

# Reveal

Heidi Poxon  
Cray Inc.

# Porting to a Hybrid or Many-core System

# A Porting and Optimization Strategy for Hybrid and Many-core Systems

- Maximize on-node communication between MPI ranks
- Relieve on-node shared resource contention by pairing threads or processes that perform different work (for example computation with off-node communication) on the same node
- Add parallelism to MPI ranks to take advantage of cores within a node while minimizing network injection contention
- Accelerate work intensive parallel loops



# Approach to Adding Parallelism

## 1. Identify possible accelerator kernels

- Determine where to add additional levels of parallelism
  - Assumes MPI application is functioning correctly on X86
  - Find top serial work-intensive loops (perftools + CCE loop work estimates)

## 2. Perform parallel analysis, scoping and vectorization

- Split loop work among threads
  - Do parallel analysis and restructuring on targeted high level loops
  - Use CCE loopmark feedback, Reveal loopmark and source browsing

## 3. Move to OpenMP and then to OpenACC

- Add parallel directives and acceleration extensions
  - Insert OpenMP directives (Reveal scoping assistance)
  - Run on X86 to verify application and check for performance improvements
  - Convert desired OpenMP directives to OpenACC

## 4. Analyze performance from optimizations

# Step 1 - Identify possible accelerator kernels

# Loop Work Estimates

- **Helps identify high-level serial loops to parallelize**
  - Based on runtime analysis, approximates how much work exists within a loop
  - Provides min, max and average trip counts that can be used to approximate work and help carve up loop on GPU

# Collecting Loop Statistics

- Load PrgEnv-cray module
- Load perftools module
- Compile **AND** link with `-h profile_generate`
- Instrument binary for tracing
  - `pat_build -w my_program`
- Run application
- Create report with loop statistics
  - `pat_report my_program.xf > loops_report`



# Example Report – Inclusive Loop Time

**Table 2:** Loop Stats by Function (from `-hprofile_generate`)

Loop Incl Time Total	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
8.995914	100	25	0	25	sweepy_.LOOP.1.li.33
8.995604	2500	25	0	25	sweepy_.LOOP.2.li.34
8.894750	50	25	0	25	sweepz_.LOOP.05.li.49
8.894637	1250	25	0	25	sweepz_.LOOP.06.li.50
4.420629	50	25	0	25	sweepx2_.LOOP.1.li.29
4.420536	1250	25	0	25	sweepx2_.LOOP.2.li.30
4.387534	50	25	0	25	sweepx1_.LOOP.1.li.29
4.387457	1250	25	0	25	sweepx1_.LOOP.2.li.30
2.523214	187500	107	0	107	riemann_.LOOP.2.li.63
1.541299	20062500	12	0	12	riemann_.LOOP.3.li.64
0.863656	1687500	104	0	108	parabola_.LOOP.6.li.67



