

# More on Application profiling and optimization

Ilias Katsardis  
[ikatsardis@cray.com](mailto: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



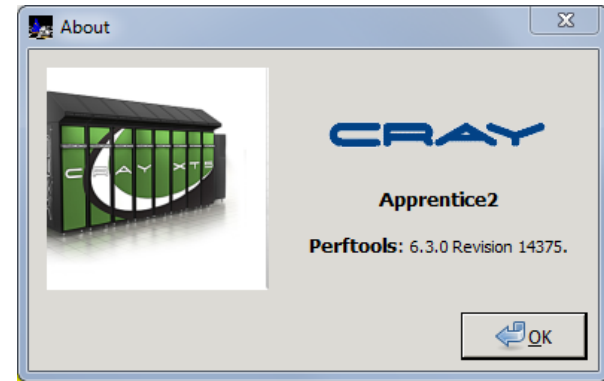
# To use Cray Apprentice<sup>2</sup>

- 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**





Apprentice2 (on eslogin006)

File Help

▼ About Apprentice2 x ▼ Espresso+pat+47254-3184t.ap2 x

Overview x

### Function/Region Profile

40.7% = MPI\_Waitall  
18.9% = calc\_... orce\_parts  
6.1% = MPI\_Recv

### Profile

#### CPU

Computation  
45.07%

Programming\_Model  
54.93%

IO  
0.0%

### Memory Utilization

Process HiMem (MBytes) 34.779

### Load Imbalance

4.49s = MPI\_Waitall  
2.65s = calc\_... orce\_parts  
1.46s = MPI\_Recv

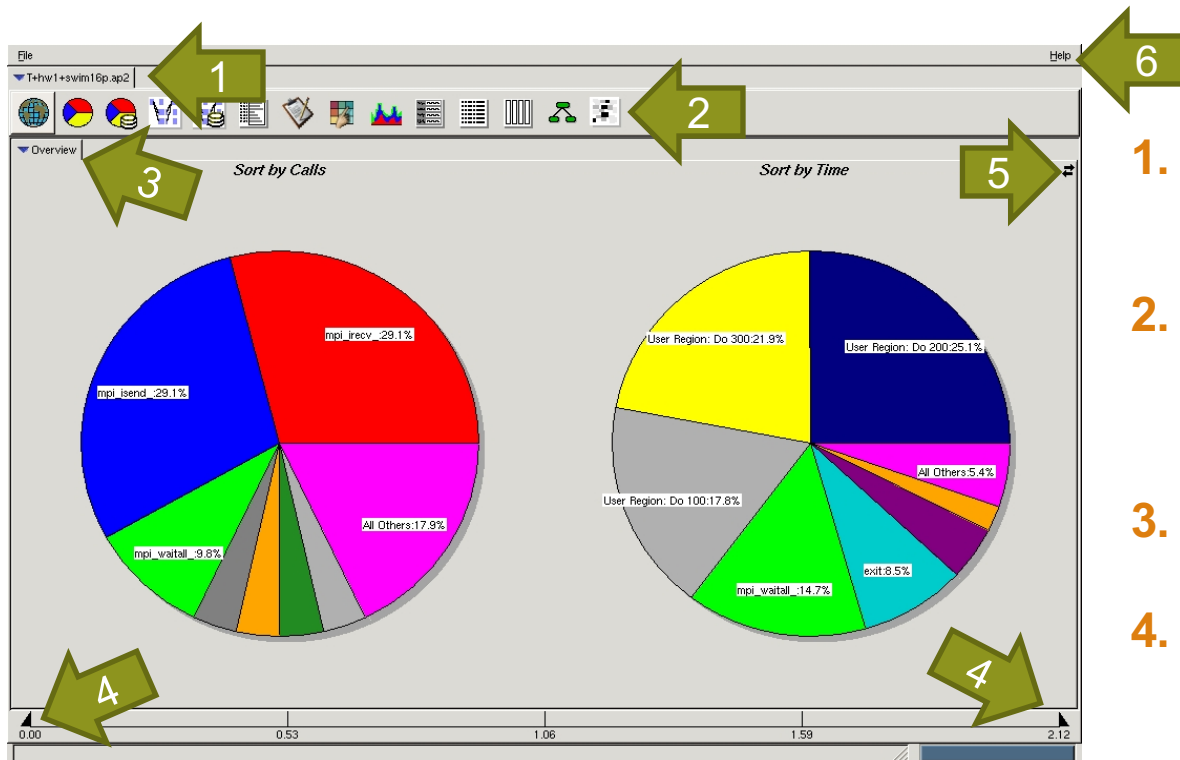
### Data Movement

MPI Msg MBytes 944.003

Wallclock time: 60,000000s

Espresso+pat+47254-3184t.ap2 (54,192 events in 0,255s)

