Reveal

Overview

- Next generation integrated performance analysis and code optimization tool.
- Extends existing performance measurement, analysis, and visualization technology.
- Combines run-time performance statistics and program source code visualization.
- Help to understand which high level serial loops could benefit from improved parallelism.
- Provides enhanced loopmark listing functionality and dependency information for targeted loops.
- Assists users optimizing code by providing variable scoping feedback and suggested compiler directives

Overview

- Take a program library as input to enable browsing source code with compiler optimization information. Enhanced loopmark functionality.
- Use `-h pl=/path/program_library` and `-hwp` flags to create a program library.
- Can also take performance data files *.ap2 containing loop work estimates to assist with navigation to loops that are good candidates for parallelization. This procedure is explained in the loop instrumentation section.
- More information can be found in man reveal after loading the perftools module and in the hands-on sessions.

- Requires codes to be compiled with CCE

Reveal

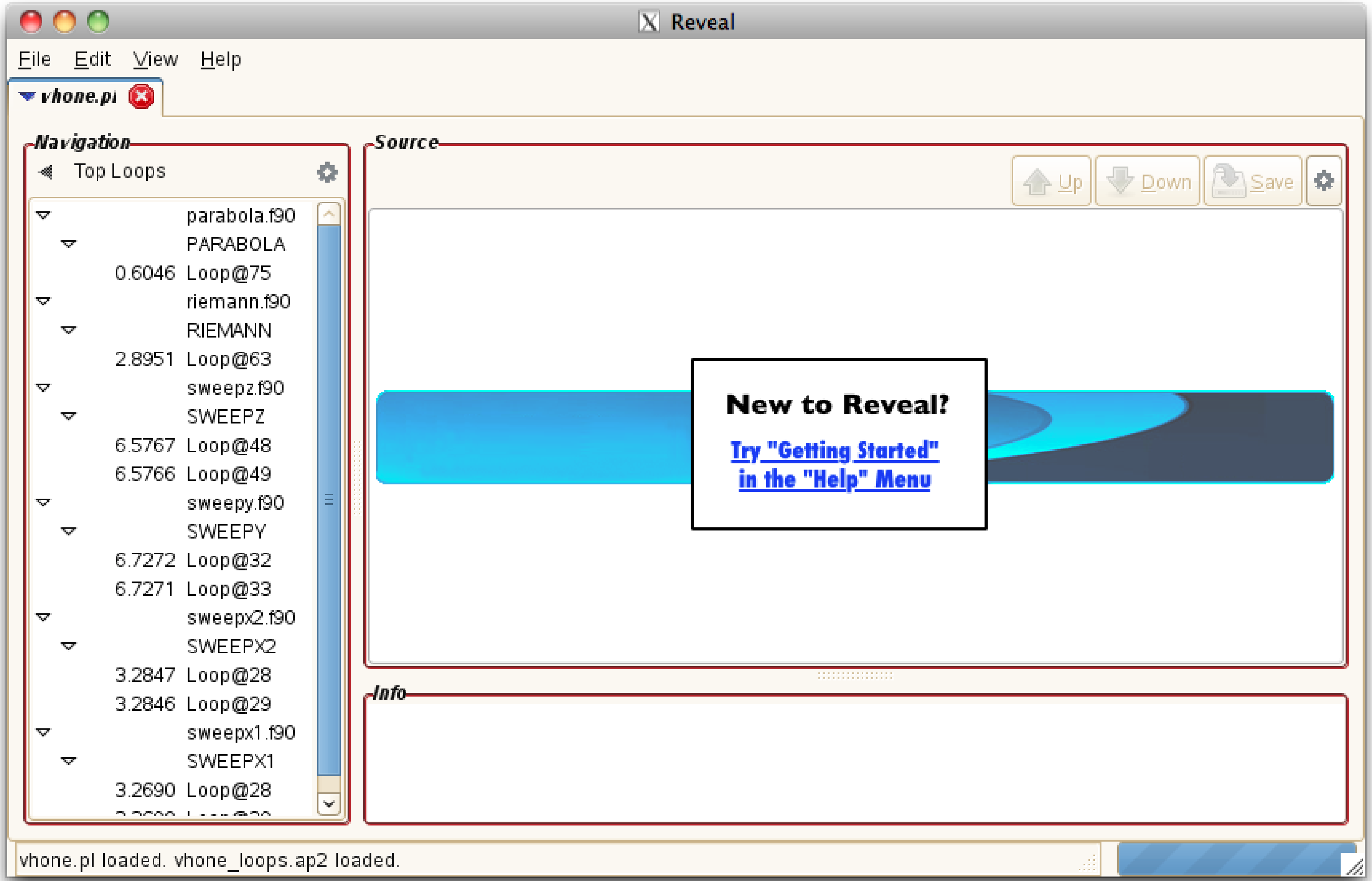
New analysis and code restructuring assistant…

