Cray Reveal Webinar: A Tool to Help Porting to Manycore

Heidi Poxon
Technical Lead & Sr. Manager, Performance Tools
Cray Inc.

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Future Architecture Directions

- **Nodes are becoming more parallel**
  - More processors per node
  - More threads per processor
  - Vector lengths are getting longer
  - Memory hierarchy is becoming more complex
  - Scalar performance is not increasing and will start decreasing

- **For the next decade, HPC systems will have the same basic architecture:**
  - Message passing between nodes
  - Multithreading within the node (pure MPI will not do)
  - Vectorization at the lowest level (SSE, AVX, GPU, Phi)

Future Application Directions

- **Threading on node as well as vectorization is becoming more important – need more parallelism exploited in applications due to increasing number of cores and threads**

- **Current petascale applications are not structured to take advantage of these architectures**
  - Currently 80-90% of applications use a single level of parallelism
    - MPI or PGAS between cores of the MPP system

  - Looking forward, application developers are faced with a significant task in preparing their applications for the future
    - Codes must be converted to use multiple levels of parallelism
    - More complex memory hierarchies will require user intervention to achieve good performance

Three Levels of Parallelism Required

1. Developers will continue to use MPI between nodes or sockets

2. Developers must address using a shared memory programming paradigm on the node

3. Developers must vectorize low level looping structures

While there is a potential acceptance of new languages for addressing all levels directly. Most developers cannot afford this approach until they are assured that the new language will be accepted and the generated code is within a reasonable performance range

When to Move to a Hybrid Programming Model

- **When code is network bound**
  - Look at collective time, excluding sync time: this goes up as network becomes a problem
  - Look at point-to-point wait times: if these go up, network may be a problem
- **When MPI starts leveling off**
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue
- **When contention of shared resources increases**
- **When you want to exploit heterogeneous nodes**

Approach to Adding Parallelism

1. **Identify key high-level loops**
   - 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 Reveal + CCE for scoping, loopmark and source browsing

3. **Add OpenMP layer of parallelism**
   - Insert OpenMP directives (with Reveal directive building assistance)
     - Run on X86 to verify application and check for performance improvements

4. **Analyze performance for further optimizations, specifically vectorization of innermost loops**

Challenges

- **Investigate parallelizability of high level looping structures**
  - Often times one level of loop is not enough, must have several parallel loops
    - Need a large number of loop iterations to feed the GPU threads
  - User must understand which high level DO loops have independent iterations
  - Without tools, variable scoping of high level loops is very difficult
    - Loops must be more than independent, their variable usage must adhere to private data local to a thread or global shared across all the threads
    - Independence can be complicated to understand (and even runtime dependent)
- **Investigate vectorizability of lower level DO loops**

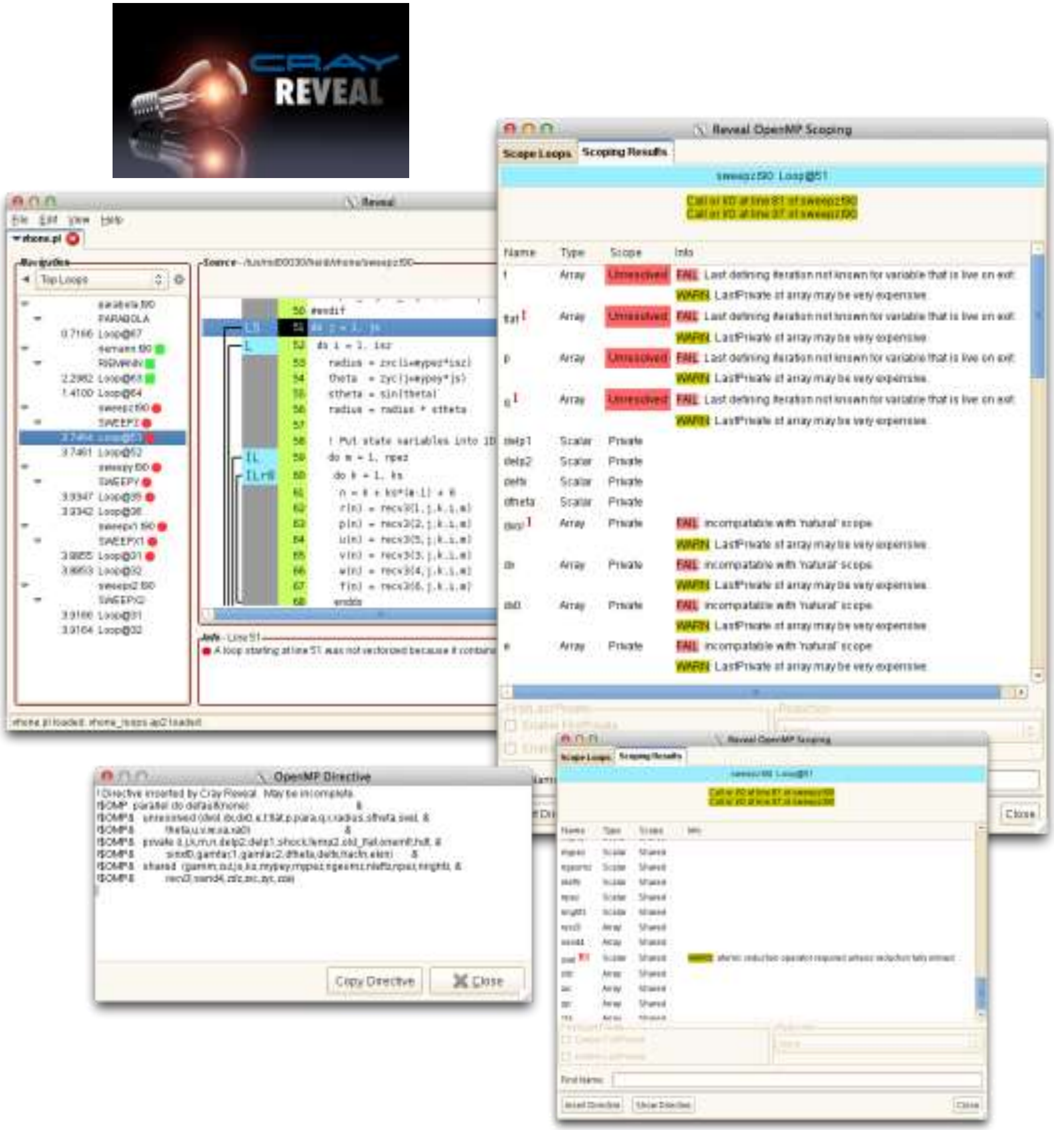
The Problem – How Do I Parallelize This Loop?

- How do I know this is a good loop to parallelize?
- What prevents me from parallelizing this loop?
- Can I get help building a directive?

```
subroutine sweepz
…
do j = 1, js
 do i = 1, isz
   radius = zxc(i+mypez*isz)
   theta  = zyc(j+mypey*js)
   do m = 1, npez
    do k = 1, ks
     n = k + ks*(m-1) + 6
     r(n) = recv3(1,j,k,i,m)
     p(n) = recv3(2,j,k,i,m)
     u(n) = recv3(5,j,k,i,m)
     v(n) = recv3(3,j,k,i,m)
     w(n) = recv3(4,j,k,i,m)
     f(n) = recv3(6,j,k,i,m)
    enddo
   enddo
   …
   call ppmlr
   do k = 1, kmax
     n = k + 6
     xa  (n) = zza(k)
     dx  (n) = zdz(k)
     xa0(n) = zza(k)
     dx0(n) = zdz(k)
     e   (n) = p(n)/(r(n)*gamm)+0.5 &
        *(u(n)**2+v(n)**2+w(n)**2)
   enddo
   call ppmlr
…
 enddo
enddo
```

```
subroutine ppmlr

call boundary
call flatten
call paraset(nmin-4, nmax+5, para, dx, xa)

call parabola(nmin-4,nmax+4,para,p,dp,p6,pl,flat)
call parabola(nmin-4,nmax+4, para,r,dr,r6,rl,flat)
call parabola(nmin-4,nmax+4,para,u,du,u6,ul,flat)

call states(pl,ul,rl,p6,u6,r6,dp,du,dr,plft,ulft,&
            rlft,prgh,urgh,rrgh)
call riemann(nmin-3,nmax+4,gam,prgh,urgh,rrgh,&
             plft,ulft,rlft pmid umid)
call evolve(umid, pmid)  ← contains more calls

call remap ← contains more calls

call volume(nmin,nmax,ngeom,radius,xa,dx,dvol)

call remap ← contains more calls

return
End
```

Simplifying the Task with Reveal



```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                    &
```

- Navigate to relevant loops to parallelize
- Identify parallelization and scoping issues
- Get feedback on issues down the call chain (shared reductions, etc.)
- Optionally insert parallel directives into source
- Validate scoping correctness on existing directives

COMPUTE | STORE | ANALYZE

Using Reveal with Performance Statistics

*Optionally **create loop statistics** using the Cray performance tools to determine which loops have the most work*

- **Helps identify high-level serial loops to parallelize**
  - Based on runtime analysis, approximates how much work exists within a loop

- **Provides the following statistics**
  - Min, max and average trip counts
  - Inclusive time spent in loops
  - Number of times a loop was executed

Collecting Loop Work Estimates

- Load PrgEnv-cray module (must use CCE)
- Load perftools module

- Compile **AND** link with –h profile_generate
  - cc –h profile_generate –o my_program my_program.c

