More on Application profiling and optimization

Ilias Katsardis
ikatsardis@cray.com

Agenda

- A Tour of the Apprentice2 GUI
- Optimizations for MPI – Rank Reordering

A tour of the Apprentice2 GUI

The Three Stages of Profiling with perftools and CrayPat

1. **Instrumentation**
   - Build executable of an instrumented version of your application
2. **Running your application and Data Collection**
   - Run the instrumented version of your application
   - Transparent collection via CrayPat's run-time library

3. **Analysis: Sampling / Tracing**
   - Interpret and visualize data using post-mortem tools:
     1. `pat_report`: a command line tool for generating text reports
     2. **Cray Apprentice2: a graphical performance analysis tool**
     3. **Reveal**: graphical performance analysis and code restructuring tool

Profile Visualization with Cray Apprentice2

Cray Apprentice2

- **Features:**
  - Call graph profile
  - Communication statistics
  - Time-line view
    - Communication
    - I/O
  - Activity view
  - Pair-wise communication statistics
  - Text reports
  - Source code mapping
- **Helps identify:**
  - Load imbalance
  - Excessive communication
  - Network contention
  - Excessive serialization
  - I/O Problems

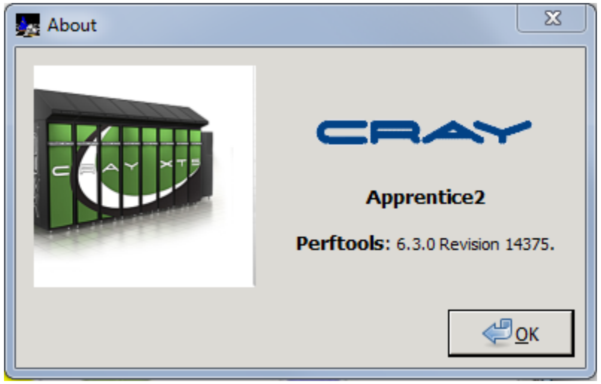
To use Cray Apprentice²

- **You can run app2 on the login nodes:**
  - You need an X session
    - `ssh -X <system name>`
    - and software to catch X windows on your local machine
  - You need `app2` in your path
    - `module load perftools-base`
  - The *.ap2 file contains the information (produced by pat_report)
    - `app2 data_file_name.ap2`
    - or you can load the ap2 file from the GUI

- **There is also a client version of app2**
  - You can run this on your local machine
  - Contact your site administrator for details on how to install this
  - Then just need to copy the *.ap2 file to this machine

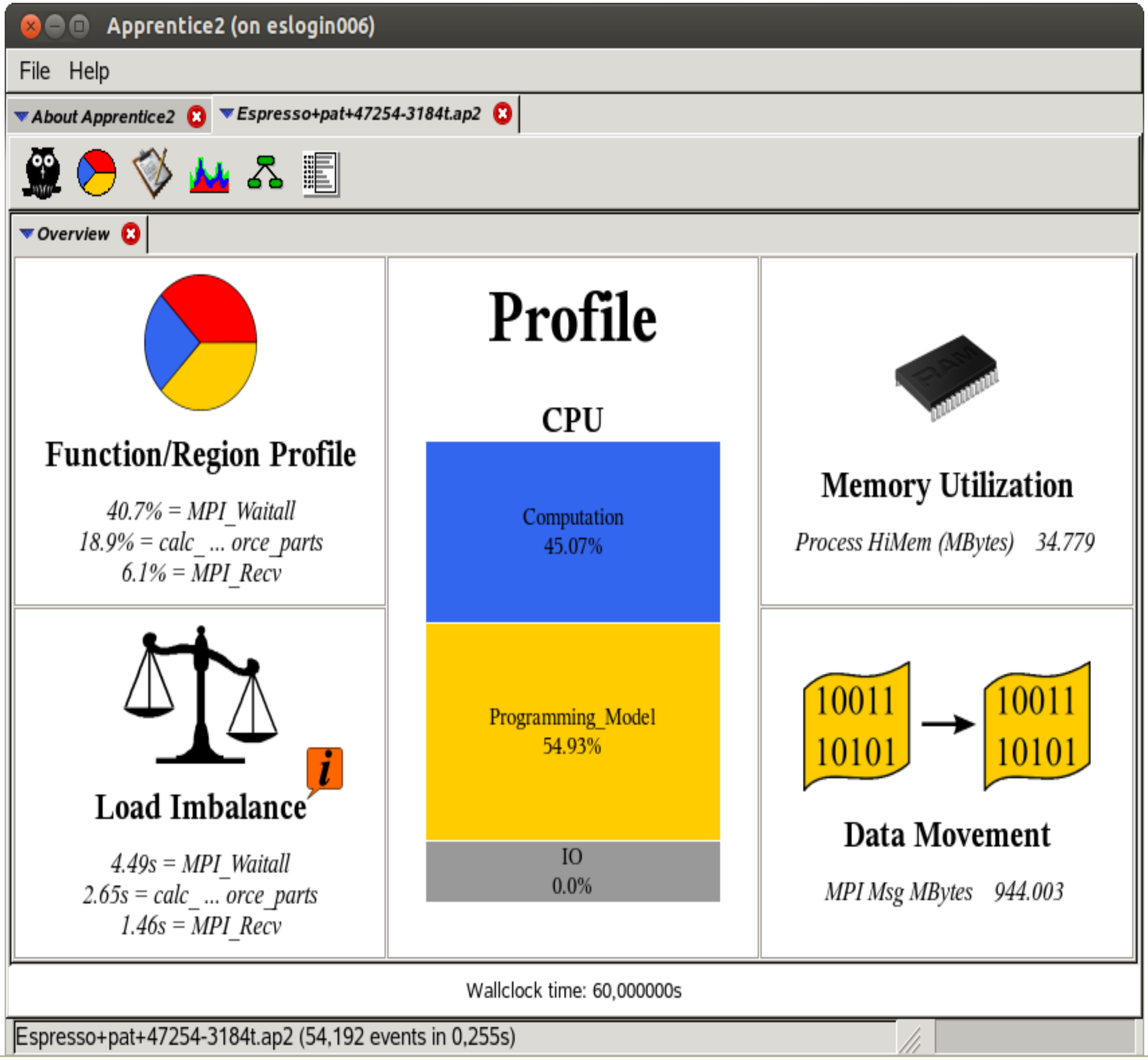
Installing Apprentice2 on Laptop

From a login node

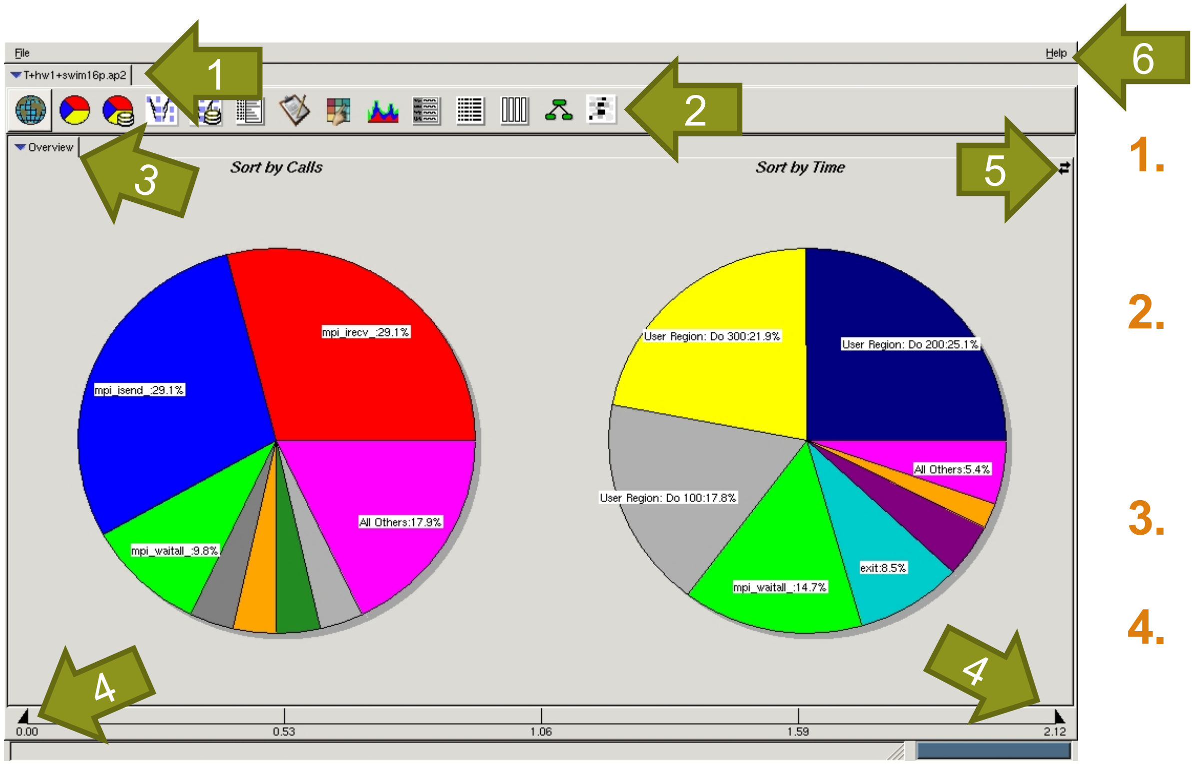


- `> module load perftools-base`
- **Go to:**
  - $CRAYPAT_ROOT/share/desktop_installers/
- **Download .dmg or .exe installer to laptop**
- **Double click on installer and follow directions to install**

Cray Apprentice²



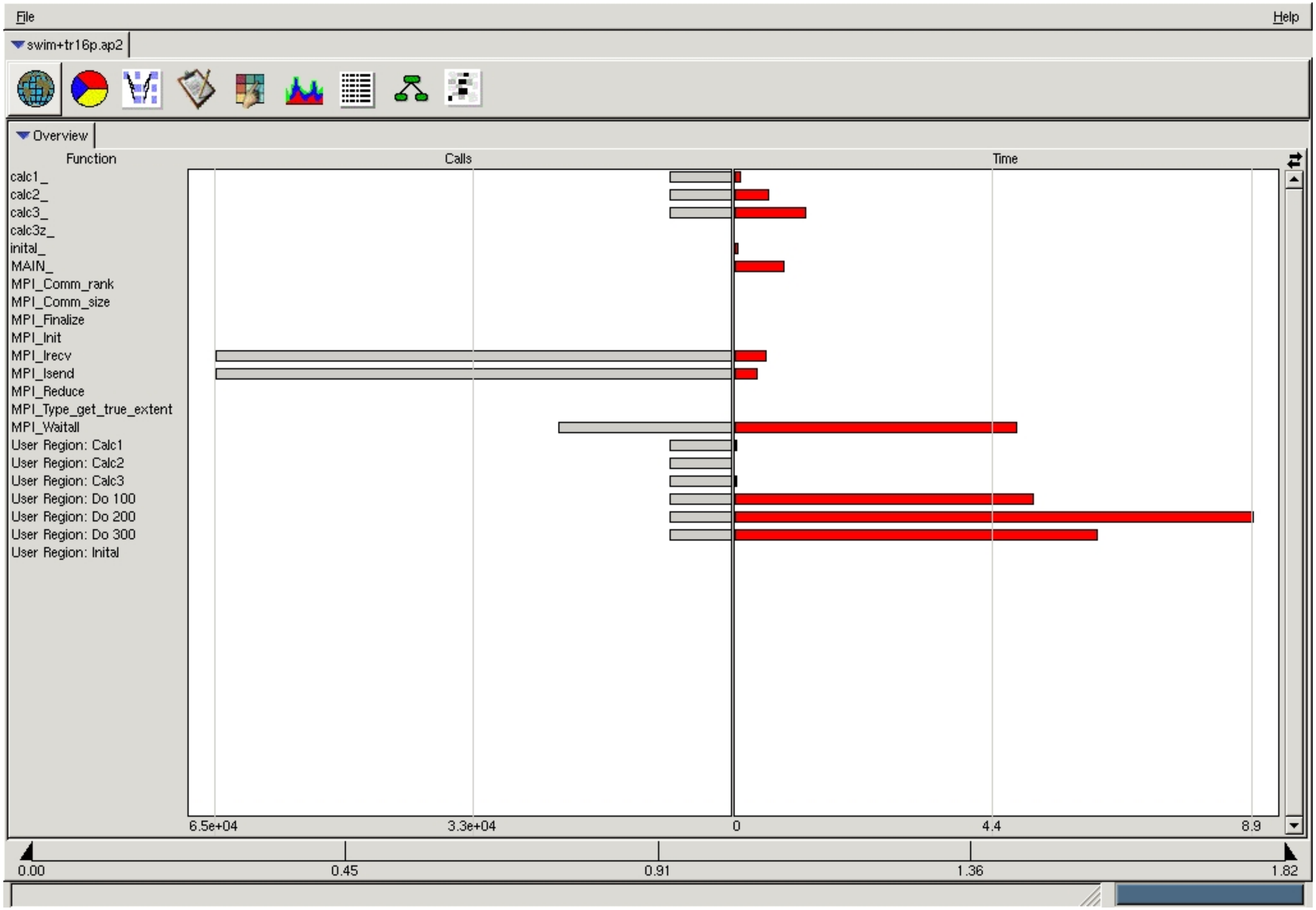
Statistics Overview: Pie Chart



1. Data tab: shows the name of the data file currently displayed
2. Report toolbar: show the reports that can be displayed for the data currently selected
3. Report tabs: show the reports
4. On many reports, the total duration of the experiment is shown as a graduated bar at the bottom of the window
5. Change view from pie chart to bar graph
6. Help menu