- **Uses both the performance toolset and CCE's program library functionality to provide static and runtime analysis information**
- **Assists user with the code optimization phase by correlating source code with analysis to help identify which areas are key candidates for optimization**

Key Features

- **Annotated source code with compiler optimization information**
  - **Provides feedback on critical dependencies that prevent optimizations**
- **Scoping analysis**
  - **Identifies shared, private and ambiguous arrays**
    - **Allows user to privatize ambiguous arrays**
    - **Allows user to override dependency analysis**
- **Source code navigation**
  - **Uses performance data collected through CrayPat**

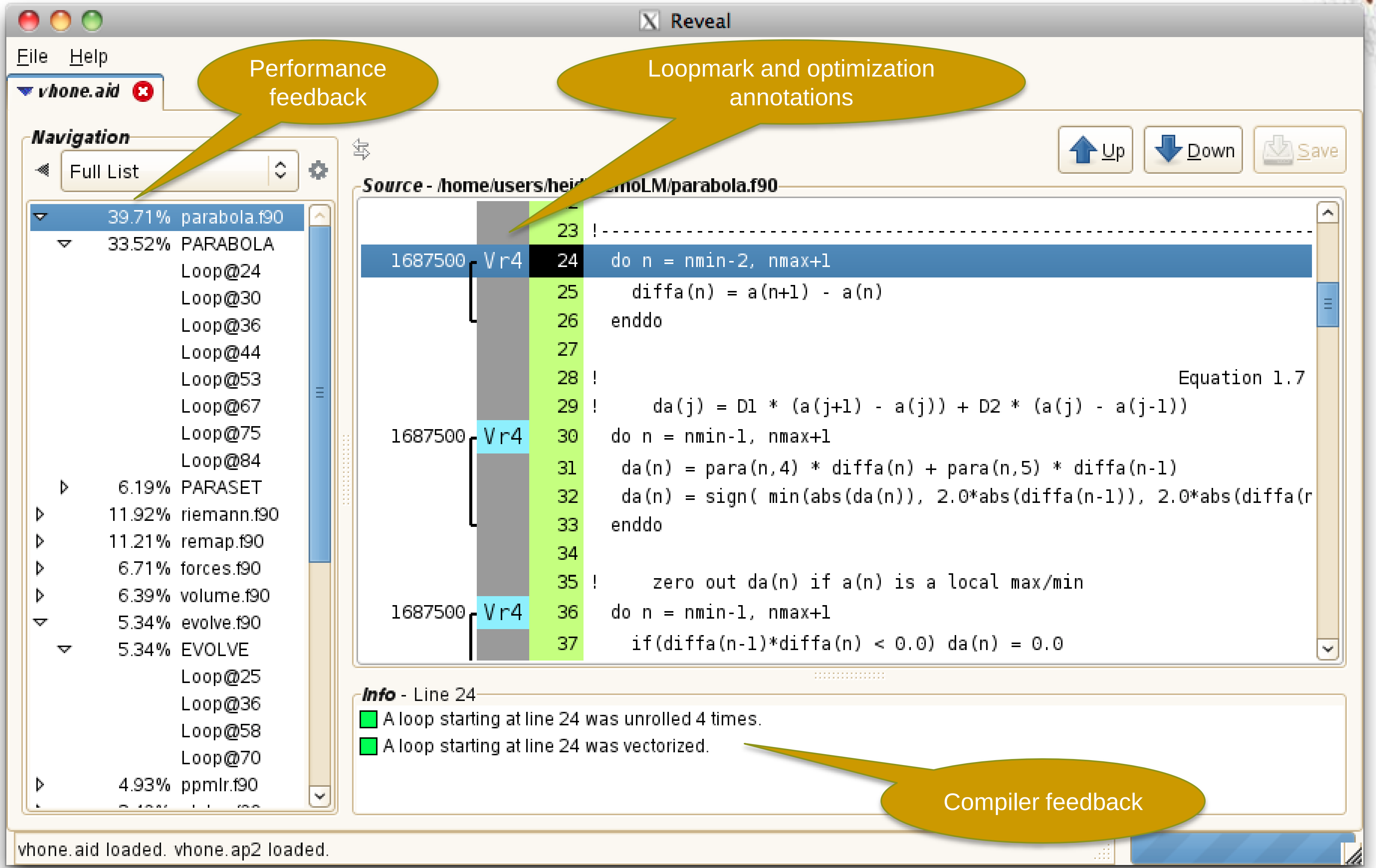
Reveal with Loop Work Estimates



```
              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
```