- Instrument binary for tracing
  - pat_build –w my_program

- Run application

- Create report with loop statistics
  - pat_report my_program.xf > loops_report

pat_report produces report plus .ap2 file that can be used with Reveal

Example Report – Inclusive Loop Time

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

    Loop   |     Loop  |    Loop  |  Loop   |  Loop   |Function=/.LOOP[.]
    Incl   |      Hit  |   Trips  | Trips   | Trips   | PE=HIDE
    Time   |           |     Avg  |   Min   |   Max   |
   Total   |           |          |         |         |
|------------------------------------------------------------------------
| 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
```

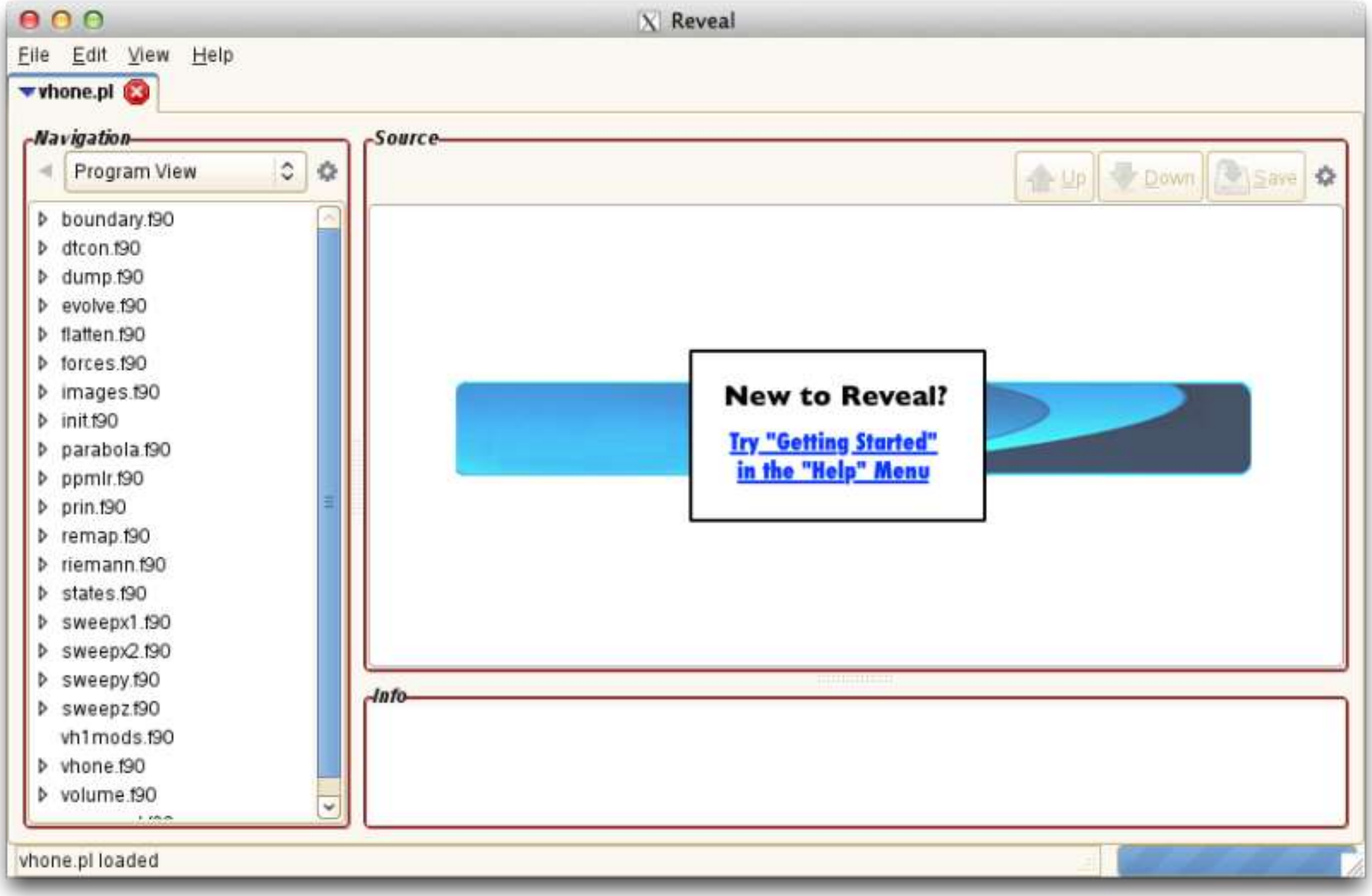
How to Use Reveal

- Generate a program library for your application with CCE
  - `> cc -h pl=himeno.pl -hwp himeno.c`
  - `> ftn -h pl=vhone.pl -hwp file1.f90`

Optionally add whole program analysis for more aggressive inlining

- Launch Reveal
  - `> module load perftools`
  - Use with compiler information only (no need to run program):
    - `> reveal vhone.pl`
  - Use with compiler + loop work estimates (include performance data)
    - `> reveal vhone.pl vhone_loops.ap2`

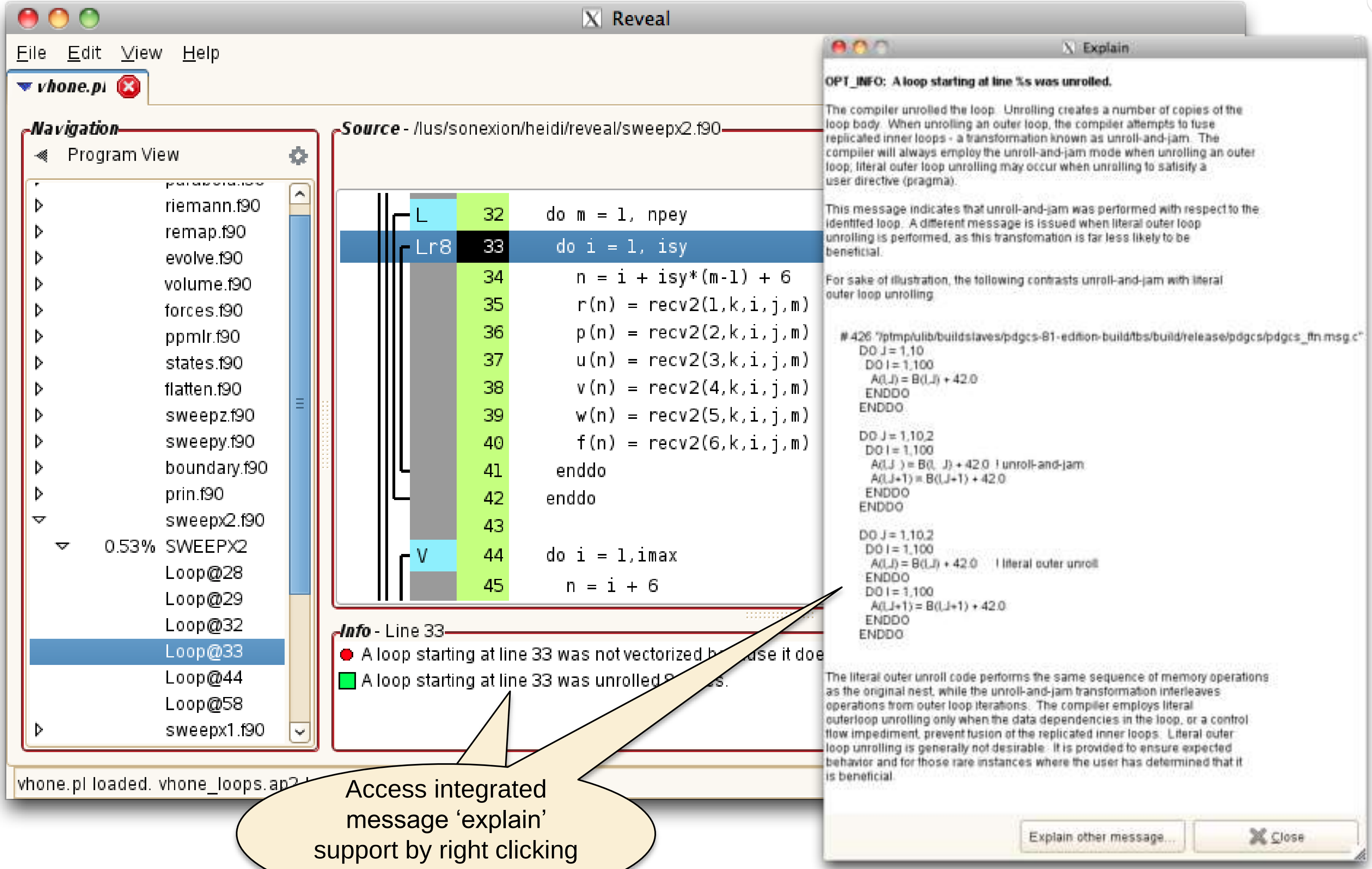
Browse Source and Compiler Optimizations



boundary.f90
dtcon.f90
dump.f90
evolve.f90
flatten.f90
forces.f90
images.f90
init.f90
parabola.f90
ppmlr.f90
prin.f90
remap.f90
riemann.f90
states.f90
sweepx1.f90
sweepx2.f90
sweepy.f90
sweepz.f90
vh1mods.f90
vhone.f90
volume.f90

vhone.pl loaded

Access Cray Compiler Message Information



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

```
L    32   do m = 1, npey
Lr8  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
V    44   do i = 1,imax
     45     n = i + 6
```

● A loop starting at line 33 was not vectorized b… use it doe

🟩 A loop starting at line 33 was unrolled 8 …s.

vhone.pl loaded. vhone_loops.ap2

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 identified loop. A different message is issued when literal outer loop unrolling is performed, as this transfomation is far less likely to be beneficial.

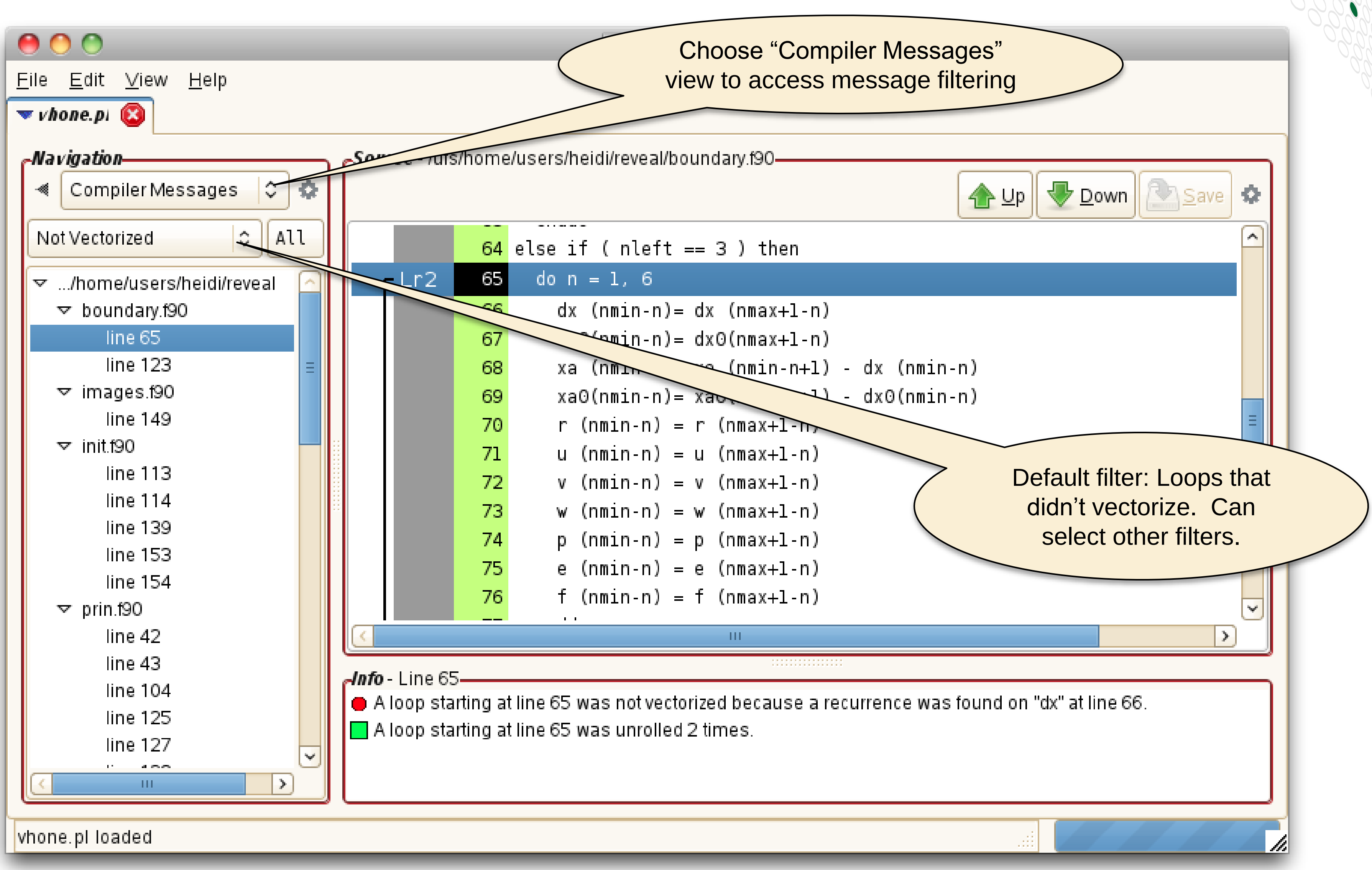
For sake of illustration, the following contrasts unroll-and-jam with literal outer loop unrolling.

```
 # 426 "/ptmp/ulib/buildslaves/pdgcs-81-edition-build/tbs/build/release/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.