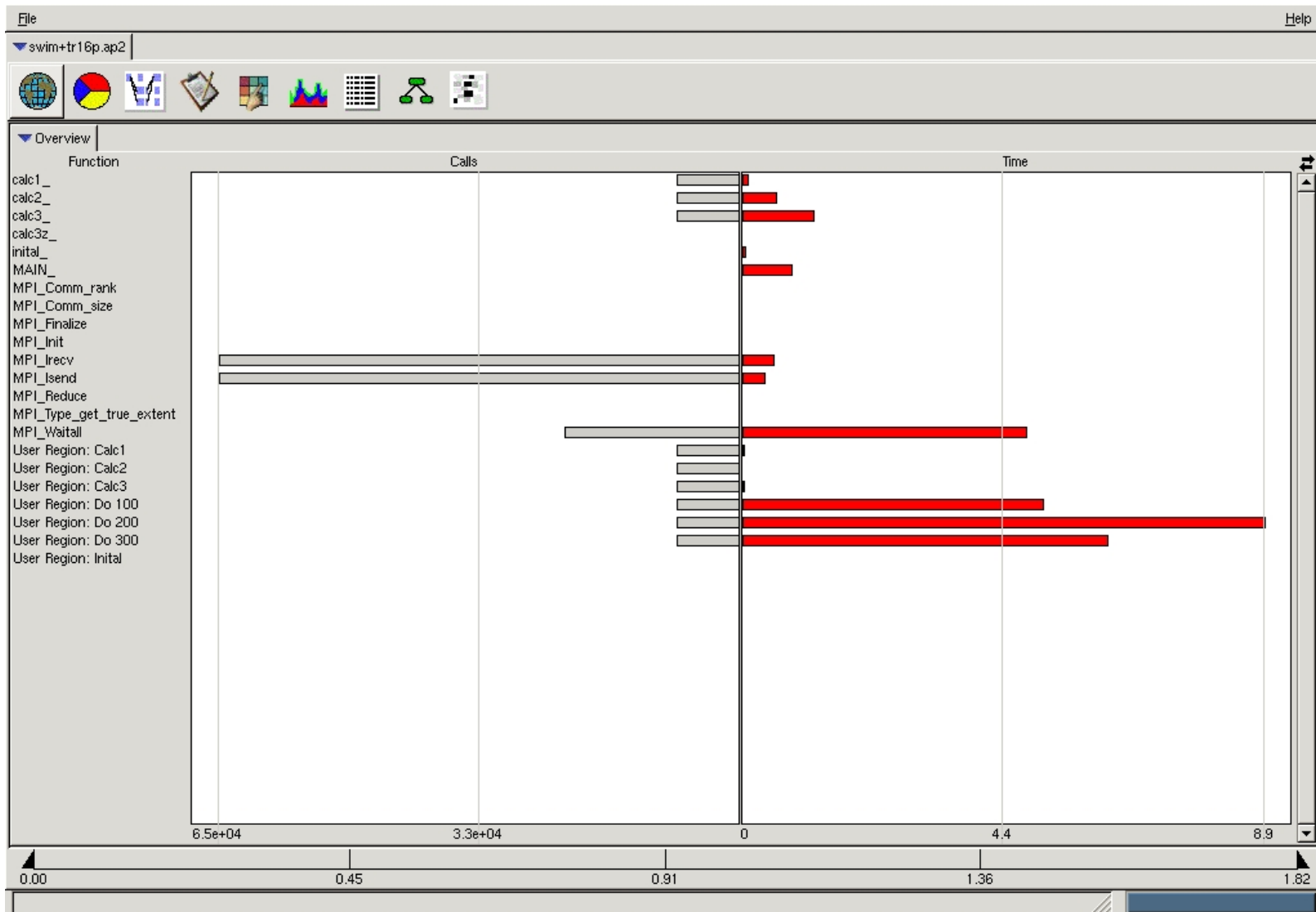
# Statistics Overview: Pie Chart



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

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

Overview Function

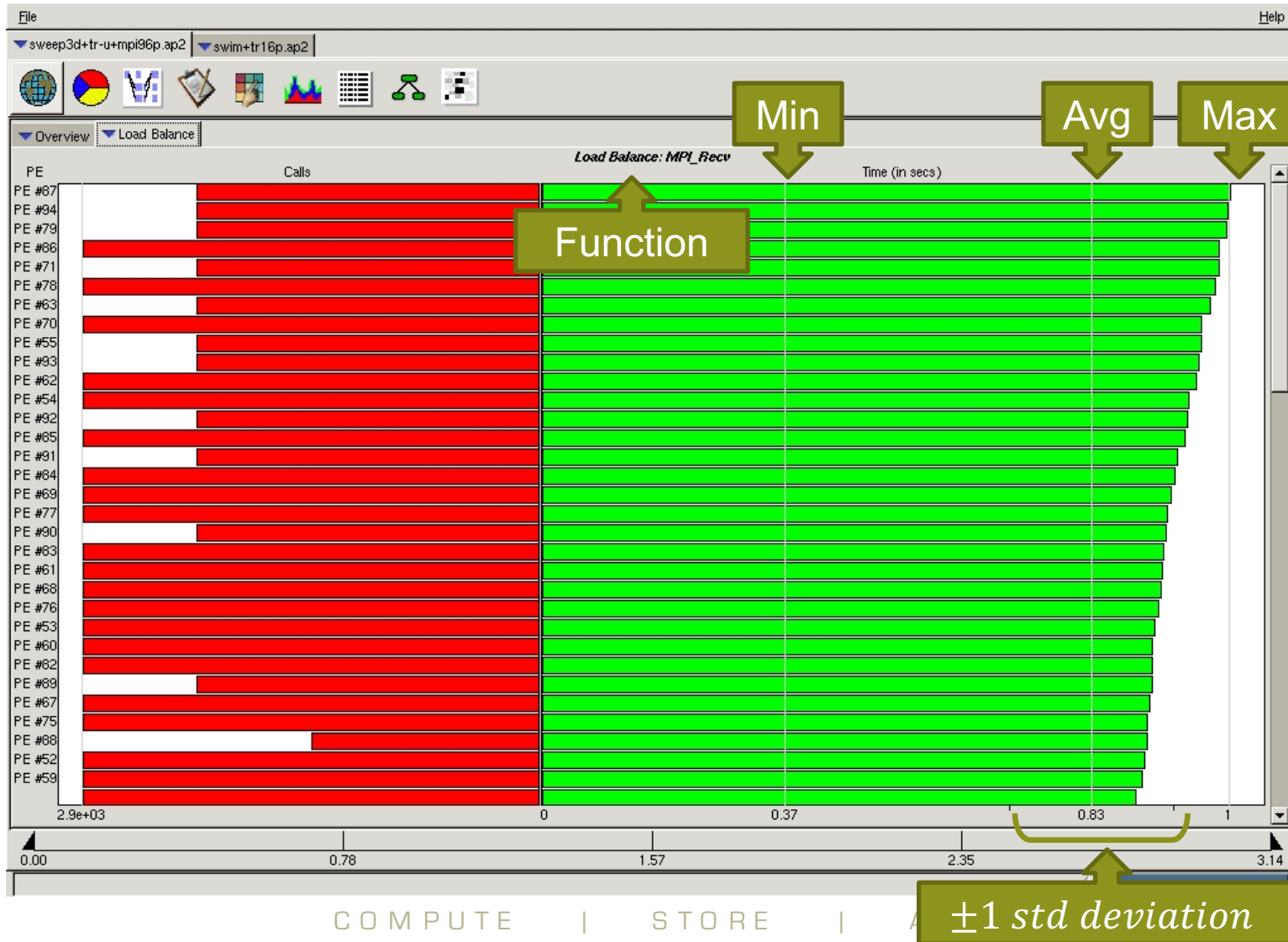
Time	Percent	Hits	Callsites	Imbalance %	Potential Savings	Function	Line	File
124.175511	63.29	576	1	5.63	0.15	sweep_	116	Aus/nid00036/ldr/Apps/sweep3d/sweep.f
40.211774	20.50	118080	1	23.40	0.25	mpi_rcv_	0	==NA==
16.319527	8.32	48	1	48.26	0.30	exit	35	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/stdlib/exit.c
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	Aus/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	Aus/nid00036/ldr/Apps/sweep3d/flux_err.f
0.382130	0.19	118080	2	10.98	0.00	snd_real_	135	Aus/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/fofwrite.c
0.185067	0.09	118080	2	17.30	0.00	rcv_real_	164	Aus/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.067832	0.03	48	1	4.56	0.00	initialize_	42	Aus/nid00036/ldr/Apps/sweep3d/initialize.f
0.059407	0.03	48	1	4.99	0.00	initxs_	77	Aus/nid00036/ldr/Apps/sweep3d/initialize.f
0.041371	0.02	48	1	23.84	0.00	inner_	72	Aus/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
0.018902	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	Aus/nid00036/ldr/Apps/sweep3d/octant.f
0.002457	0.00	576	1	18.79	0.00	global_real_max_	321	Aus/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.002083	0.00	48	1	89.55	0.00	MAIN_	72	Aus/nid00036/ldr/Apps/sweep3d/driver.f
0.001588	0.00	576	1	39.10	0.00	global_int_sum_	373	Aus/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.001393	0.00	48	1	10.23	0.00	inner_auto_	69	Aus/nid00036/ldr/Apps/sweep3d/inner_auto.f
0.000982	0.00	48	1	97.74	0.00	task_init_	24	Aus/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.000739	0.00	384	2	27.87	0.00	global_real_sum_	347	Aus/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/fofopen.c
0.000499	0.00	48	1	7.54	0.00	initsnc_	175	Aus/nid00036/ldr/Apps/sweep3d/initialize.f

0.00 1.15 2.30 3.45 4.61

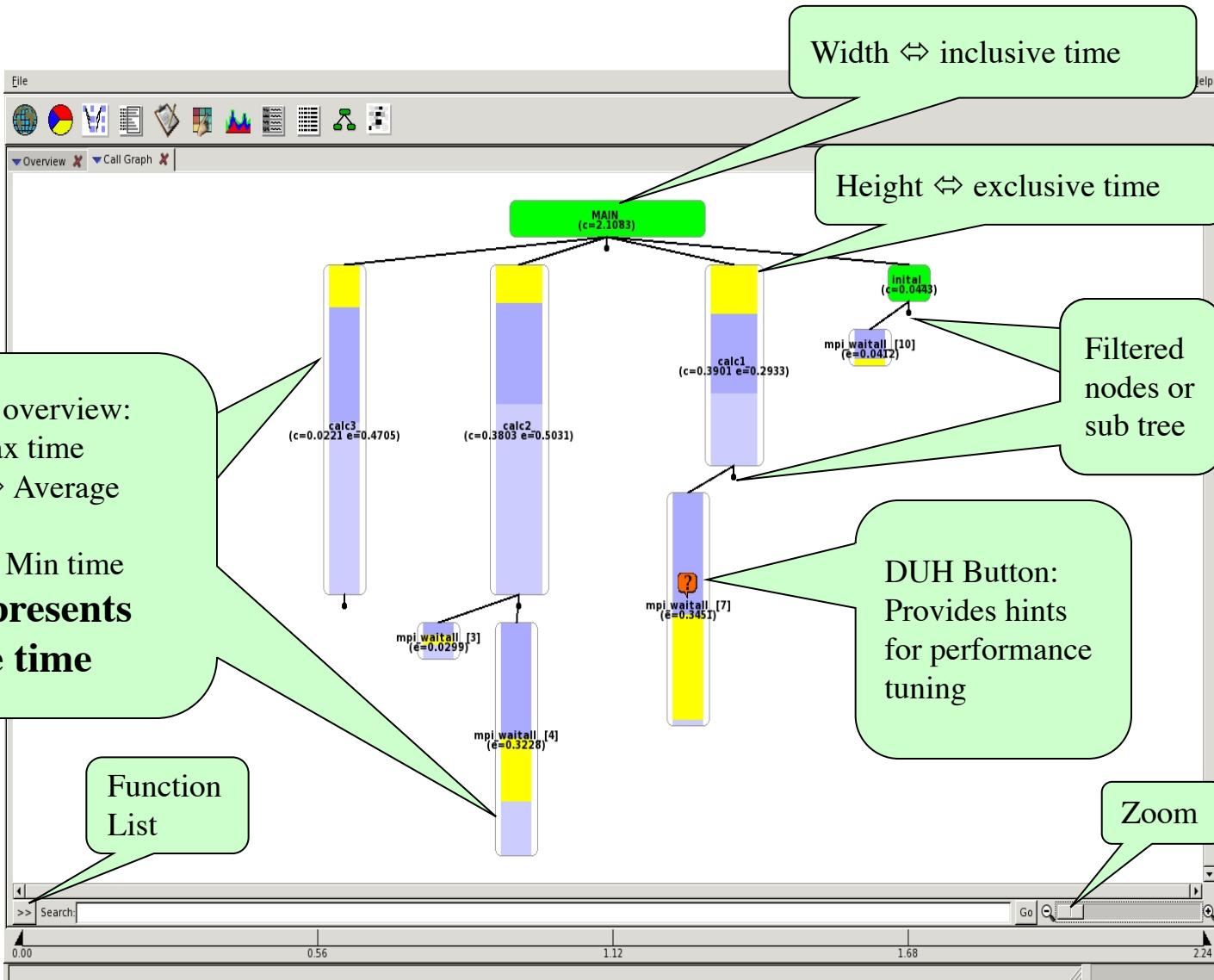


# Load Balance View (Aggregated from Overview)

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



# Call Tree View



# Call Tree View – Function List

**Right mouse click: Node menu e.g., hide/unhide children**

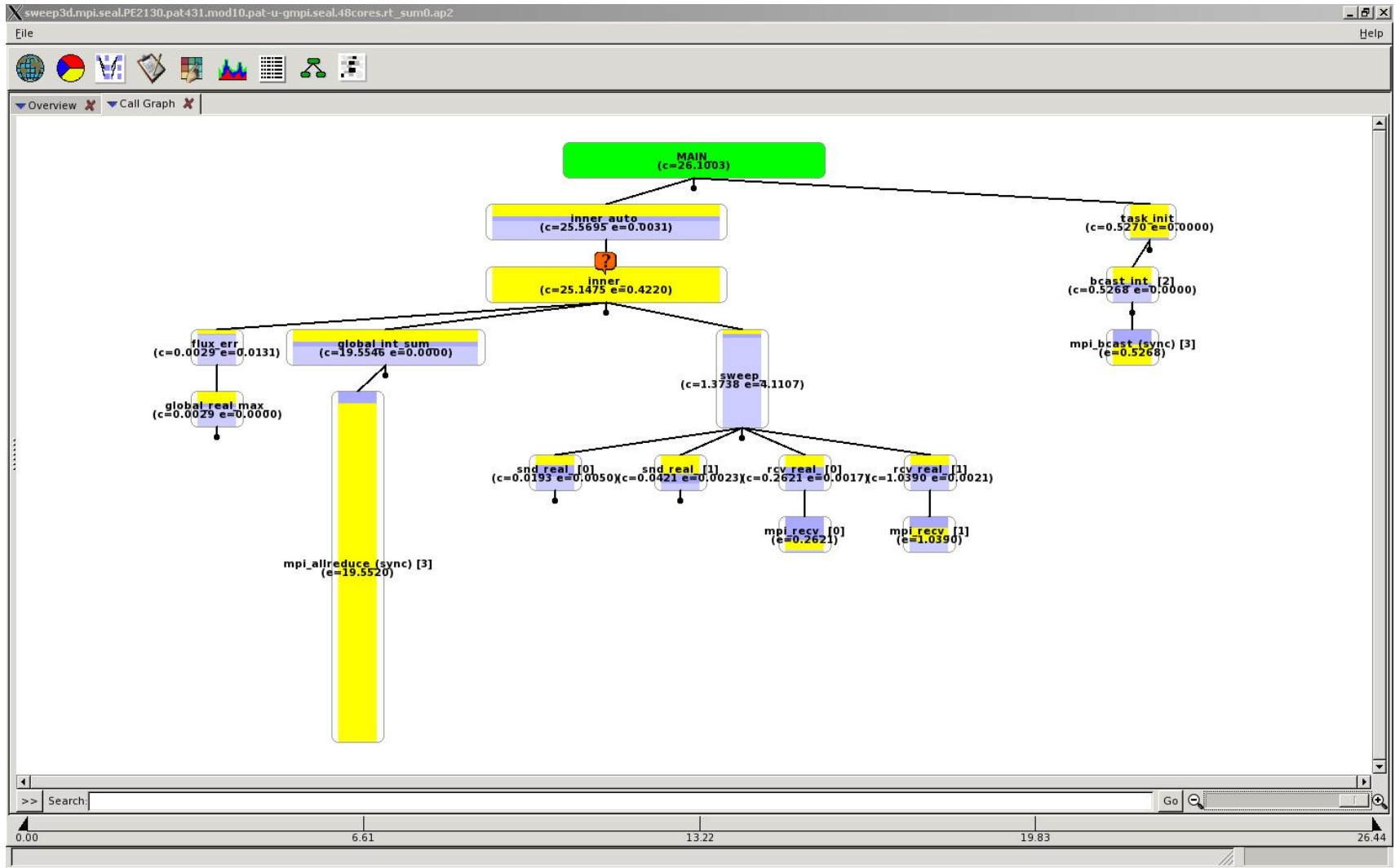
**Right mouse click: View menu: e.g., Filter**