vhone.pl loaded. vhone_loops.ap2 loaded.

Visualize CCE's Loopmark with Performance Profile



```
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/hei…moLM/parabola.f90

```
                  23  !-----------------------------------------------------------------
1687500  Vr4      24    do n = nmin-2, nmax+1
                  25      diffa(n) = a(n+1) - a(n)
                  26    enddo
                  27
                  28  !                                                    Equation 1.7
                  29  !     da(j) = D1 * (a(j+1) - a(j)) + D2 * (a(j) - a(j-1))
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(r
                  33    enddo
                  34
                  35  !     zero out da(n) if a(n) is a local max/min
1687500  Vr4      36    do n = nmin-1, nmax+1
                  37      if(diffa(n-1)*diffa(n) < 0.0) da(n) = 0.0
```

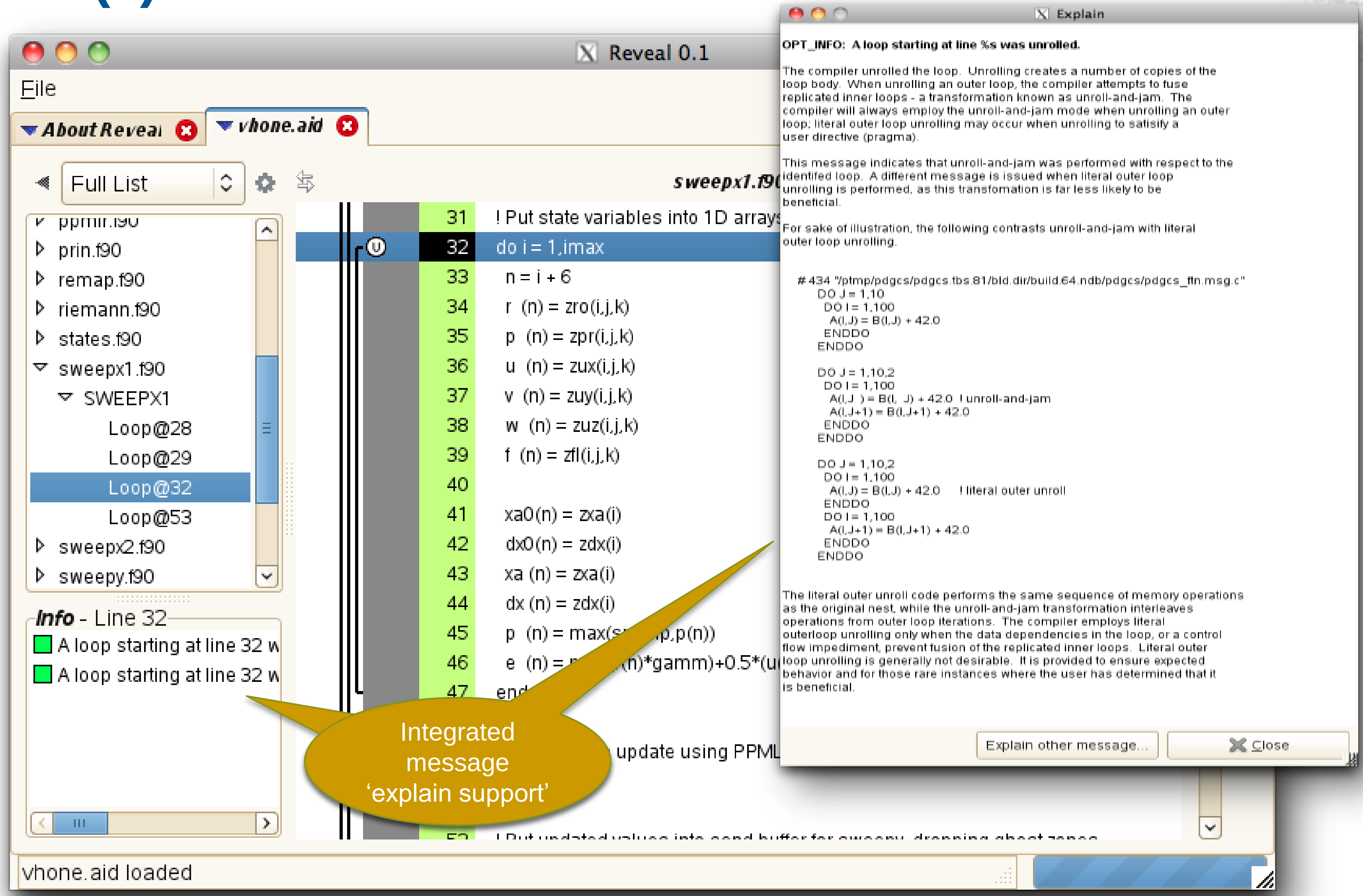
Info - Line 24

🟩 A loop starting at line 24 was unrolled 4 times.

🟩 A loop starting at line 24 was vectorized.

vhone.aid loaded. vhone.ap2 loaded.

Visualize CCE's Loopmark with Performance Profile (2)