Access integrated message 'explain' support by right clicking on message

Navigate Code via Compiler Messages



```
64 else if ( nleft == 3 ) then
65   do n = 1, 6
       dx (nmin-n)= dx (nmax+1-n)
67          (nmin-n)= dx0(nmax+1-n)
68     xa (nmin                  (nmin-n+1) - dx (nmin-n)
69     xa0(nmin-n)= xa0                 ) - dx0(nmin-n)
70     r (nmin-n) = r (nmax+1-n)
71     u (nmin-n) = u (nmax+1-n)
72     v (nmin-n) = v (nmax+1-n)
73     w (nmin-n) = w (nmax+1-n)
74     p (nmin-n) = p (nmax+1-n)
75     e (nmin-n) = e (nmax+1-n)
76     f (nmin-n) = f (nmax+1-n)
```

Info - Line 65

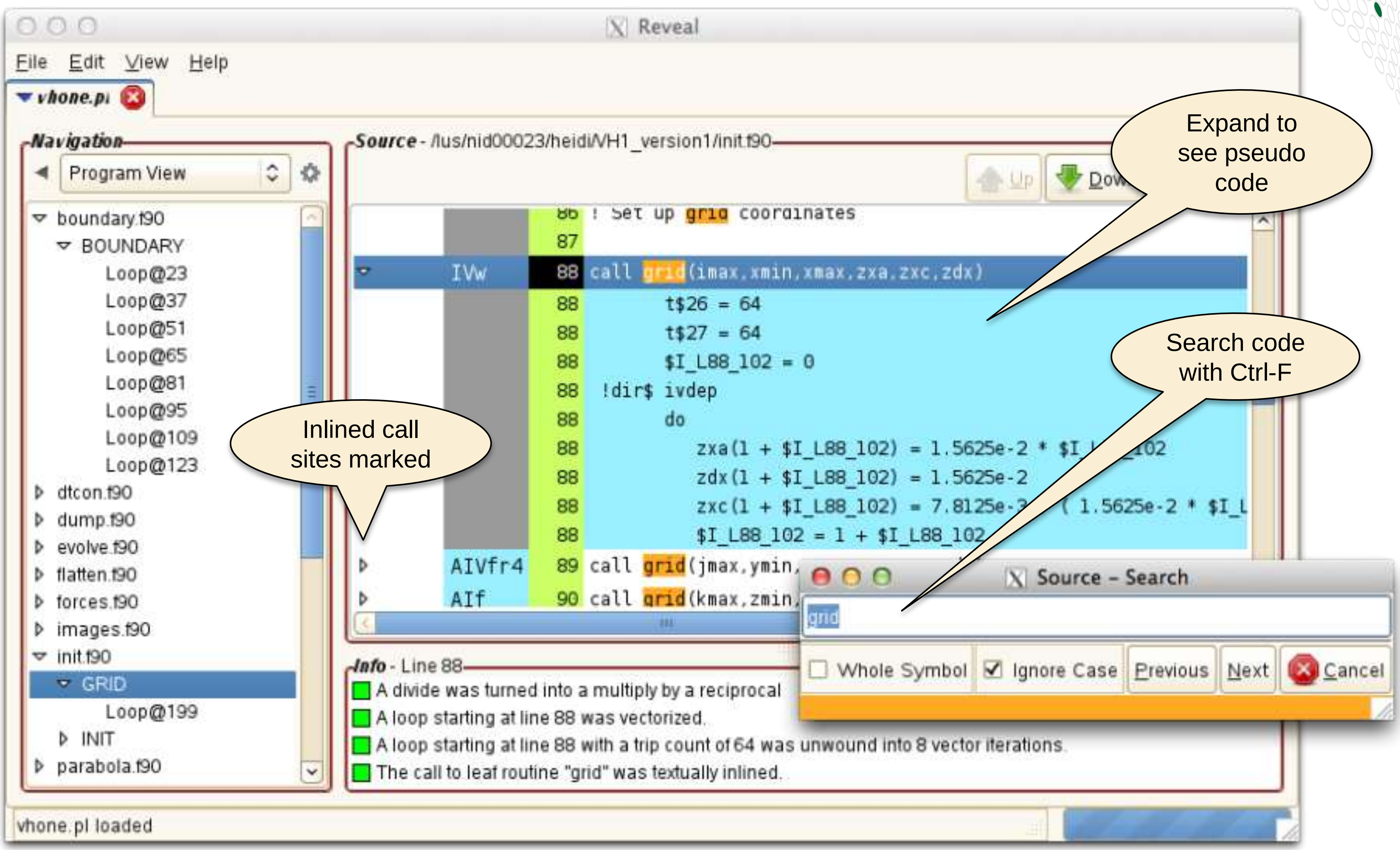
- 🟥 A loop starting at line 65 was not vectorized because a recurrence was found on "dx" at line 66.
- 🟩 A loop starting at line 65 was unrolled 2 times.

vhone.pl loaded

Choose “Compiler Messages” view to access message filtering

Default filter: Loops that didn’t vectorize. Can select other filters.

View Pseudo Code for Inlined Functions



```
86  ! Set up grid coordinates
87
88  call grid(imax,xmin,xmax,zxa,zxc,zdx)
88          t$26 = 64
88          t$27 = 64
88          $I_L88_102 = 0
88   !dir$ ivdep
88          do
88              zxa(1 + $I_L88_102) = 1.5625e-2 * $I_L   102
88              zdx(1 + $I_L88_102) = 1.5625e-2
88              zxc(1 + $I_L88_102) = 7.8125e-3   ( 1.5625e-2 * $I_L
88              $I_L88_102 = 1 + $I_L88_102
89  call grid(jmax,ymin,
90  call grid(kmax,zmin,
```

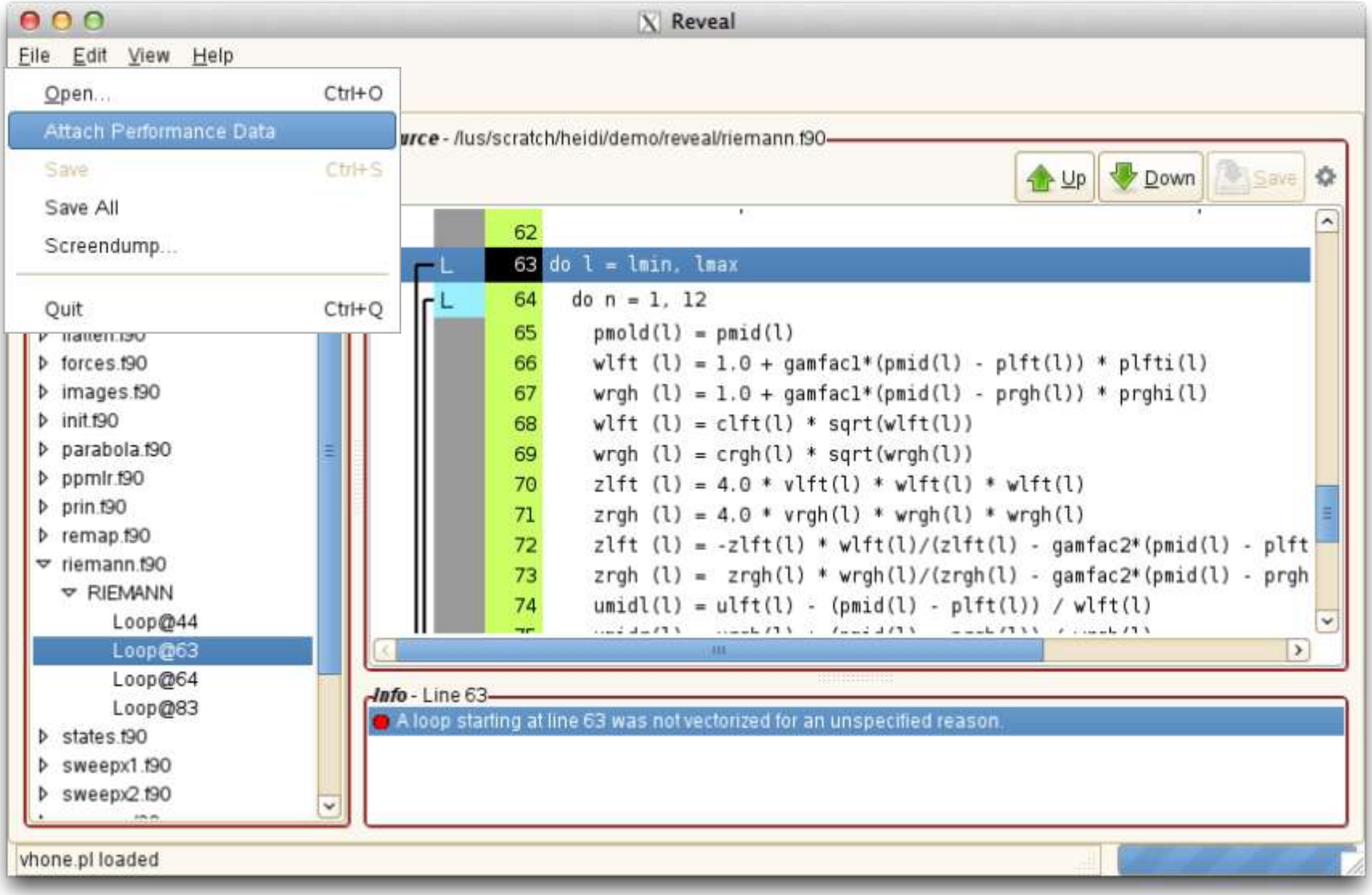
- 🟩 A divide was turned into a multiply by a reciprocal
- 🟩 A loop starting at line 88 was vectorized.
- 🟩 A loop starting at line 88 with a trip count of 64 was unwound into 8 vector iterations.
- 🟩 The call to leaf routine "grid" was textually inlined.