**Sort options**  
 % Time,  
 Time,  
 Imbalance %  
 Imbalance time

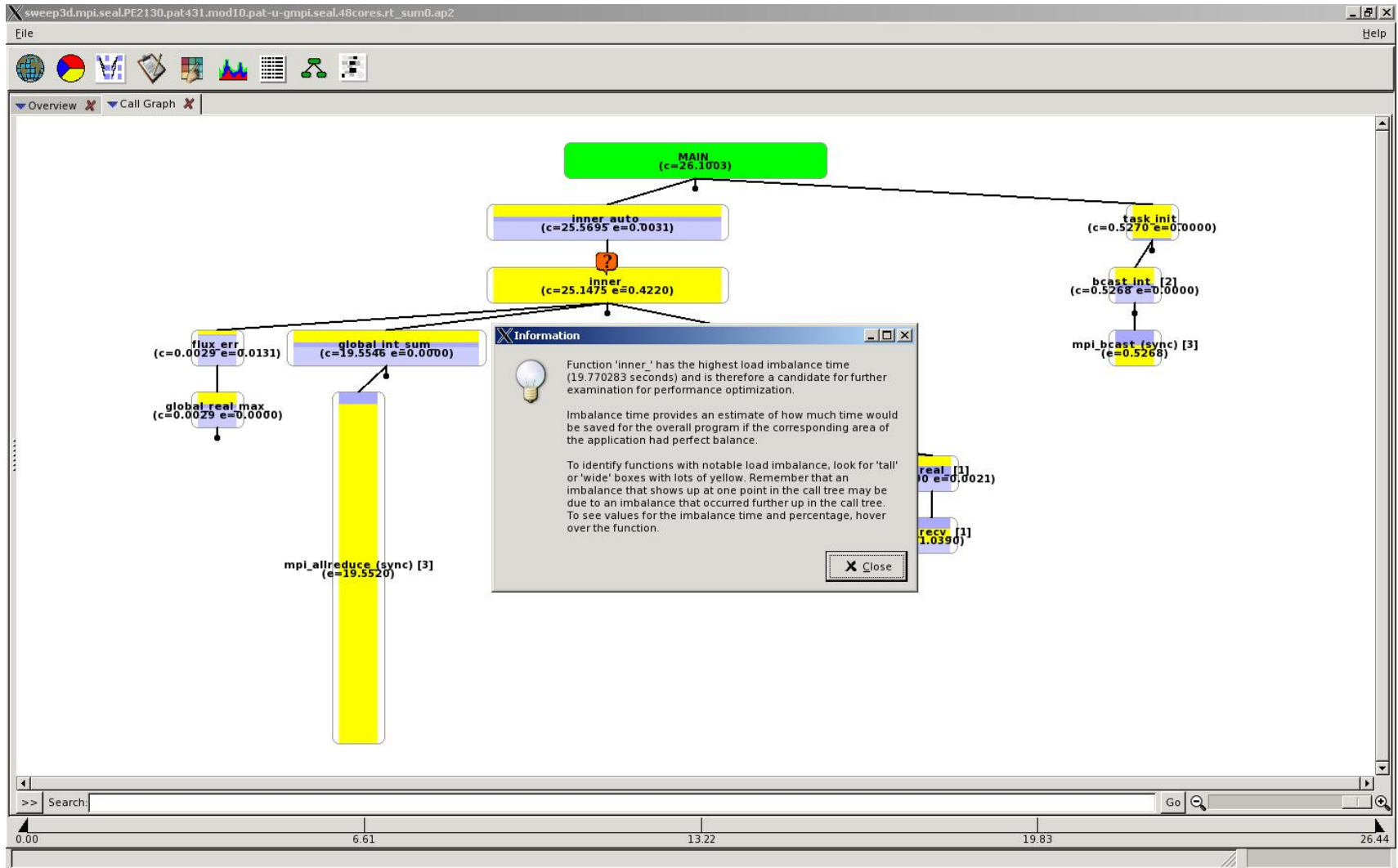
**Function List off**

lmb Time	Name
0.3702	mpi_waitall_ [7]
0.3103	mpi_waitall_ [4]
0.1586	mpi_waitall_ [10]
0.1226	mpi_waitall_ [6]
0.1108	mpi_waitall_ [1]
0.1017	mpi_waitall_ [3]
0.0917	calc1_
0.0673	calc3_
0.0649	calc2_
0.0249	mpi_waitall_ [9]
0.0161	mpi_isend_ [13]
0.0129	mpi_irecv_ [10]
0.0117	mpi_isend_ [10]
0.0090	mpi_waitall_ [0]
0.0084	mpi_isend_ [7]
0.0072	mpi_irecv_ [13]
0.0070	mpi_isend_ [4]
0.0065	mpi_irecv_ [4]
0.0048	mpi_irecv_ [7]
0.0031	mpi_waitall_ [2]
0.0029	mpi_reduce_(sync)
0.0025	mpi_waitall_ [5]
0.0001	mpi_reduce_
0.0000	mpi_waitall_ [8]
0.0000	mpi_irecv_ [18]
0.0000	mpi_isend_ [16]
0.0000	mpi_finalize_
0.0000	mpi_comm_rank
0.0000	mpi_init
0.0000	mpi_comm_size_

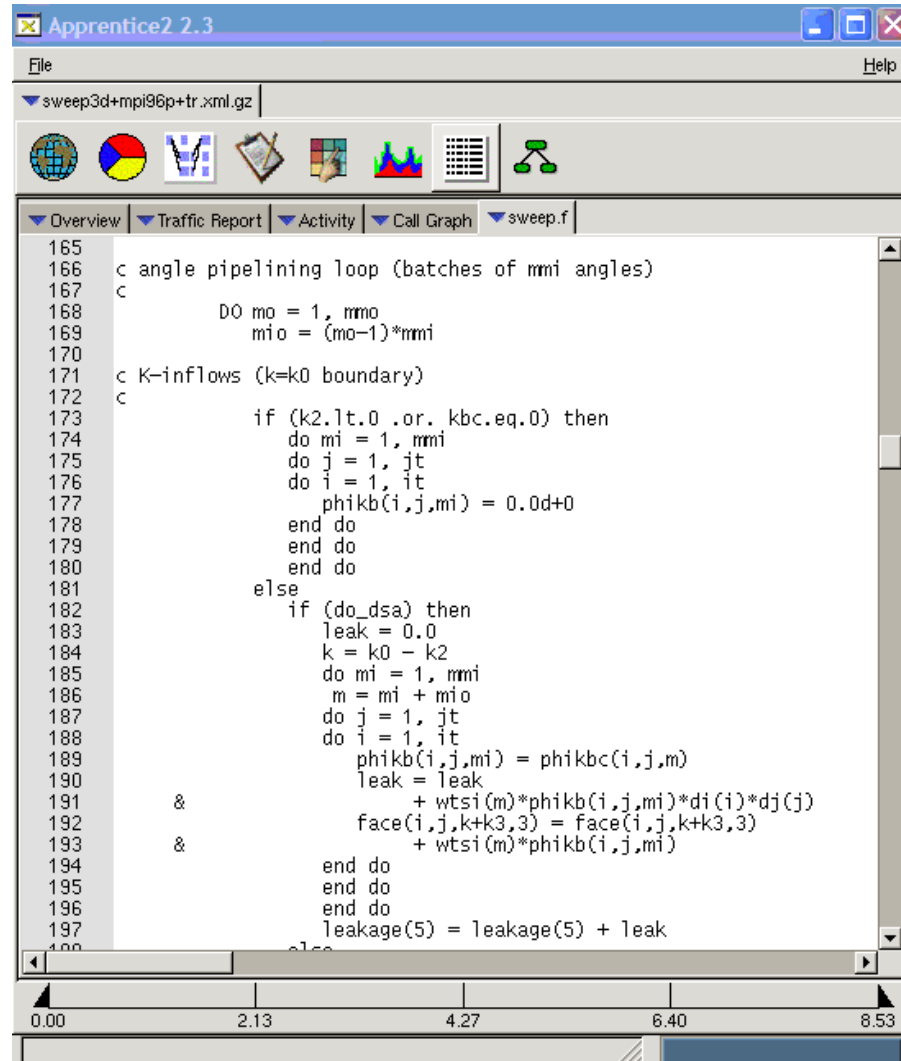
# Call Tree Visualization



# Discrete Unit of Help (DUH Button)



# Source Mapping from Call Graph view



```

Apprentice2 2.3
File Help
sweep3d+mpi96p+tr.xml.gz
Overview Traffic Report Activity Call Graph sweep.f
165
166 c angle pipelining loop (batches of mmi angles)
167 c
168     DO mo = 1, rmo
169     mio = (mo-1)*rmi
170
171 c K-inflows (k=k0 boundary)
172 c
173     if (k2.lt.0 .or. kbc.eq.0) then
174         do mi = 1, rmi
175         do j = 1, jt
176         do i = 1, it
177             phikb(i,j,mi) = 0.0d+0
178         end do
179         end do
180     end do
181     else
182         if (do_dsa) then
183             leak = 0.0
184             k = k0 - k2
185             do mi = 1, rmi
186             m = mi + mio
187             do j = 1, jt
188             do i = 1, it
189                 phikb(i,j,mi) = phikbc(i,j,m)
190                 leak = leak
191                 & + wtsi(m)*phikb(i,j,mi)*di(i)*dj(j)
192                 face(i,j,k+k3,3) = face(i,j,k+k3,3)
193                 & + wtsi(m)*phikb(i,j,mi)
194             end do
195             end do
196         end do
197         leakage(5) = leakage(5) + leak
198     end if
199
200

```

0.00 2.13 4.27 6.40 8.53



# 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



The screenshot shows the Cray Apprentice2 interface. At the top, there are menu bars for "File" and "Help". Below them are two window tabs: "About Apprentice2" and "swim+pat+10302-0tap2". A toolbar with various icons is visible. The main window is titled "Text" and contains the following text:

```
CrayPat/X: Version 5.2 Revision 7190 (xf 7034) 09/06/11 02:52:12
Number of PEs (MPI ranks): 16
Numbers of PEs per Node: 16
Numbers of Threads per PE: 1
Number of Cores per Socket: 12
Execution start time: Thu Apr 7 09:50:13 2011
System type and speed: x86_64 2000 MHz
Current path to data file: swim+pat+10302-0t.ap2

Notes for table 1:

Table option:
-O profile
Options implied by table option:
-d ti%@0.95,ti,imb_ti,imb_ti%,tr -b gr,fu,pe=HIDE,th=HIDE

The Total value for Time, Calls is the sum for the Group values.
The Group value for Time, Calls is the sum for the Function values.
The Function value for Time, Calls is the avg for the PE values.
The PE value for Time, Calls is the max for the Thread values.
(To specify different aggregations, see: pat_help report options s1)

This table shows only 1 line with Time = 0.00
```

At the bottom of the window, a progress bar shows values: 0.00, 49.30, 98.59, 147.89, 197.18. Below the progress bar, the text "swim+pat+10302-0t.ap2 (1.373s)" is displayed. At the very bottom, the words "COMPUTE | STORE | ANALYZE" are visible.