```
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) = r        (n)*gamm)+0.5*(u
47  end
           update using PPML
53  ! Put updated values into send buffer for sweepy, dropping ghost zones
```

🟩 A loop starting at line 32 w

🟩 A loop starting at line 32 w

vhone.aid loaded

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 satisify a user directive (pragma).

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

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

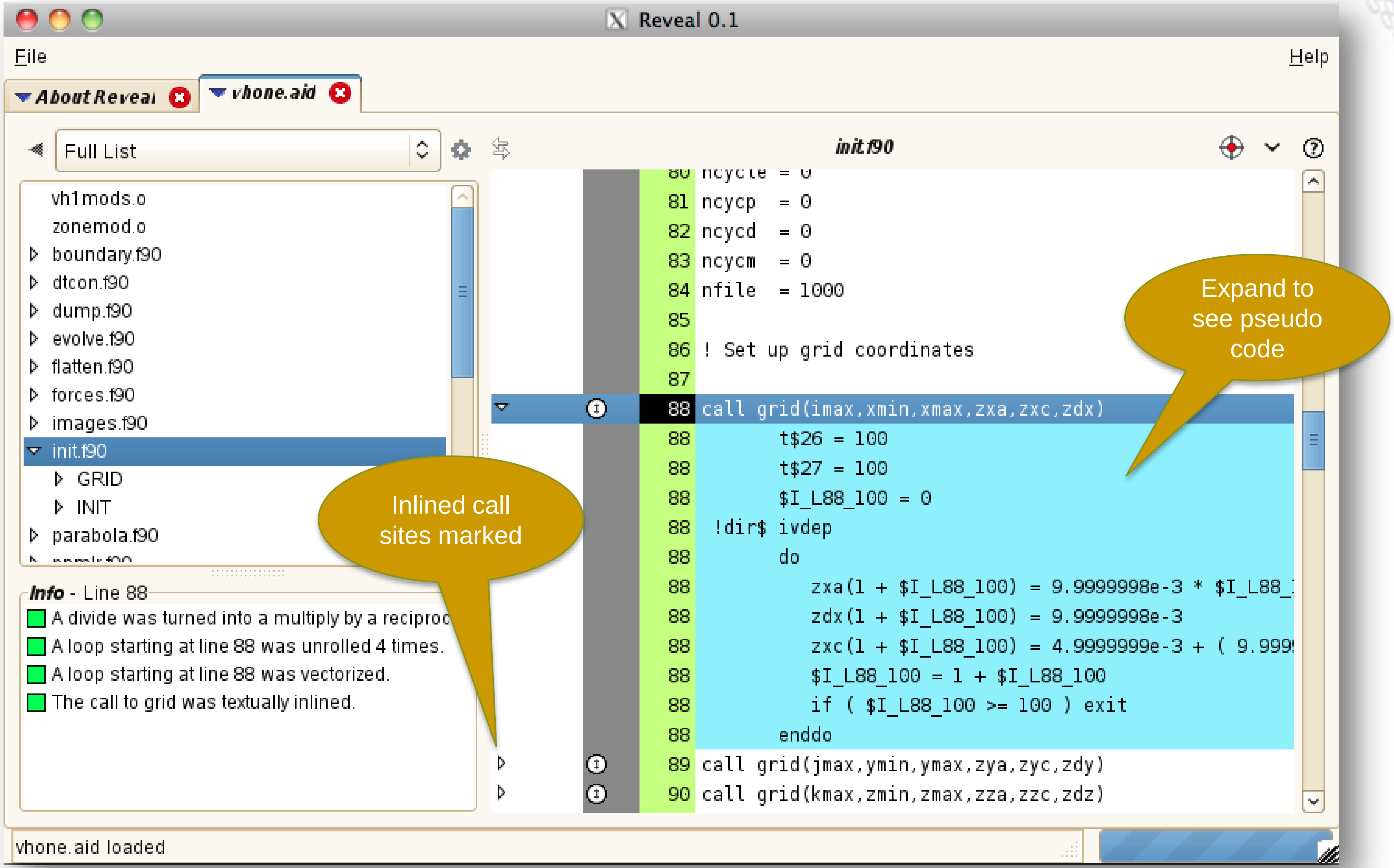
```
  # 434 "/ptmp/pdgcs/pdgcs.tbs.81/bld.dir/build.64.ndb/pdgcs/pdgcs_ftn.msg.c"
      DO J = 1,10
       DO I = 1,100
        A(I,J) = B(I,J) + 42.0
       ENDDO
      ENDDO