Expand to see pseudo code

Search code with Ctrl-F

Inlined call sites marked

Add Performance Data to Find Top Loops

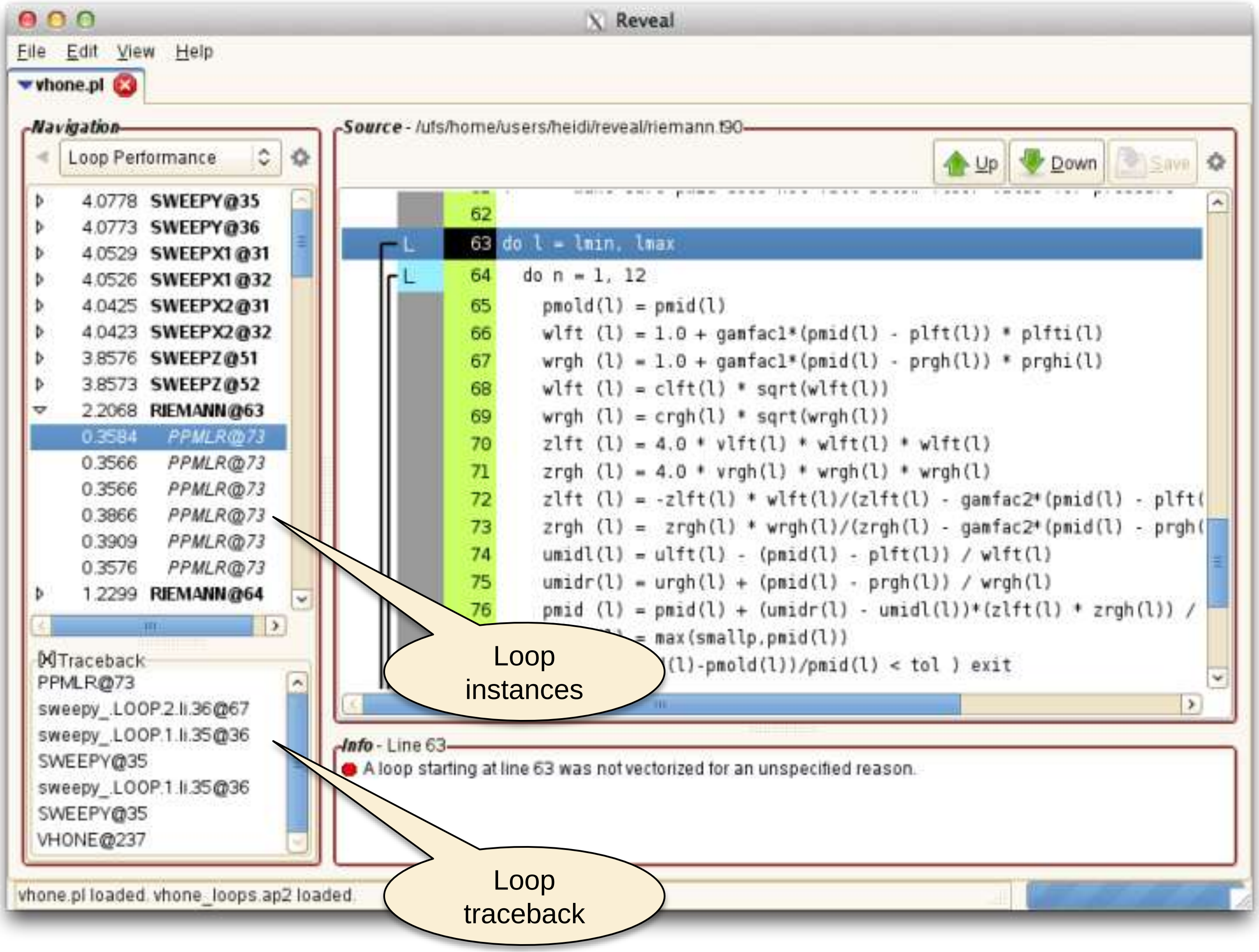


```
62
63  do l = lmin, lmax
64    do n = 1, 12
65      pmold(l) = pmid(l)
66      wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67      wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68      wlft (l) = clft(l) * sqrt(wlft(l))
69      wrgh (l) = crgh(l) * sqrt(wrgh(l))
70      zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71      zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72      zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft
73      zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh
74      umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
```

● A loop starting at line 63 was not vectorized for an unspecified reason.

vhone.pl loaded

View Loops through Call Chain



4.0778	**SWEEPY@35**
4.0773	**SWEEPY@36**
4.0529	**SWEEPX1@31**
4.0526	**SWEEPX1@32**
4.0425	**SWEEPX2@31**
4.0423	**SWEEPX2@32**
3.8576	**SWEEPZ@51**
3.8573	**SWEEPZ@52**
2.2068	**RIEMANN@63**
0.3584	*PPMLR@73*
0.3566	*PPMLR@73*
0.3566	*PPMLR@73*
0.3866	*PPMLR@73*
0.3909	*PPMLR@73*
0.3576	*PPMLR@73*
1.2299	**RIEMANN@64**

Traceback

```
PPMLR@73
sweepy_.LOOP.2.li.36@67
sweepy_.LOOP.1.li.35@36
SWEEPY@35
sweepy_.LOOP.1.li.35@36
SWEEPY@35
VHONE@237
```

```
62
63 do l = lmin, lmax
64   do n = 1, 12
65     pmold(l) = pmid(l)
66     wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67     wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68     wlft (l) = clft(l) * sqrt(wlft(l))
69     wrgh (l) = crgh(l) * sqrt(wrgh(l))
70     zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71     zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72     zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(
73     zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(
74     umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75     umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76     pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) /
                = max(smallp,pmid(l))
                (l)-pmold(l))/pmid(l) < tol ) exit
```

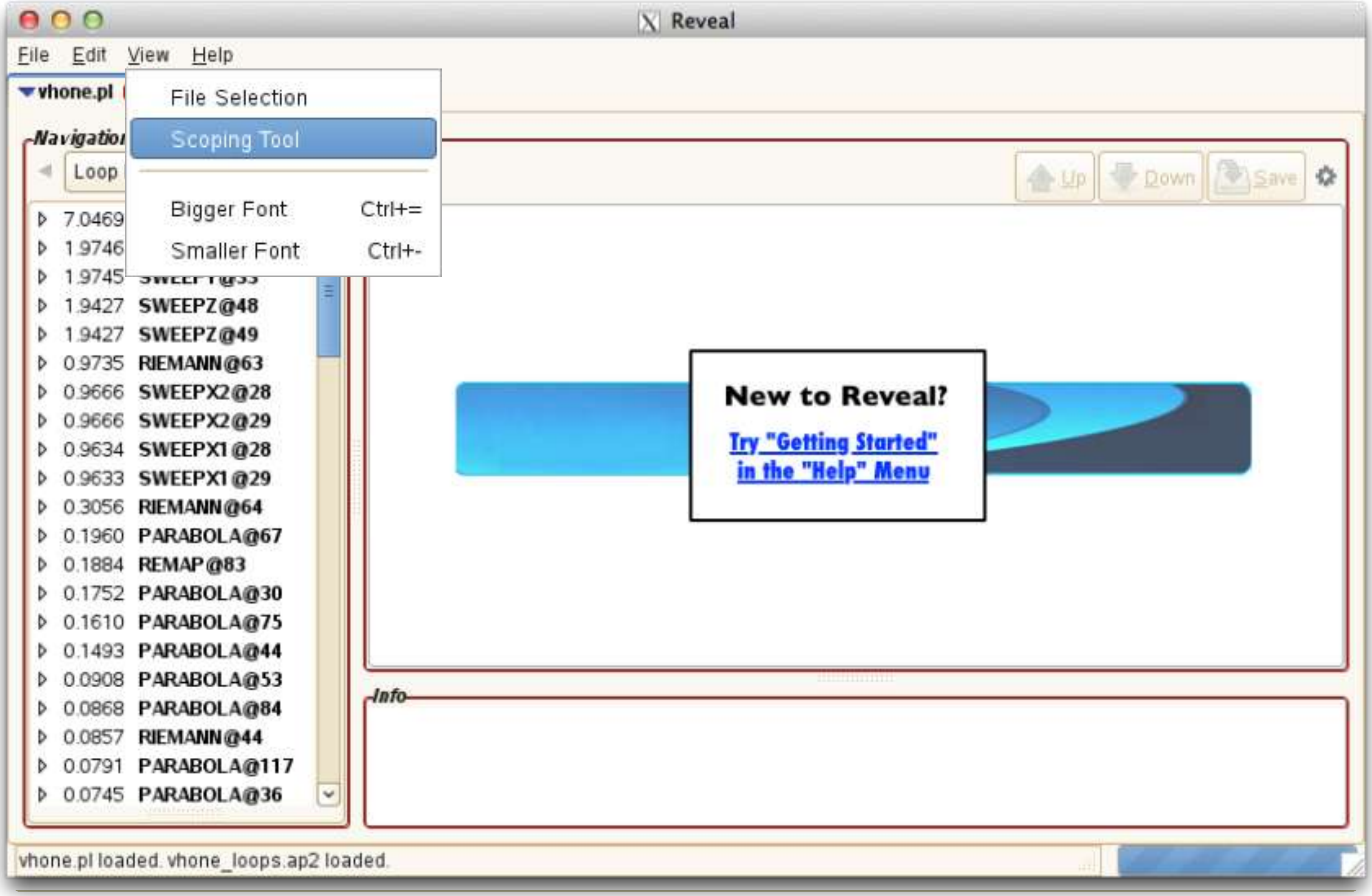
A loop starting at line 63 was not vectorized for an unspecified reason.

vhone.pl loaded. vhone_loops.ap2 loaded.