**Step 2 - Perform parallel  
analysis, scoping and  
vectorization**

**&**

**Step 3 - Move to  
OpenMP and then to  
OpenACC**





# Reveal

## New code analysis and restructuring assistant...

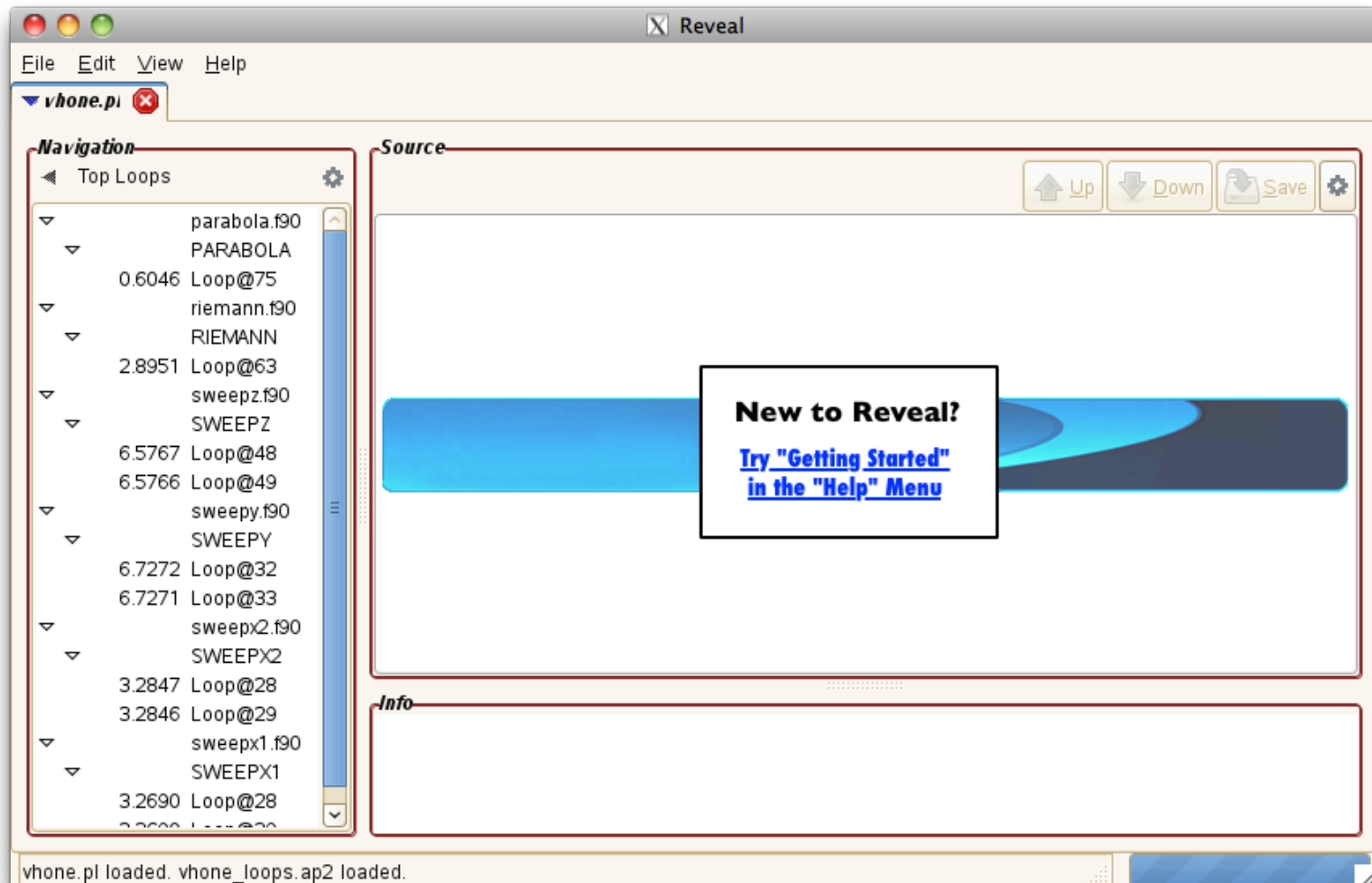
- Uses both the performance toolset and CCE's program library functionality to provide static and runtime analysis information
- **Key Features**
  - **Annotated source code** with compiler optimization information
    - Feedback on critical dependencies that prevent optimizations
  - **Scoping analysis**
    - Identify, shared, private and ambiguous arrays
      - Allow user to privatize ambiguous arrays
      - Allow user to override dependency analysis
  - **Source code navigation** based on performance data collected through CrayPat