New text table icon

Right click for table generation options





# Generating New pat\_report Tables

- Profile
- Custom...

---

- Source
- Calltree
- Callers

---

- Show Notes
- Show All PE's
- Show HWPC
- Use Thresholds

---

- Select All
- Select None

---

- Panel Actions >

---

- Panel Help



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

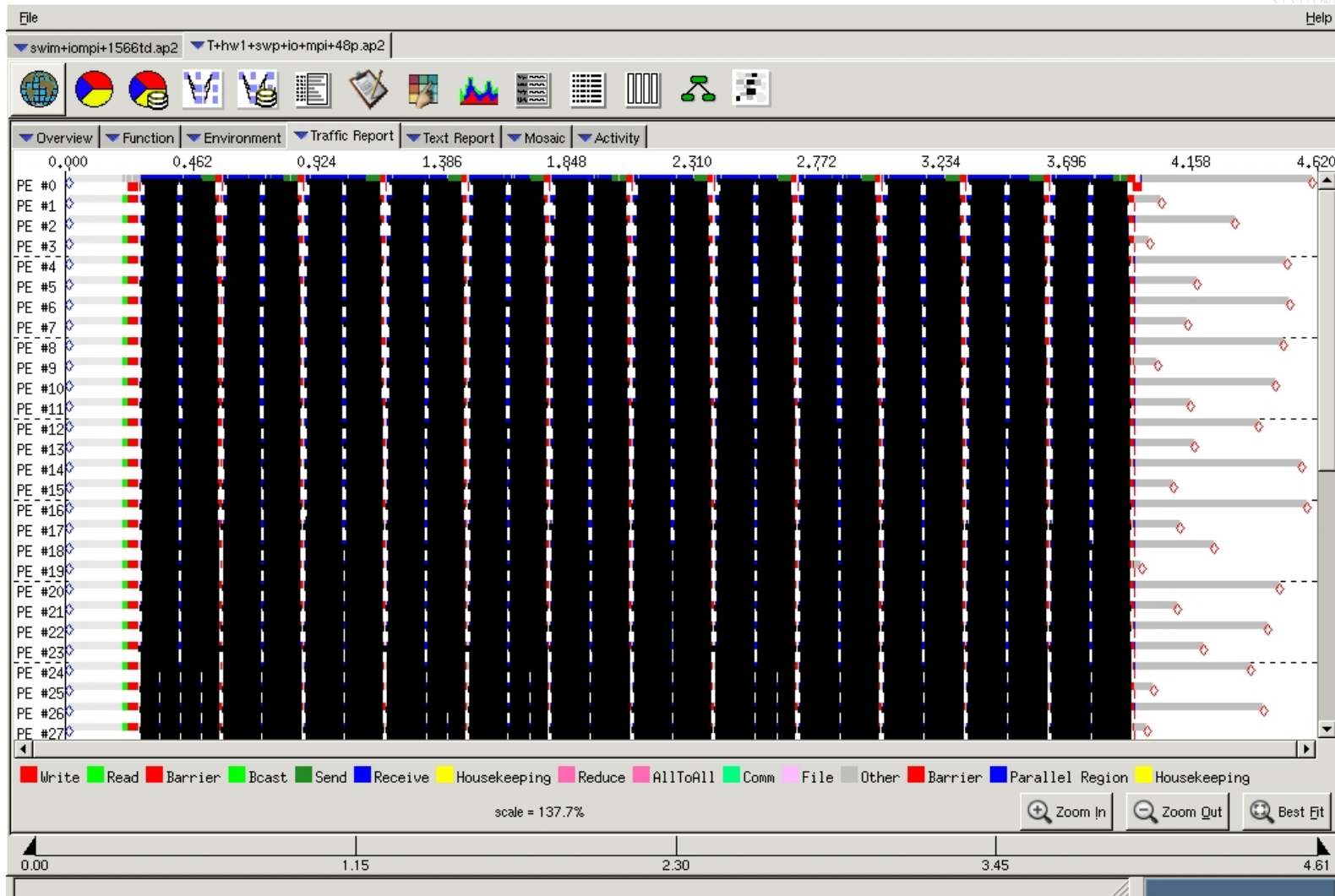
# Timeline views with Cray Apprentice<sup>2</sup>



# 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`**

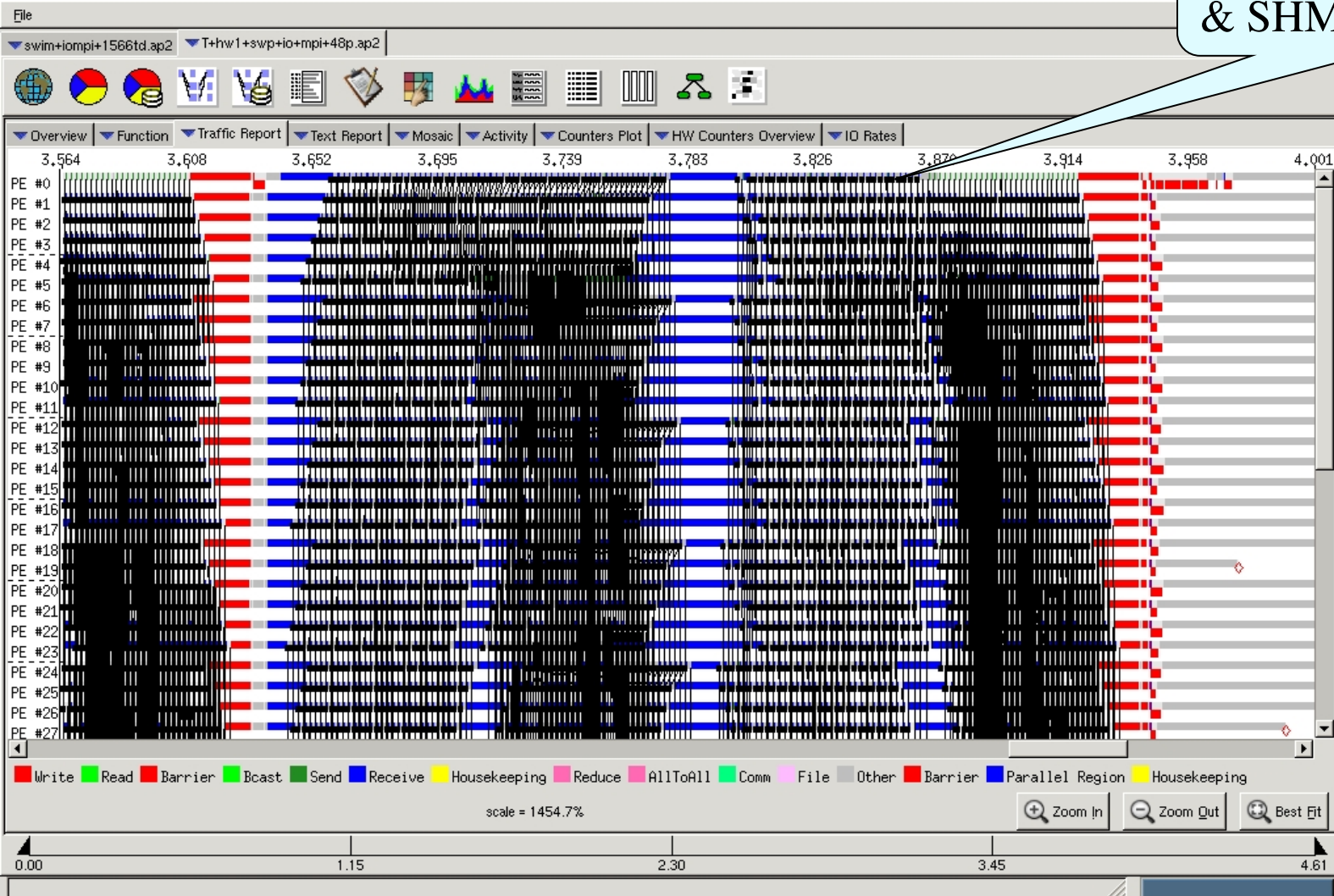
# Time Line View (Sweep3D)



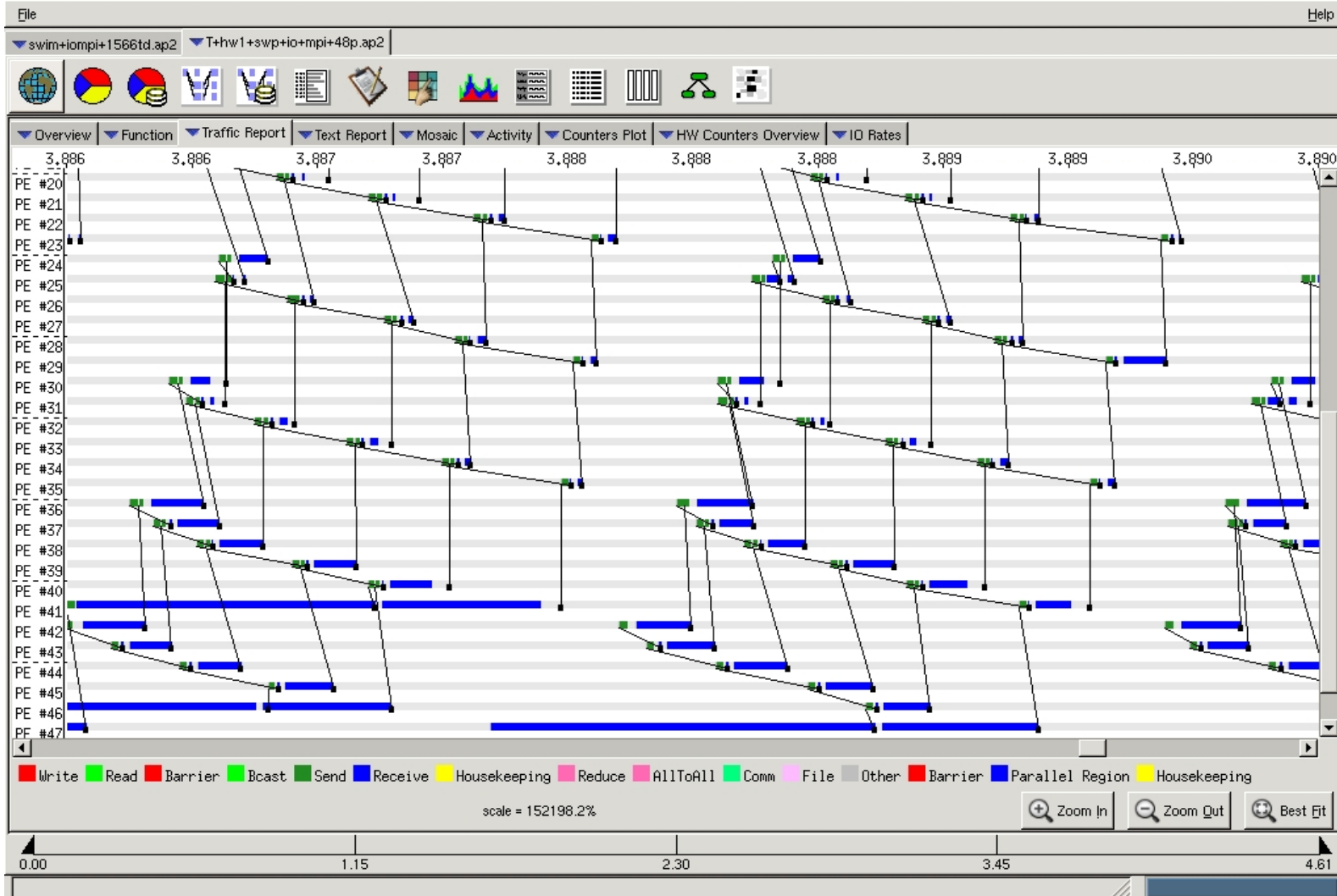
# Time Line View (Zoom)