Loop instances

Loop traceback

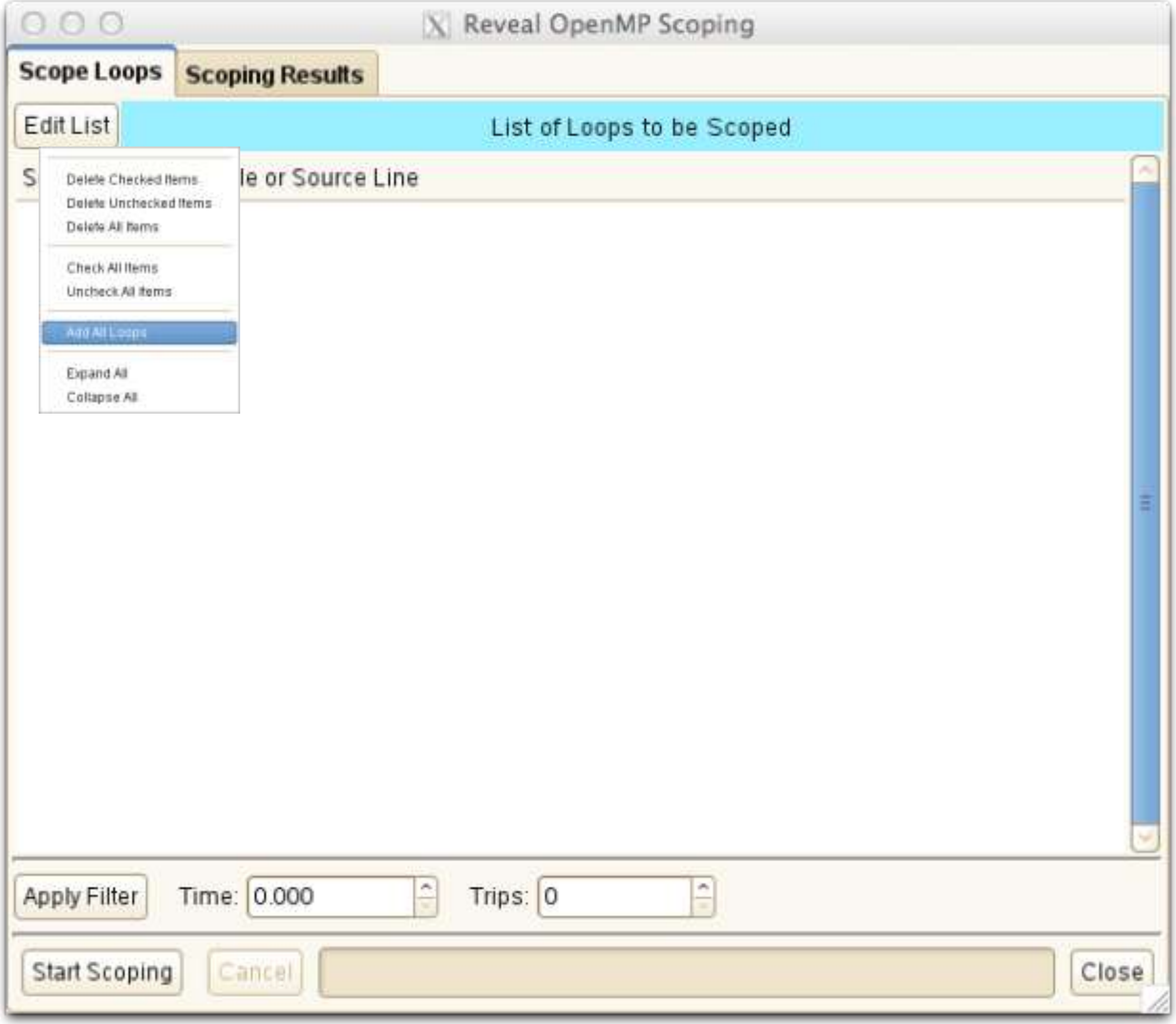
Scope Top Time Consuming Loops



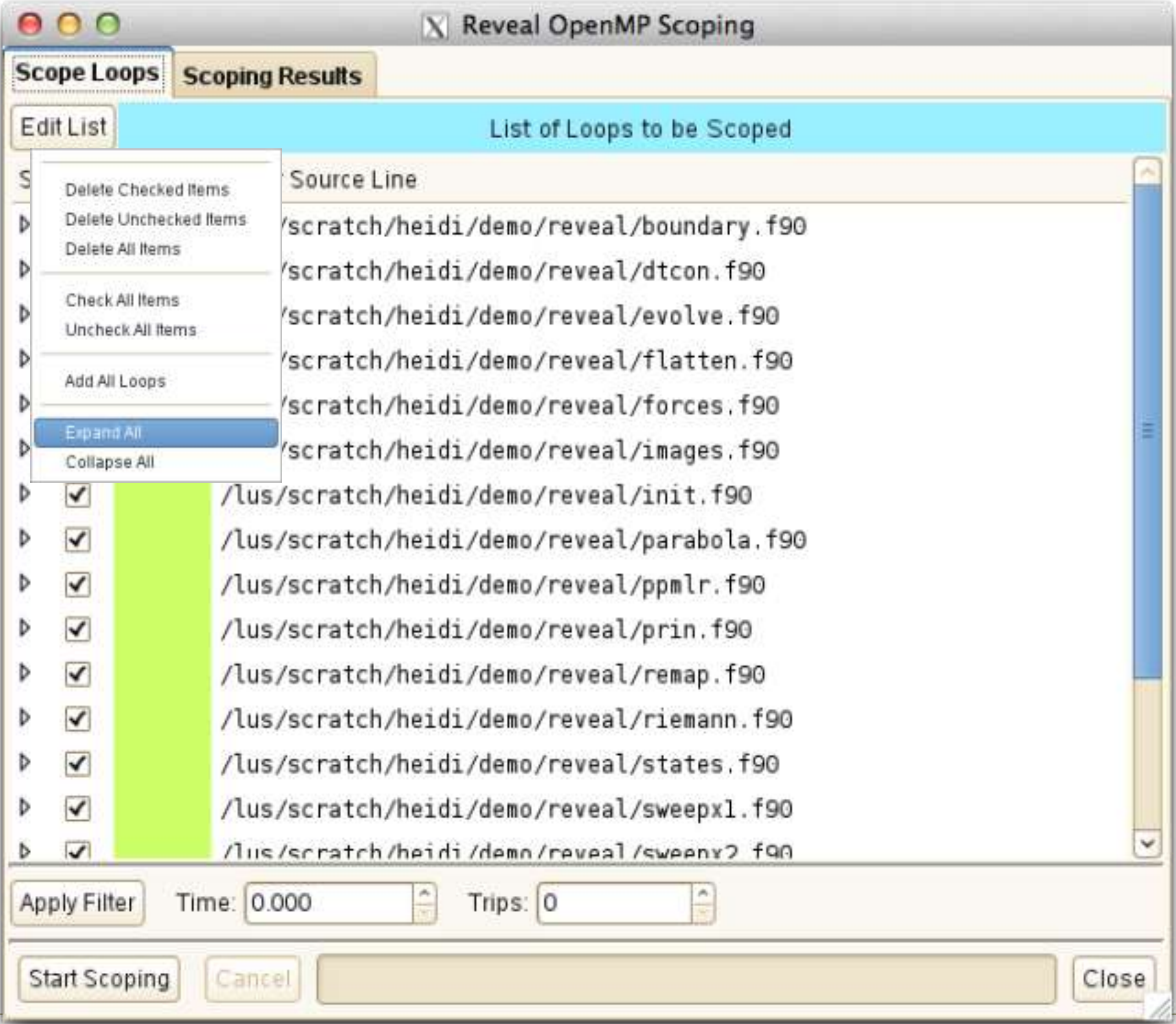
7.0469	
1.9746	
1.9745	SWEEPY@33
1.9427	SWEEPZ@48
1.9427	SWEEPZ@49
0.9735	RIEMANN@63
0.9666	SWEEPX2@28
0.9666	SWEEPX2@29
0.9634	SWEEPX1@28
0.9633	SWEEPX1@29
0.3056	RIEMANN@64
0.1960	PARABOLA@67
0.1884	REMAP@83
0.1752	PARABOLA@30
0.1610	PARABOLA@75
0.1493	PARABOLA@44
0.0908	PARABOLA@53
0.0868	PARABOLA@84
0.0857	RIEMANN@44
0.0791	PARABOLA@117
0.0745	PARABOLA@36

vhone.pl loaded. vhone_loops.ap2 loaded.

Include All Loops as Initial Candidates



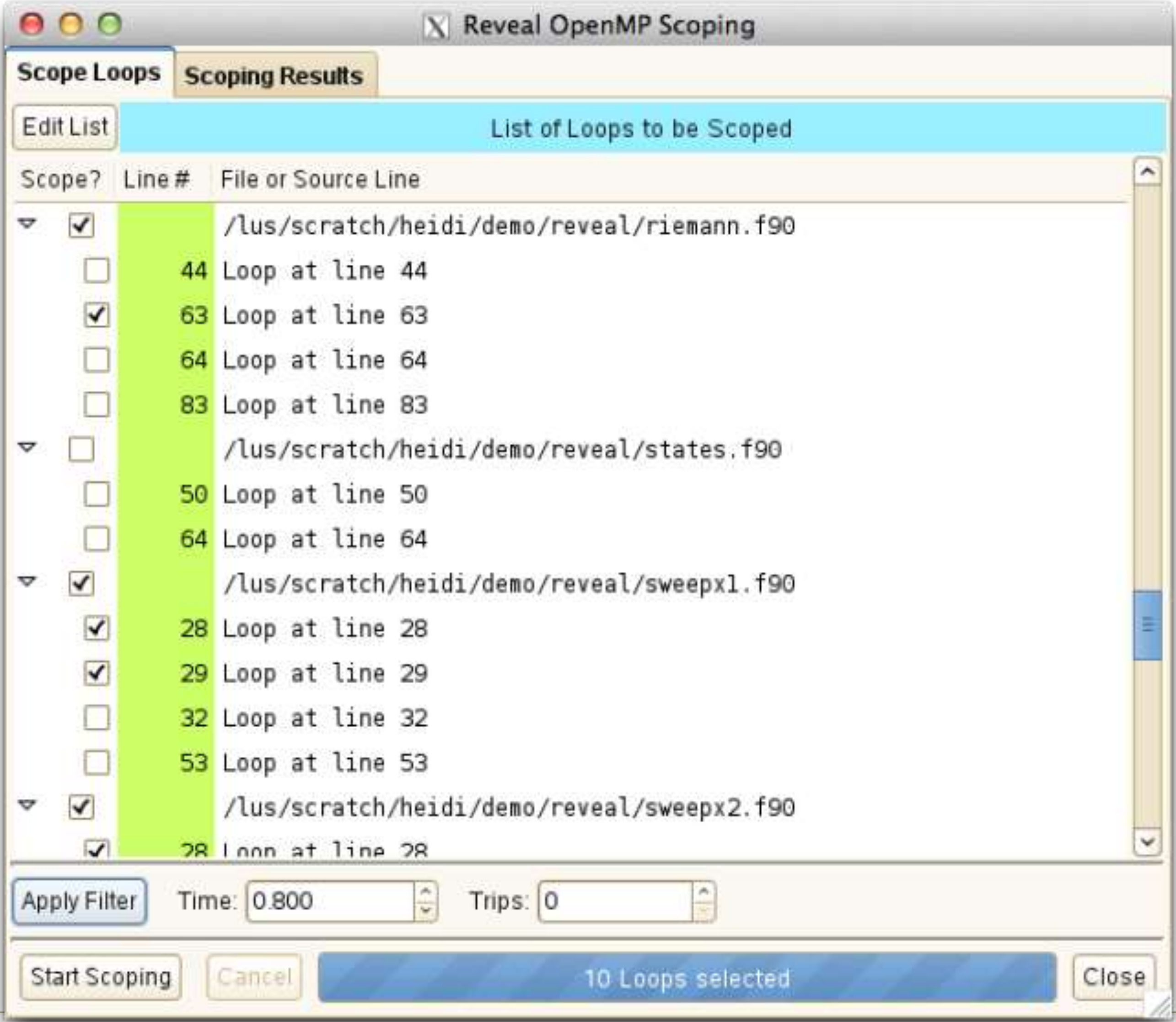
Include All Loops as Initial Candidates (2)



S		Source Line
		/scratch/heidi/demo/reveal/boundary.f90
		/scratch/heidi/demo/reveal/dtcon.f90
		/scratch/heidi/demo/reveal/evolve.f90
		/scratch/heidi/demo/reveal/flatten.f90
		/scratch/heidi/demo/reveal/forces.f90
		/scratch/heidi/demo/reveal/images.f90
☑		/lus/scratch/heidi/demo/reveal/init.f90
☑		/lus/scratch/heidi/demo/reveal/parabola.f90
☑		/lus/scratch/heidi/demo/reveal/ppmlr.f90
☑		/lus/scratch/heidi/demo/reveal/prin.f90
☑		/lus/scratch/heidi/demo/reveal/remap.f90
☑		/lus/scratch/heidi/demo/reveal/riemann.f90
☑		/lus/scratch/heidi/demo/reveal/states.f90
☑		/lus/scratch/heidi/demo/reveal/sweepx1.f90
☑		/lus/scratch/heidi/demo/reveal/sweepx2.f90