Note that report toolbar ONLY what you have decide to collected with pat_build

Statistics Overview: Bar Graph



Function Profile View

File | Help

swim+iompi+1566td.ap2 | T+hw1+swp+io+mpi+48p.ap2

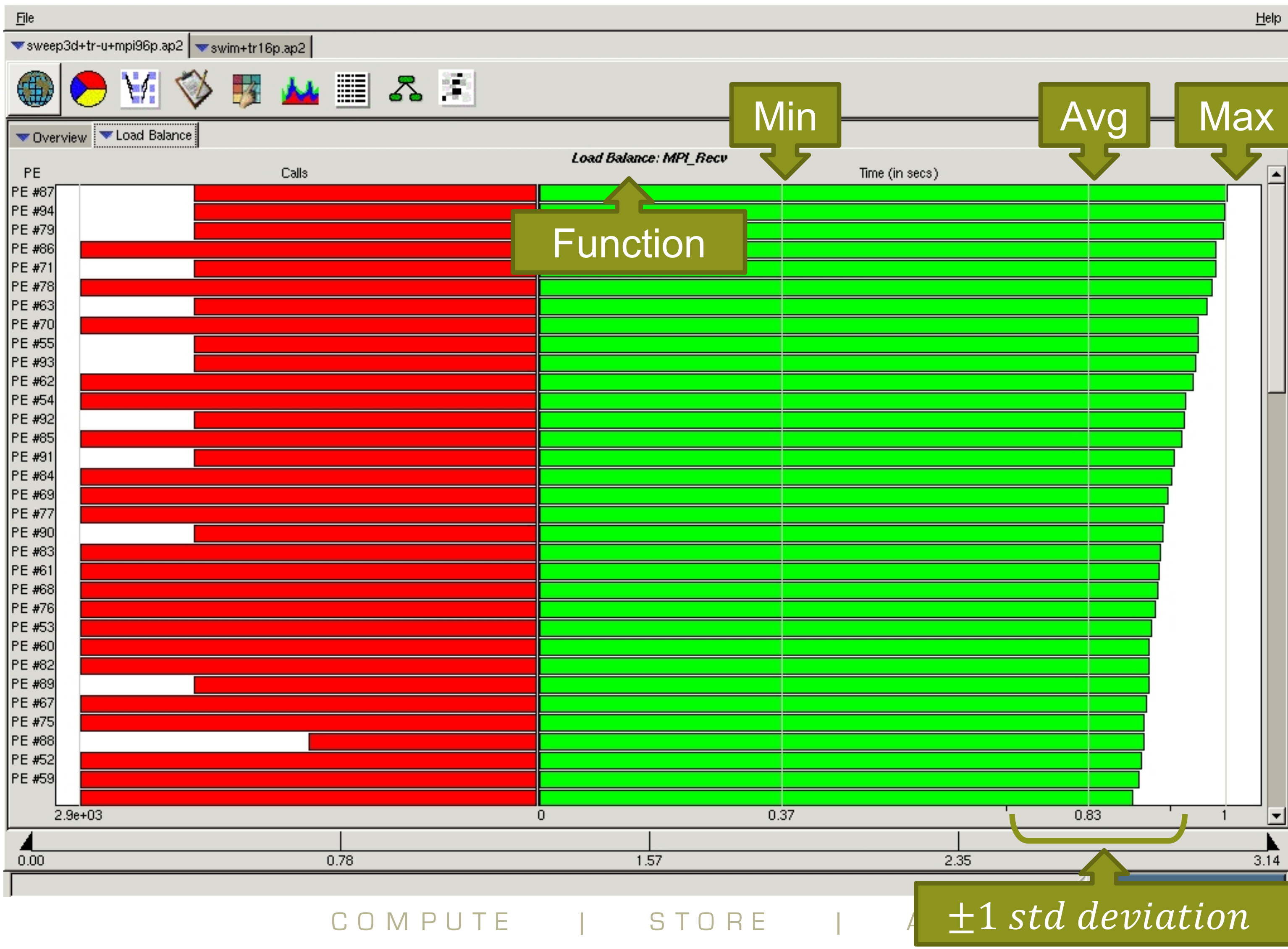
Overview | Function

Time	Percent	Hits	Callsites	Imbalance %	Potential Savings	Function	Line	File
124.175511	63.29	576	1	5.63	0.15	sweep_	116	/us/nid00036/ldr/Apps/sweep3d/sweep.f
40.211774	20.50	118080	1	23.40	0.25	mpi_recv_	0	==NA==
16.319527	8.32	48	1	48.26	0.30	exit	35	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/stdlib/exit.c
6.173236	3.15	1536	3	50.00	0.12	mpi_allreduce_	0	==NA==
2.760376	1.41	118080	1	17.58	0.01	mpi_send_	0	==NA==
2.250029	1.15	576	1	2.62	0.00	source_	18	/us/nid00036/ldr/Apps/sweep3d/source.f
1.984620	1.01	144	1	2.59	0.00	mpi_barrier_	0	==NA==
0.867359	0.44	192	2	2.47	0.00	mpi_bcast_	0	==NA==
0.416231	0.21	576	1	2.60	0.00	flux_err_	17	/us/nid00036/ldr/Apps/sweep3d/flux_err.f
0.382130	0.19	118080	2	10.98	0.00	snd_real_	135	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.237772	0.12	309	1	95.76	0.07	fwrite	36	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/libio/iofwrite.c
0.185067	0.09	118080	2	17.30	0.00	rcv_real_	164	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.067832	0.03	48	1	4.56	0.00	initialize_	42	/us/nid00036/ldr/Apps/sweep3d/initialize.f
0.059407	0.03	48	1	4.99	0.00	initxs_	77	/us/nid00036/ldr/Apps/sweep3d/initialize.f
0.041371	0.02	48	1	23.84	0.00	inner_	72	/us/nid00036/ldr/Apps/sweep3d/inner.f
0.023948	0.01	8	1		0.00	fputc	35	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/libio/fputc.c
0.016902	0.01	68	1		0.00	getc	36	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/libio/getc.c
0.008104	0.00	4992	2	28.14	0.00	octant_	17	/us/nid00036/ldr/Apps/sweep3d/octant.f
0.002457	0.00	576	1	18.79	0.00	global_real_max_	321	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.002083	0.00	48	1	69.55	0.00	MAIN_	72	/us/nid00036/ldr/Apps/sweep3d/driver.f
0.001588	0.00	576	1	39.10	0.00	global_int_sum_	373	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.001393	0.00	48	1	10.23	0.00	inner_auto_	69	/us/nid00036/ldr/Apps/sweep3d/inner_auto.f
0.000982	0.00	48	1	97.74	0.00	task_init_	24	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.000739	0.00	384	2	27.87	0.00	global_real_sum_	347	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.000662	0.00	2	1		0.00	fopen	106	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/libio/iofopen.c
0.000499	0.00	48	1	7.54	0.00	initsnc_	175	/us/nid00036/ldr/Apps/sweep3d/initialize.f

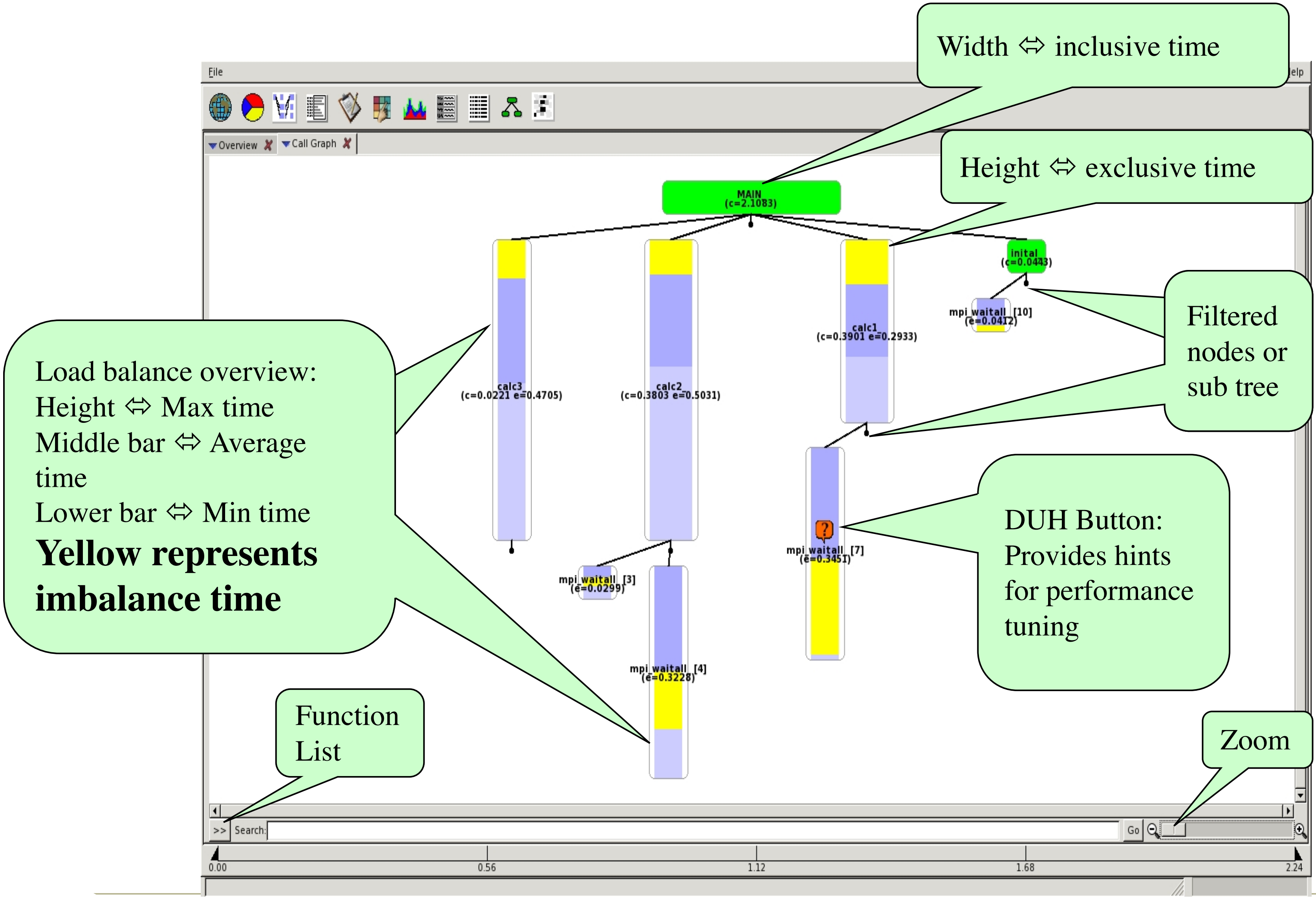
0.00 1.15 2.30 3.45 4.61

Load Balance View (Aggregated from Overview)

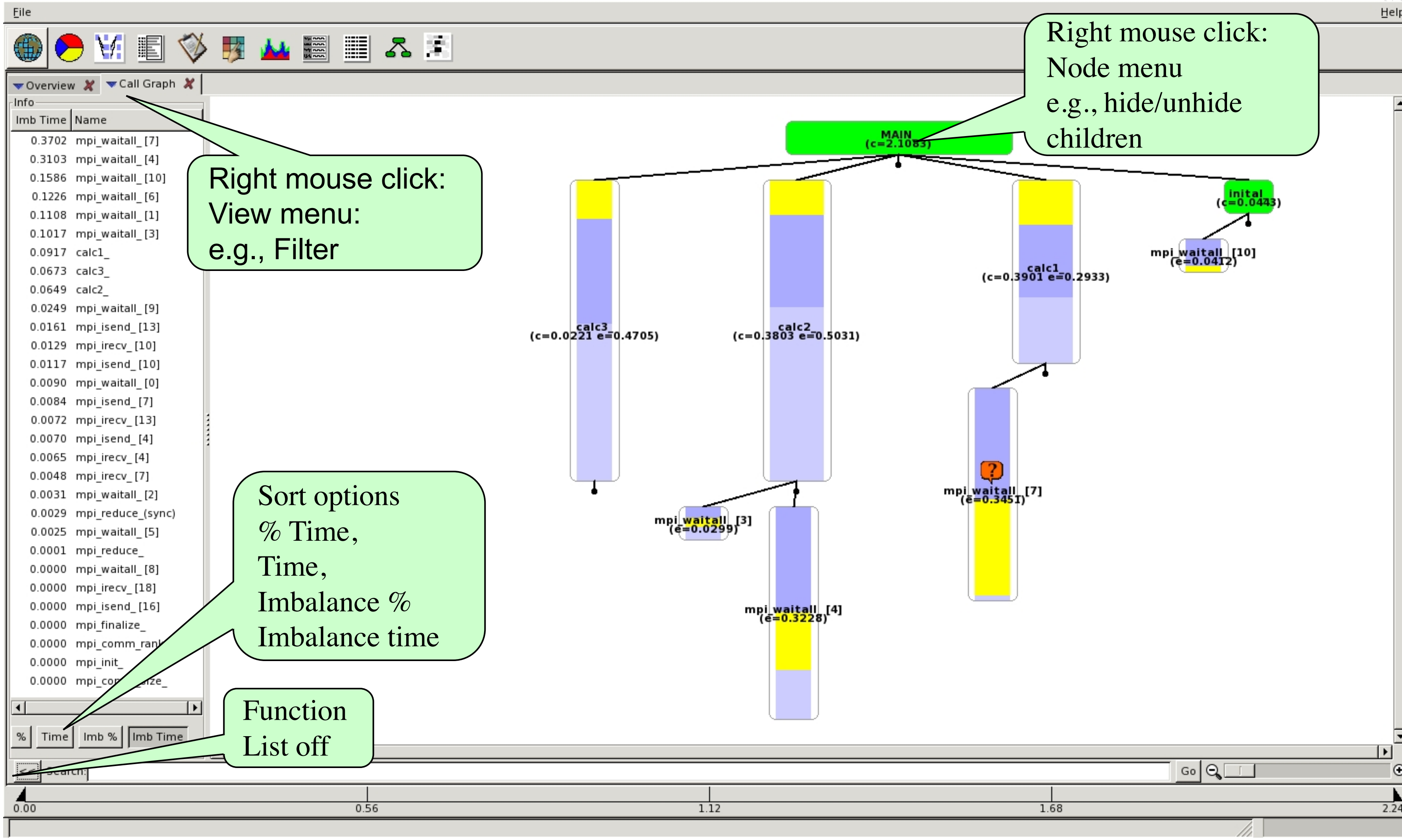
By clicking on a give function, we can show the breakdown per each PE