      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 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.

View Pseudo Code for Inlined Functions

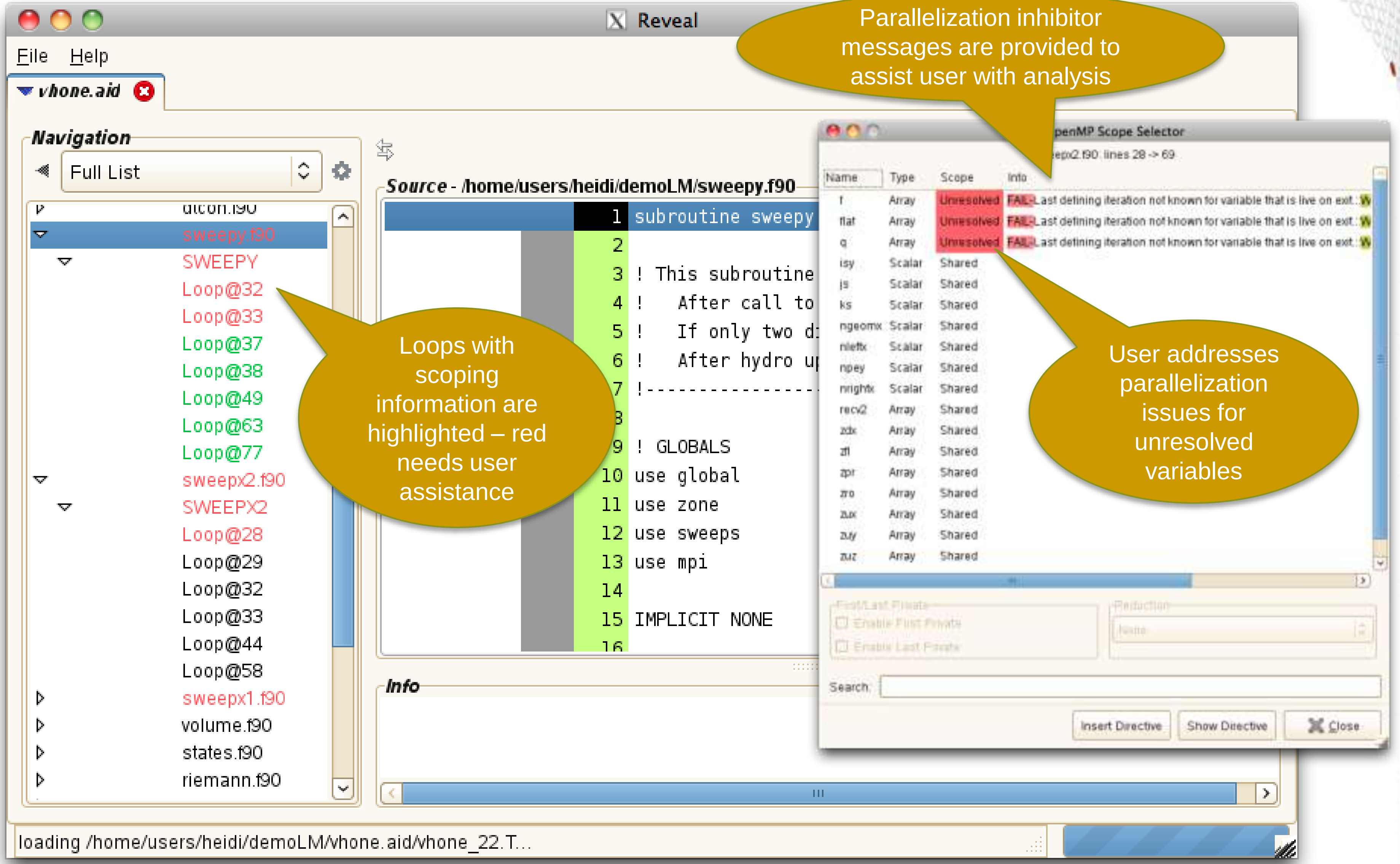


Info - Line 88

- 🟩 A divide was turned into a multiply by a reciproc
- 🟩 A loop starting at line 88 was unrolled 4 times.
- 🟩 A loop starting at line 88 was vectorized.
- 🟩 The call to grid was textually inlined.

```
80 ncycle = 0
81 ncycp  = 0
82 ncycd  = 0
83 ncycm  = 0
84 nfile  = 1000
85
86 ! Set up grid coordinates
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_
88          zdx(1 + $I_L88_100) = 9.9999998e-3
88          zxc(1 + $I_L88_100) = 4.9999999e-3 + ( 9.999
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)
```

Scoping Assistance – Review Scoping Results



dtcon.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

```
subroutine sweepy

! This subroutine
!   After call to
!   If only two di
!   After hydro up
!----------------

! GLOBALS
use global
use zone
use sweeps
use mpi

IMPLICIT NONE
```

epx2.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	
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	

