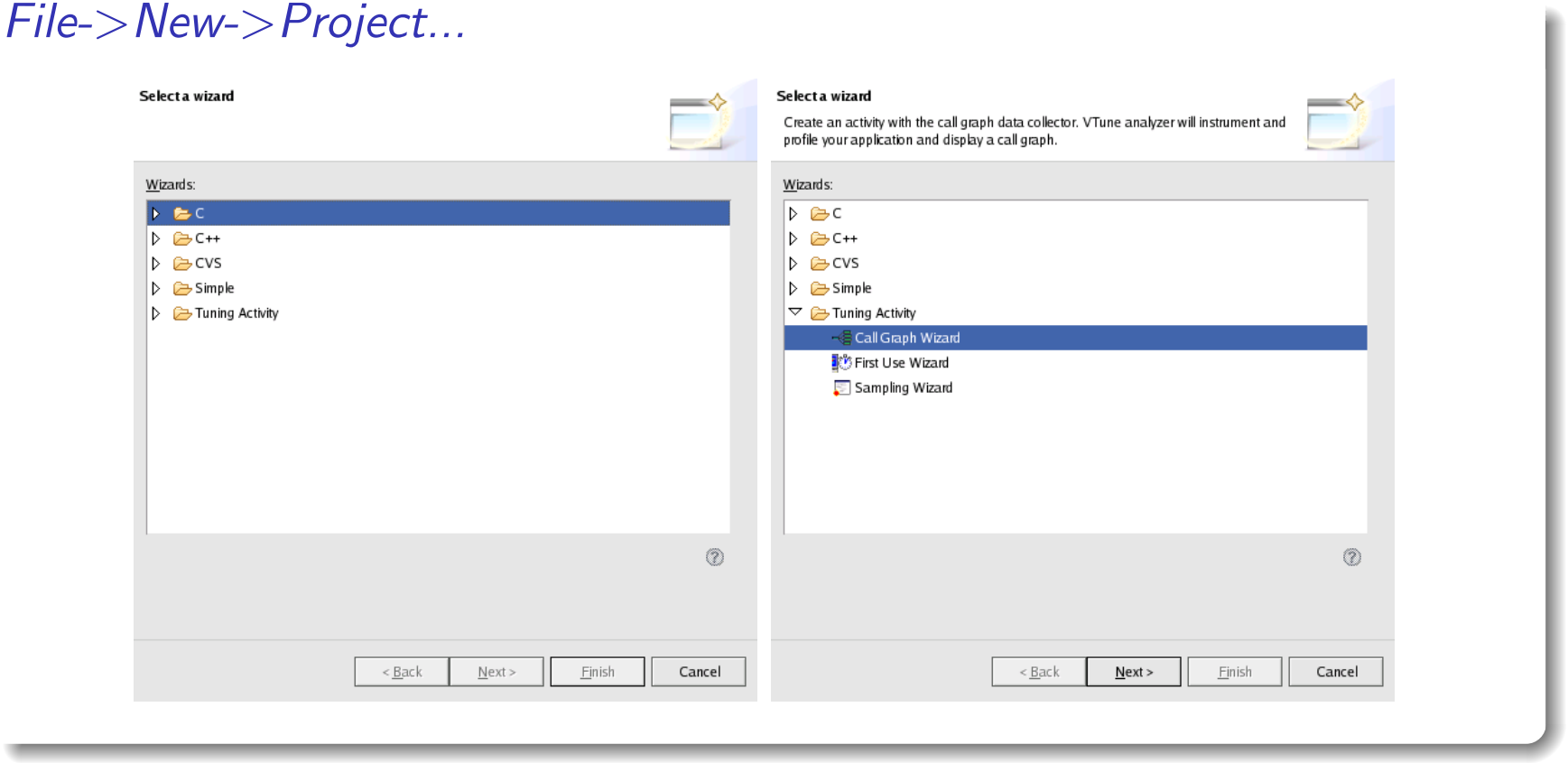
Sampling

Events



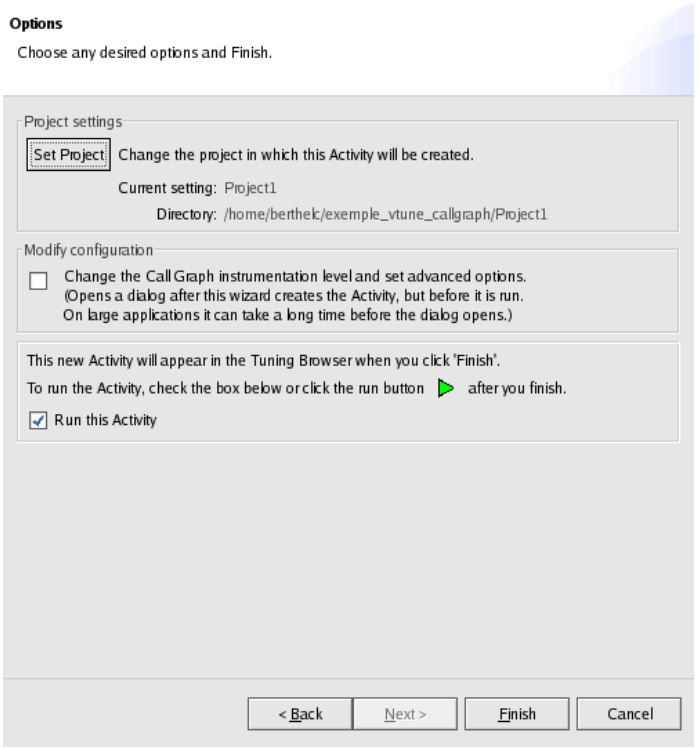
Call Graph

File->New->Project...