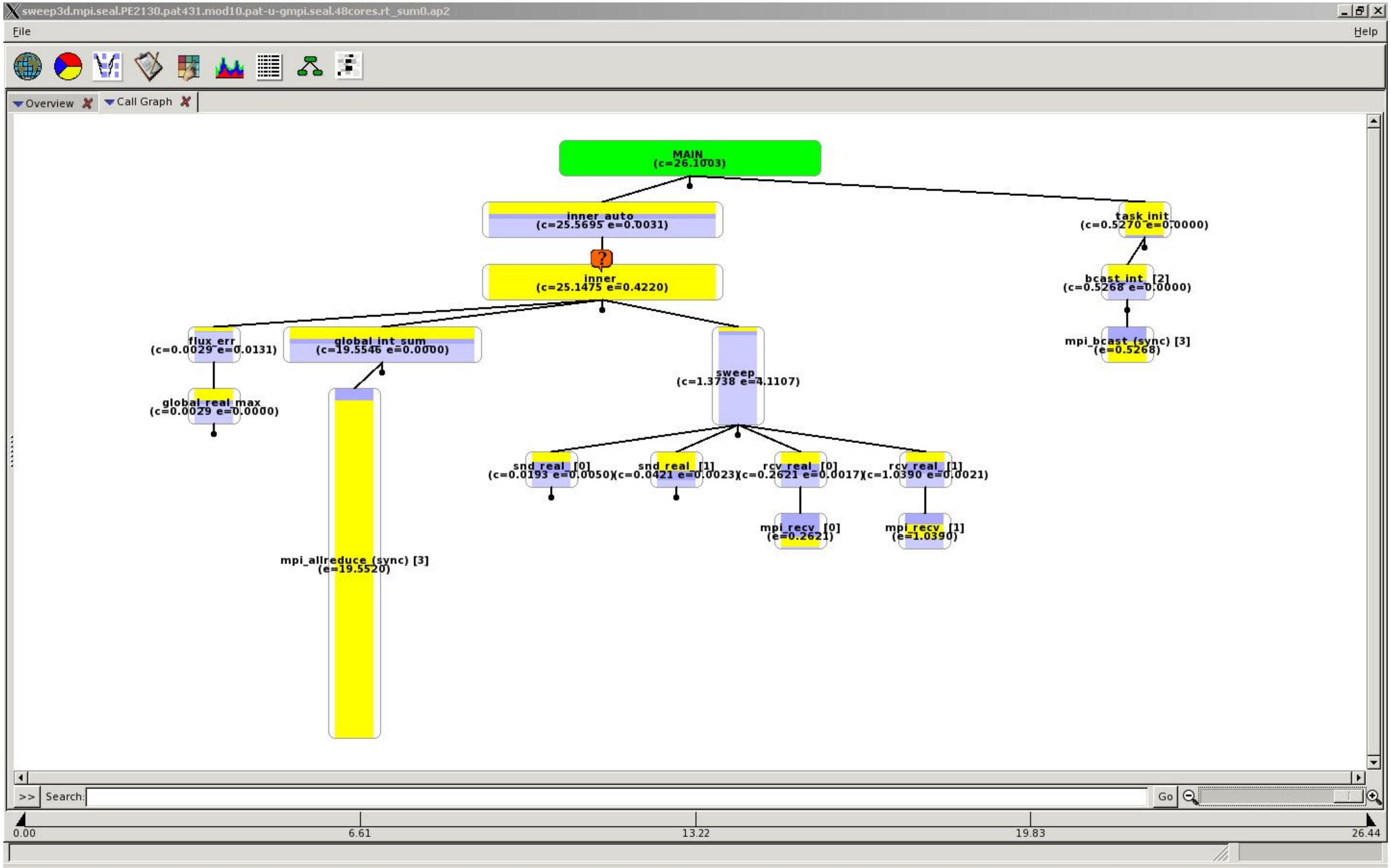
Call Tree View



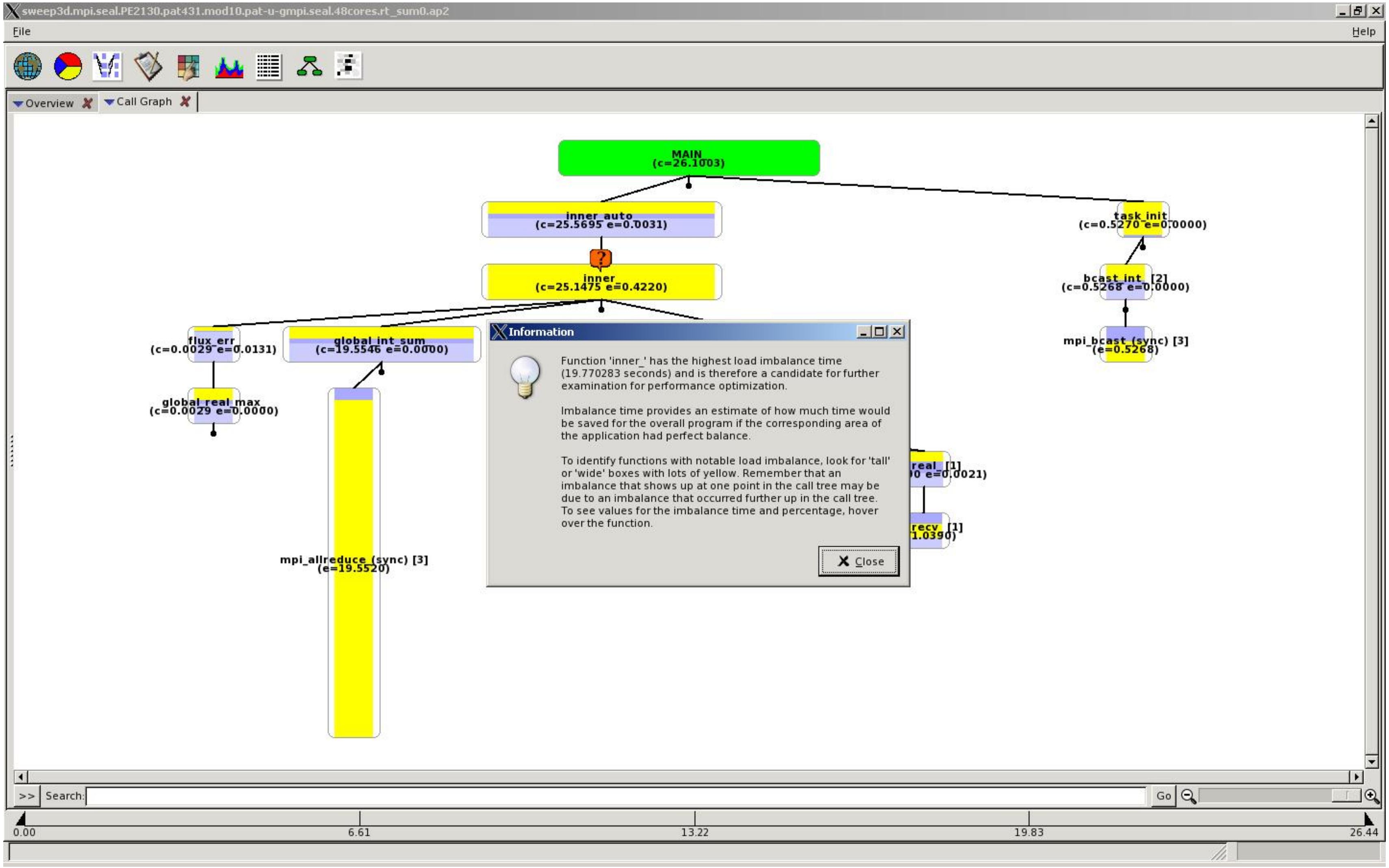
Call Tree View – Function List



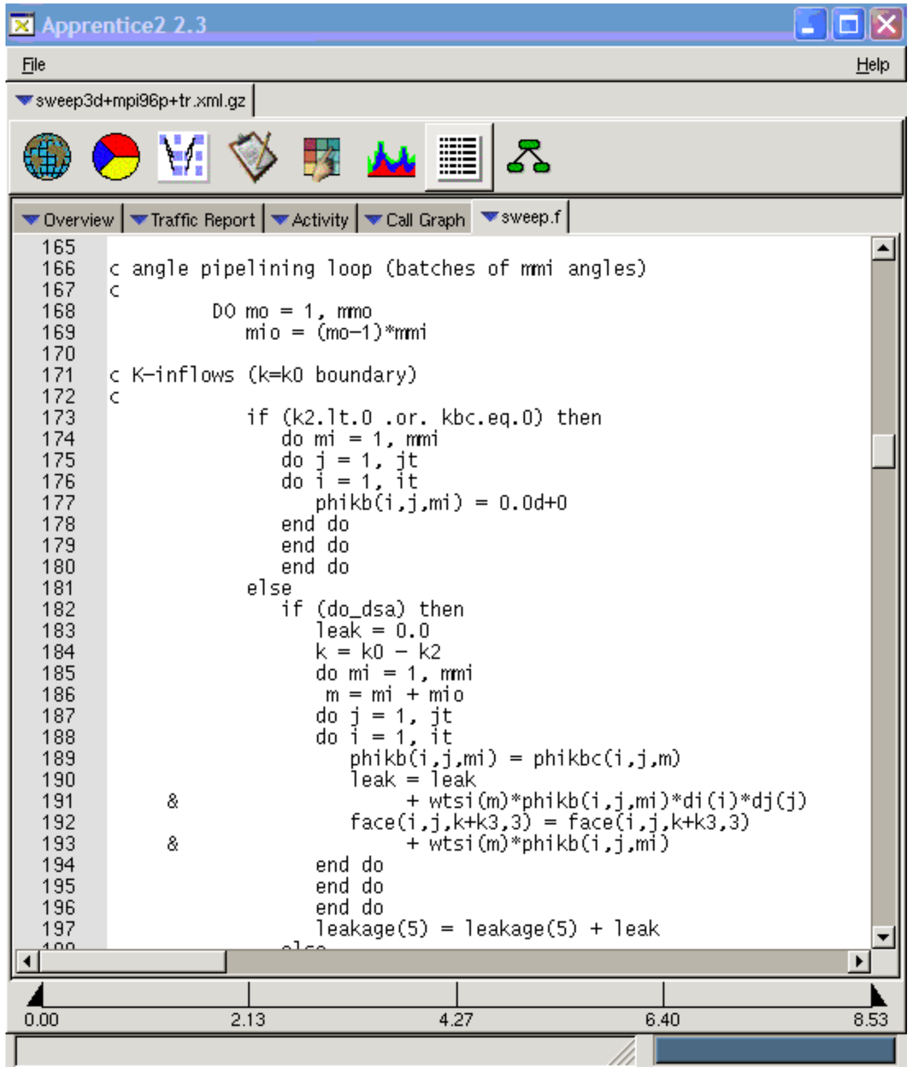
Call Tree Visualization



Discrete Unit of Help (DUH Button)