Time: 0.000 Trips: 0

Apply Filter to Select Only Top Loops

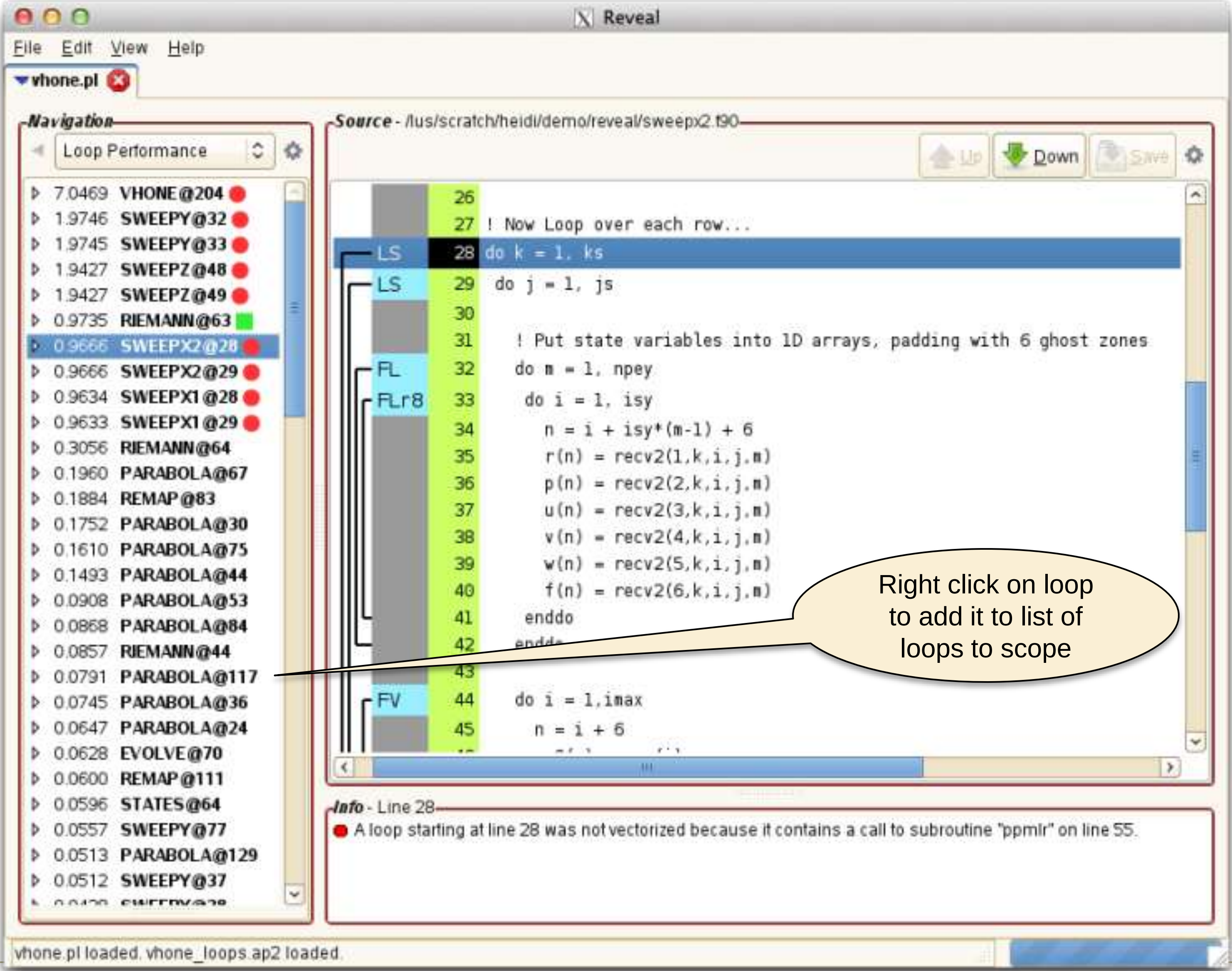


Scope?	Line #	File or Source Line
☑		/lus/scratch/heidi/demo/reveal/riemann.f90
□	44	Loop at line 44
☑	63	Loop at line 63
□	64	Loop at line 64
□	83	Loop at line 83
□		/lus/scratch/heidi/demo/reveal/states.f90
□	50	Loop at line 50
□	64	Loop at line 64
☑		/lus/scratch/heidi/demo/reveal/sweepx1.f90
☑	28	Loop at line 28
☑	29	Loop at line 29
□	32	Loop at line 32
□	53	Loop at line 53
☑		/lus/scratch/heidi/demo/reveal/sweepx2.f90
☑	28	Loop at line 28

Time: 0.800 Trips: 0

10 Loops selected

View Scoping Results



```
7.0469  VHONE@204 ●
1.9746  SWEEPY@32 ●
1.9745  SWEEPY@33 ●
1.9427  SWEEPZ@48 ●
1.9427  SWEEPZ@49 ●
0.9735  RIEMANN@63 🟩
0.9666  SWEEPX2@28 ●
0.9666  SWEEPX2@29 ●
0.9634  SWEEPX1@28 ●
0.9633  SWEEPX1@29 ●
0.3056  RIEMANN@64
0.1960  PARABOLA@67
0.1884  REMAP@83
0.1752  PARABOLA@30
0.1610  PARABOLA@75
0.1493  PARABOLA@44
0.0908  PARABOLA@53
0.0868  PARABOLA@84
0.0857  RIEMANN@44
0.0791  PARABOLA@117
0.0745  PARABOLA@36
0.0647  PARABOLA@24
0.0628  EVOLVE@70
0.0600  REMAP@111
0.0596  STATES@64
0.0557  SWEEPY@77
0.0513  PARABOLA@129
0.0512  SWEEPY@37
```

```
      26
      27  ! Now Loop over each row...
LS    28  do k = 1, ks
LS    29   do j = 1, js
      30
      31     ! Put state variables into 1D arrays, padding with 6 ghost zones
FL    32     do m = 1, npey
FLr8  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
FV    44     do i = 1,imax
      45       n = i + 6
```

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

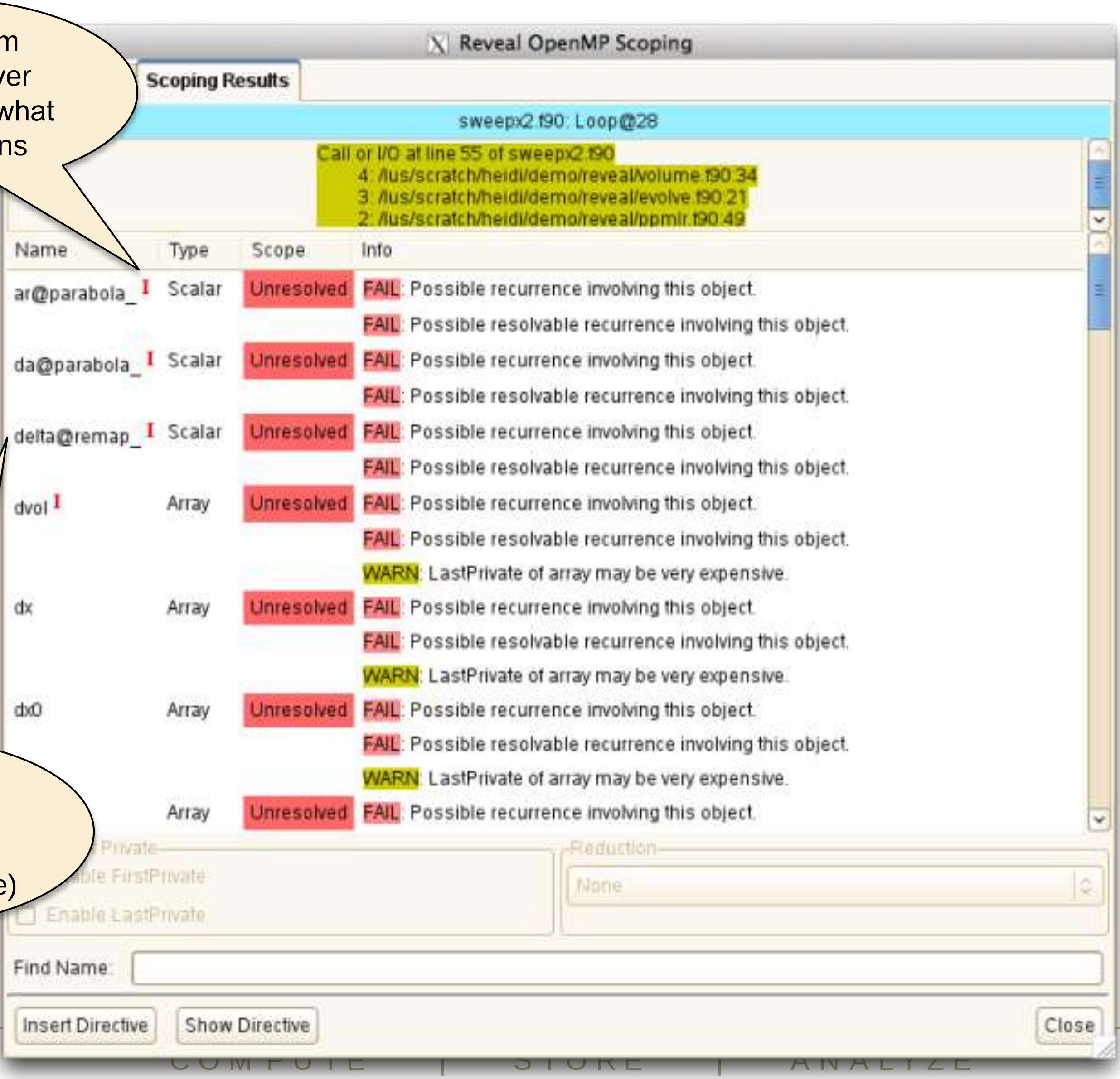
vhone.pl loaded. vhone_loops.ap2 loaded.

Right click on loop to add it to list of loops to scope

COMPUTE | STORE | ANALYZE

Reveal Gives Feedback on Scoping Results

Variable from inlining – hover over ‘I’ to see what symbol means



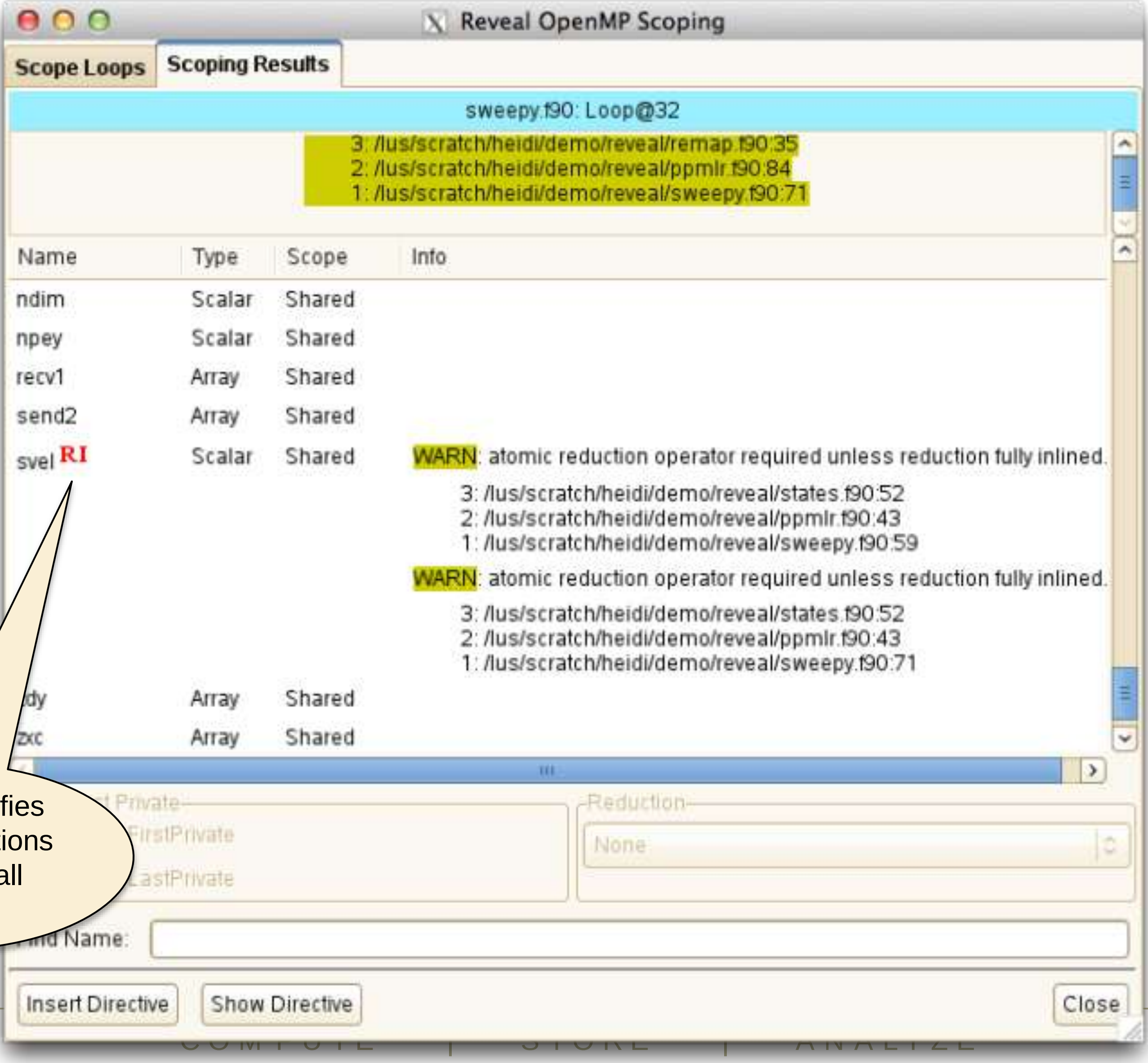
sweepx2.f90: Loop@28

```
Call or I/O at line 55 of sweepx2.f90
    4: /lus/scratch/heidi/demo/reveal/volume.f90:34
    3: /lus/scratch/heidi/demo/reveal/evolve.f90:21
    2: /lus/scratch/heidi/demo/reveal/ppmlr.f90:49
```

Name	Type	Scope	Info
ar@parabola_ I	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
da@parabola_ I	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
delta@remap_ I	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
dvol I	Array	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object. WARN: LastPrivate of array may be very expensive.
dx	Array	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object. WARN: LastPrivate of array may be very expensive.
dx0	Array	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object. WARN: LastPrivate of array may be very expensive.
	Array	Unresolved	FAIL: Possible recurrence involving this object.