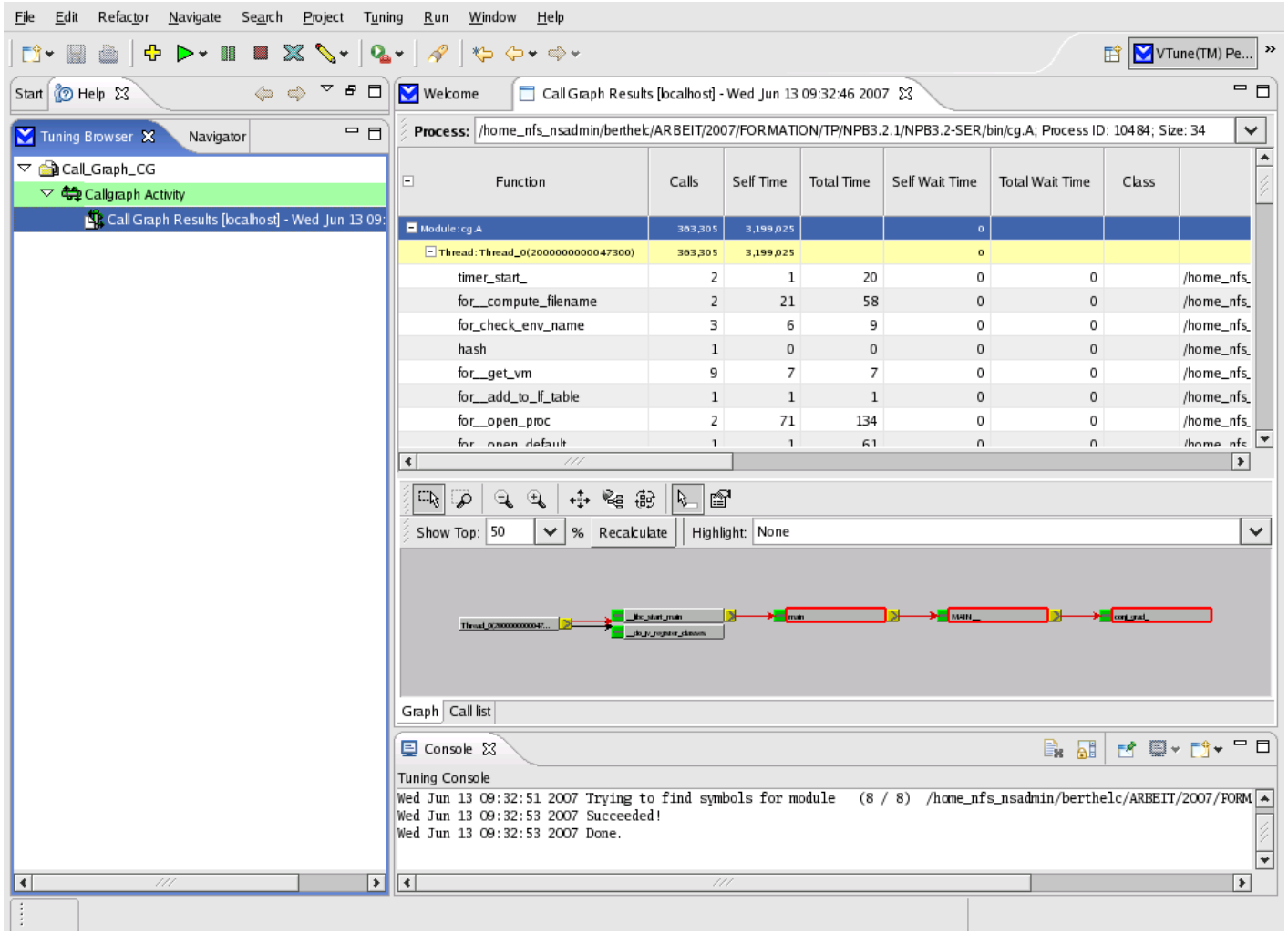
Call Graph

Options

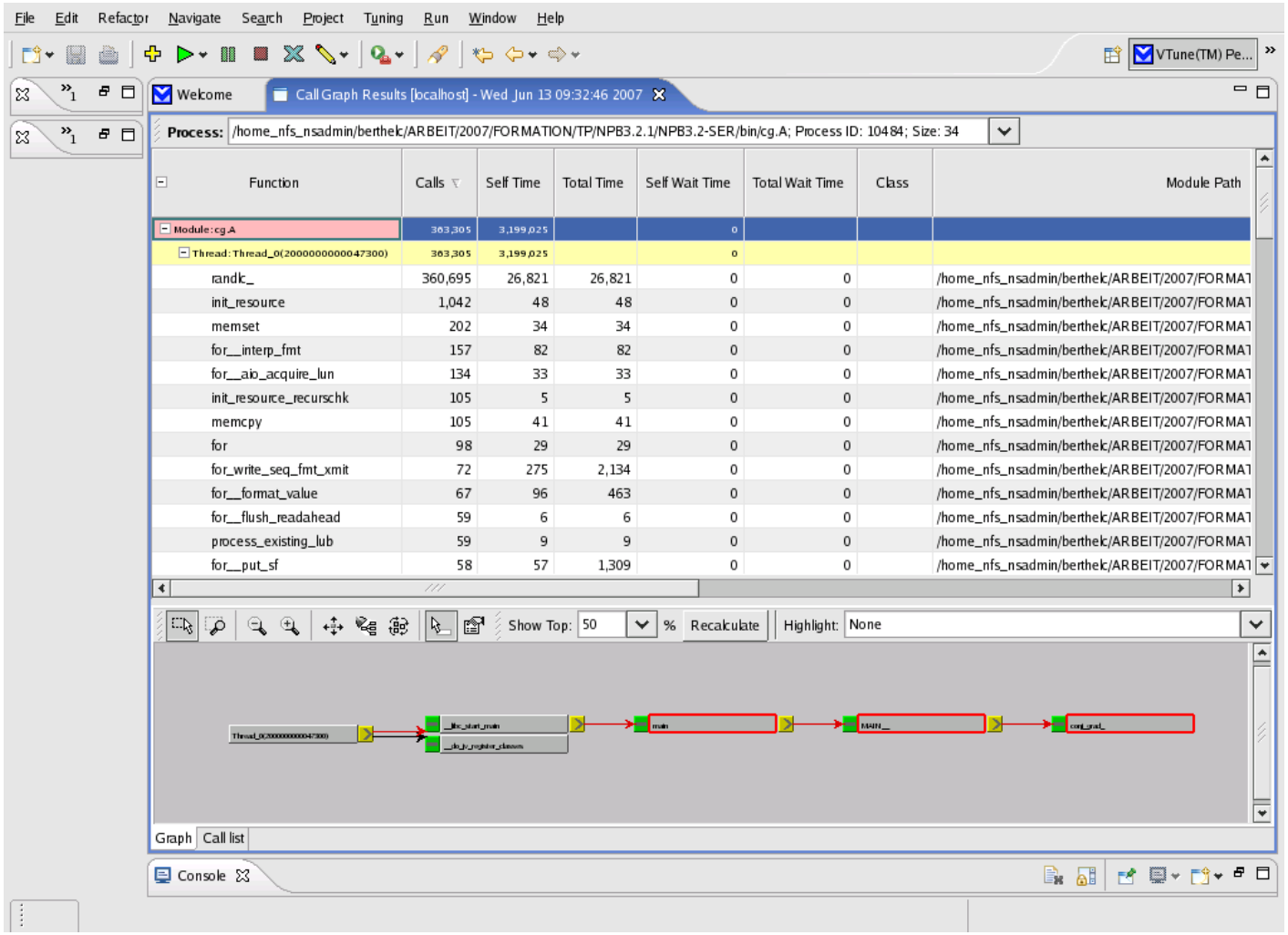


Call Graph

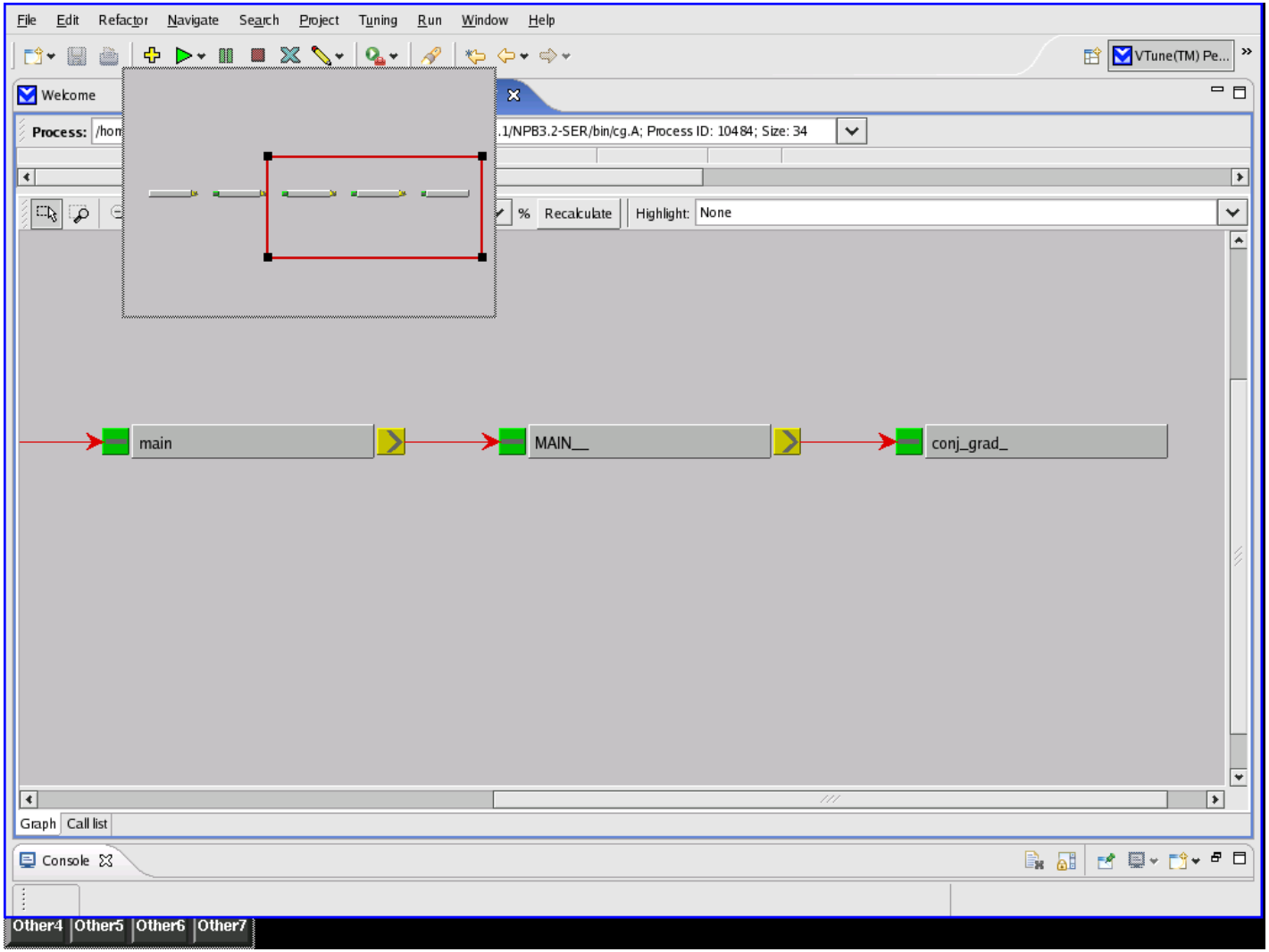
Results



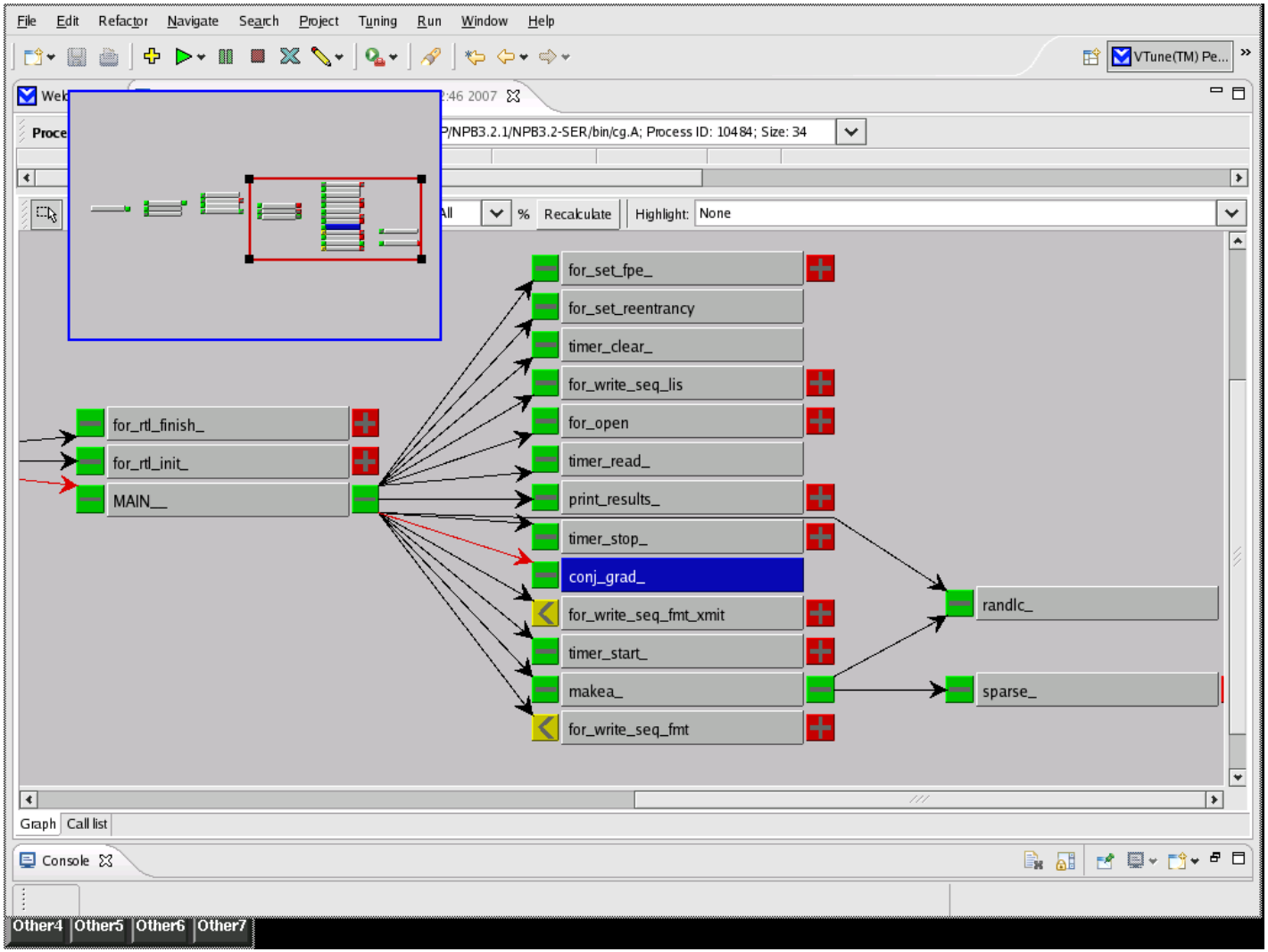
Call Graph



Call Graph



Call Graph



- Bubble Analysis