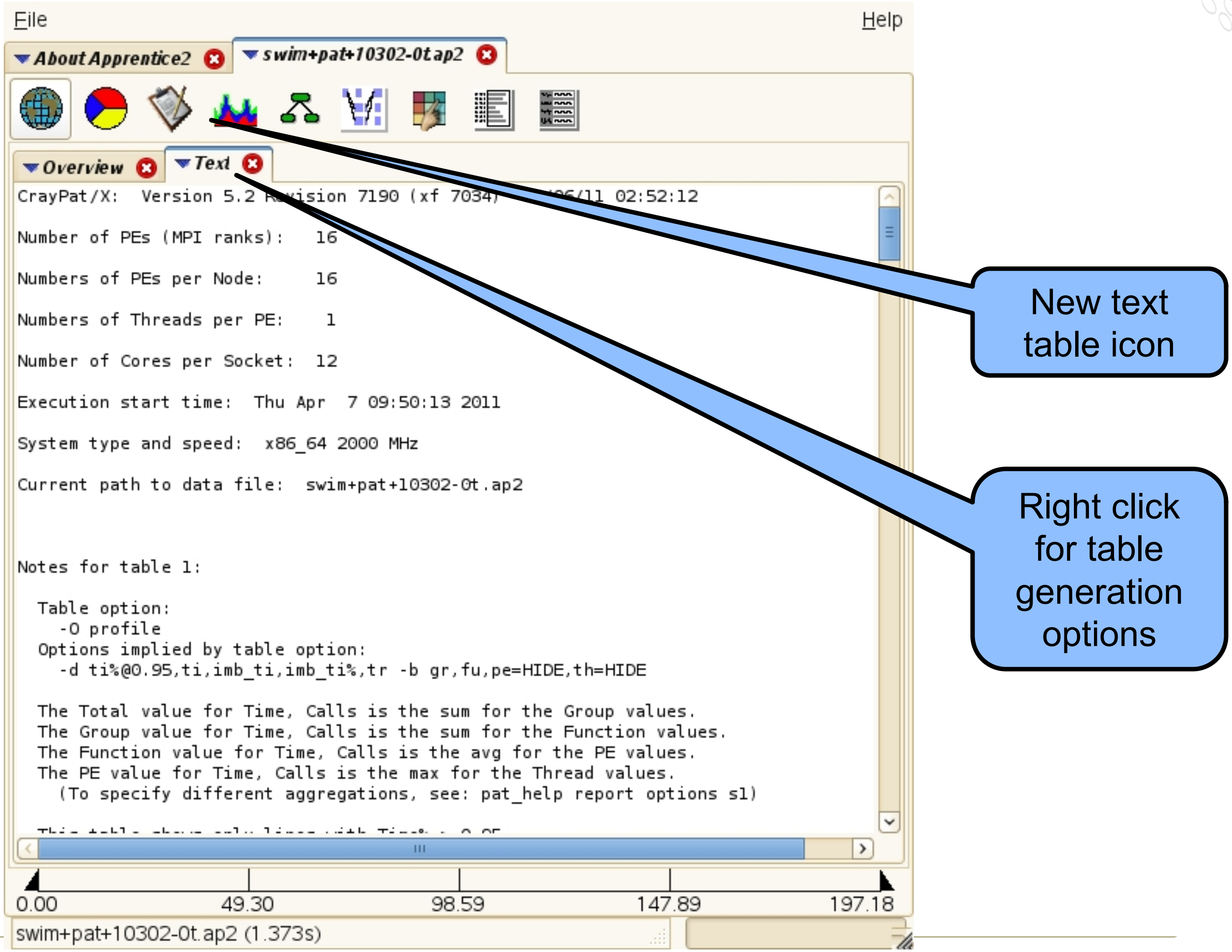
Source Mapping from Call Graph view



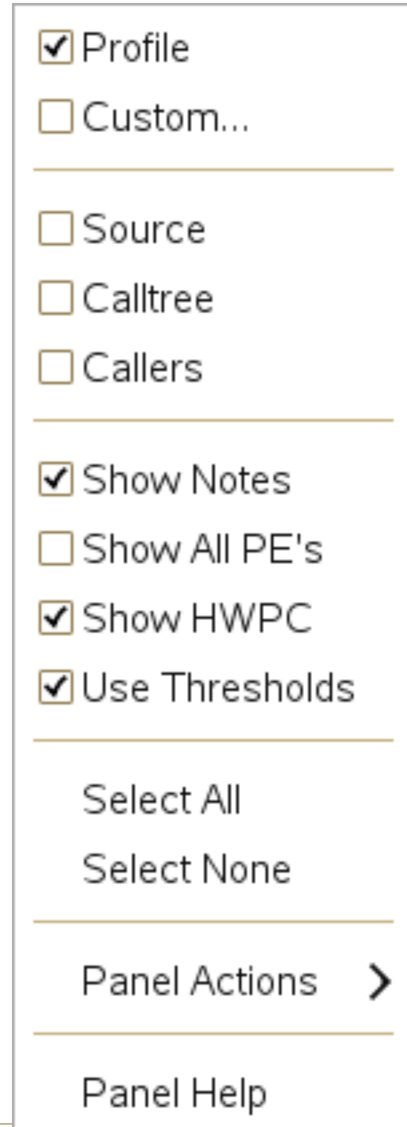
pat_report Tables in Cray Apprentice2

- **Complementary performance data available in one place**
- **Most reports easily accessible**
  - using drop-down menu for easy navigation
- **Can easily generate new views of performance data**
- **Provides mechanism for more in depth explanation of data presented**

Example of pat_report Tables in Cray Apprentice2



Generating New pat_report Tables



Reduce ap2 file information

- **Sometimes the amount of data in ap2 file can be large**
  - Very long-running applications
  - Applications running on a large number of PEs

- **The app2 command supports two options to help**
  - `--limit` and `--limit_per_pe`
  - Restrict the amount of data being read in from the ap2 file
  - use K, M, and G abbreviations for kilo, mega, and giga
- **`--limit` sets a global limit on data size.**
- **`--limit_per_pe` sets limit per PE**
  - `--limit_per_pe` generally preferable (not always, but generally)
    - preserves full parallism in analysis

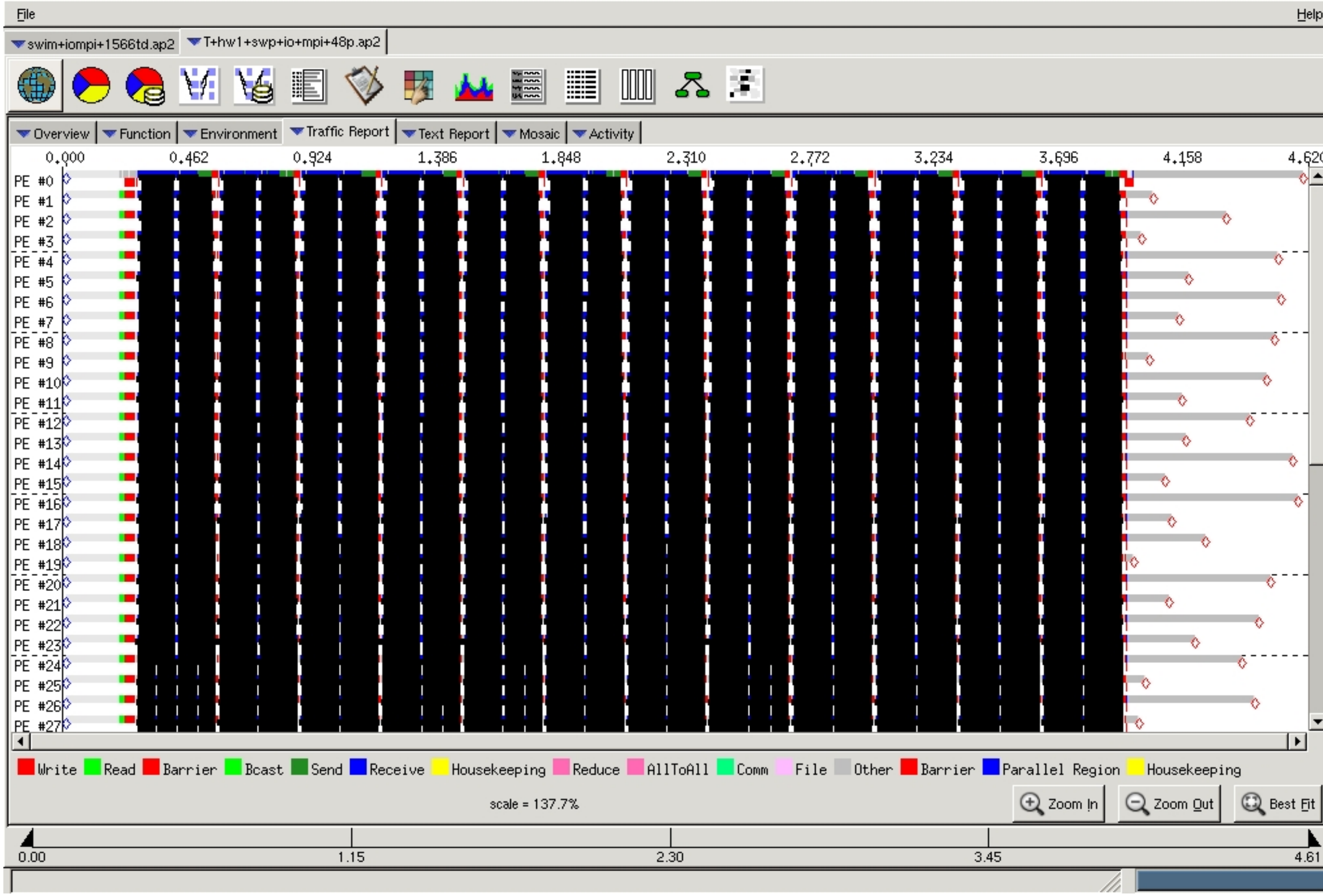
- **Example: first 3M data items**
  - `app2 --limit 3M data_file_name.ap2 &`

Timeline views with Cray Apprentice^2

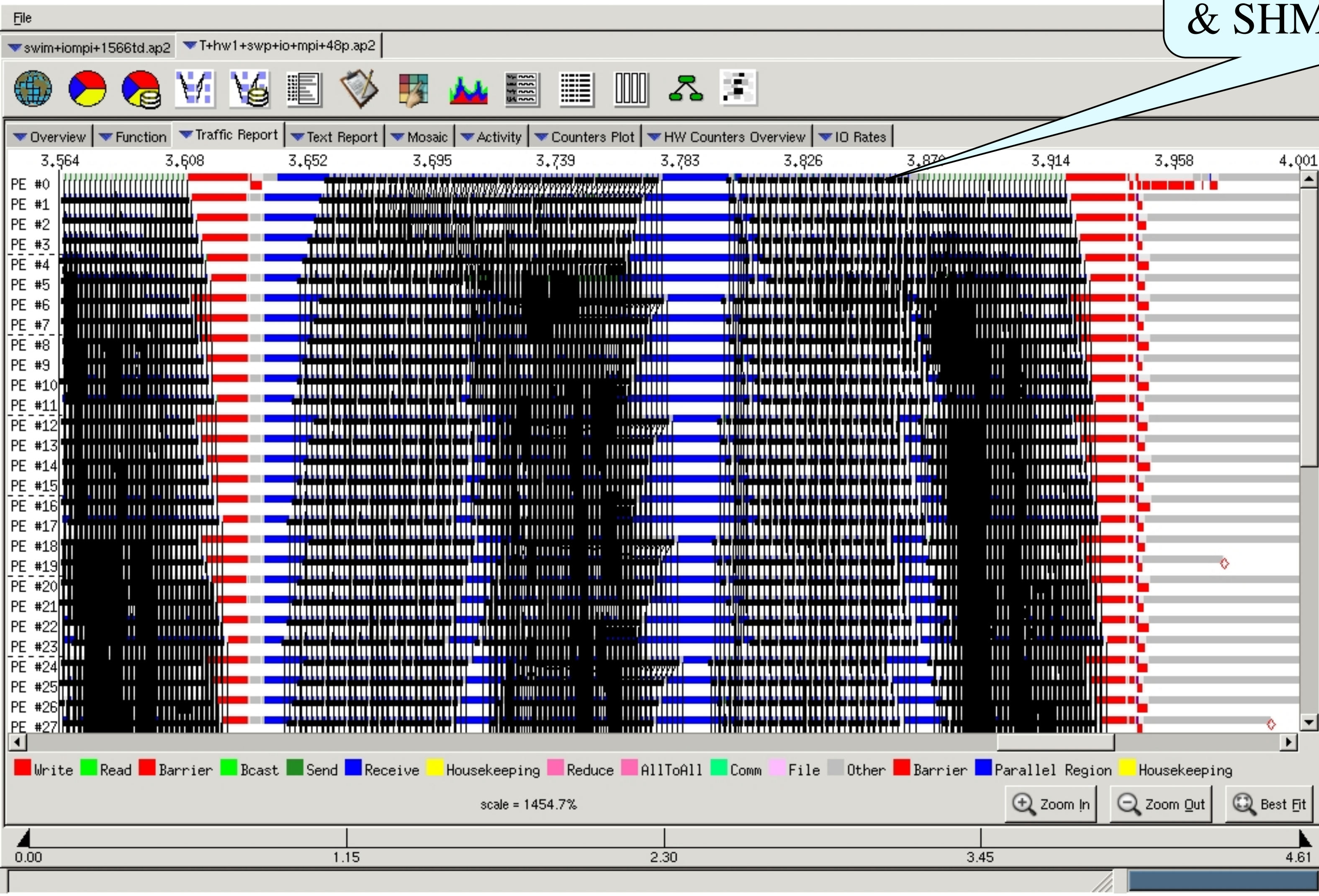
Tracing

- **Show tracing results (Time Live View)**
  - Information broken out by communication type (read, write, barrier, and so on)
- **Only true function calls can be traced**
  - Functions that are inlined by the compiler or that have local scope in a compilation unit cannot be traced
- **Enabled with `pat_build -g, -u, -T` or `–w` options**
- **Full trace (sequence of events) enabled by setting `Pat_RT_SUMMARY=0`**

Time Line View (Sweep3D)



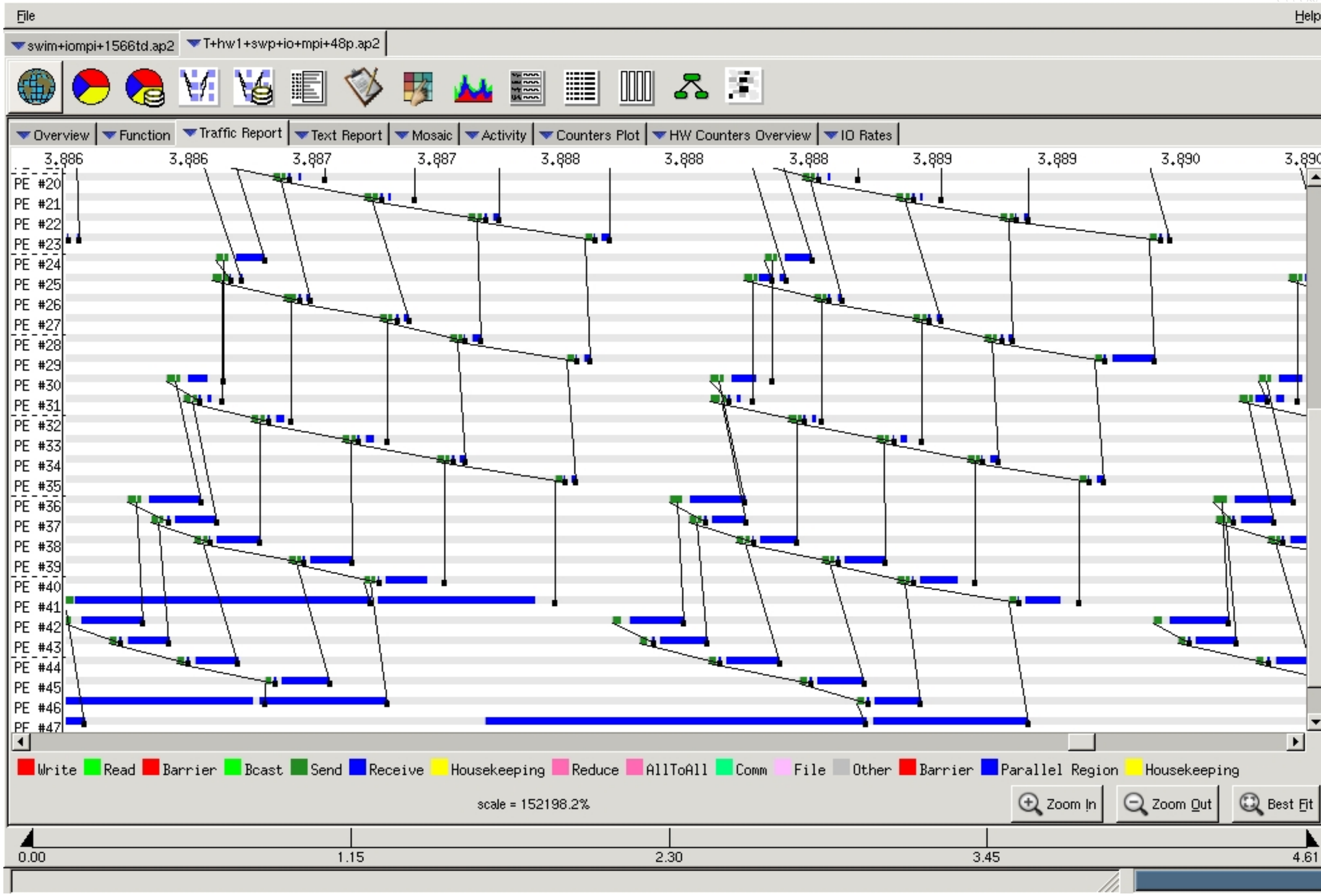
Time Line View (Zoom)