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**

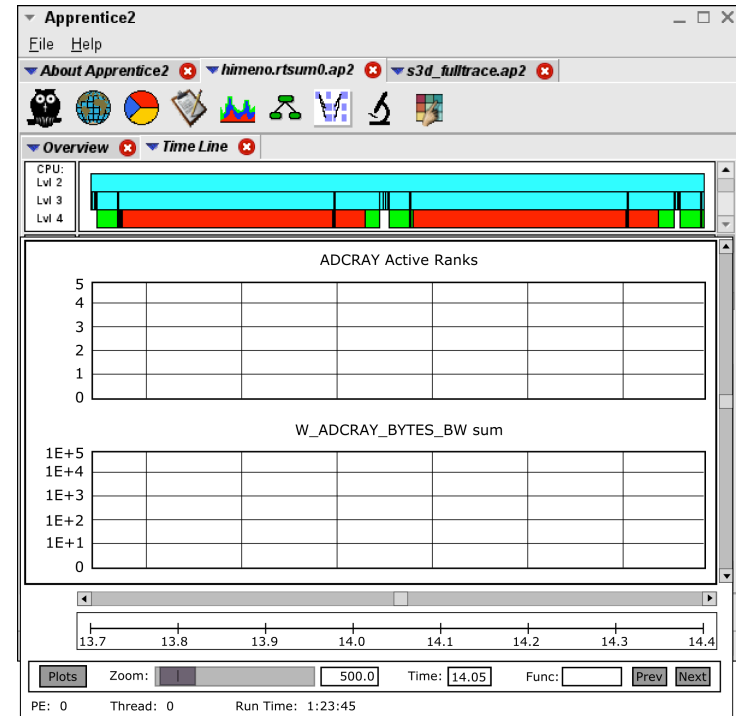
- Available only if hardware counter information was captured

- 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



**Select Plots and Scale**

<input checked="" type="checkbox"/>	Ranks	Linear	<input checked="" type="checkbox"/>	Log	<input type="checkbox"/>
<input checked="" type="checkbox"/>	BW Ag	Linear	<input checked="" type="checkbox"/>	Log	<input type="checkbox"/>
<input checked="" type="checkbox"/>	BW In	Linear	<input type="checkbox"/>	Log	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Plot 4	Linear	<input type="checkbox"/>	Log	<input type="checkbox"/>

PE  Thread

# 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

**Navigation**

- 39.71% parabola.f90
  - 33.52% PARABOLA
    - Loop@24
    - Loop@30
    - Loop@36
    - Loop@44
    - Loop@53
    - Loop@67
    - Loop@75
    - Loop@84
  - 6.19% PARASET
  - 11.92% riemann.f90
  - 11.21% remap.f90
  - 6.71% forces.f90
  - 6.39% volume.f90
  - 5.34% evolve.f90
  - 5.34% EVOLVE
    - Loop@25
    - Loop@36
    - Loop@58
    - Loop@70
  - 4.93% ppmlr.f90

**Source - /home/users/heid.../noLM/parabola.f90**

```

23 !-----
1687500 Vr4 24 do n = nmin-2, nmax+1
25     diffa(n) = a(n+1) - a(n)
26     enddo
27
28 !
29 !           Equation 1.7
1687500 Vr4 30 do n = nmin-1, nmax+1
31     da(n) = para(n,4) * diffa(n) + para(n,5) * diffa(n-1)
32     da(n) = sign( min(abs(da(n)), 2.0*abs(diffa(n-1))), 2.0*abs(diffa(n-1)))
33     enddo
34
1687500 Vr4 35 !           zero out da(n) if a(n) is a local max/min
36 do n = nmin-1, nmax+1
37     if(diffa(n-1)*diffa(n) < 0.0) da(n) = 0.0

```

**Info - Line 24**

- A loop starting at line 24 was unrolled 4 times.
- A loop starting at line 24 was vectorized.

**Performance feedback**

**Loopmark and optimization annotations**

**Compiler feedback**

vhone.aid loaded. vhone.ap2 loaded.

# Visualize CCE's Loopmark with Performance Profile (2)



The screenshot shows the Reveal 0.1 interface. The main window displays a code editor with a loop starting at line 32 marked with a 'U' icon. The code includes:

```
31 ! Put state variables into 1D arrays
32 do i = 1,imax
33   n = i + 6
34   r (n) = zro(i,j,k)
35   p (n) = zpr(i,j,k)
36   u (n) = zux(i,j,k)
37   v (n) = zuy(i,j,k)
38   w (n) = zuz(i,j,k)
39   f (n) = zfl(i,j,k)
40
41   xa0(n) = zxa(i)
42   dx0(n) = zdx(i)
43   xa (n) = zxa(i)
44   dx (n) = zdx(i)
45   p (n) = max(s...p,p(n))
46   e (n) = ...*gamm)+0.5*(u
47 enddo
```

An 'Info' window at the bottom left shows the message: "A loop starting at line 32 w". A yellow callout bubble points to this message with the text: "Integrated message 'explain support'".

An 'Explain' dialog box is open on the right, titled "OPT\_INFO: A loop starting at line %s was unrolled." It contains the following text:

The compiler unrolled the loop. Unrolling creates a number of copies of the loop body. When unrolling an outer loop, the compiler attempts to fuse replicated inner loops - a transformation known as unroll-and-jam. The compiler will always employ the unroll-and-jam mode when unrolling an outer loop; literal outer loop unrolling may occur when unrolling to satisfy a user directive (pragma).

This message indicates that unroll-and-jam was performed with respect to the identified loop. A different message is issued when literal outer loop unrolling is performed, as this transformation is far less likely to be beneficial.

For sake of illustration, the following contrasts unroll-and-jam with literal outer loop unrolling.

```
# 434 "/ptmp/pdgcs/pdgcs.tbs.81/bld.dir/build.64.ndb/pdgcs/pdgcs_ftn.msg.c"
DO J = 1,10
DO I = 1,100
A(I,J) = B(I,J) + 42.0
ENDDO
ENDDO

DO J = 1,10,2
DO I = 1,100
A(I,J) = B(I,J) + 42.0 ! unroll-and-jam
A(I,J+1) = B(I,J+1) + 42.0
ENDDO
ENDDO

DO J = 1,10,2
DO I = 1,100
A(I,J) = B(I,J) + 42.0 ! literal outer unroll
ENDDO
DO I = 1,100
A(I,J+1) = B(I,J+1) + 42.0
ENDDO
ENDDO
```

The dialog box also includes a detailed explanation of the difference between unroll-and-jam and literal outer unrolling, and buttons for "Explain other message..." and "Close".

# View Pseudo Code for Inlined Functions

The screenshot shows the Reveal 0.1 IDE interface. On the left, a file browser shows a tree view of files, with `init.f90` selected and expanded to show sub-files like `GRID` and `INIT`. The main editor window displays the source code for `init.f90`. Line 88, `call grid(imax, xmin, xmax, zxa, zxc, zdx)`, is highlighted in blue and marked with an information icon. A yellow callout bubble points to this line with the text "Inlined call sites marked". Below the code, an "Info" panel lists optimization details for line 88, including: "A divide was turned into a multiply by a reciprocal", "A loop starting at line 88 was unrolled 4 times.", "A loop starting at line 88 was vectorized.", and "The call to grid was textually inlined." A second yellow callout bubble points to the expanded pseudo code for the `grid` function, with the text "Expand to see pseudo code". The pseudo code shows the initialization of variables `t$26`, `t$27`, and `$I_L88_100`, followed by a loop that updates `zxa`, `zdx`, and `zxc` based on `$I_L88_100`, and finally exits the loop when `$I_L88_100` reaches 100. The status bar at the bottom indicates "vhone.aid loaded".

# Scoping Assistance – Review Scoping Results

**Navigation**

- Full List
- qicon.f90
- sweepy.f90**
  - SWEEPY
  - Loop@32
  - Loop@33
  - Loop@37
  - Loop@38
  - Loop@49
  - Loop@63
  - Loop@77
  - sweepx2.f90
  - SWEEPX2
  - Loop@28
  - Loop@29
  - Loop@32
  - Loop@33
  - Loop@44
  - Loop@58
  - sweepx1.f90
  - volume.f90
  - states.f90
  - riemann.f90

**Source - /home/users/heidi/demoLM/sweepy.f90**

```

1 subroutine sweepy
2
3 ! This subroutine
4 ! After call to
5 ! If only two d
6 ! After hydro u
7 ! -----
8
9 ! GLOBALS
10 use global
11 use zone
12 use sweeps
13 use mpi
14
15 IMPLICIT NONE
16

```

**ParallelMP Scope Selector**

Name	Type	Scope	Info
f	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit:...
flat	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit:...
q	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit:...
isy	Scalar	Shared	
js	Scalar	Shared	
ks	Scalar	Shared	
ngeomx	Scalar	Shared	
nleftx	Scalar	Shared	
npey	Scalar	Shared	
nrightx	Scalar	Shared	
recv2	Array	Shared	
zdx	Array	Shared	
zfl	Array	Shared	
zpr	Array	Shared	
zro	Array	Shared	
zux	Array	Shared	
zuy	Array	Shared	
zuz	Array	Shared	

**Info**

loading /home/users/heidi/demoLM/vhone.aid/vhone\_22.T...

Parallelization inhibitor messages are provided to assist user with analysis

Loops with scoping information are highlighted – red needs user assistance

User addresses parallelization issues for unresolved variables



# Scoping Assistance – User Resolves Issues

The screenshot shows the Reveal IDE interface. On the left, a sidebar displays a list of OpenMP tips, with the following text highlighted:

**Reduction in an inlined function**

**Scoping conflict with inlined variable**

**Scoping conflict with locally visible array**

An array requires conflicting scopes at different locations. It may be possible to declare and use a different array for the private array uses.

A yellow callout bubble points to this sidebar with the text: "Use Reveal's OpenMP parallelization tips".