- Latency Analysis
- First dimension: The node
- Second dimension The interconnect
  MPI Analyser
  Intel Trace Analyzer
- Questions

Communications: first Level

MPIAnalyser

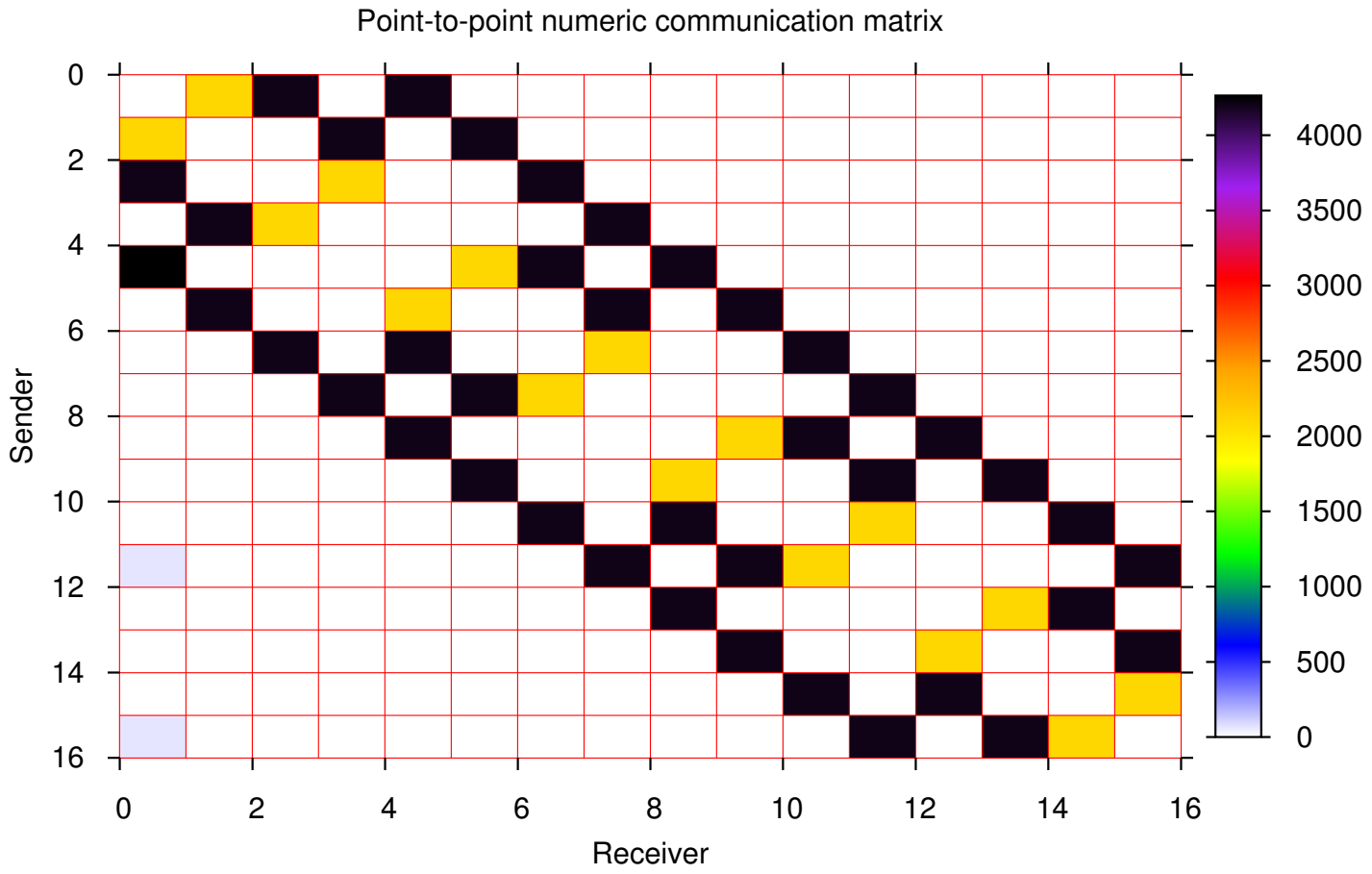
MPIAnalyser is a profiling tool dedicated to MPI applications.