User Functions, MPI & SHMEM Line

I/O Line

Time Line View (Fine Grain Zoom)



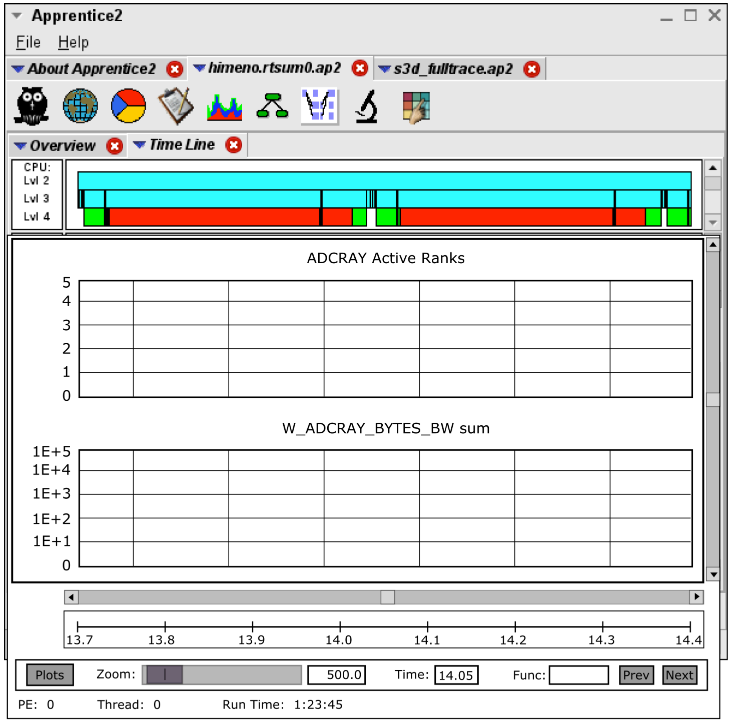
Other Cray Apprentice2 Reports

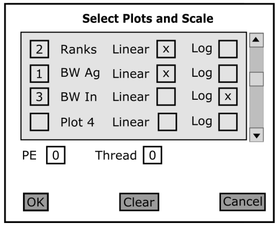
- **Environment reports**
  - Provide general information about the conditions under which the data file currently being examined was created
- **Traffic Report**
  - shows internal PE-to-PE traffic over time. T
    - information is broken down by comm. type (read, write, barrier etc.)
- **I/O Rates Report**
  - table listing quantitative information about program's I/O usage.
    - look for I/O activities that have low average rates and high data volumes;
    - this may indicate that the file should be moved to a different file system.
- **Hardware reports**
  - Available only if hardware counter information was captured

- **Full description at: http://docs.cray.com/books/S-2376-63/S-2376-63.pdf**

I/O display in Apprentice2

- New feature which allows user to study MPI I/O and file I/O activity over time, associating back to call tree





Compiler feedback and variable scoping with Reveal

Reveal

- **For an OpenMP port a developer has to understand the scoping of the variables, i.e. whether variables are shared or private.**
- **Reveal is Cray's next-generation integrated performance analysis and code optimization tool.**
  - Source code navigation using whole program analysis (data provided by the Cray compilation environment.)
  - Coupling with performance data collected during execution by CrayPAT. Understand which high level serial loops could benefit from parallelism.
  - Enhanced loop mark listing functionality.
  - Dependency information for targeted loops
  - Assist users optimize code by providing variable scoping feedback and suggested compile directives.

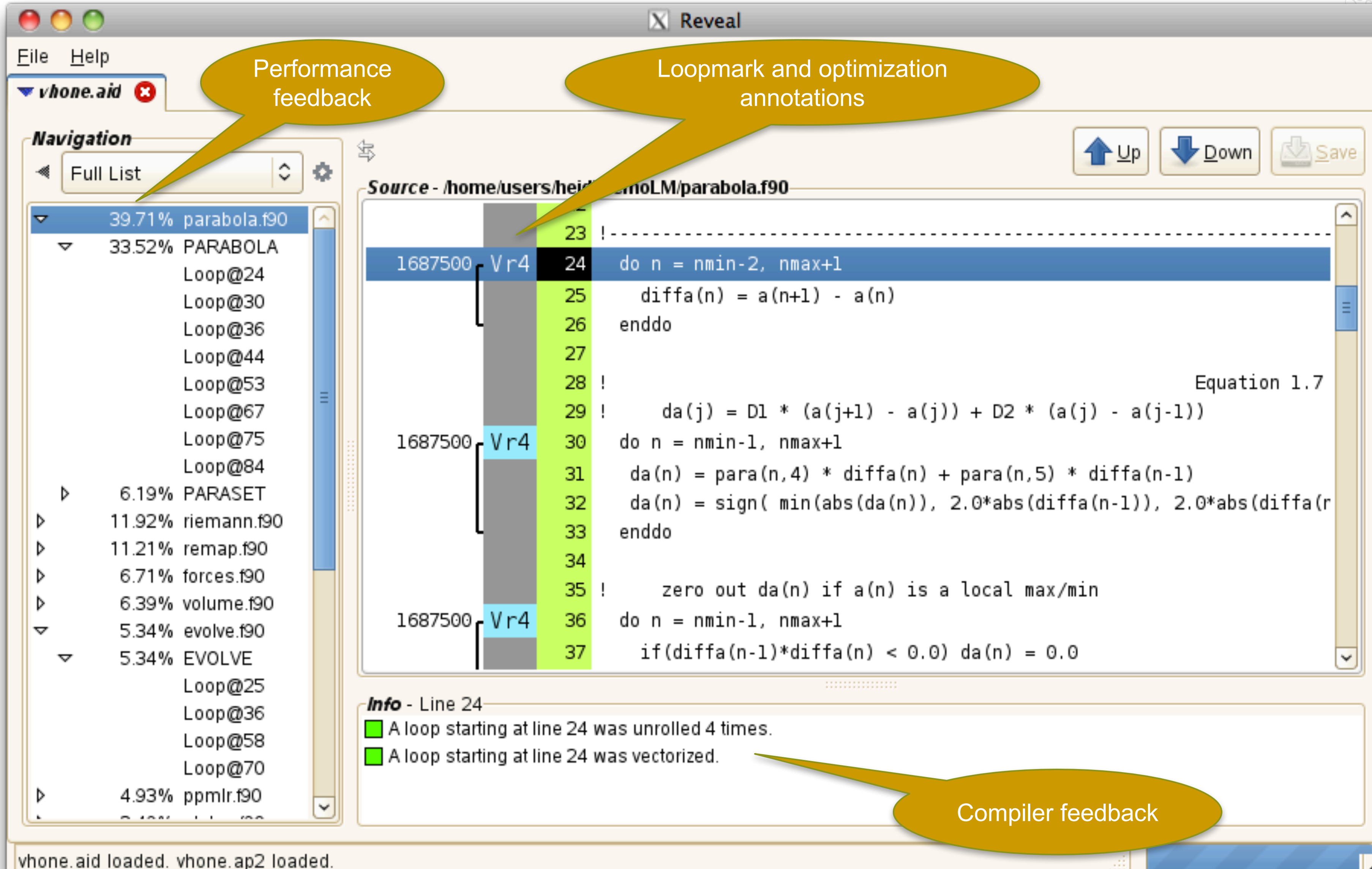
Input to Reveal

- Recompile to generate program library
- Performance data from a separate loop timing trace experiment
- Launch Reveal

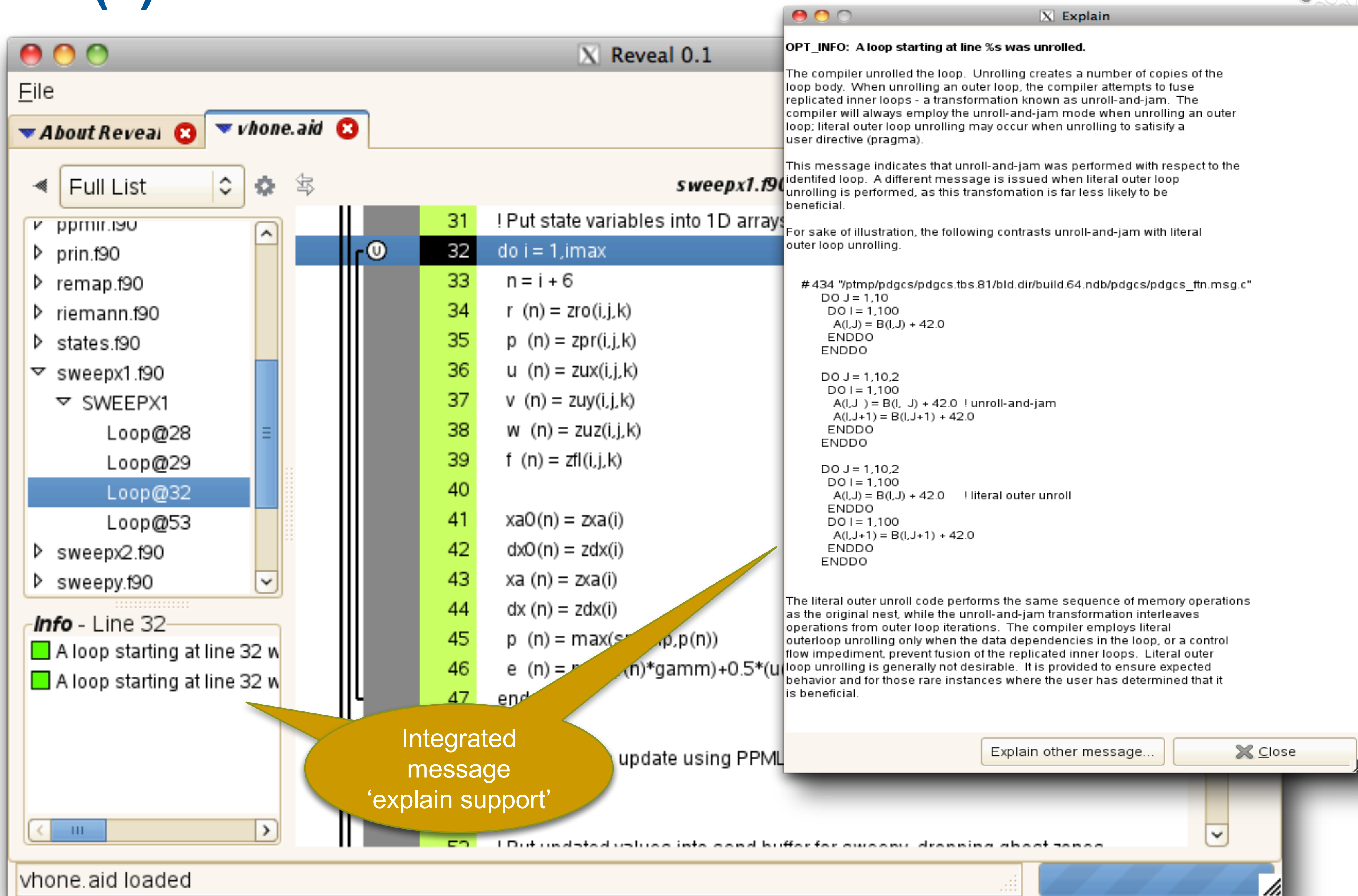
```
$> module load perftools
$> ftn -O3 -hpl=my_program.pl -c my_program_file1.f90
$> reveal my_program.pl my_program.ap2 &
```

- You can omit the *.ap2 and inspect only compiler feedback.
- Note that the `profile_generate` option disables most automatic compiler optimizations, which is why Cray recommends generating this data separately from generating the program_library file.