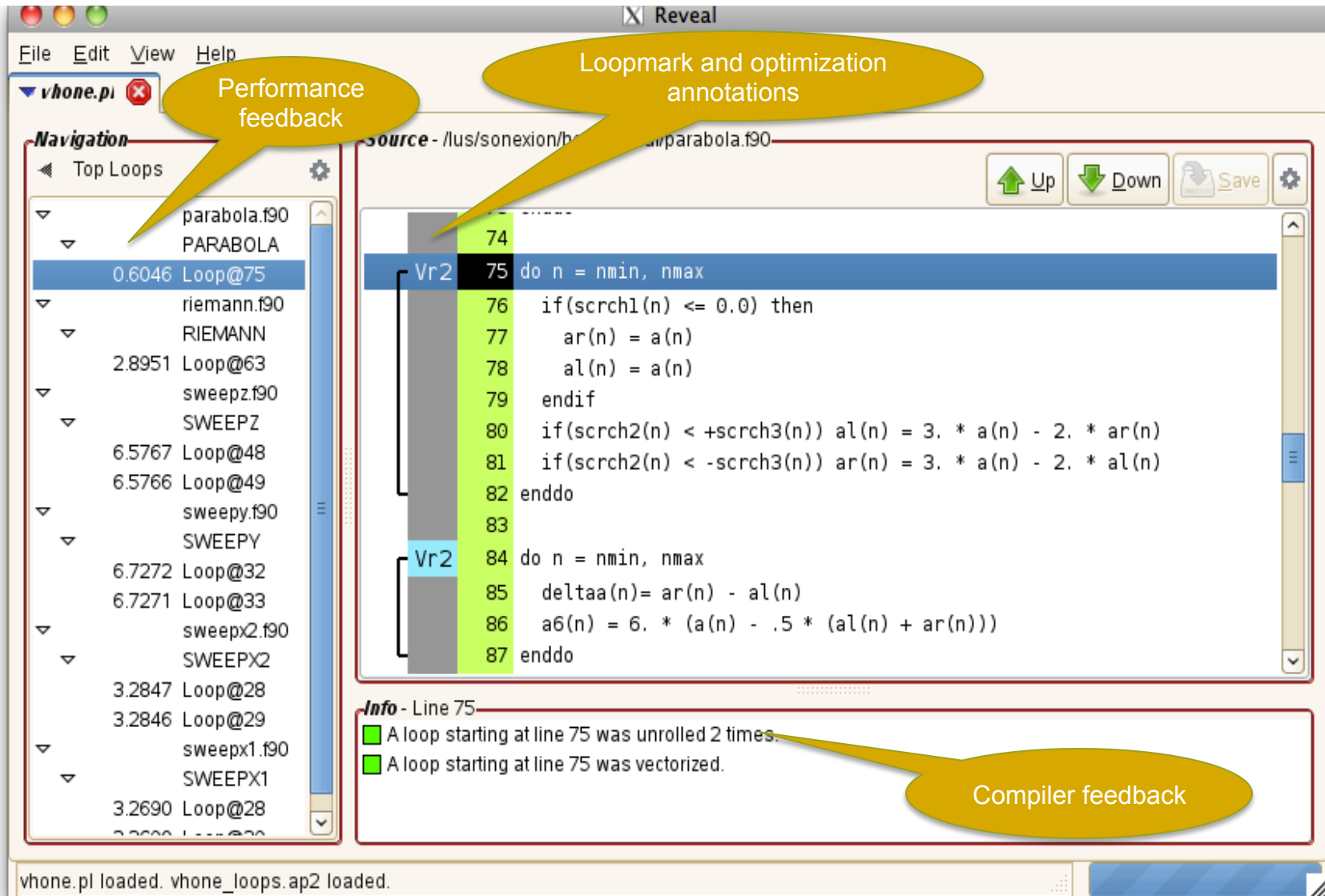
# How to Use

- **Optionally create loop statistics using Cray performance tools to determine which loops have the most work**
  - >
- **Compile your application with Cray CCE to generate a program library**
  - > `ftn -h pl=vhone.pl -c file1.f90`
- **Run reveal**
  - Compiler information only:
    - > `reveal vhone.pl`
  - Compiler + loop work estimates
    - > `reveal vhone.pl vhone_loops.ap2`

# Reveal with Loop Work Estimates



# Visualize Loopmark with Performance Information



The screenshot shows the Reveal IDE interface with the following components:

- Navigation Panel (Left):** Lists various loops and their performance metrics. A yellow callout labeled "Performance feedback" points to the "0.6046 Loop@75" entry.
- Source Editor (Center):** Displays the source code for "parabola.f90". A yellow callout labeled "Loopmark and optimization annotations" points to the loop starting at line 75, which is marked with a blue bar and the label "Vr2".
- Info Panel (Bottom):** Provides compiler feedback. A yellow callout labeled "Compiler feedback" points to the information: "A loop starting at line 75 was unrolled 2 times." and "A loop starting at line 75 was vectorized."

**Navigation Panel Data:**

Loop Name	Performance Metric
parabola.f90	
PARABOLA	
0.6046 Loop@75	
riemann.f90	
RIEMANN	
2.8951 Loop@63	
sweepz.f90	
SWEEPZ	
6.5767 Loop@48	
6.5766 Loop@49	
sweepy.f90	
SWEEPY	
6.7272 Loop@32	
6.7271 Loop@33	
sweepx2.f90	
SWEEPX2	
3.2847 Loop@28	
3.2846 Loop@29	
sweepx1.f90	
SWEEPX1	
3.2690 Loop@28	

**Source Code (parabola.f90):**

```

74
75 do n = nmin, nmax
76   if(scrch1(n) <= 0.0) then
77     ar(n) = a(n)
78     al(n) = a(n)
79   endif
80   if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
81   if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
82 enddo
83
84 do n = nmin, nmax
85   deltaa(n)= ar(n) - al(n)
86   a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
87 enddo
  
```

**Info - Line 75:**

- A loop starting at line 75 was unrolled 2 times.
- A loop starting at line 75 was vectorized.

vhone.pl loaded. vhone\_loops.ap2 loaded.

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



The screenshot shows the 'Reveal' window with the 'vhone.pl' file loaded. The 'Navigation' pane on the left shows a tree view of the file, with '0.53% SWEEPX2' expanded. The 'Source' pane shows the code for 'sweepx2.f90', with lines 32-45 highlighted. The 'Info' pane at the bottom shows a message about line 33: 'A loop starting at line 33 was not vectorized because it does not have a constant stride' (indicated by a red dot) and 'A loop starting at line 33 was unrolled 8 times' (indicated by a green square). A yellow callout bubble points to the 'Info' pane with the text 'Integrated message "explain support"'. The 'Explain' window on the right shows the message 'OPT\_INFO: A loop starting at line %s was unrolled.' and provides a detailed explanation of the unroll-and-jam transformation, contrasting it with literal outer loop unrolling. The 'Explain' window also has buttons for 'Explain other message...' and 'Close'.