- Light: it uses few resources and so does not slow down the application.
- Easy to run: it is used to characterize the MPI communications in a program. Communication matrices are constructed with it. Profilecomm is a "post-mortem" tool.

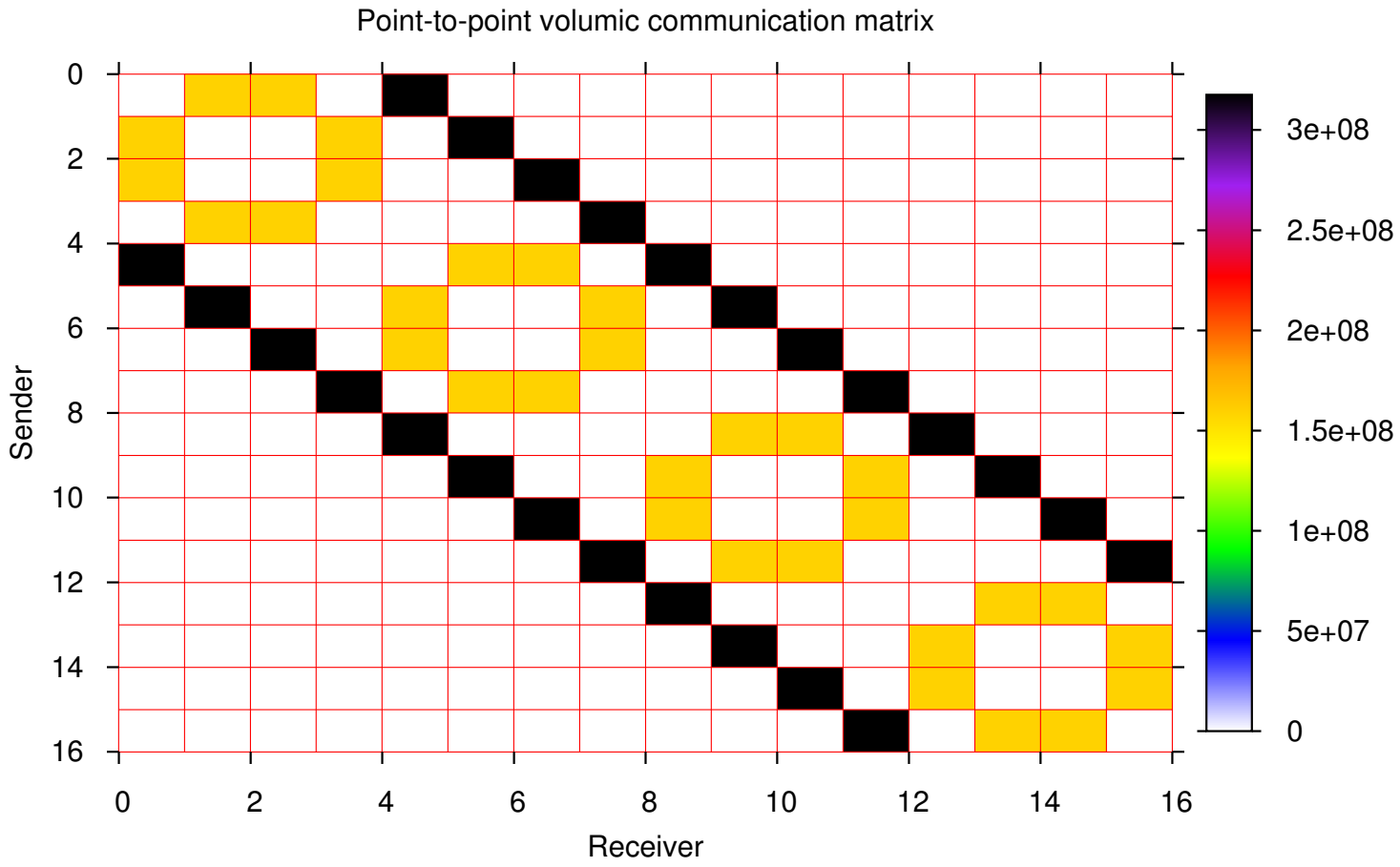
How to use it ?

- Link with -lmpianalyser
- Use environment
  - MPIANALYSER_PROFILECOMM=1
  - MPIANALYSER_GLOBAL=1
- Run your application
- Use tool: readpfc

Point to Point numeric command matrix



Point to Point Volumic communication matrix



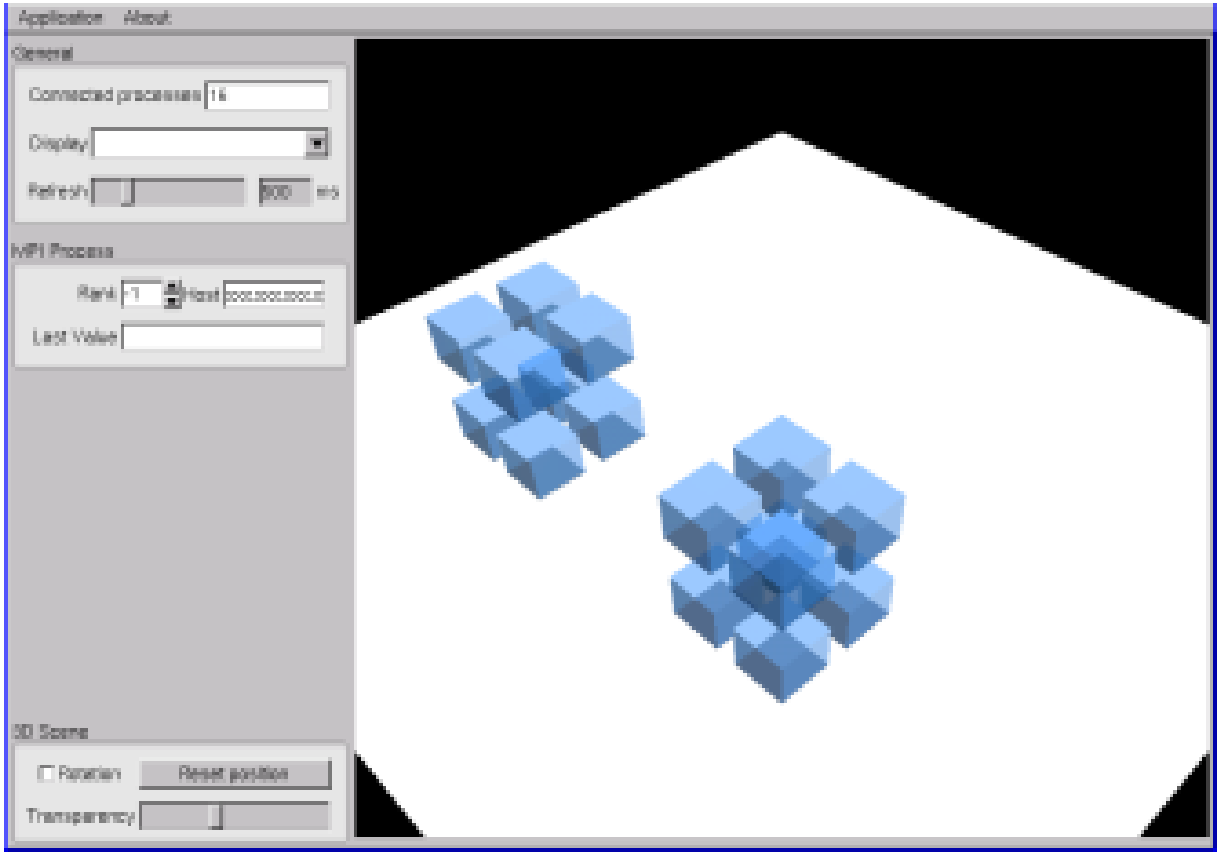
How to use it ?