The main editor window shows a code snippet from `/sweepx2.f90`:

```

Loop over each row.
do m = 1, ks
  do i = 1, js
    ! Put state variables i
    do n = 1, npey
      r8 = r8 + ...
    enddo
  enddo
enddo

```

Below the code, an "Info" window displays the following message:

**Info - Line 28**

A loop starting at line 28 was not vectorized because it contains a call to subroutine "ppmlr" on line 55. Loop has been flattened. Loop has been flattened.

On the right, the "OpenMP Scope Selector" window is open, showing a table of variables and their scopes:

Name	Type	Scope	Info
f	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit...
flat	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit...
q	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit...
isy	Scalar	Shared	
js	Scalar	Shared	
ks	Scalar	Shared	
ngeomx	Scalar	Shared	
nletx	Scalar	Shared	
npey	Scalar	Shared	
nrightx	Scalar	Shared	
recv2	Array	Shared	
zdx	Array	Shared	
zll	Array	Shared	
zpr	Array	Shared	
zro	Array	Shared	
zux	Array	Shared	
zuy	Array	Shared	
zuz	Array	Shared	

A yellow callout bubble points to the `recv2` variable in the table with the text: "Click on variable to view all occurrences in loop".

At the bottom of the IDE, a status bar shows the file path: `loading /home/users/heidi/demoLM/vhone.aid/vhone_22.T...`

# Scoping Assistance – Generate Directive

Reveal generates example OpenMP directive

The screenshot shows the Reveal IDE interface with several windows open:

- Source - /home/users/heidi/demoLM/sweep1.f90**: Displays Fortran code with line numbers 28-43. Line 29 is highlighted in black, and lines 30-43 are highlighted in green.
- OpenMP Directive**: Shows generated OpenMP directives:
 

```

      !$OMP parallel do default(none) &
      !$OMP shared (gamm,send1,zdx,zff,zpr,zro,zux,zuy,zuz,zxa) &
      !$OMP lastprivate (dx,dx0,e,f,p,r,u,v,w,xa,xa0)
      
```
- OpenMP Scope Selector**: A dialog box for 'sweep1.f90: lines 29 -> 63'. It contains a table of variables and their scoping options:
 

Name	Type	Scope	Info
dx	Array	Private	WARN: LastPrivate of array may be very expensive.
dx0	Array	Private	WARN: LastPrivate of array may be very expensive.
e	Array	Private	WARN: LastPrivate of array may be very expensive.
f	Array	Private	WARN: LastPrivate of array may be very expensive.
p	Array	Private	WARN: LastPrivate of array may be very expensive.
r	Array	Private	WARN: LastPrivate of array may be very expensive.
u	Array	Private	WARN: LastPrivate of array may be very expensive.
v	Array	Private	WARN: LastPrivate of array may be very expensive.
w	Array	Private	WARN: LastPrivate of array may be very expensive.

 Below the table are options for 'First/Last Private' (Enable First Private, Enable Last Private) and a 'Resolution' dropdown set to 'None'.
- Info - Line 29**: Provides a warning:
 

```

      ■ A loop starting at line 29 was not vectorized because it contains a call to subroutine "ppmlr" on line 50.
      Loop has been flattened.
      Loop has been flattened.
      
```
- Navigation**: A sidebar showing a tree view of code elements like 'sweepx2.f90', 'SWEEPX2', 'Loop@28', etc., with 'Loop@29' selected.

# Automatic parallelization 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

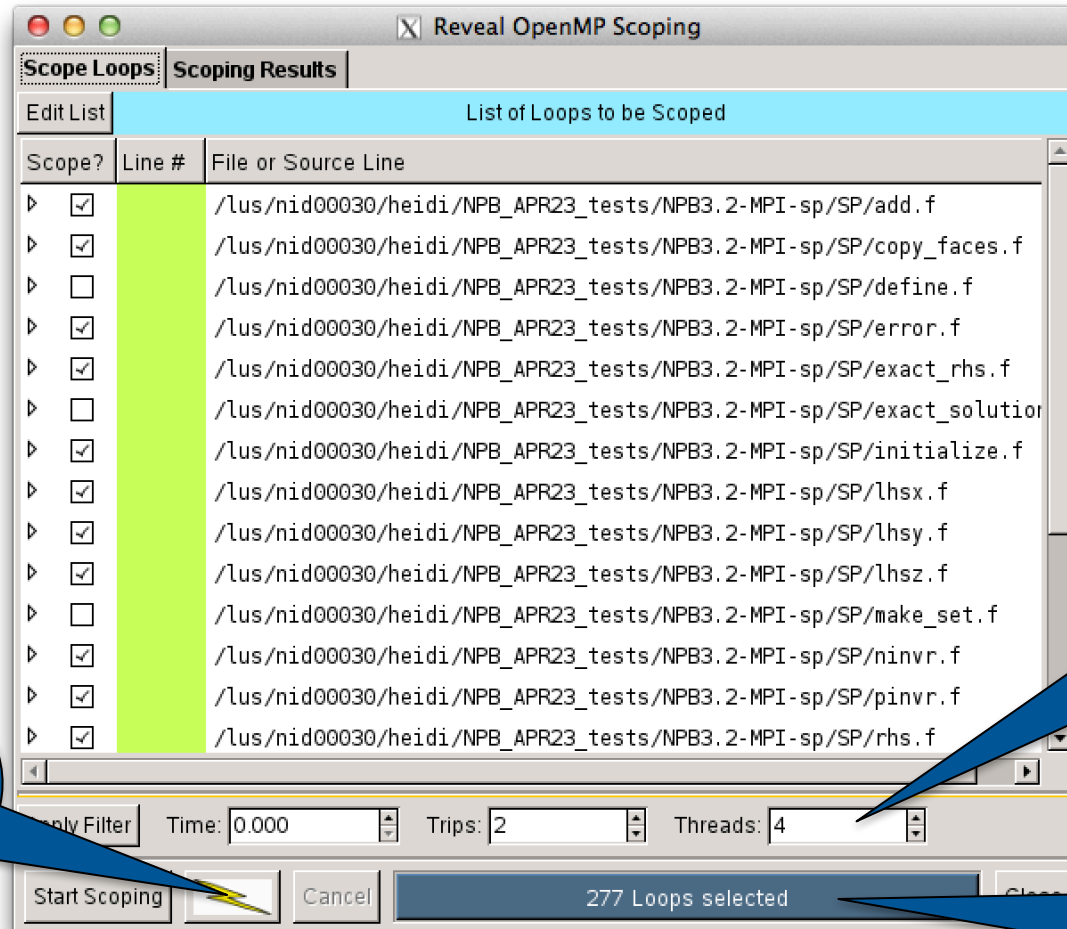
## Auto parallelize important loops

- 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



**Scope Loops** | **Scoping Results**

Edit List | List of Loops to be Scoped

Scope?	Line #	File or Source Line
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/add.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/copy_faces.f
<input type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/define.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/error.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/exact_rhs.f
<input type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/exact_solution
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/initialize.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/lhsx.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/lhsy.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/lhsz.f
<input type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/make_set.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/ninvr.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/pinvr.f
<input checked="" type="checkbox"/>		/lus/nid00030/heidi/NPB_APPR23_tests/NPB3.2-MPI-sp/SP/rhs.f

Apply Filter | Time: 0.000 | Trips: 2 | Threads: 4

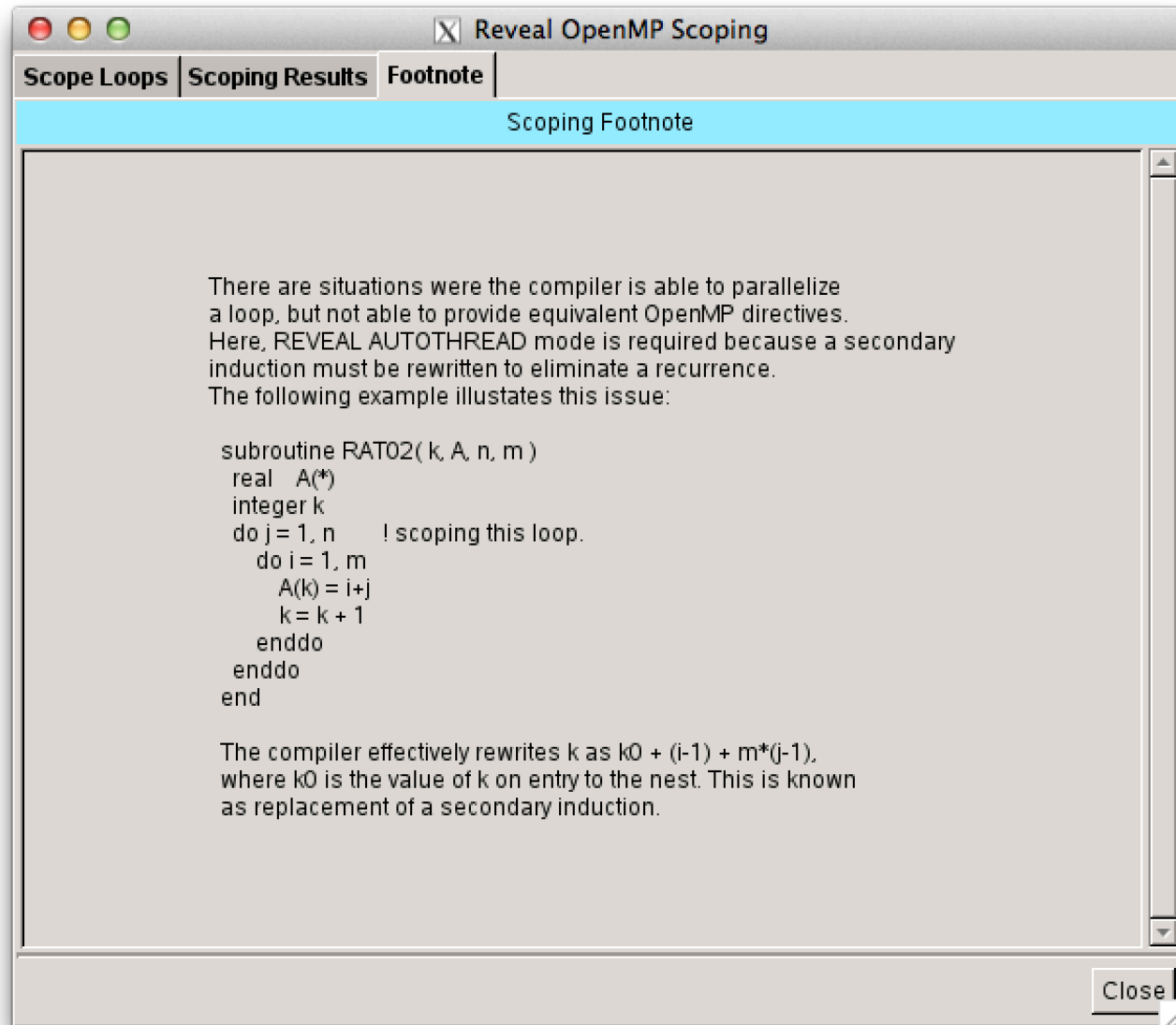
Start Scoping | Cancel | 277 Loops selected