**Navigation**

- Program View
- riemann.f90
- remap.f90
- evolve.f90
- volume.f90
- forces.f90
- ppmlr.f90
- states.f90
- flatten.f90
- sweepz.f90
- sweepx.f90
- boundary.f90
- prin.f90
- sweepx2.f90
  - 0.53% SWEEPX2
    - Loop@28
    - Loop@29
    - Loop@32
    - Loop@33
    - Loop@44
    - Loop@58
  - sweepx1.f90

**Source - /lus/sonexion/heidi/reveal/sweepx2.f90**

```
32 do m = 1, npey
33 do i = 1, isy
34   n = i + isy*(m-1) + 6
35   r(n) = recv2(1,k,i,j,m)
36   p(n) = recv2(2,k,i,j,m)
37   u(n) = recv2(3,k,i,j,m)
38   v(n) = recv2(4,k,i,j,m)
39   w(n) = recv2(5,k,i,j,m)
40   f(n) = recv2(6,k,i,j,m)
41 enddo
42 enddo
43
44 do i = 1,imax
45   n = i + 6
```

**Info - Line 33**

- A loop starting at line 33 was not vectorized because it does not have a constant stride
- A loop starting at line 33 was unrolled 8 times

**OPT\_INFO: A loop starting at line %s was unrolled.**

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.

```
# 426 "/tmp/ulib/buildslaves/pdgc8-81-edition-build/tbs/build/release/pdgc8/pdgc8_ftn.msg.c"
DO J = 1,10
DO I = 1,100
A(I,J) = B(I,J) + 42.0
ENDDO
ENDDO

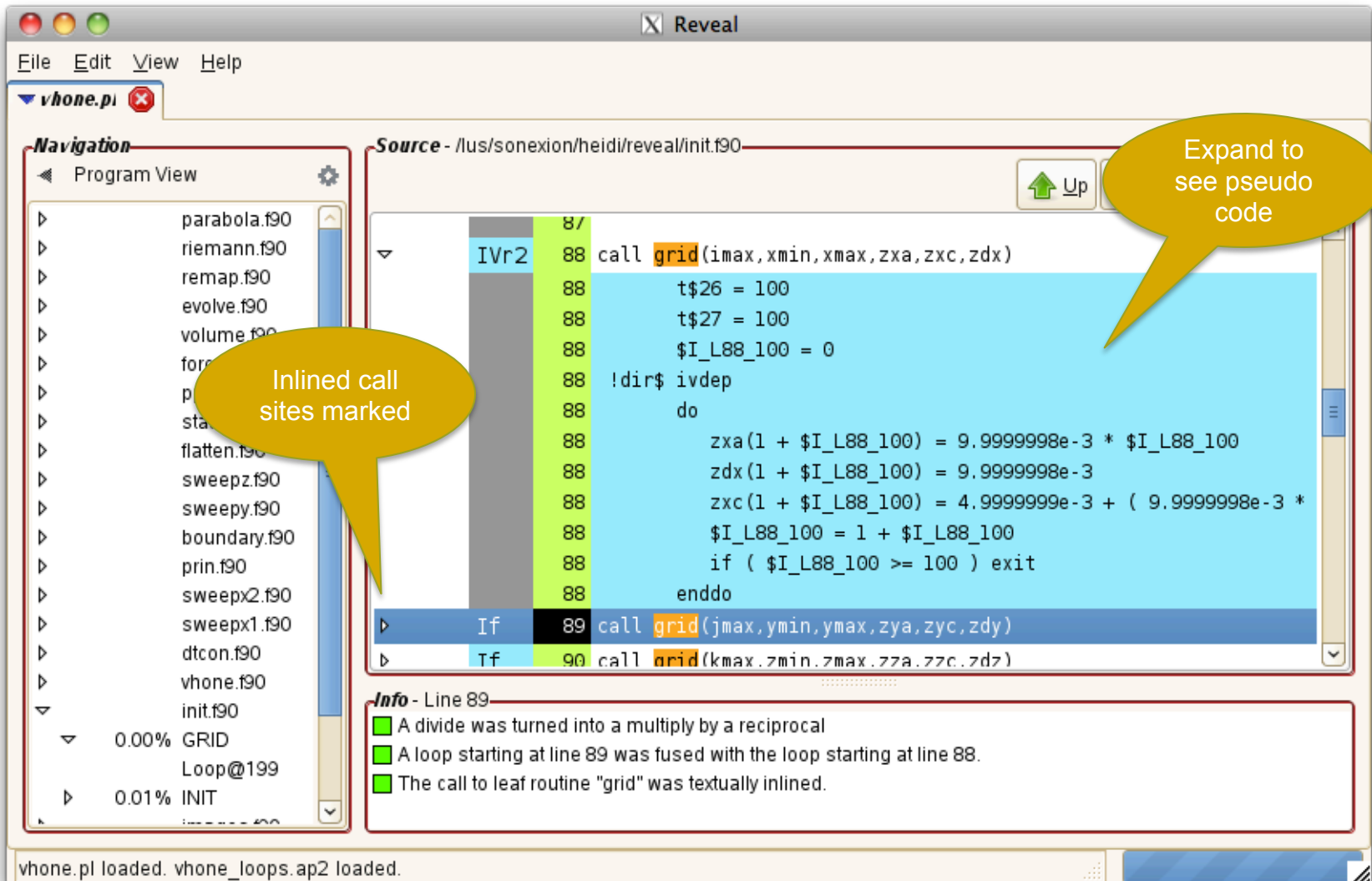
DO J = 1,10,2
DO I = 1,100
A(I,J) = B(I,J) + 42.0 ! unroll-and-jam
A(I,J+1) = B(I,J+1) + 42.0
ENDDO
ENDDO

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 literal outer unroll code performs the same sequence of memory operations as the original nest, while the unroll-and-jam transformation interleaves operations from outer