On Compute node

```
export MPIANALYSER_GLOBAL=1
export MPIANALYSER_DATA_COLL=1
export MPIANALYSER_OUTPUT_SOCKET=1
export MPIANALYSER_CALL=1
export MPIANALYSER_VOLUME=1
export MPIANALYSER_TIME=1
export MPIANALYSER_VISU_SERVER_IP=xxx.xxx.xxx.xxx
export MPIANALYSER_VISU_SERVER_PORT=12345
```

On Vizual node

```
export MPIANALYSER_VISU_SERVER_PATH=/opt/mpi/mpibull2-X/lib/mpianalyser/
export MPIANALYSER_VISU_SERVER_PORT=12345
mpibull2-visualize
```


Application
About
General
Connected processes
Display
Refresh
MPI Process
Rank
Host
Last Value
3D Scene
Rotation
Reset position
Transparency

Communications: advanced Level

Intel Trace Tools : Collector ans Analyser

- Timeline Views and Parallelism Display
- Communication Statistics
- Execution Statistics
- Profiling Library
- MPI Bull support Intel Trace Tools

How to use it ?

Level 1

- Environment: source itacvars.sh or use a module file
- Link with -L$VT_LIB_DIR -lVT -ldwarf -lelf -lvtunwind -lnsl -lm -ldl -lpthread or $VT_ADD_LIBS
- Run your application
- Run traceanalyzer

Level 2 with Class

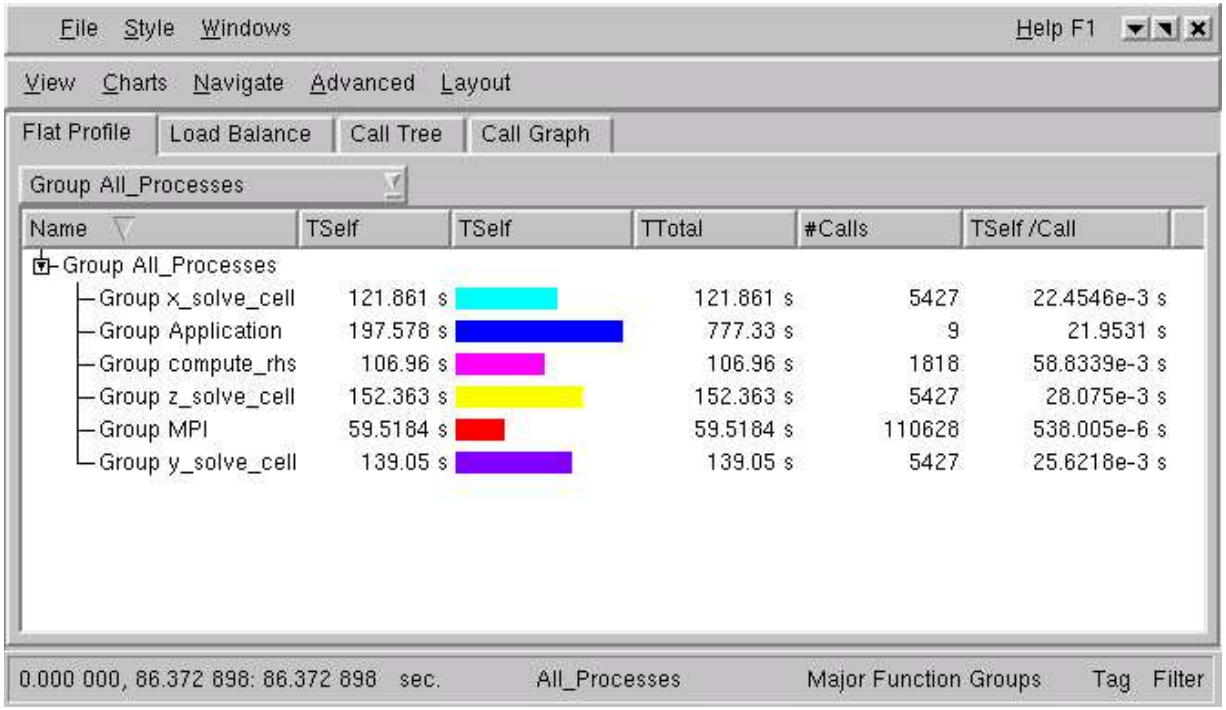
- At the beginning

```
include "VT.inc"
integer function1,VT_ierr,class1
call VTCLASSDEF("GODFINE1",class1,VT_ierr)
call VTFUNCDEF("GODFINE1",class1,function1,VT_ierr)
call VTENTER(function1,VT_NOSCL,VT_ierr)
```

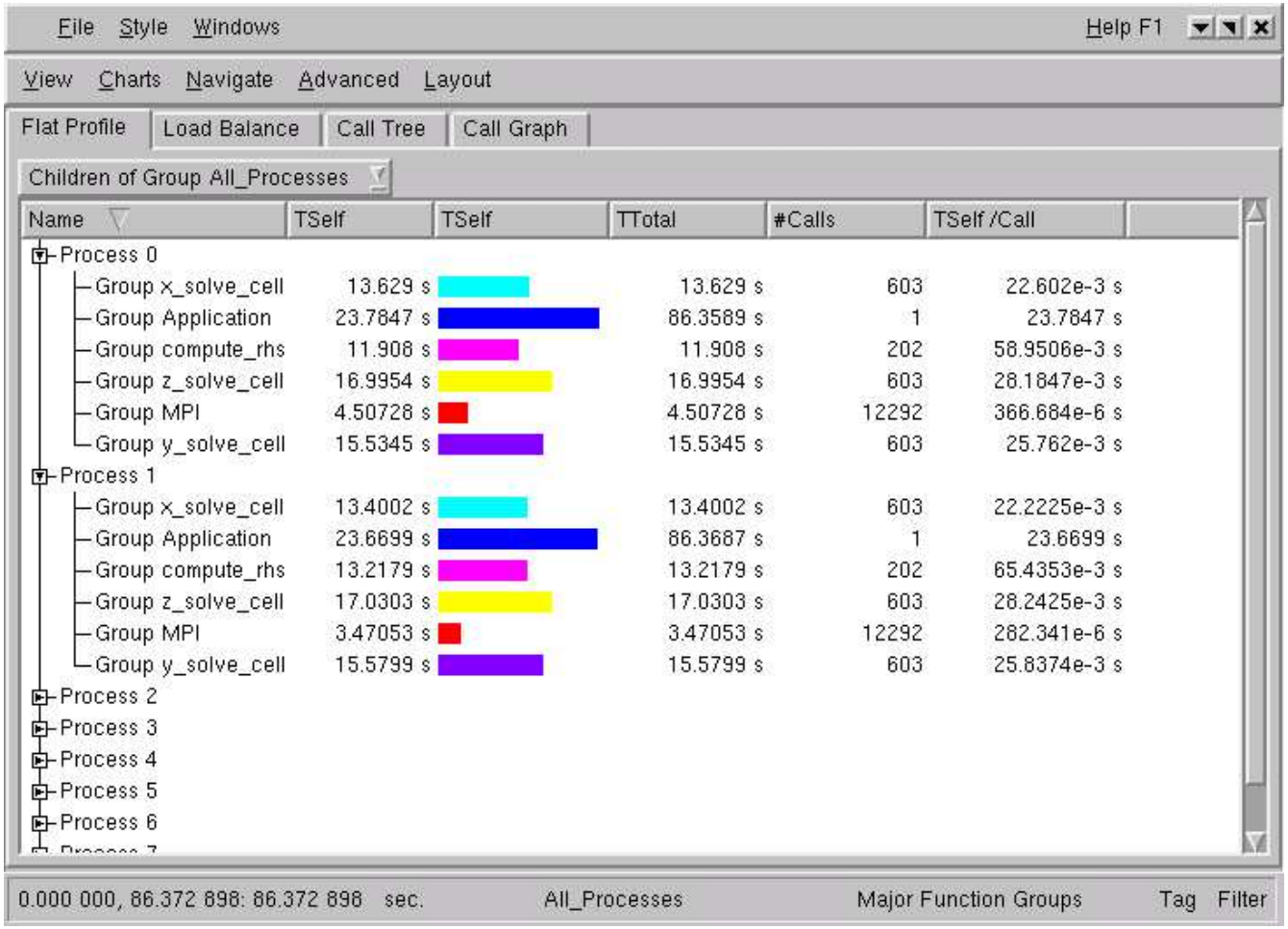
- At the end

```
call VTLEAVE(function1,VT_ierr)
```