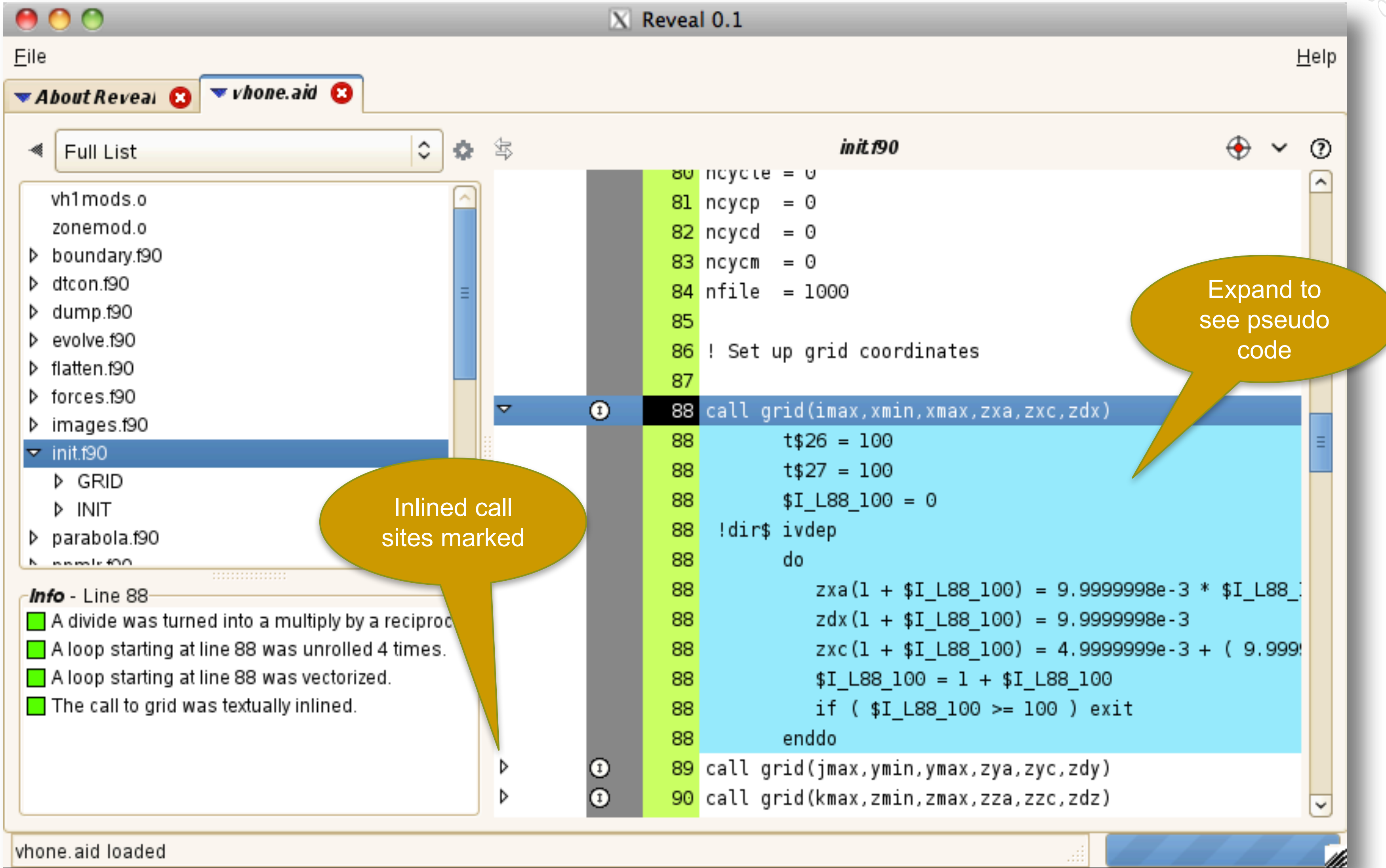
Visualize CCE's Loopmark with Performance Profile



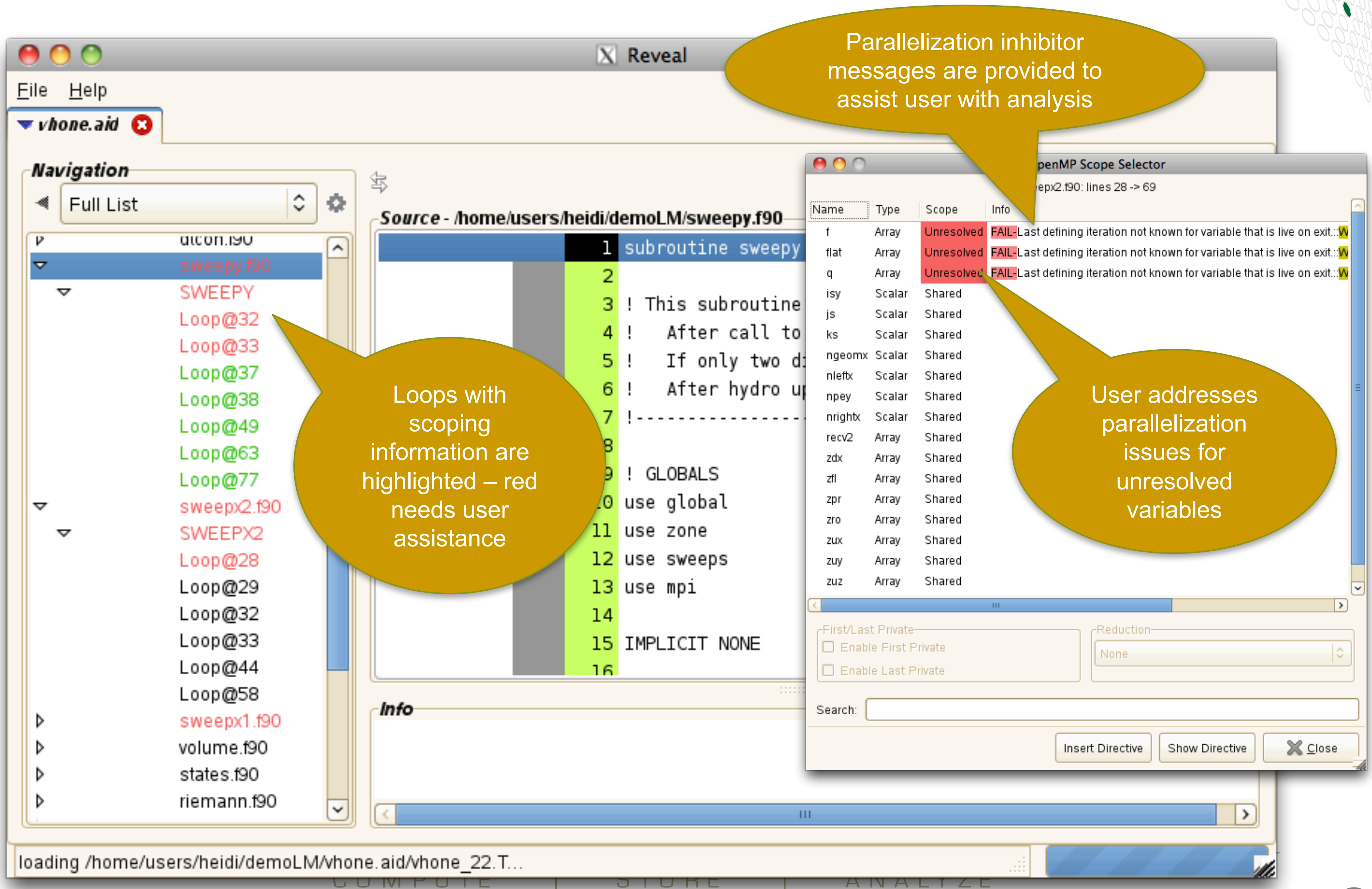
Visualize CCE's Loopmark with Performance Profile (2)



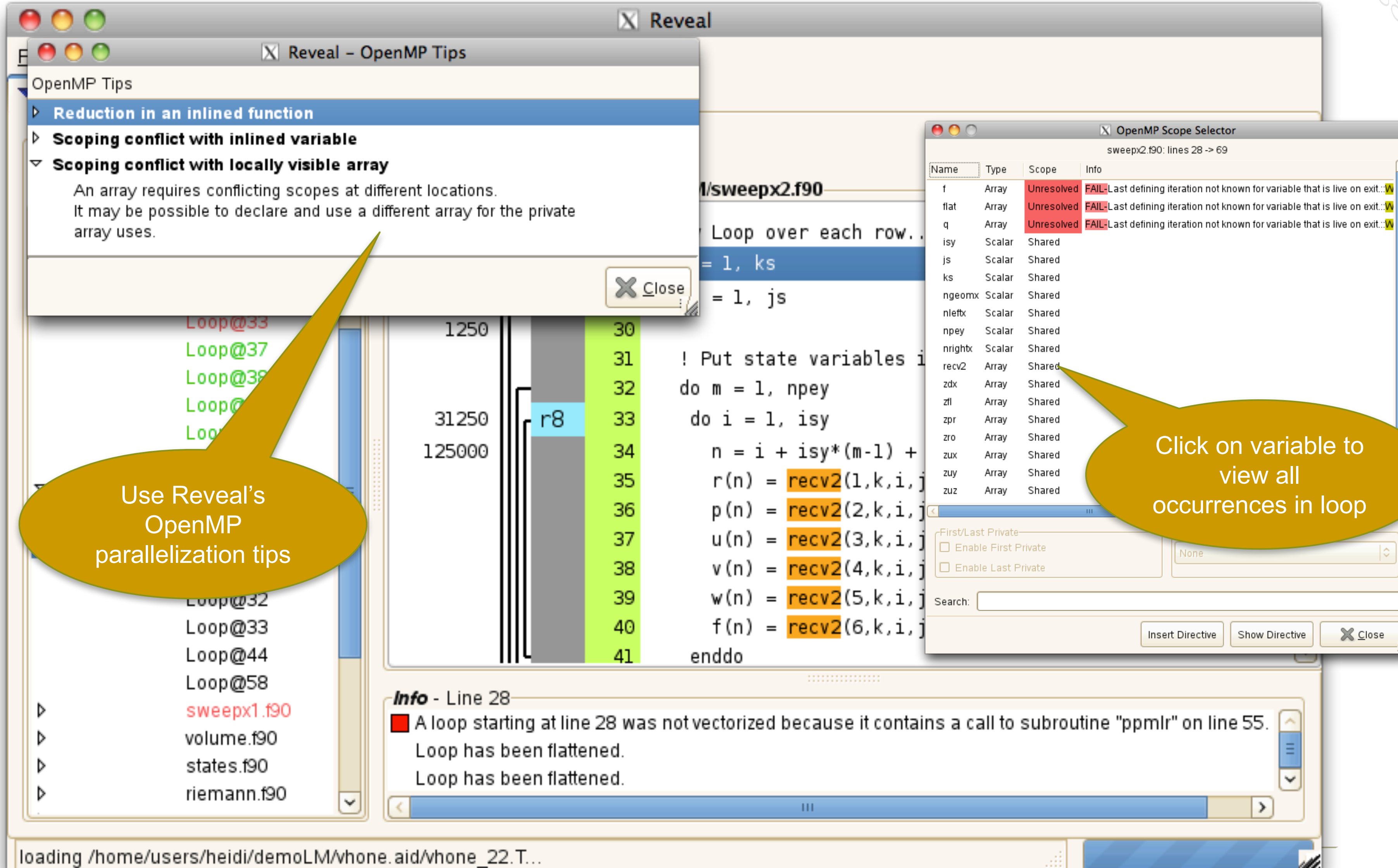
View Pseudo Code for Inlined Functions



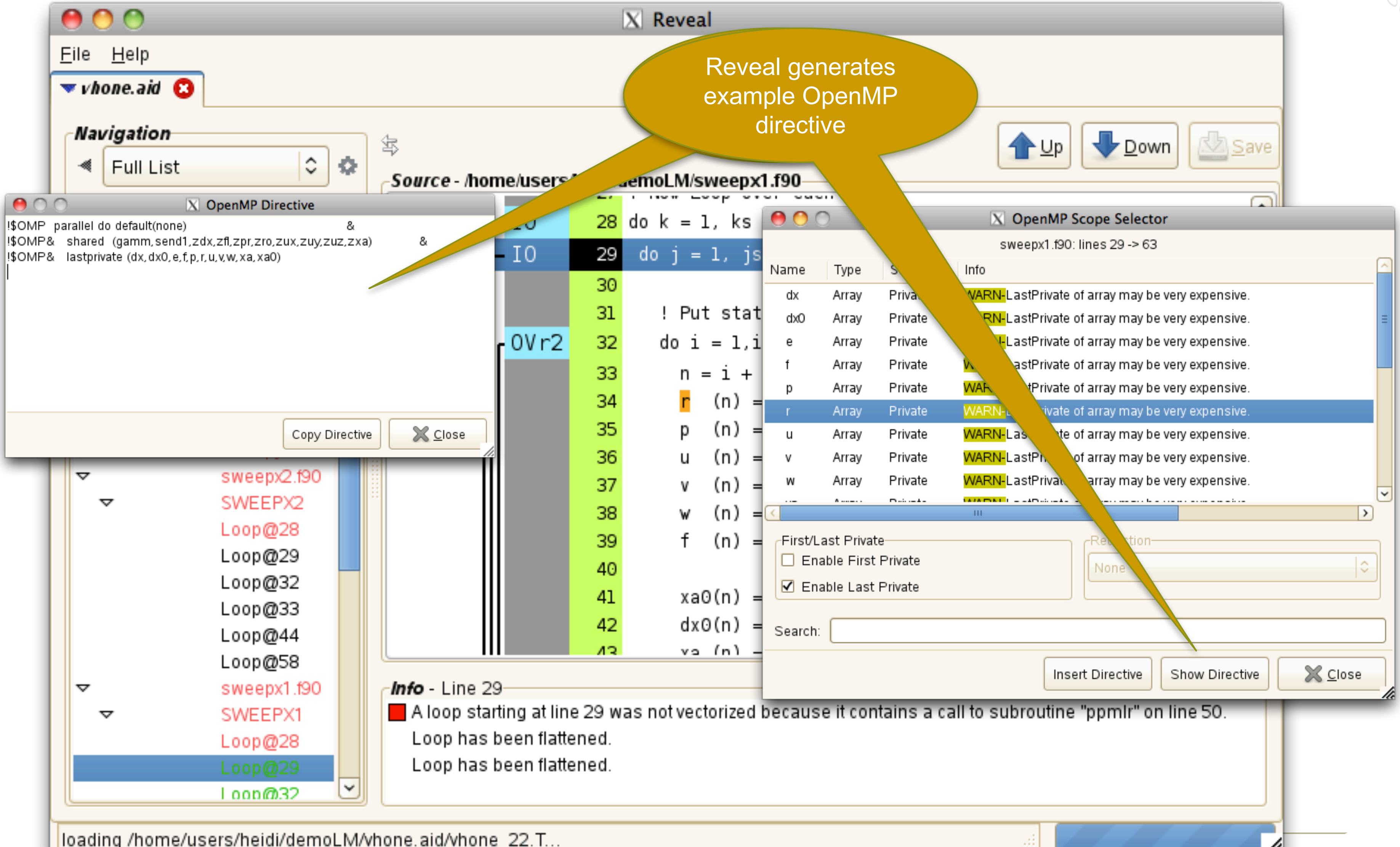
Scoping Assistance – Review Scoping Results



Scoping Assistance – User Resolves Issues



Scoping Assistance – Generate Directive



Automatic paralellization with Reveal

Reveal Auto-Parallelization

- Use an automated procedure to create loop work estimates for use with Reveal
- Build an experimental binary that includes automatic runtime-assisted parallelization
- Explore if high-level loops that contain subroutine calls can be automatically parallelized
- Goal is to assist the user with adding additional levels of parallelism to their program