loop iterations. The compiler employs literal outerloop unrolling only when the data dependencies in the loop, or a control flow impediment, prevent fusion of the replicated inner loops. Literal outer loop unrolling is generally not desirable. It is provided to ensure expected behavior and for those rare instances where the user has determined that it is beneficial.

Explain other message... Close

# View Pseudo Code for Inlined Functions



The screenshot shows the Cray Reveal IDE interface. The top menu bar includes File, Edit, View, and Help. The main window is titled "Source - /lus/sonexion/heidi/reveal/init.f90". The left sidebar shows a "Navigation" pane with a "Program View" tree listing various files like parabola.f90, riemann.f90, remap.f90, etc. The main editor area displays Fortran code with line numbers 87, 88, 89, and 90. Line 88 contains a call to the `grid` function. A yellow callout bubble points to the `grid` function call, stating "Expand to see pseudo code". Another yellow callout bubble points to the `grid` function call in line 89, stating "Inlined call sites marked". The bottom status bar shows "vhone.pl loaded. vhone\_loops.ap2 loaded."

**Navigation Pane:**

- parabola.f90
- riemann.f90
- remap.f90
- evolve.f90
- volume.f90
- for...
- p...
- sta...
- flatten.f90
- sweepz.f90
- sweepy.f90
- boundary.f90
- prin.f90
- sweepx2.f90
- sweepx1.f90
- dtcon.f90
- vhone.f90
- init.f90
- 0.00% GRID
- Loop@199
- 0.01% INIT

**Source Code:**