See where variable came from (@function_name)

Reveal Points Out Parallelization Issues



sweepy.f90: Loop@32

```
3: /lus/scratch/heidi/demo/reveal/remap.f90:35
2: /lus/scratch/heidi/demo/reveal/ppmlr.f90:84
1: /lus/scratch/heidi/demo/reveal/sweepy.f90:71
```

Name	Type	Scope	Info
ndim	Scalar	Shared	
npey	Scalar	Shared	
recv1	Array	Shared	
send2	Array	Shared	
svel RI	Scalar	Shared	WARN: atomic reduction operator required unless reduction fully inlined. 3: /lus/scratch/heidi/demo/reveal/states.f90:52 2: /lus/scratch/heidi/demo/reveal/ppmlr.f90:43 1: /lus/scratch/heidi/demo/reveal/sweepy.f90:59 WARN: atomic reduction operator required unless reduction fully inlined. 3: /lus/scratch/heidi/demo/reveal/states.f90:52 2: /lus/scratch/heidi/demo/reveal/ppmlr.f90:43 1: /lus/scratch/heidi/demo/reveal/sweepy.f90:71
dy	Array	Shared	
zxc	Array	Shared	

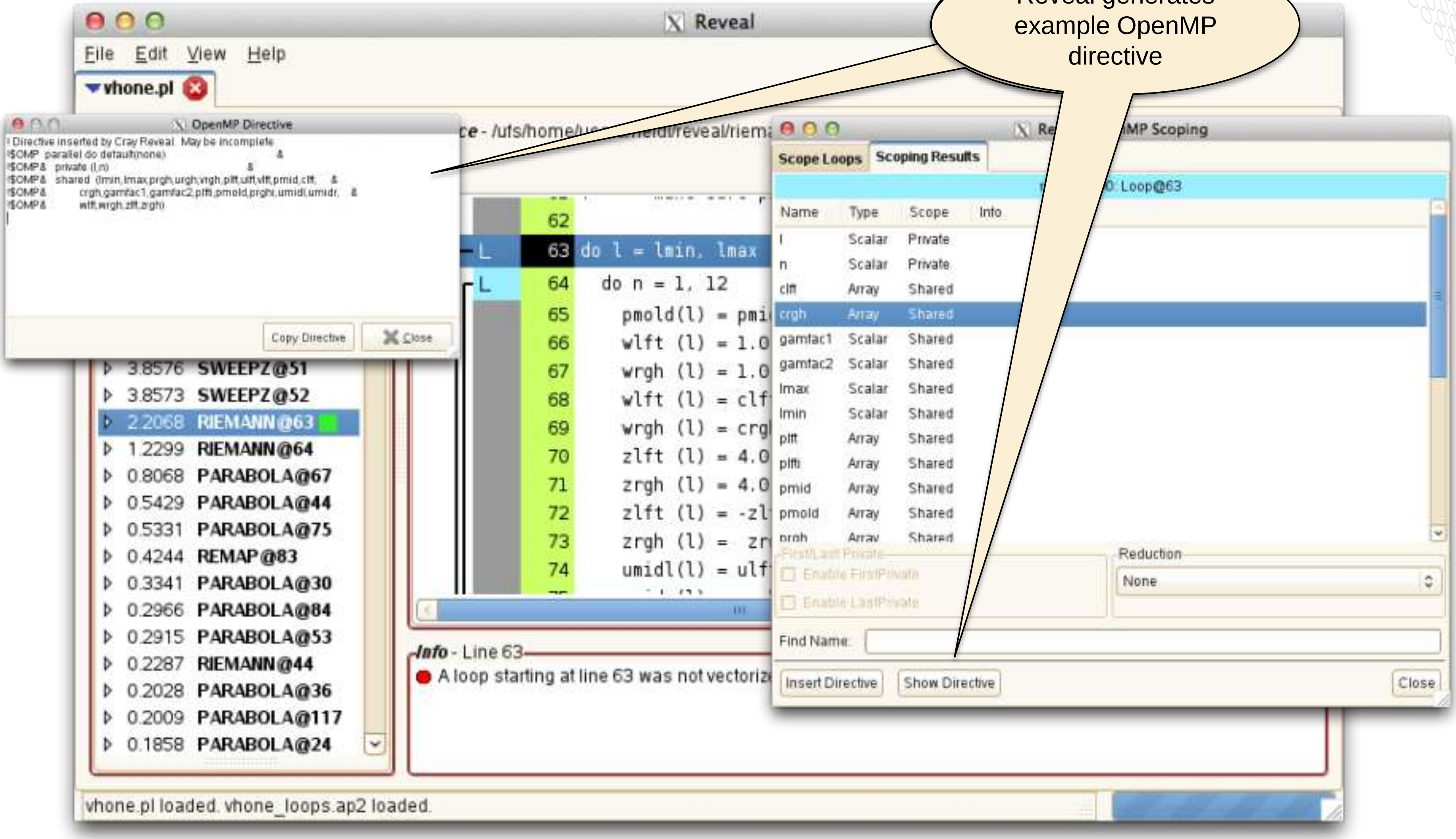
Private
FirstPrivate
LastPrivate

Reduction
None

ind Name:

Reveal identifies shared reductions down the call chain

Generate Directive



```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                                      &
!$OMP&   private (l,n)                                              &
!$OMP&   shared  (lmin,lmax,prgh,urgh,vrgh,plft,ulft,vlft,pmid,clft,    &
!$OMP&          crgh,gamfac1,gamfac2,plfti,pmold,prghi,umidl,umidr,     &
!$OMP&          wlft,wrgh,zlft,zrgh)
```

3.8576	SWEEPZ@51
3.8573	SWEEPZ@52
2.2068	RIEMANN@63
1.2299	RIEMANN@64
0.8068	PARABOLA@67
0.5429	PARABOLA@44
0.5331	PARABOLA@75
0.4244	REMAP@83
0.3341	PARABOLA@30
0.2966	PARABOLA@84
0.2915	PARABOLA@53
0.2287	RIEMANN@44
0.2028	PARABOLA@36
0.2009	PARABOLA@117
0.1858	PARABOLA@24

ce - /ufs/home/u.../reveal/riem...

```
62
63 do l = lmin, lmax
64    do n = 1, 12
65       pmold(l) = pmi
66       wlft (l) = 1.0
67       wrgh (l) = 1.0
68       wlft (l) = clf
69       wrgh (l) = crg
70       zlft (l) = 4.0
71       zrgh (l) = 4.0
72       zlft (l) = -zl
73       zrgh (l) =  zr
74       umidl(l) = ulf
```

Info - Line 63

● A loop starting at line 63 was not vectoriz

Name	Type	Scope	Info
l	Scalar	Private	
n	Scalar	Private	
clft	Array	Shared	
crgh	Array	Shared	
gamfac1	Scalar	Shared	
gamfac2	Scalar	Shared	
lmax	Scalar	Shared	
lmin	Scalar	Shared	
plft	Array	Shared	
plfti	Array	Shared	
pmid	Array	Shared	
pmold	Array	Shared	
prgh	Array	Shared	

First/Last Private

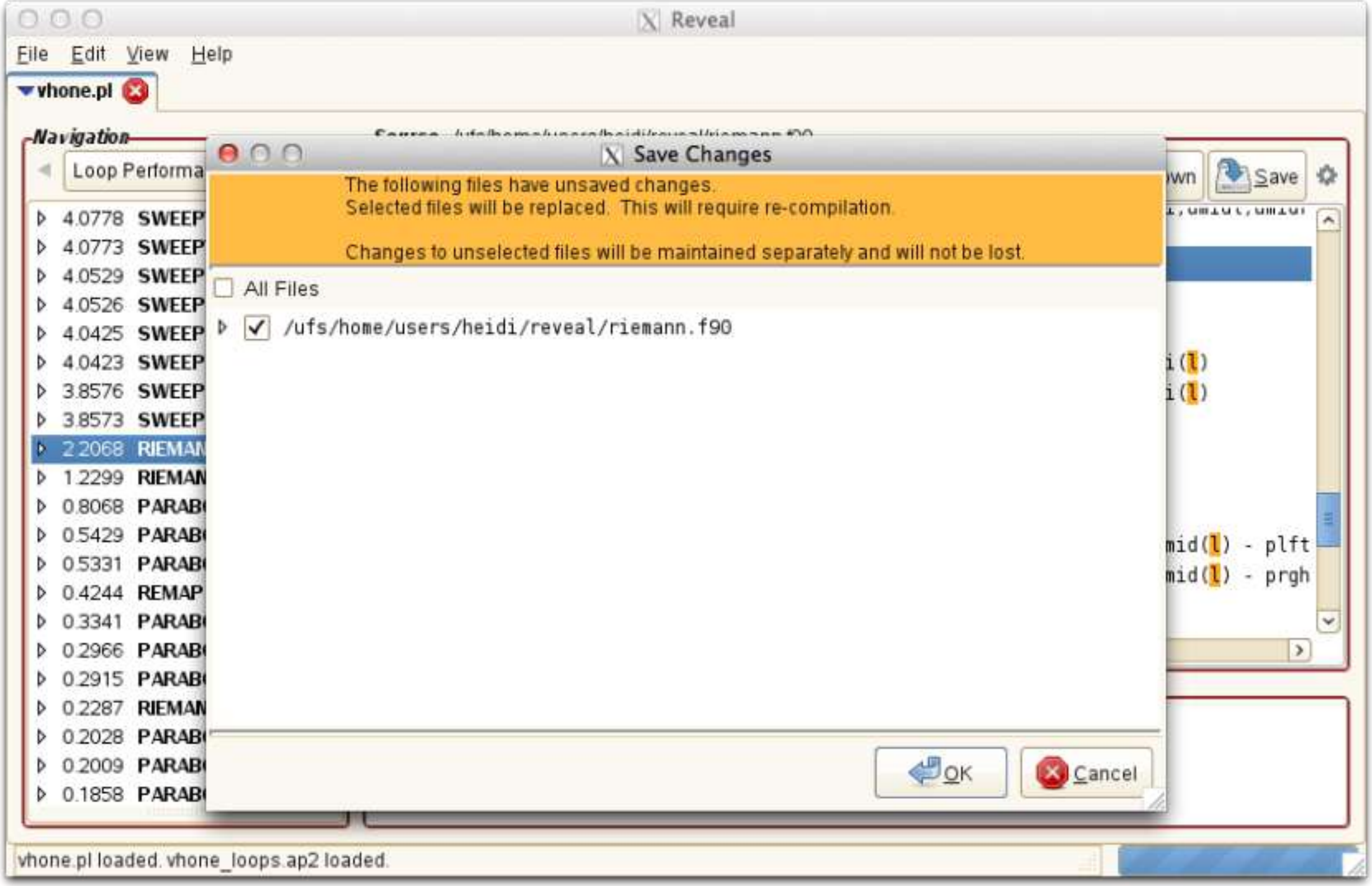
☐ Enable FirstPrivate

☐ Enable LastPrivate

vhone.pl loaded. vhone_loops.ap2 loaded.