Reveal Auto-Parallelization Recipe

Perform performance analysis run

- Loop level tracing using module load perftools-lite-loops

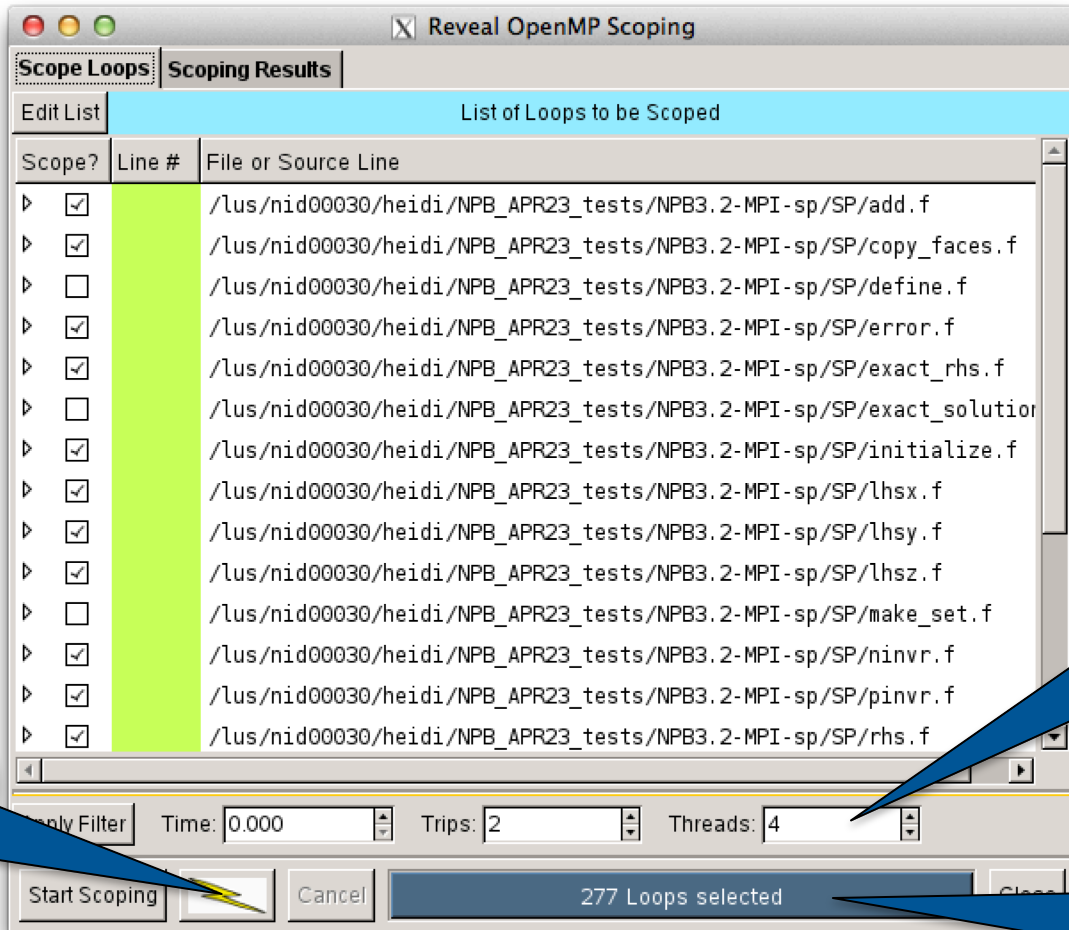
Auto parallelize important loops

- Recompile with program library option, i.e. -hpl=objcode.exe.pl
- reveal objcode.exe.pl *.ap2, select loops, perform loop scoping

Run experimental binary and compare wallclock against performance baseline

Optionally add parallel directives to code

Reveal: Create Experimental Binary



Thread Count to Use at Runtime

One Step to Create New Binary (only from perftools 6.3.2)

277 Loop Candidates for Parallelization

Reveal: Example of Parallelization Hints

Reveal OpenMP Scoping

Scope Loops | Scoping Results | Footnote

Scoping Footnote

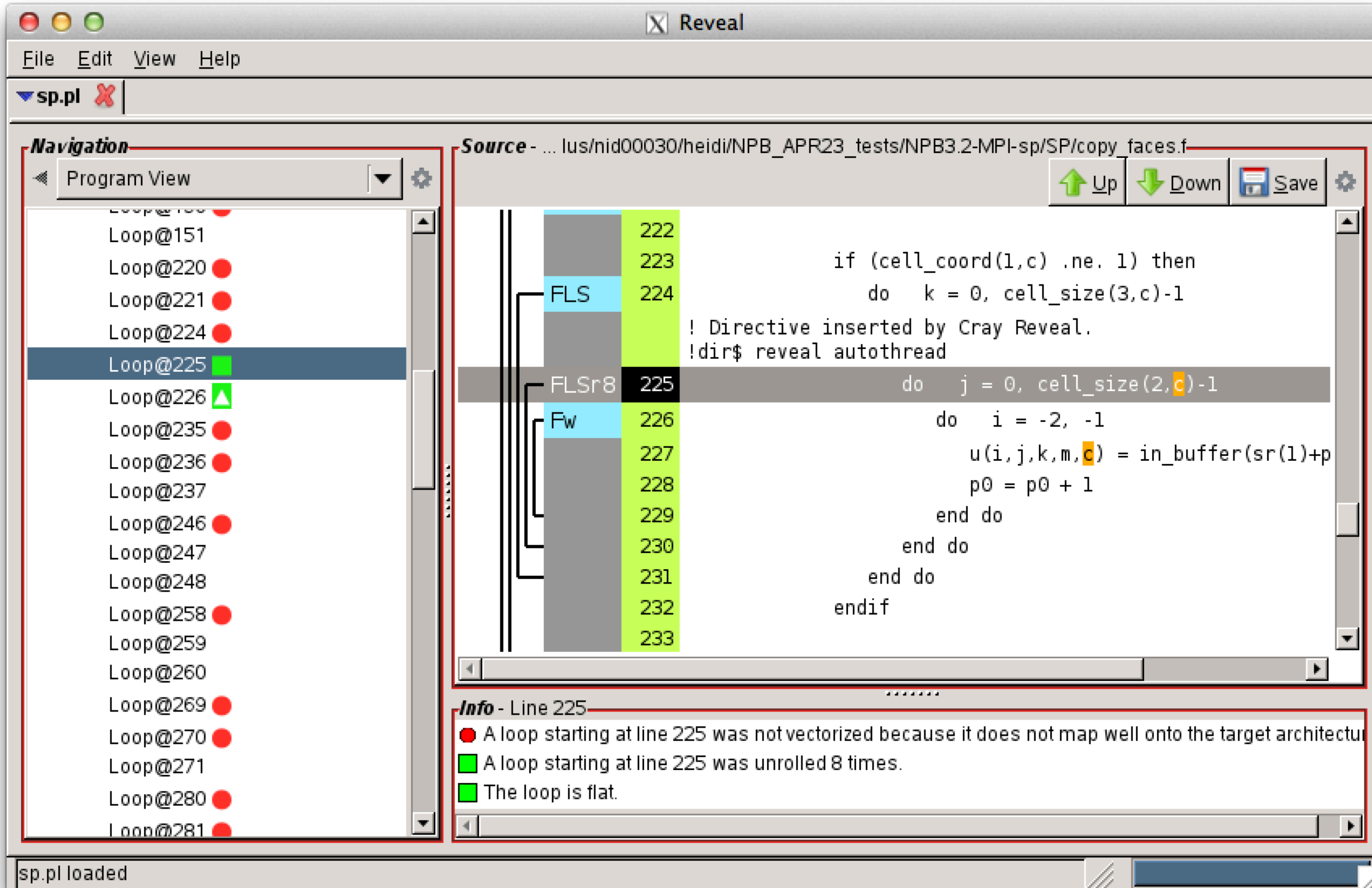
There are situations were the compiler is able to parallelize
a loop, but not able to provide equivalent OpenMP directives.
Here, REVEAL AUTOTHREAD mode is required because a secondary
induction must be rewritten to eliminate a recurrence.
The following example illustates this issue:

```
subroutine RAT02( k, A, n, m )
  real    A(*)
  integer k
  do j = 1, n        ! scoping this loop.
     do i = 1, m
        A(k) = i+j
        k = k + 1
     enddo
  enddo
end
```

The compiler effectively rewrites k as k0 + (i-1) + m*(j-1),
where k0 is the value of k on entry to the nest. This is known
as replacement of a secondary induction.

Close

Reveal: New Autothread Directive



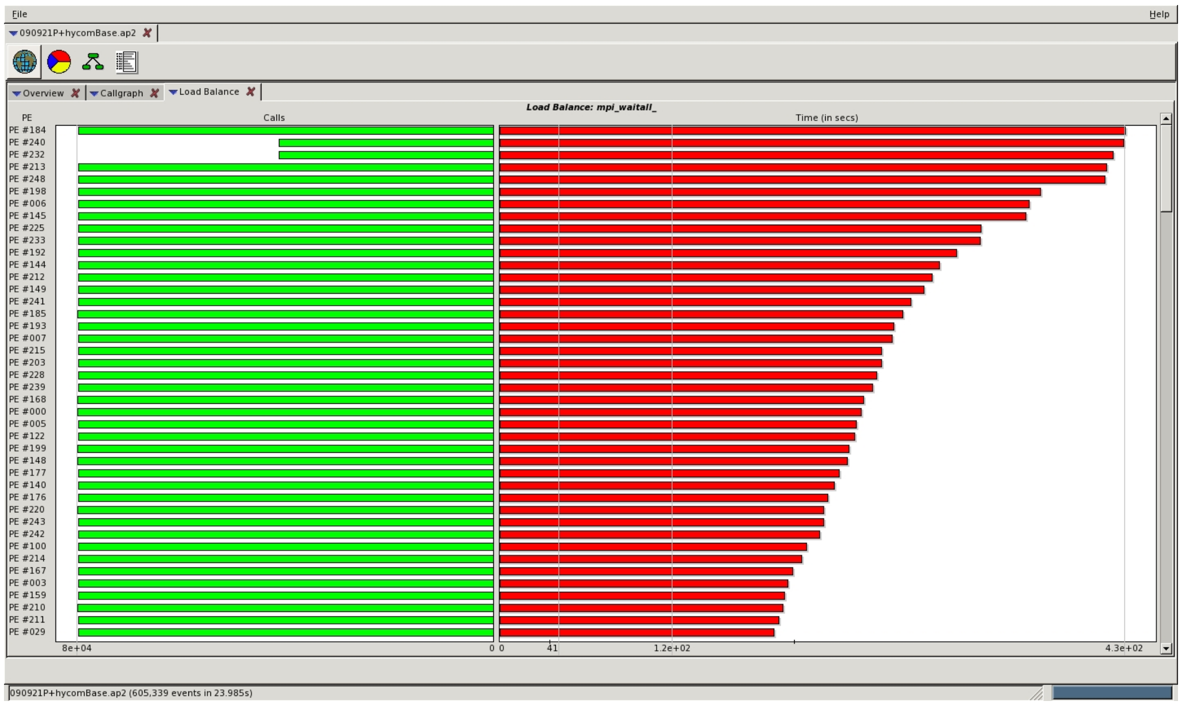
Reveal: New Autothread Directive

- Optional directive available when loop cannot be parallelized via OpenMP directives (without code rewrite)
- Loop directive
- Inlines all calls within a loop
- No runtime threshold for directive
- Correctness ensured

Optimisations for MPI

Rank Reordering

- Sometimes an MPI application is not well balanced



- The MPI library can reorder the ranks at runtime based on the setting of MPICH_RANK_REORDER_METHOD