First/Last Private
☐ Enable First Private
☐ Enable Last Private
Reduction
None

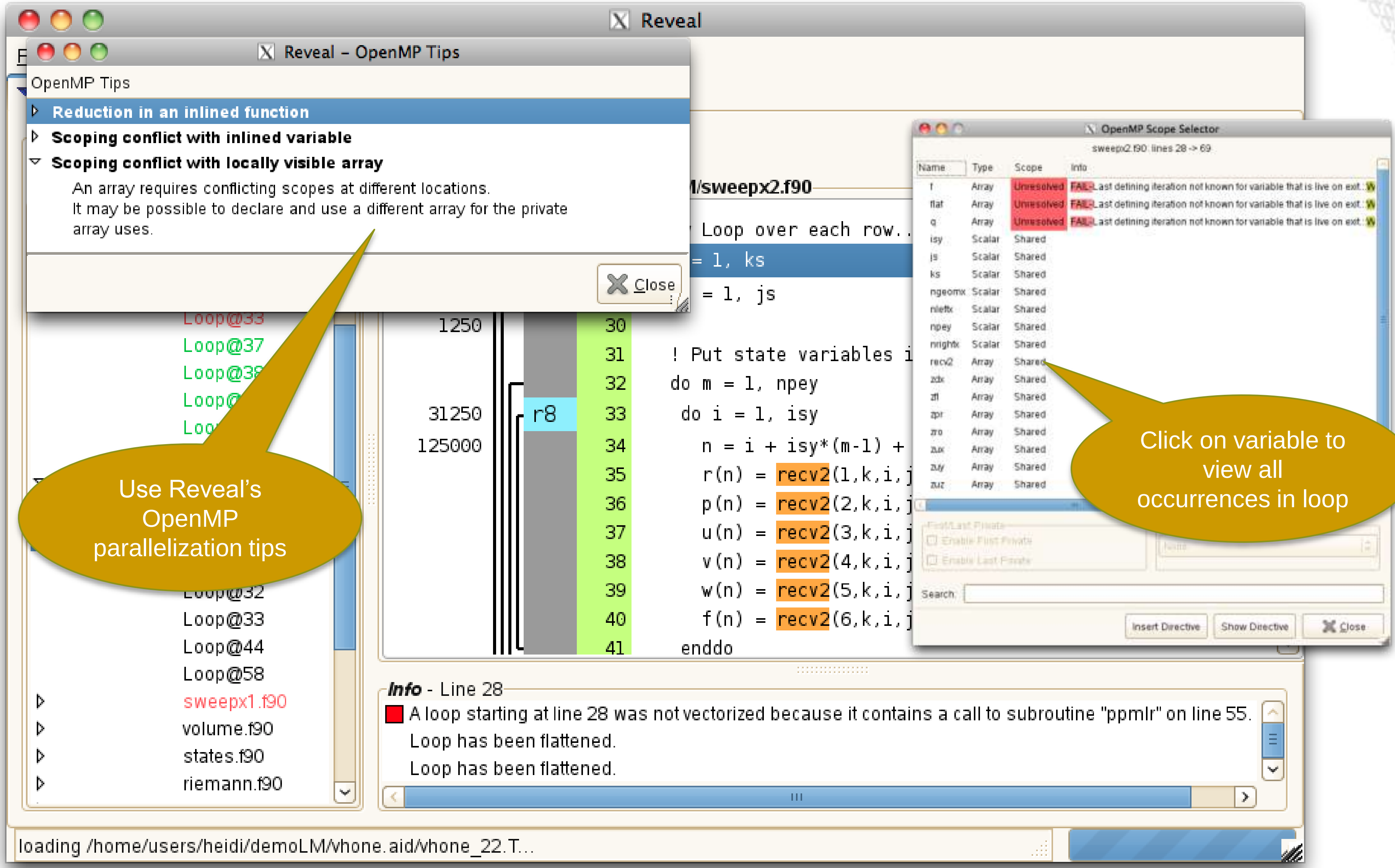
loading /home/users/heidi/demoLM/vhone.aid/vhone_22.T...

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



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.