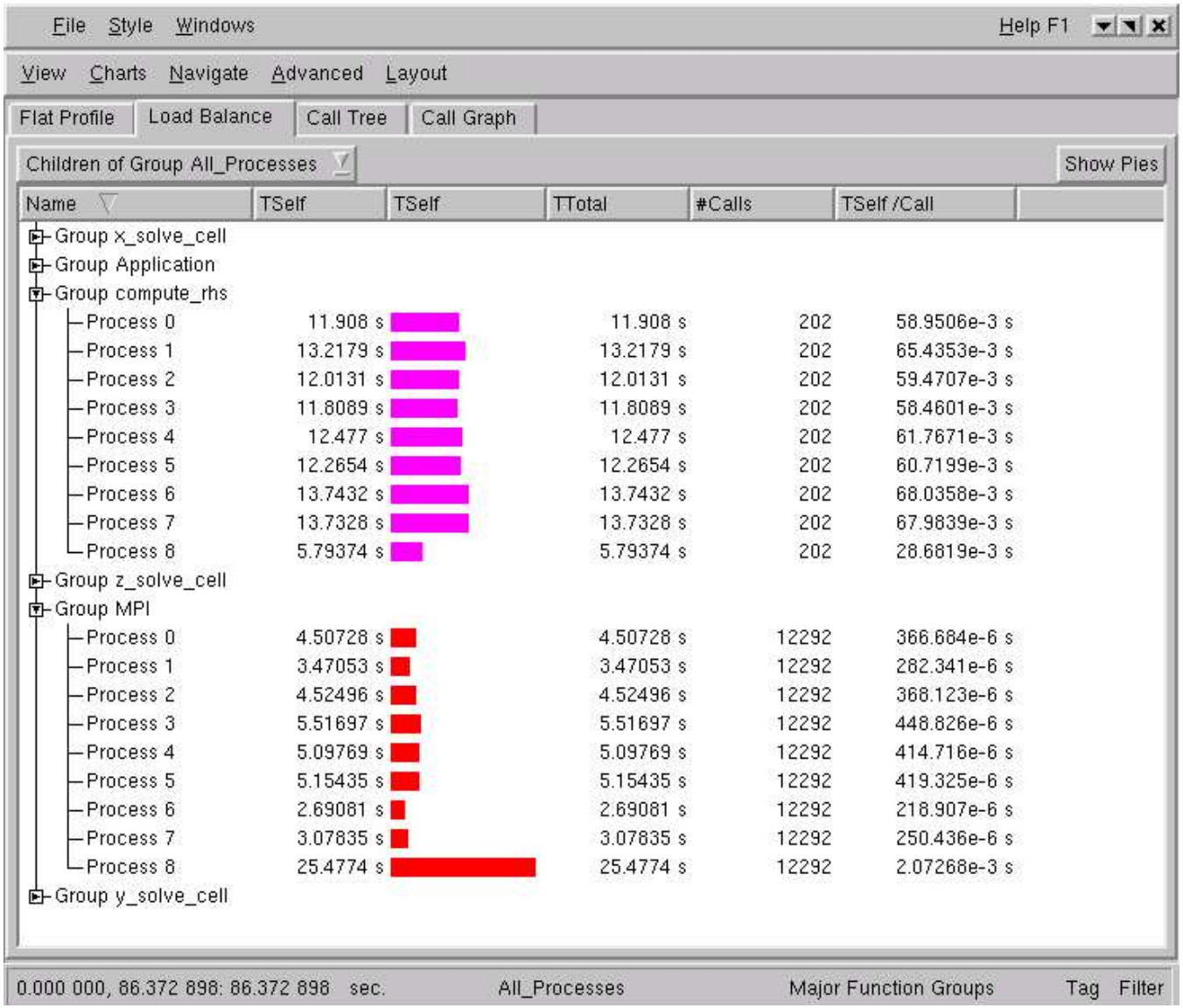
Example BT from NAS



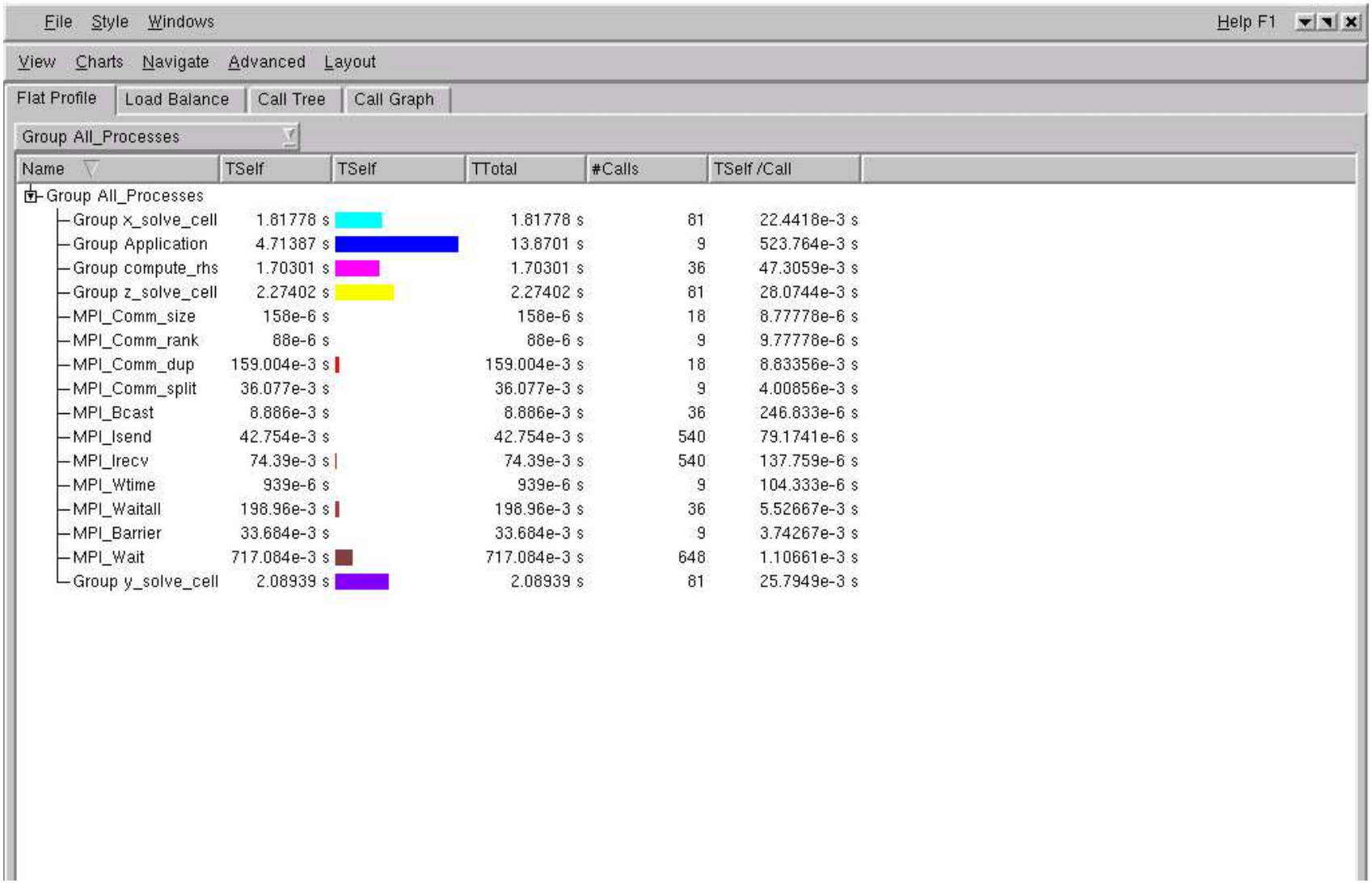
Example



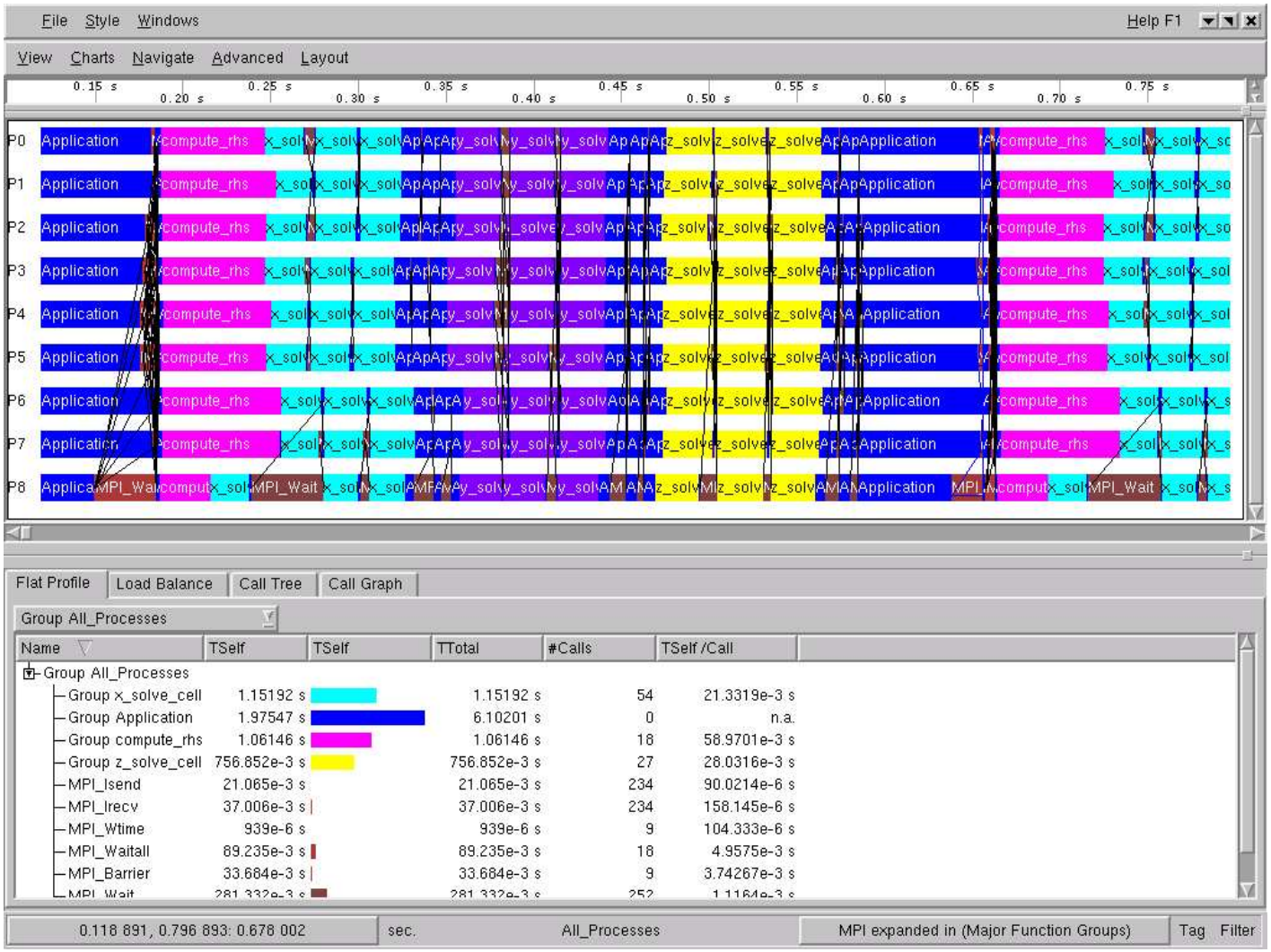
Example: Load balance



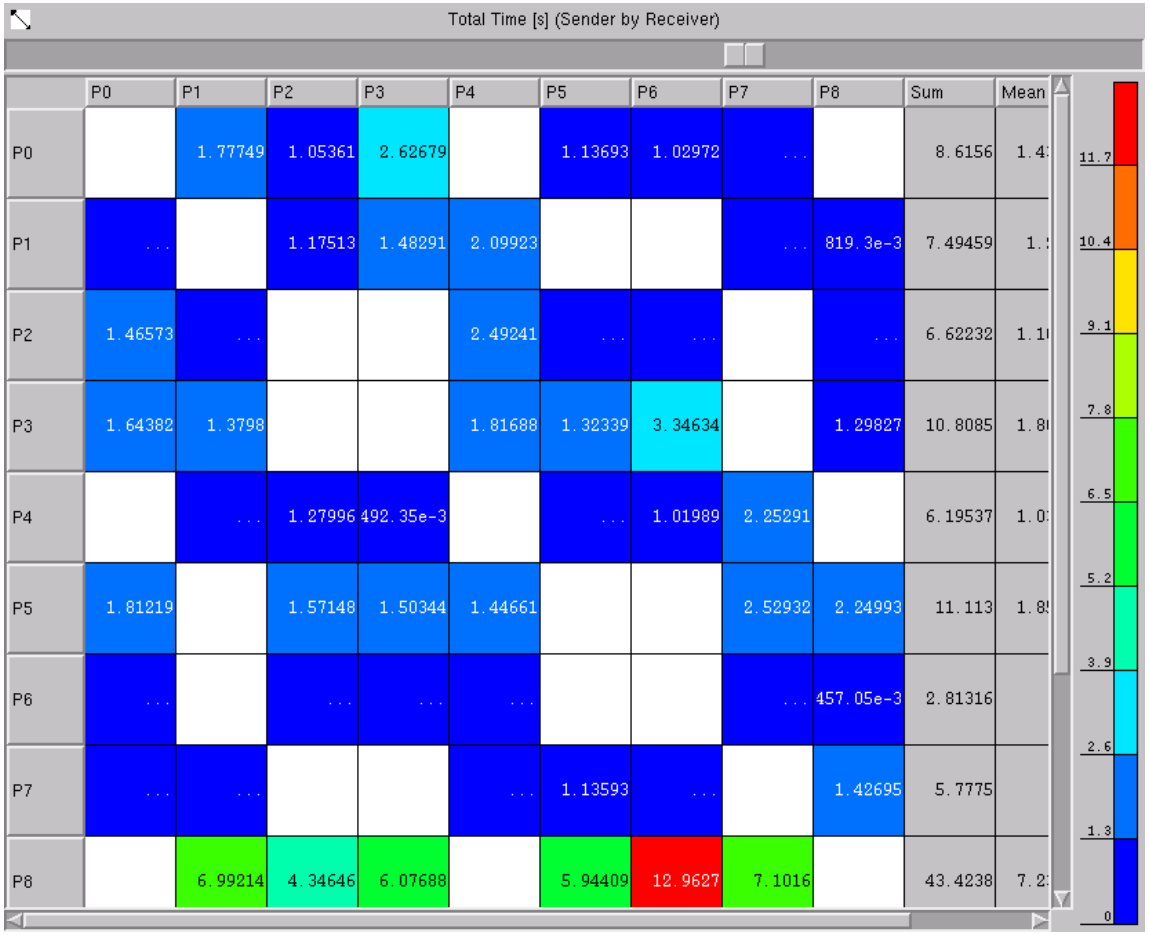
Example: Ungroup MPI



Example



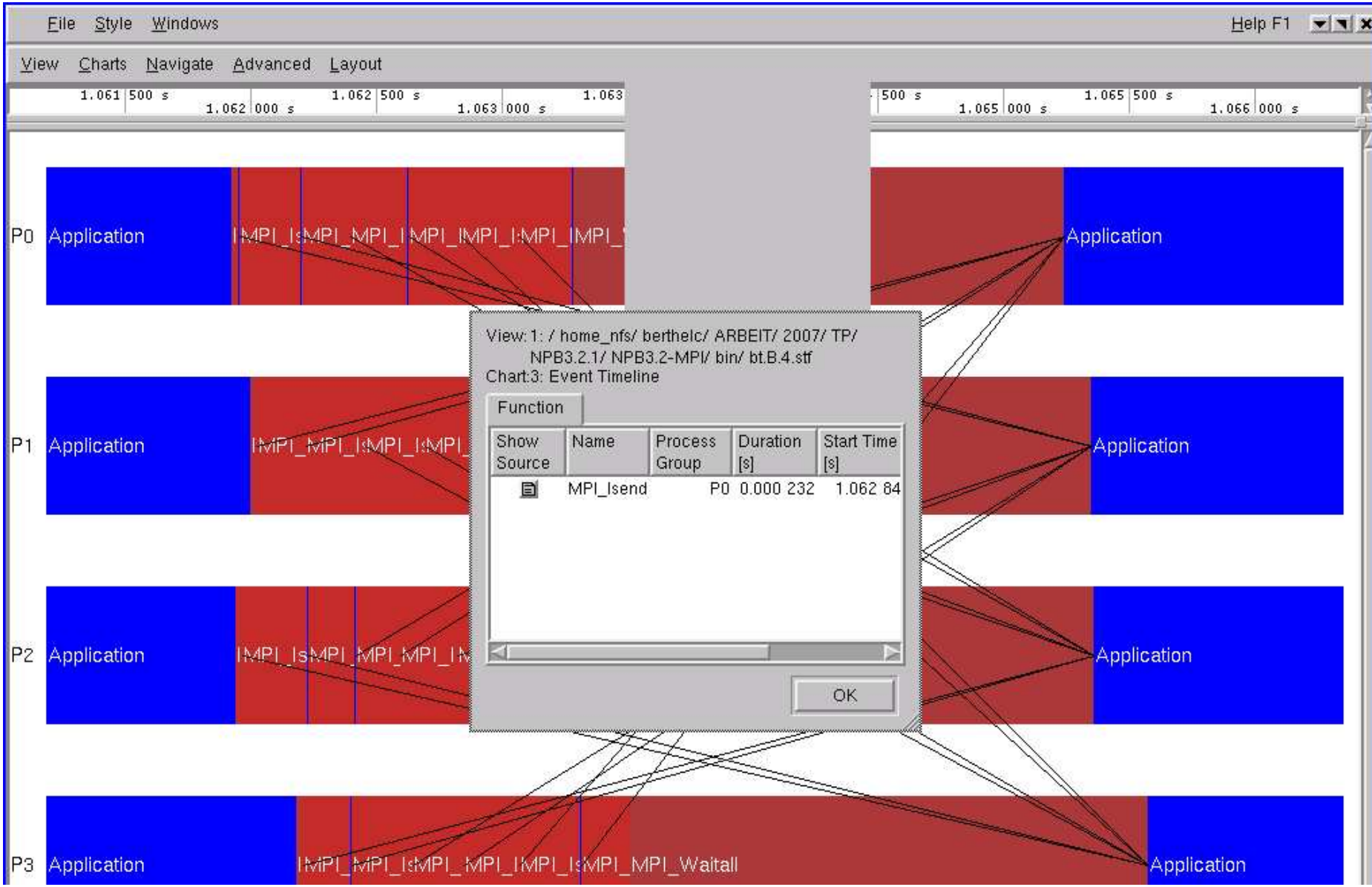
Example

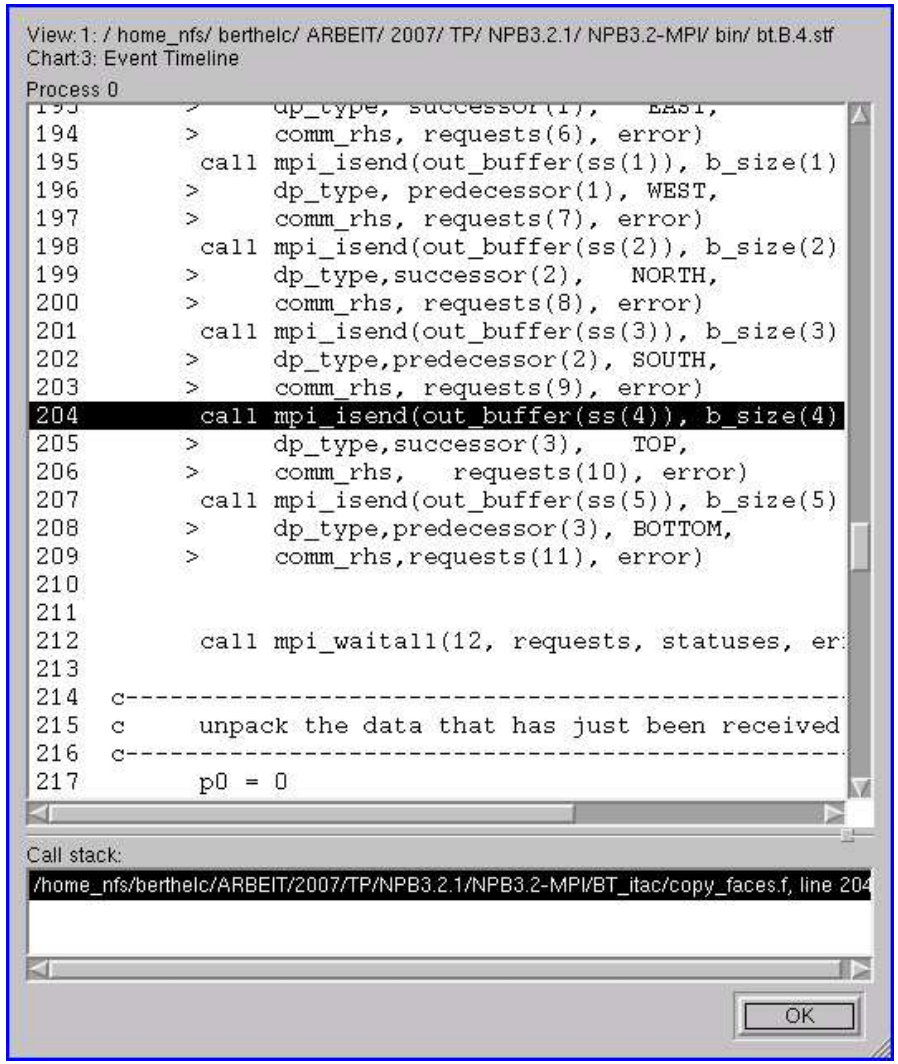


Source and Profiling

Compile and Run

- You need compile and link with -g
- You need export VT_PCTRACE=5


File Style Windows
Help F1
View Charts Navigate Advanced Layout
1.061 500 s
1.062 000 s
1.062 500 s
1.063 000 s
1.063
1.065 500 s
1.066 000 s
P0
P1
P2
P3
Application
MPI_Waitall
View: 1: / home_nfs/ berthelc/ ARBEIT/ 2007/ TP/ NPB3.2.1/ NPB3.2-MPI/ bin/ bt.B.4.stf
Chart:3: Event Timeline
Function
Show Source
Name
Process Group
Duration [s]
Start Time [s]
MPI_Isend P0 0.000 232 1.062 84
OK


View: 1: / home_nfs/ berthelc/ ARBEIT/ 2007/ TP/ NPB3.2.1/ NPB3.2-MPI/ bin/ bt.B.4.stf