Reveal generates example OpenMP directive

Optionally Insert Directive Into Source



The following files have unsaved changes.
Selected files will be replaced. This will require re-compilation.

Changes to unselected files will be maintained separately and will not be lost.

□ All Files

☑ /ufs/home/users/heidi/reveal/riemann.f90

vhone.pl loaded. vhone_loops.ap2 loaded.

Reveal Inserts Directive Into Source

```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                                      &
!$OMP&   unresolved (dvol,dx,dx0,e,f,flat,p,para,q,r,radius,svel,u,v,w, &
!$OMP&            xa,xa0)                                              &
!$OMP&   private (i,j,k,m,n,$$_n,delp2,delp1,shock,temp2,old_flat,     &
!$OMP&            onemfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn, &
!$OMP&            ekin)                                                &
!$OMP&   shared  (gamm,isy,js,ks,mypey,ndim,ngeomy,nlefty,npey,nrighty, &
!$OMP&            recv1,send2,z[illegible]zxc,zya)
do k = 1, ks
 do i = 1, isy
   radius = zxc(i+mypey*isy)

   ! Put state variables into 1D arrays, padding with 6 ghost zones
   do m = 1, npey
    do j = 1, js
     n = j + js*(m-1) + 6
     r(n) = recv1(1,k,j,i,m)
     p(n) = recv1(2,k,j,i,m)
     u(n) = recv1(4,k,j,i,m)
     v(n) = recv1(5,k,j,i,m)
     w(n) = recv1(3,k,j,i,m)
     f(n) = recv1(6,k,j,i,m)
    enddo
   enddo

   do j = 1, jmax
     n = j + 6
     …
```

Reveal generates OpenMP directive with illegal clause marking variables that need addressing

Resolve Private Array Concerns for dvol, etc.

```
From file vh1mods.f90:
. . .

! module sweeps
!========================================================================
! Data structures used in 1D sweeps, dimensioned maxsweep  (set in sweepsize.mod)
!----------------------------------------------------------------------

use sweepsize

integer :: nmin, nmax, ngeom, nleft, nright                   ! number of first and last real zone
real, dimension(maxsweep) :: r, p, e, q, u, v, w              ! fluid variables
real, dimension(maxsweep) :: xa, xa0, dx, dx0, dvol           ! coordinate values
real, dimension(maxsweep) :: f, flat                          ! flattening parameter
real, dimension(maxsweep,5) :: para                           ! parabolic interpolation coefficients
real :: radius, theta, stheta

!$omp threadprivate(dvol,dx,dx0,e,f,flat,p,para,q,r,radius,theta,stheta,u,v,w,xa,xa0)
```

For OpenMP these need to be made task_private

Resolve Shared Reductions

Original

```
hdt    = 0.5*dt
do n = nmin-4, nmax+4
  Cdtdx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
  svel      = max(svel,Cdtdx(n))
  Cdtdx (n) = Cdtdx(n)*hdt
  fCdtdx(n) = 1. - fourthd*Cdtdx(n)
enddo
```

Restructured – One Approach

```
hdt    = 0.5*dt
!$omp critical
do n = nmin-4, nmax+4
  Cdtdx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
  svel      = max(svel,Cdtdx(n))
  Cdtdx (n) = Cdtdx(n)*hdt
  fCdtdx(n) = 1. - fourthd*Cdtdx(n)
enddo
!$omp end critical
```

For OpenMP need to have a critical region around setting of svel

Resolve Shared Reductions (Continued)

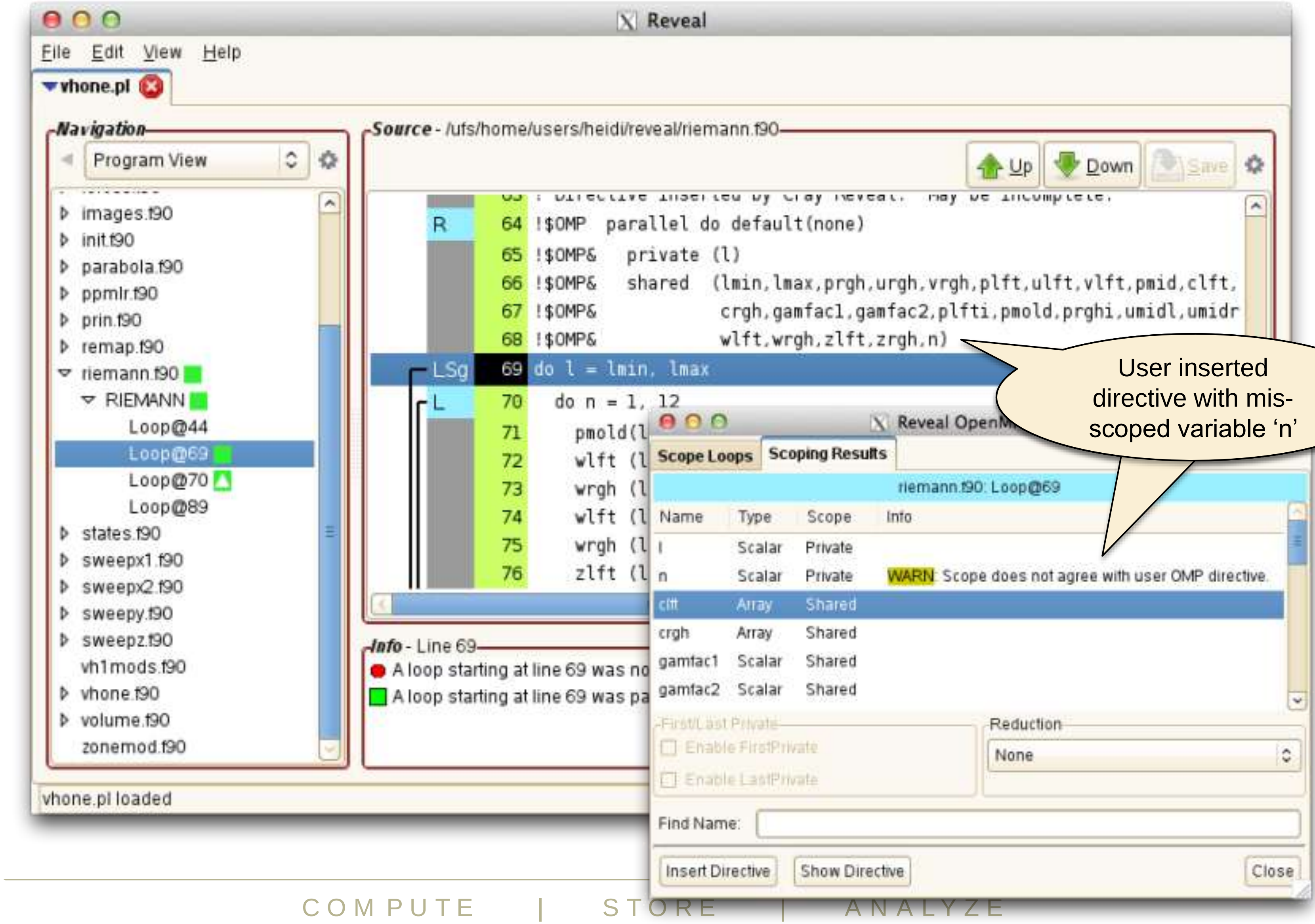
Original

```
hdt   = 0.5*dt
do n = nmin-4, nmax+4
  Cdtdx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
  svel      = max(svel,Cdtdx(n))
  Cdtdx (n) = Cdtdx(n)*hdt
  fCdtdx(n) = 1. - fourthd*Cdtdx(n)
enddo
```

Restructured – Better Approach

```
hdt   = 0.5*dt
Svel0 = 0.0
do n = nmin-4, nmax+4
  Cdtdx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
  svel0(n)      = max(svel(n),Cdtdx(n))
  Cdtdx (n) = Cdtdx(n)*hdt
  fCdtdx(n) = 1. - fourthd*Cdtdx(n)
Enddo
!$omp critical
Do n = nmin-4, nmax +4
  Svel = max(svel0(n),svel)
Enddo
!$omp end critical
```

Use Reveal to Validate User Inserted Directives



Source - /ufs/home/users/heidi/reveal/riemann.f90

Navigation:

- images.f90
- init.f90
- parabola.f90
- ppmlr.f90
- prin.f90
- remap.f90
- riemann.f90 🟩
  - RIEMANN 🟩
    - Loop@44
    - Loop@69 🟩
    - Loop@70
    - Loop@89
- states.f90
- sweepx1.f90
- sweepx2.f90
- sweepy.f90
- sweepz.f90
- vh1mods.f90
- vhone.f90
- volume.f90
- zonemod.f90

```
     63 ! Directive inserted by Cray Reveal.  May be incomplete.
R    64 !$OMP  parallel do default(none)
     65 !$OMP&   private (l)
     66 !$OMP&   shared  (lmin,lmax,prgh,urgh,vrgh,plft,ulft,vlft,pmid,clft,
     67 !$OMP&             crgh,gamfac1,gamfac2,plfti,pmold,prghi,umidl,umidr
     68 !$OMP&             wlft,wrgh,zlft,zrgh,n)
LSg  69 do l = lmin, lmax
L    70   do n = 1, 12
     71     pmold(l
     72     wlft (l
     73     wrgh (l
     74     wlft (l
     75     wrgh (l
     76     zlft (l
```

Info - Line 69

- 🟥 A loop starting at line 69 was no
- 🟩 A loop starting at line 69 was pa

riemann.f90: Loop@69

Name	Type	Scope	Info
l	Scalar	Private	
n	Scalar	Private	WARN: Scope does not agree with user OMP directive.
clft	Array	Shared	
crgh	Array	Shared	
gamfac1	Scalar	Shared	
gamfac2	Scalar	Shared	

First/Last Private

☐ Enable FirstPrivate

☐ Enable LastPrivate

Reduction: None

Find Name:

Insert Directive | Show Directive | Close

vhone.pl loaded

User inserted directive with mis-scoped variable ‘n’

VH1 – Astrophysics Code

- **VH1 is written with high level loops and complex decision processes.**
- **Ported to hybrid MPI + OpenMP using Reveal**
- **Reveal was able to identify**
  - storage conflicts
  - private variables in modules
  - reductions down the call chain that require critical regions
- **Scoping was performed in seconds where it would have taken weeks to get correct without Reveal**

S3D - Structured Cartesian Mesh Flow Solver

- S3D, a pure MPI program, was converted to a hybrid multi-core application suited for a multi-core node with or without an accelerator.
- When the work was started, Reveal did not exist.
- Once Reveal was available, it was instrumental in identifying bugs in the scoping of extremely large loops (3000 lines of Fortran).
- There are both OpenMP and OpenACC versions of S3D that run well on both OpenMP systems and on the Titan Cray XK7 machine at Oak Ridge National Laboratory.

Summary

- Reveal can be used to simplify the task of adding OpenMP to MPI programs

- Can be used as a stepping stone for codes targeted for nodes with higher core counts and as the first step in adding OpenACC to applications to for execution on GPUs

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.