```

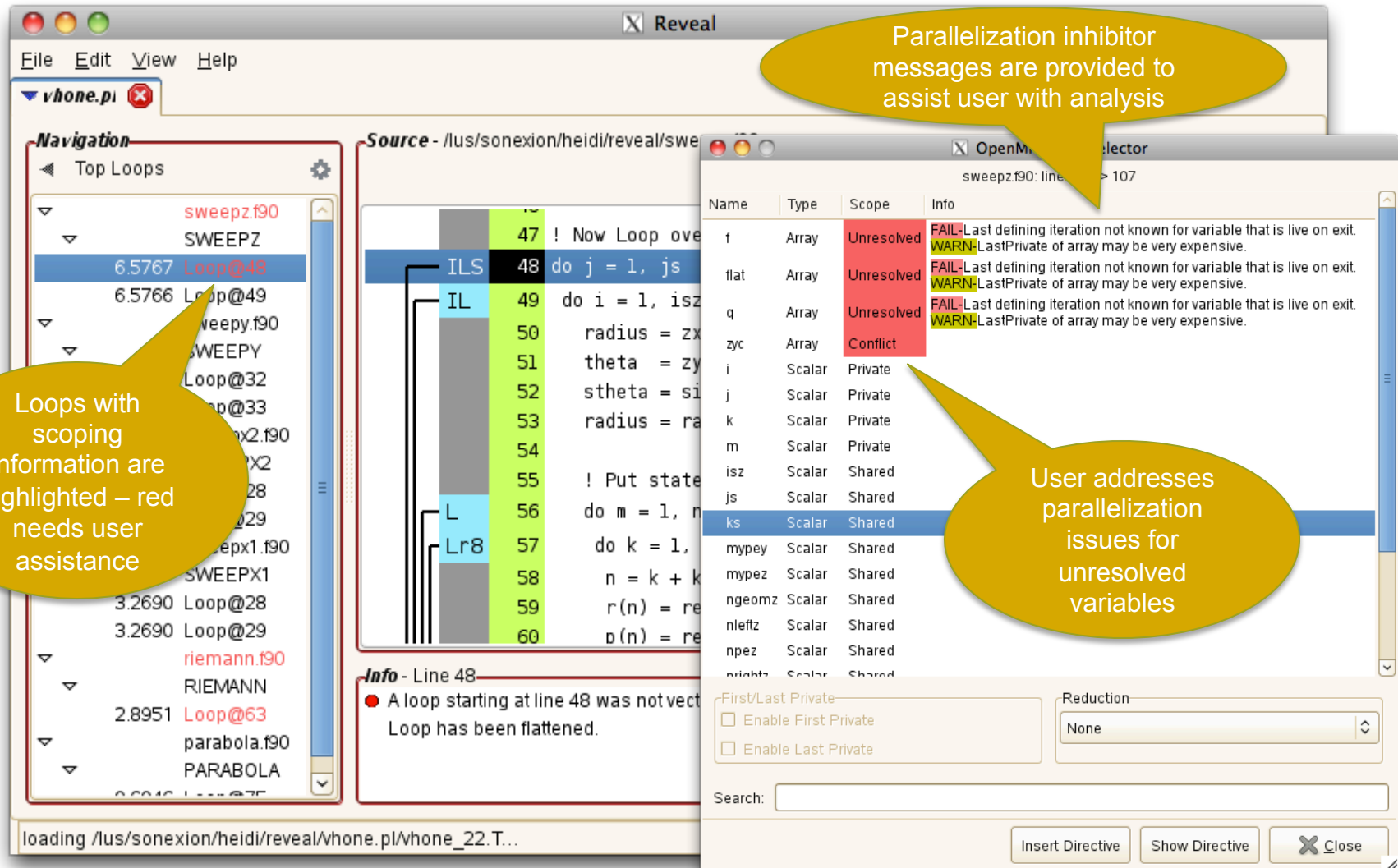
87
88 call grid(imax,xmin,xmax,zxa,zxc,zdx)
88   t$26 = 100
88   t$27 = 100
88   $I_L88_100 = 0
88   !dir$ ivdep
88   do
88     zxa(1 + $I_L88_100) = 9.9999998e-3 * $I_L88_100
88     zdx(1 + $I_L88_100) = 9.9999998e-3
88     zxc(1 + $I_L88_100) = 4.9999999e-3 + ( 9.9999998e-3 *
88       $I_L88_100 = 1 + $I_L88_100
88     if ( $I_L88_100 >= 100 ) exit
88   enddo
89 call grid(jmax,ymin,ymax,zya,zyc,zdy)
90 call grid(kmax,zmin,zmax,zza,zzc,zdz)
  
```

**Info - Line 89:**

- A divide was turned into a multiply by a reciprocal
- A loop starting at line 89 was fused with the loop starting at line 88.
- The call to leaf routine "grid" was textually inlined.



# Scoping Assistance – Review Scoping Results



**Navigation**

- Top Loops
  - sweepz.f90
    - SWEEPZ
      - 6.5767 Loop@48
      - 6.5766 Loop@49
    - sweepy.f90
      - SWEEPY
        - Loop@32
        - Loop@33
      - Loop@34
      - Loop@35
      - Loop@36
      - Loop@37
      - Loop@38
      - Loop@39
      - Loop@40
      - Loop@41
      - Loop@42
      - Loop@43
      - Loop@44
      - Loop@45
      - Loop@46
      - Loop@47
      - Loop@48
      - Loop@49
      - Loop@50
      - Loop@51
      - Loop@52
      - Loop@53
      - Loop@54
      - Loop@55
      - Loop@56
      - Loop@57
      - Loop@58
      - Loop@59
      - Loop@60
  - riemann.f90
    - RIEMANN
      - 2.8951 Loop@63
    - parabola.f90
      - PARABOLA

**Source** - /lus/sonexion/heidi/reveal/sweepz.f90