Chart:3: Event Timeline
Process 0
194 > comm_rhs, requests(6), error)
195 call mpi_isend(out_buffer(ss(1)), b_size(1)
196 > dp_type, predecessor(1), WEST,
197 > comm_rhs, requests(7), error)
198 call mpi_isend(out_buffer(ss(2)), b_size(2)
199 > dp_type,successor(2), NORTH,
200 > comm_rhs, requests(8), error)
201 call mpi_isend(out_buffer(ss(3)), b_size(3)
202 > dp_type,predecessor(2), SOUTH,
203 > comm_rhs, requests(9), error)
204 call mpi_isend(out_buffer(ss(4)), b_size(4)
205 > dp_type,successor(3), TOP,
206 > comm_rhs, requests(10), error)
207 call mpi_isend(out_buffer(ss(5)), b_size(5)
208 > dp_type,predecessor(3), BOTTOM,
209 > comm_rhs,requests(11), error)
210
211
212 call mpi_waitall(12, requests, statuses, er
213
214 c-----
215 c unpack the data that has just been received
216 c-----
217 p0 = 0
Call stack:
/home_nfs/berthelc/ARBEIT/2007/TP/NPB3.2.1/NPB3.2-MPI/BT_itac/copy_faces.f, line 204
OK

Questions ?

For Further Reading

HPC BAS4 User's Guide
REFERENCE 86 A2 29ER 03.

HPC BAS4 Application Tuning Guide
REFERENCE 86 A2 19ER 00.

Black Belt Itanium 2 Processor Performance: Use of Constrained Event Collection

Intel Itanium 2 Processor Reference Manual for Software Development and Optimization
Order Number: 251110-003

Architect of an Open World™

- (c) Copyright Bull. All rights reserved
  - ✔ Users Restricted Rights - Use, duplication or disclosure restricted.
  - ✔ Any copy of these documents should keep all copyright, logos and other proprietary notices contained herein.
  - ✔ This publication may include technical inaccuracies or typographical errors.
  - ✔ This publication is provided "AS IS" without any warranty either expressed or implied including but not limited to the implied warranties of merchantabilities or fitness of the described product.
  - ✔ Course Material Licensing Terms : No sublicensing rights.
  - ✔ For other licensing needs, please contact Bull