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.:W
flat	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit.:W
q	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit.:W
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	

```
! Loop over each row..
  = 1, ks
  = 1, js
1250

 ! Put state variables i
 do m = 1, npey
31250
  do i = 1, isy
125000
    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

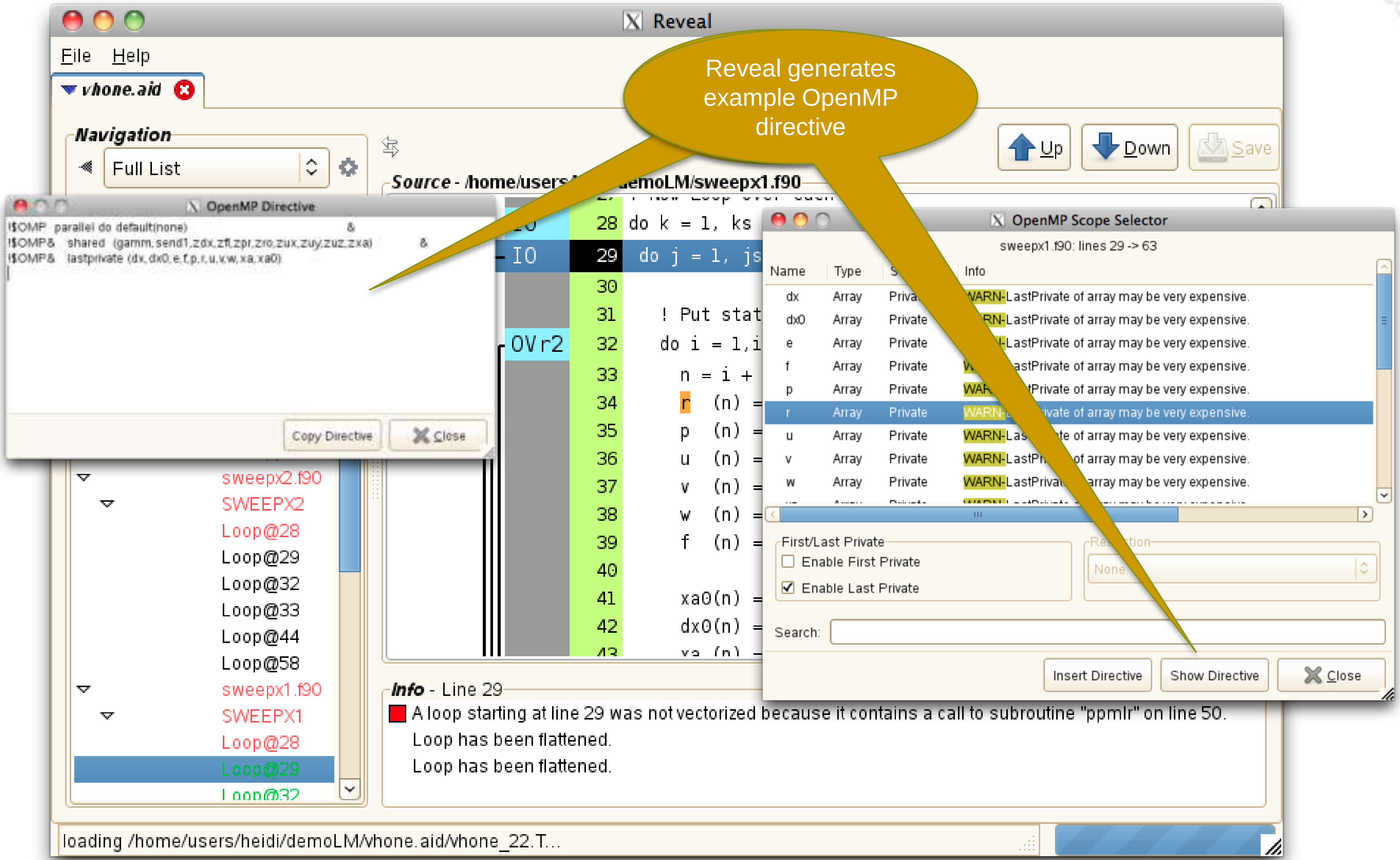
🟥 A loop starting at line 28 was not vectorized because it contains a call to subroutine "ppmlr" on line 55.
Loop has been flattened.
Loop has been flattened.

loading /home/users/heidi/demoLM/vhone.aid/vhone_22.T...

Use Reveal's OpenMP parallelization tips

Click on variable to view all occurrences in loop

Scoping Assistance – Generate 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)
```

Source - /home/users/[illegible]demoLM/sweepx1.f90

```
28  do k = 1, ks
29   do j = 1, js
30
31     ! Put stat
32     do i = 1,i
33       n = i +
34       r  (n) =
35       p  (n) =
36       u  (n) =
37       v  (n) =
38       w  (n) =
39       f  (n) =
40
41       xa0(n) =
42       dx0(n) =
```

OpenMP Scope Selector

sweepx1.f90: lines 29 -> 63

Name	Type	S[illegible]	Info
dx	Array	Priva[illegible]	WARN-LastPrivate of array may be very expensive.
dx0	Array	Private	[illegible]RN-LastPrivate of array may be very expensive.
e	Array	Private	[illegible]-LastPrivate of array may be very expensive.
f	Array	Private	W[illegible]astPrivate of array may be very expensive.
p	Array	Private	WAR[illegible]tPrivate of array may be very expensive.
r	Array	Private	WARN-[illegible]ivate of array may be very expensive.
u	Array	Private	WARN-Las[illegible]te of array may be very expensive.
v	Array	Private	WARN-LastPr[illegible] of array may be very expensive.
w	Array	Private	WARN-LastPriva[illegible]array may be very expensive.

First/Last Private
- [ ] Enable First Private
- [x] Enable Last Private

Re[illegible]tion: None

Search:

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.

loading /home/users/heidi/demoLM/vhone.aid/vhone_22.T...