```

47 ! Now Loop over
48 do j = 1, js
49 do i = 1, isz
50 radius = zxc
51 theta = zyc
52 stheta = sin(theta)
53 radius = radius * stheta
54
55 ! Put state
56 do m = 1, n
57 do k = 1, m
58 n = k + k
59 r(n) = radius
60 p(n) = radius

```

**Info** - Line 48

- A loop starting at line 48 was not vectored. Loop has been flattened.

**Open Message Selector**

Name	Type	Scope	Info
f	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit. WARN-LastPrivate of array may be very expensive.
flat	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit. WARN-LastPrivate of array may be very expensive.
q	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit. WARN-LastPrivate of array may be very expensive.
zyc	Array	Conflict	
i	Scalar	Private	
j	Scalar	Private	
k	Scalar	Private	
m	Scalar	Private	
isz	Scalar	Shared	
js	Scalar	Shared	
ks	Scalar	Shared	
mypey	Scalar	Shared	
mypez	Scalar	Shared	
ngeomz	Scalar	Shared	
nleftz	Scalar	Shared	
npez	Scalar	Shared	
nrightz	Scalar	Shared	

**First/Last Private**

- ☐ Enable First Private
- ☐ Enable Last Private

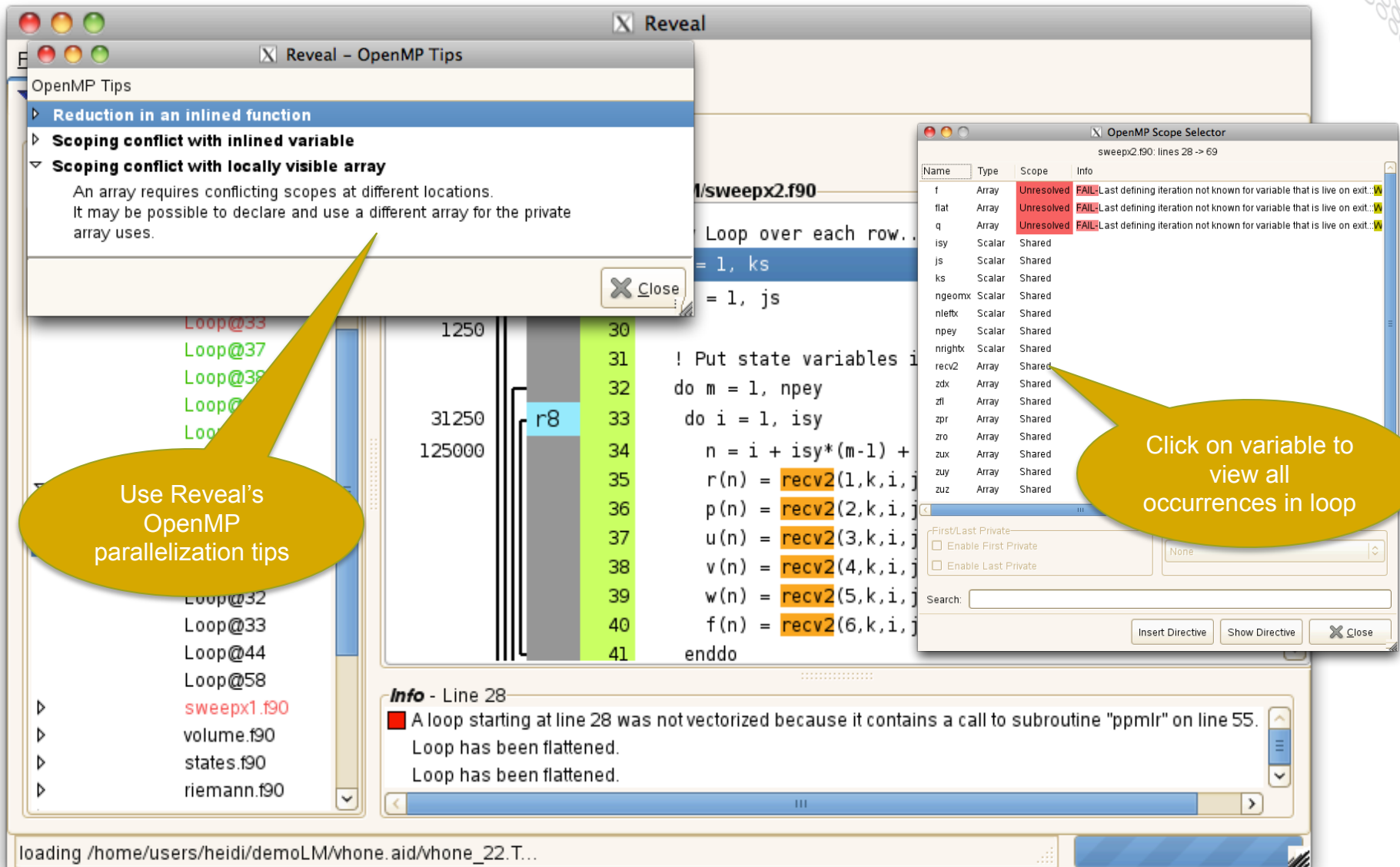
**Reduction**

None

Search:

Insert Directive Show Directive Close

# Scoping Assistance – User Resolves Issues



**OpenMP Tips**

- 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.

**OpenMP Scope Selector**  
sweepx2.f90: lines 28 -> 69

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	
zfi	Array	Shared	
zpr	Array	Shared	
zro	Array	Shared	
zux	Array	Shared	
zuy	Array	Shared	
zuz	Array	Shared	

**Code Editor:**