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

Thread Count to Use at Runtime

277 Loop Candidates for Parallelization

# Reveal: Example of Parallelization Hints



Reveal OpenMP Scoping

Scope Loops | Scoping Results | Footnote

Scoping Footnote

There are situations where 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 illustrates this issue:

```

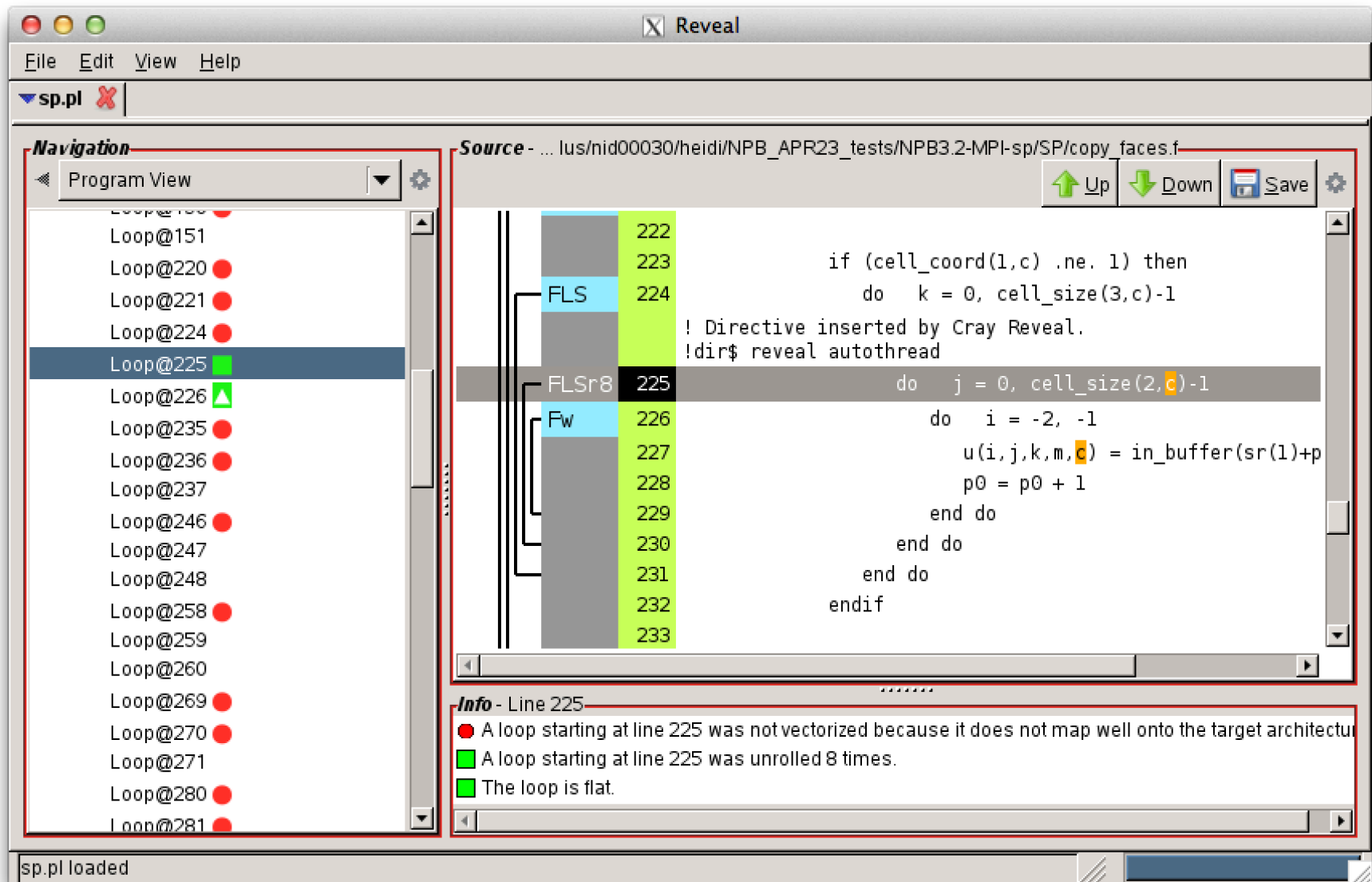
subroutine RAT02( k, A, n, m )
  real A(*)
  integer k
  do j = 1, n    ! scoping this loop.
    do i = 1, m
      A(k) = i+j
      k = k + 1
    enddo
  enddo
end

```

The compiler effectively rewrites  $k$  as  $k_0 + (i-1) + m*(j-1)$ , where  $k_0$  is the value of  $k$  on entry to the nest. This is known as replacement of a secondary induction.

Close

# Reveal: New Autothread Directive



The screenshot shows the Cray Reveal IDE interface. On the left is a 'Navigation' pane with a 'Program View' showing a list of loops from Loop@151 to Loop@281. Loop@225 is highlighted in blue. The main window displays the source code for 'copy\_faces.f'. A vertical bar on the left of the code indicates the execution flow, with labels 'FLS', 'FLSr8', and 'Fw' corresponding to lines 224, 225, and 226 respectively. Line 225 is highlighted in black, indicating it is the current line of focus. The code includes an autothread directive: `!dir$ reveal autothread`. Below the code is an 'Info' pane with the following text:

- A loop starting at line 225 was not vectorized because it does not map well onto the target architecture.
- A loop starting at line 225 was unrolled 8 times.
- The loop is flat.

The status bar at the bottom left shows 'sp.pl loaded'.





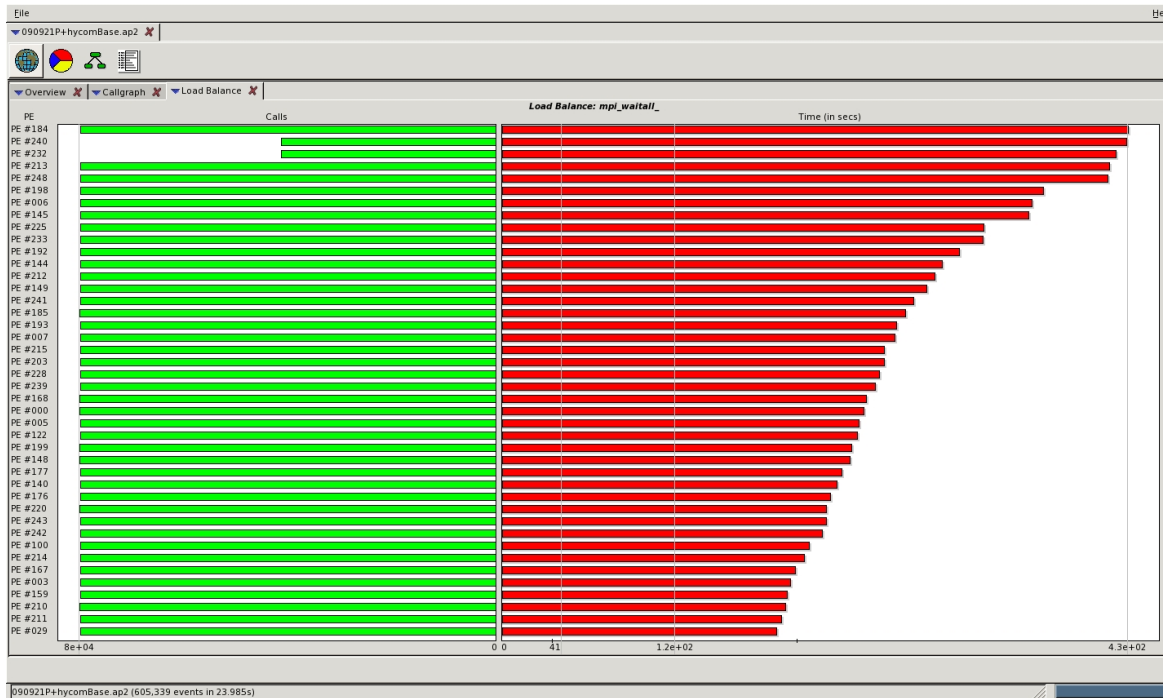
# Reveal: New Autothread Directive

- **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

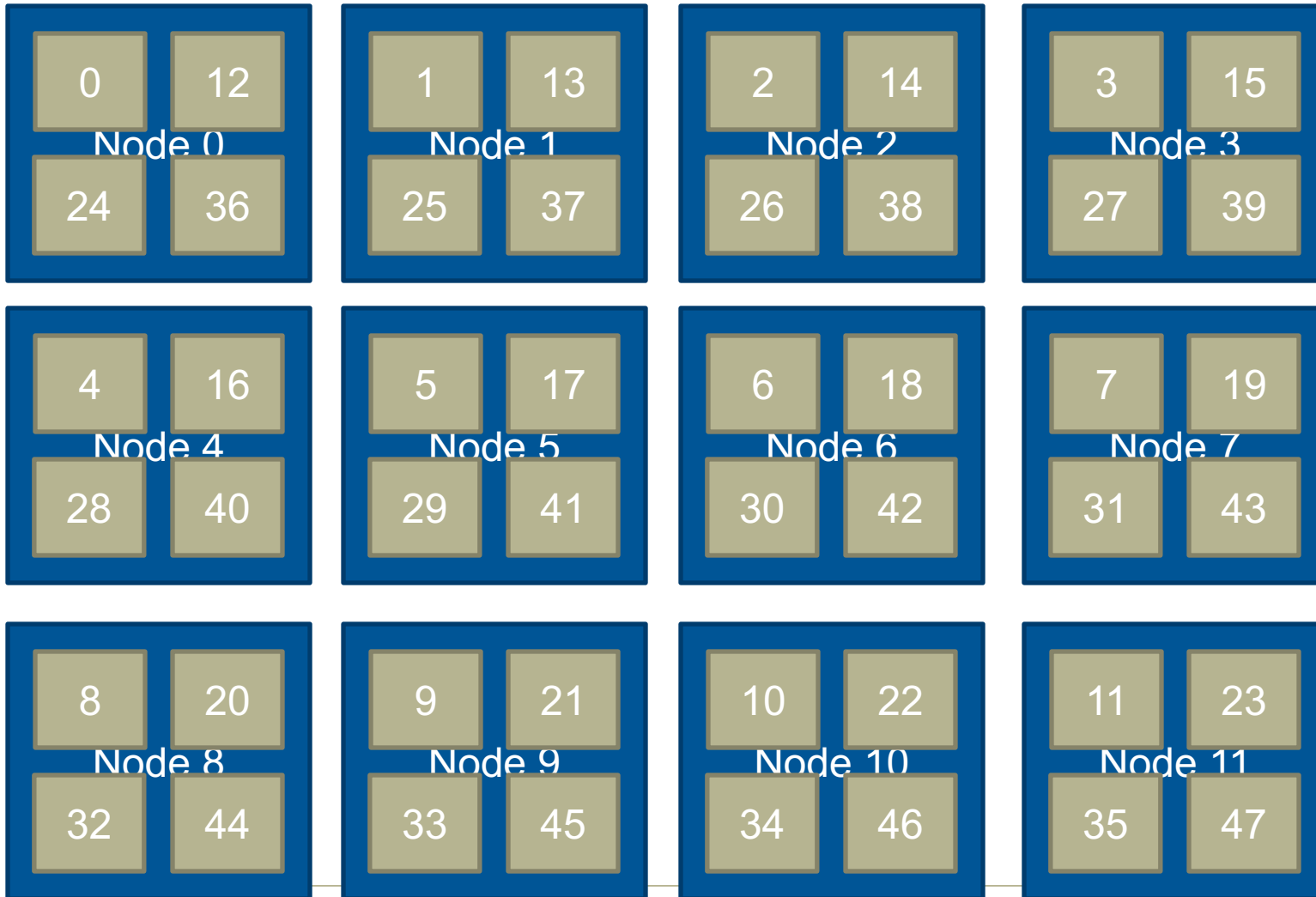


- The MPI library can reorder the ranks at runtime based on the setting of `MPICH_RANK_REORDER_METHOD`