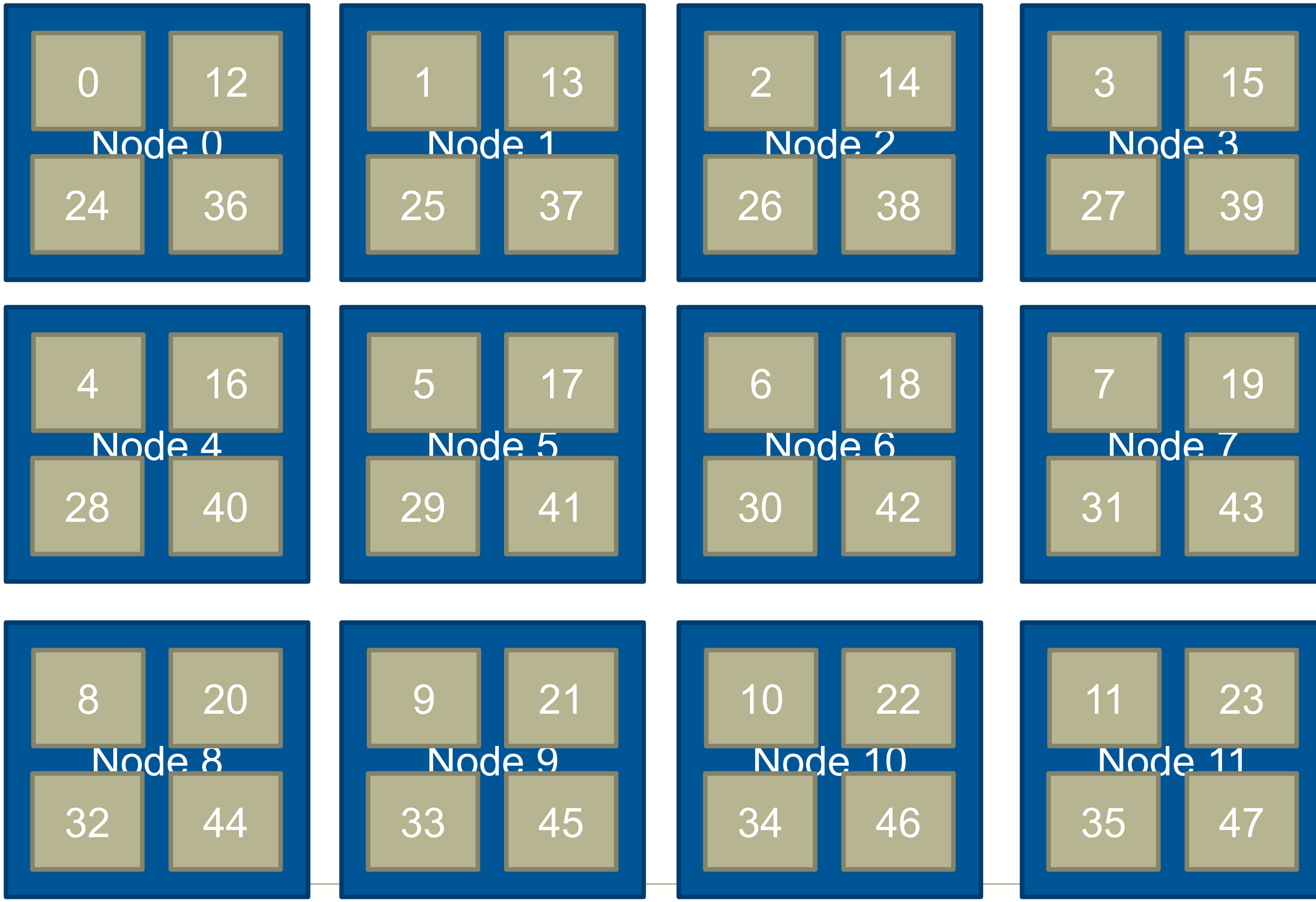
Rank Placement

- **Start with a list of nodes to run on**

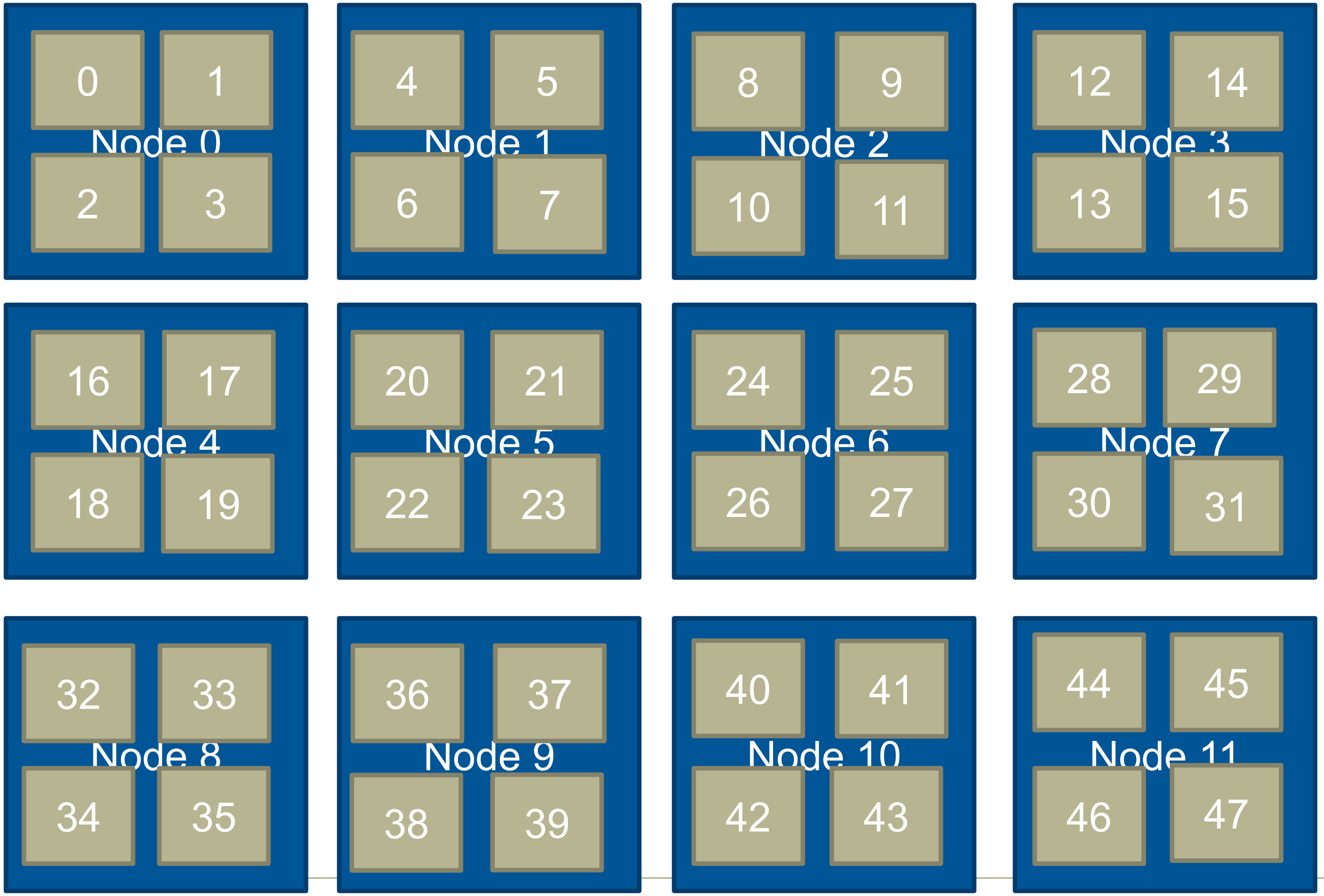
- **0: Round-robin placement**
  - Sequential ranks are allocated one per node in sequence
  - Placement starts again on first node if we reach the last node
- **1: SMP-style placement (default)**
  - Sequential ranks fill up each node in turn
  - Only then move on to the next node
- **2: Folded rank placement**
  - Similar to round-robin placement
  - except each pass over node list is in the opposite direction
- **3: Custom ordering**
  - The location of each rank in turn is specified in a list

- **Examples of these are shown on the next slide**
  - For a simplified example of four cores per node

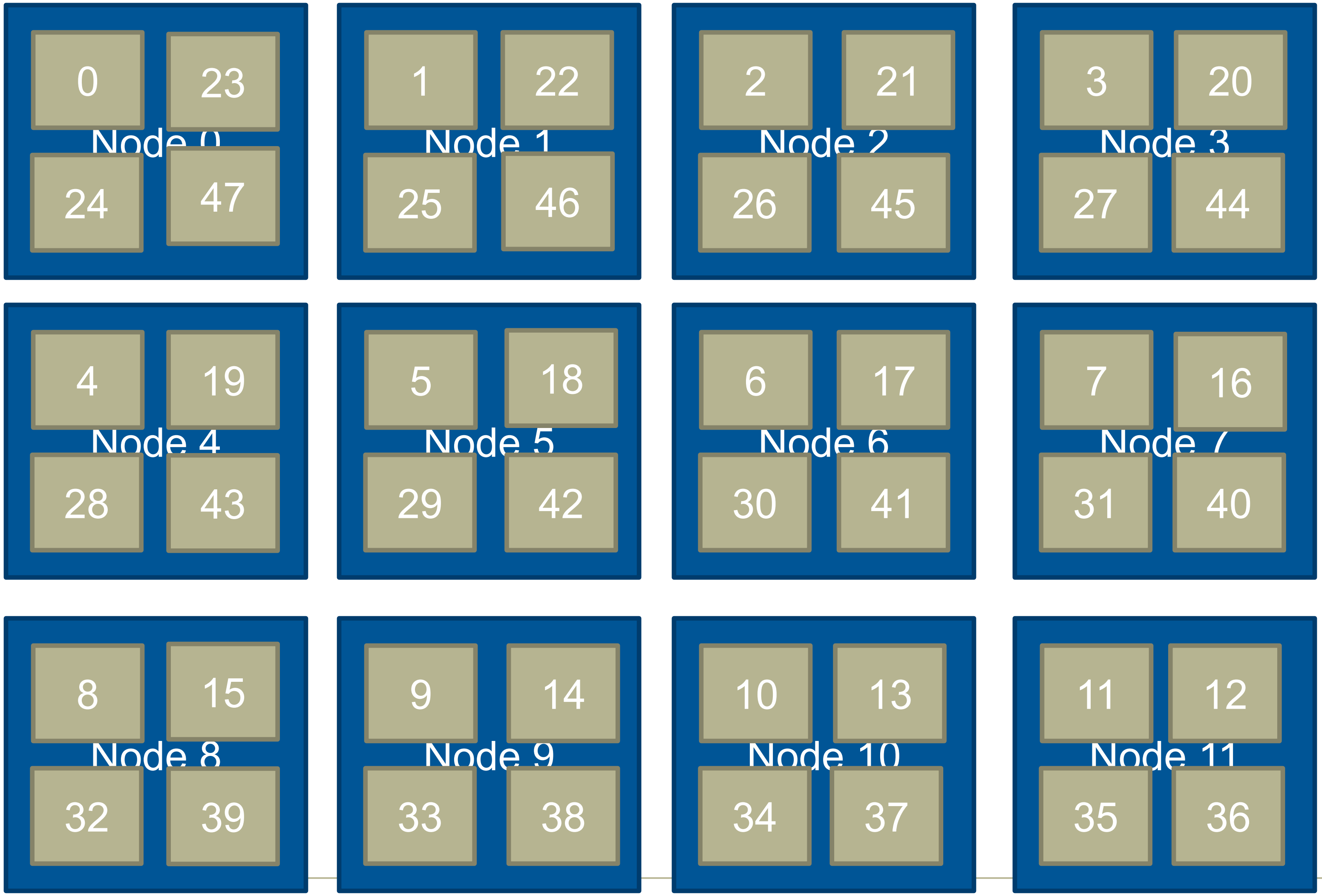
0: Round Robin Placement



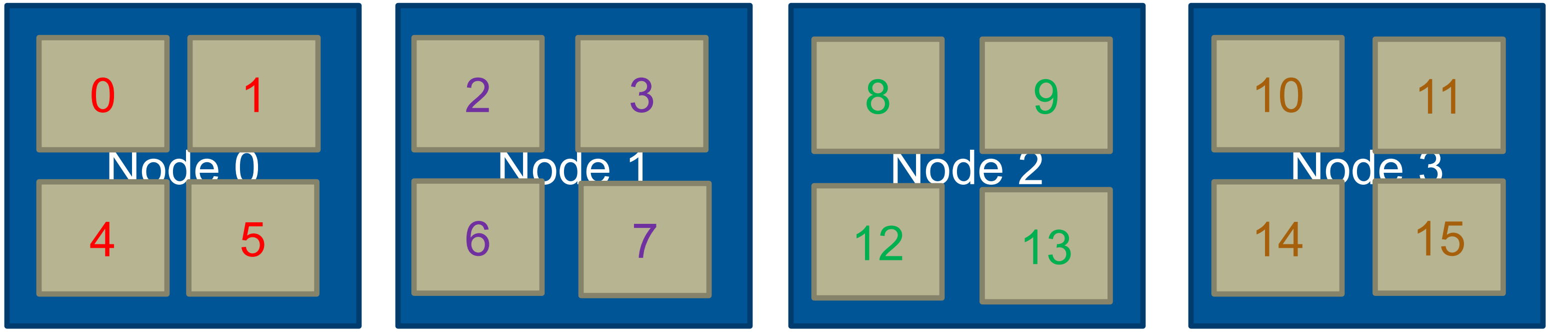
1: SMP Placement (default)



2: Folded Placement



3: Custom Example




```
MPICH_RANK_ORDER: 0,1,4,5,2,3,6,7,8,9,12,13,10,11,14,15
```

- **`MPICH_RANK_REORDER=3` enables this**
- **Ordering comes from file `MPICH_RANK_ORDER`**
  - comma separated ordered list
    - can optionally be condensed into hyphenated ranges
  - all ranks should be included in the list once and only once
- **Nodes are filled up SMP-style**
  - but not with sequential rank numbers
  - instead, take ranks sequentially from the `MPICH_RANK_ORDER` list

```
MPICH_RANK_ORDER: 0,1,4,5,2,3,6-9,12,13,10,11,14,15
```

Rank placement with CrayPat

```
MPI grid detection:

    There appears to be point-to-point MPI communication in a 20 X 16
    grid pattern. The 27.5% of the total execution time spent in MPI
    functions might be reduced with a rank order that maximizes
    communication between ranks on the same node. The effect of several
    rank orders is estimated below.

    A file named MPICH_RANK_ORDER.Grid was generated along with this
    report and contains usage instructions and the Custom rank order
    from the following table.

         Rank     On-Node      On-Node   MPICH_RANK_REORDER_METHOD
        Order    Bytes/PE    Bytes/PE%
                              of Total
                              Bytes/PE

        Custom   8.092e+09      75.00%   3
           SMP   4.580e+09      42.45%   1
          Fold   2.290e+08       2.12%   2
    RoundRobin   0.000e+00       0.00%   0
```

When testing this the time went only down to 348 from 360 seconds, but approach might become important when scaling higher

Further information from CrayPat

```
Metric-Based Rank Order:

    When the use of a shared resource like memory bandwidth is
    unbalanced across nodes, total execution time may be reduced
    with a rank order that improves the balance.  The metric used
    here for resource usage is: USER Time

    For each node, the metric values for the ranks on that node
    are summed. The maximum and average value of those sums are
    shown below for both the current rank order and a Custom rank
    order that seeks to reduce the maximum value.

    A file named MPICH_RANK_ORDER.USER_Time was generated
    along with this report and contains usage instructions and the
    Custom rank order from the following table.

      Rank     Maximum     Average    Max:Ave   Reduction in Max
     Order       Value       Value     Ratio

     Custom   3.491e+03   3.393e+03      1.03    8.77%
    Current   3.827e+03   3.393e+03      1.13
```

Rank reordering

- **Easy to experiment with**
  - defaults at least should be tested with every application…
  - CrayPat can help generate the reorder file
- **When might rank reordering be useful?**
  - If point-to-point communication consumes a significant fraction of program time and a load imbalance detected
    - e.g. for nearest-neighbour exchanges (see next slide)
  - Also shown to help for collectives (alltoall) on subcommunicators
  - Spread out I/O servers across nodes
  - If there is a good use case for exploiting the Intel hyperthreads
- **Have used this for I/O servers (NEMO) and radiation colocation (IFS)**