```

! Put state variables i
do m = 1, npey
  do i = 1, isy
    n = i + isy*(m-1) +
    r(n) = recv2(1,k,i,j)
    p(n) = recv2(2,k,i,j)
    u(n) = recv2(3,k,i,j)
    v(n) = recv2(4,k,i,j)
    w(n) = recv2(5,k,i,j)
    f(n) = recv2(6,k,i,j)
  enddo

```

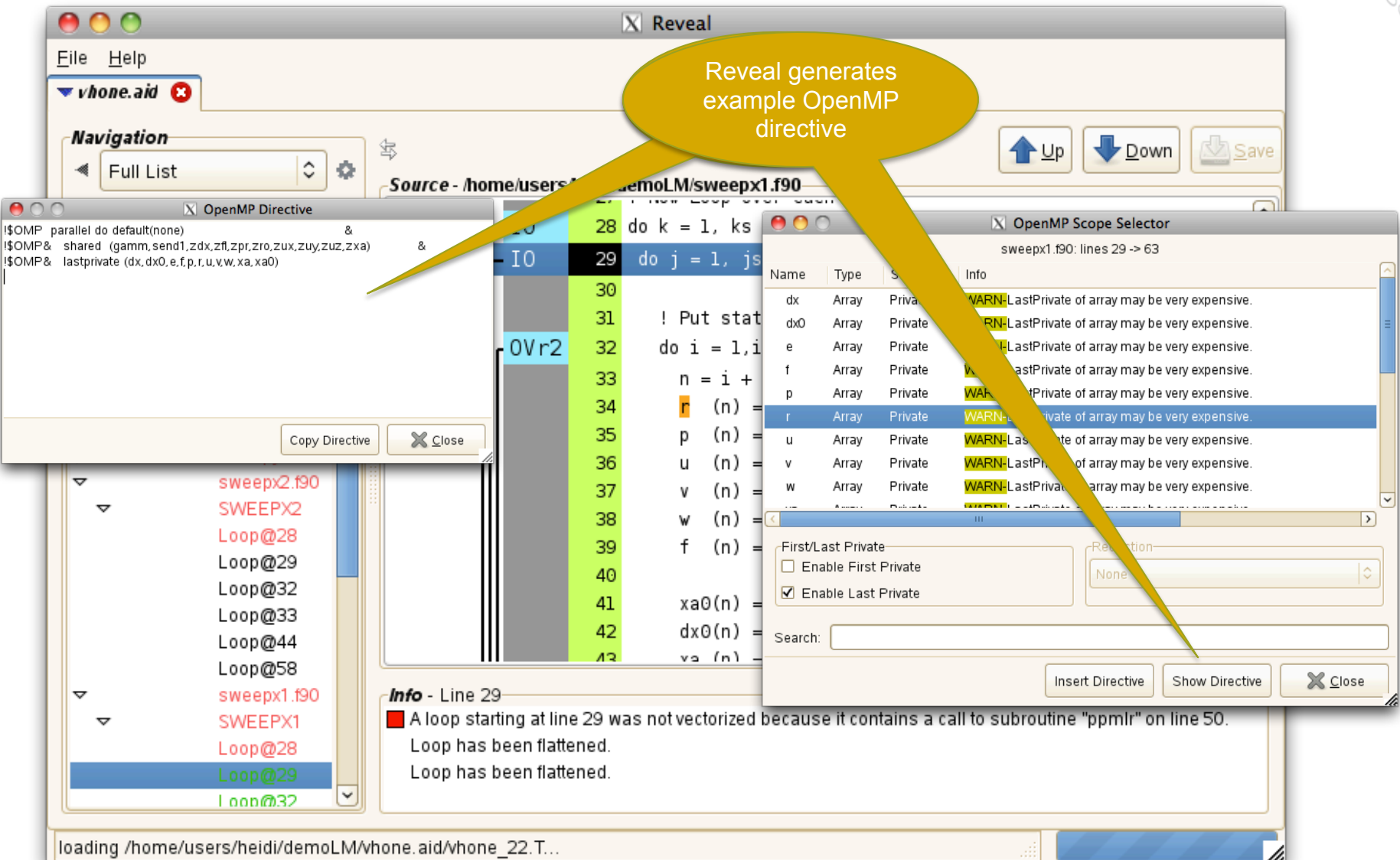
**Info - Line 28**

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.

**Use Reveal's OpenMP parallelization tips**

**Click on variable to view all occurrences in loop**

# Scoping Assistance – Generate Directive



Reveal generates example OpenMP directive

**OpenMP Directive**

```
$OMP parallel do default(none) &
$OMP shared (gamm,send1,zdx,zfl,zpr,zro,zux,zuy,zuz,zxa) &
$OMP lastprivate (dx,dx0,e,f,p,r,u,v,w,xa,xa0)
```

**OpenMP Scope Selector**

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.

**Info - Line 29**

- 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.

# Questions ?