# Rank Placement

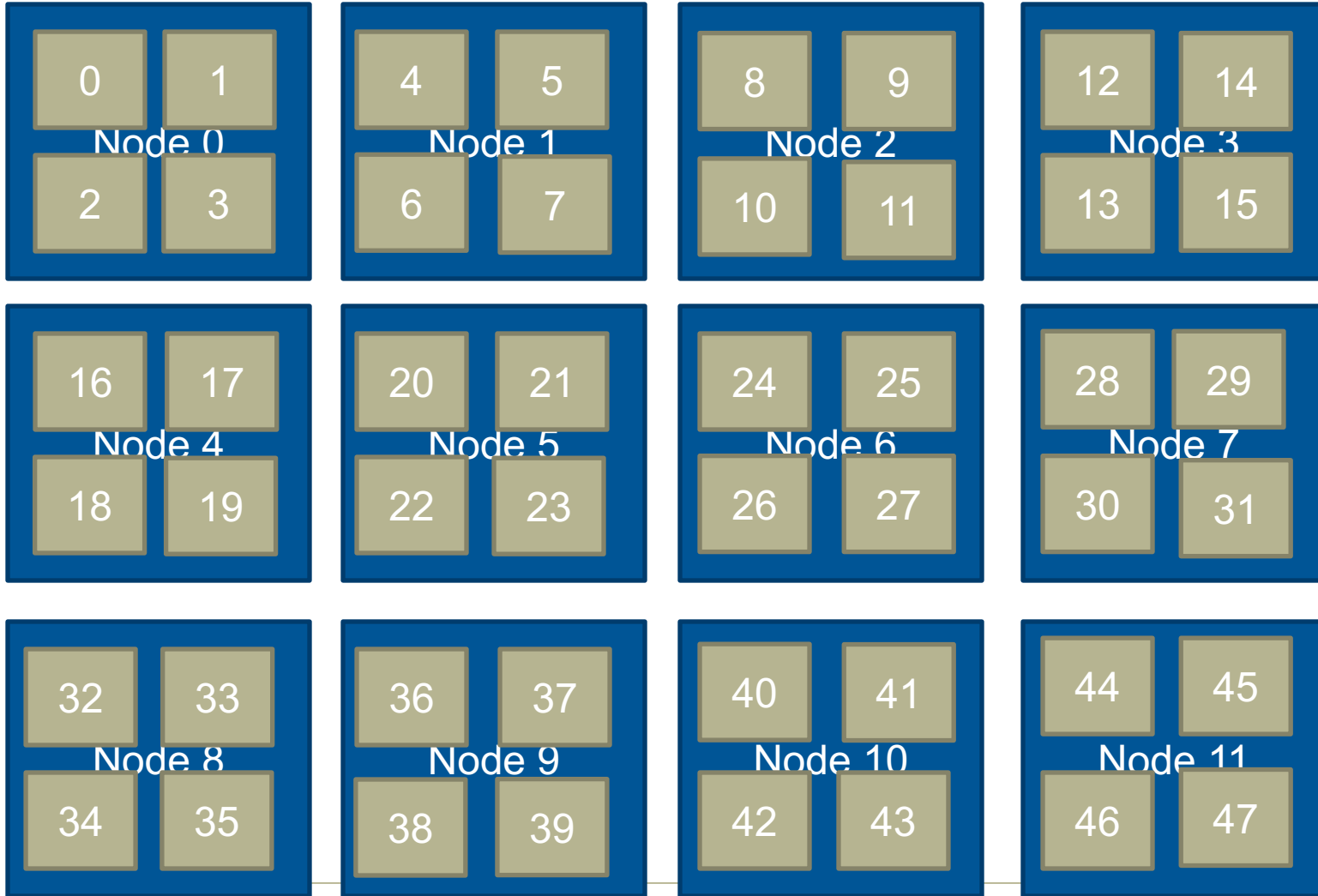
- Start with a list of nodes to run on
- **0: Round-robin placement**
  - Sequential ranks are allocated one per node in sequence
  - Placement starts again on first node if we reach the last node
- **1: SMP-style placement (default)**
  - Sequential ranks fill up each node in turn
  - Only then move 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



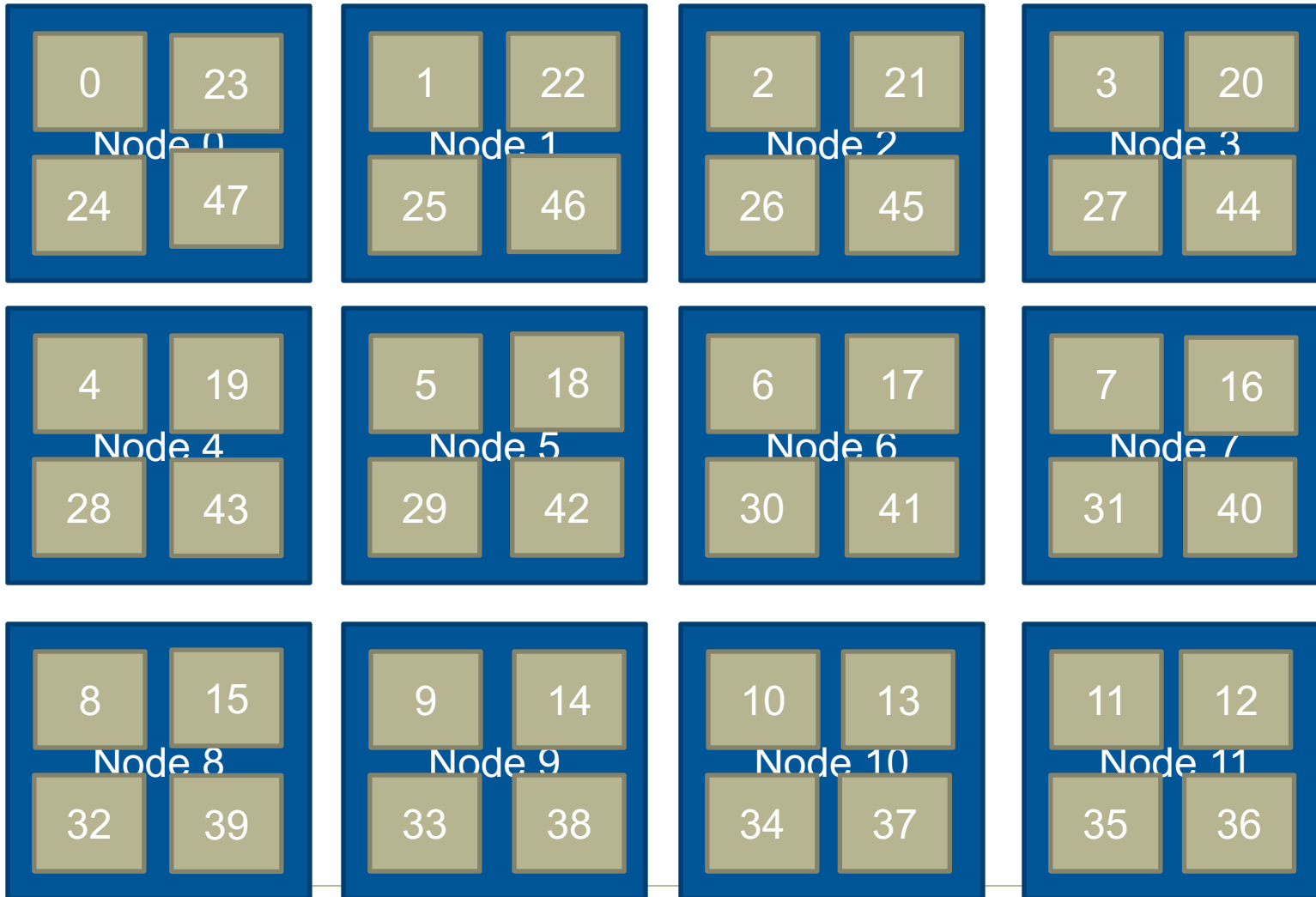
COMPUTE | STORE | ANALYZE

# 1: SMP Placement (default)



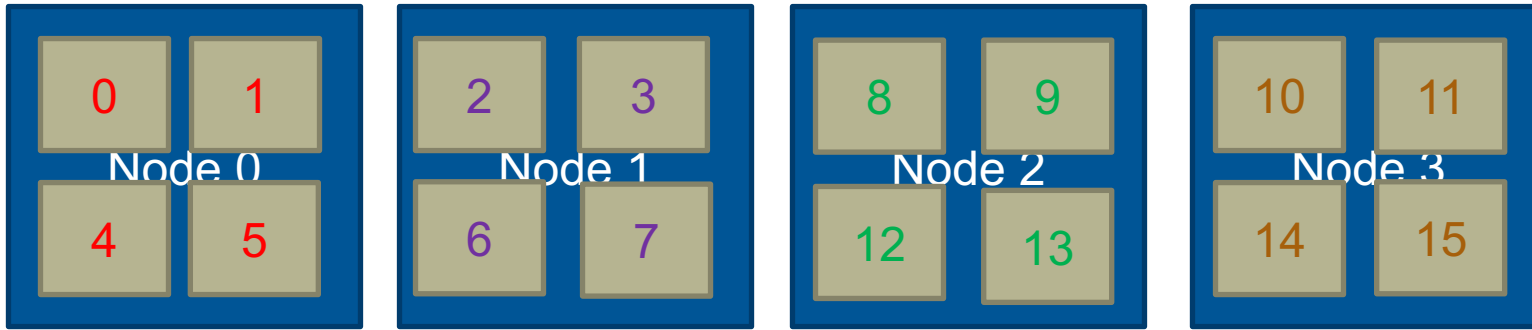
COMPUTE | STORE | ANALYZE

# 2: Folded Placement



COMPUTE | STORE | ANALYZE

### 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 Order	On-Node Bytes/PE	On-Node Bytes/PE% of Total Bytes/PE	MPICH_RANK_REORDER_METHOD
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 Order	Maximum Value	Average Value	Max:Ave Ratio	Reduction in Max
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)**