Cray Reveal: A Tool to Help Porting to Many-core and GPUs

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**

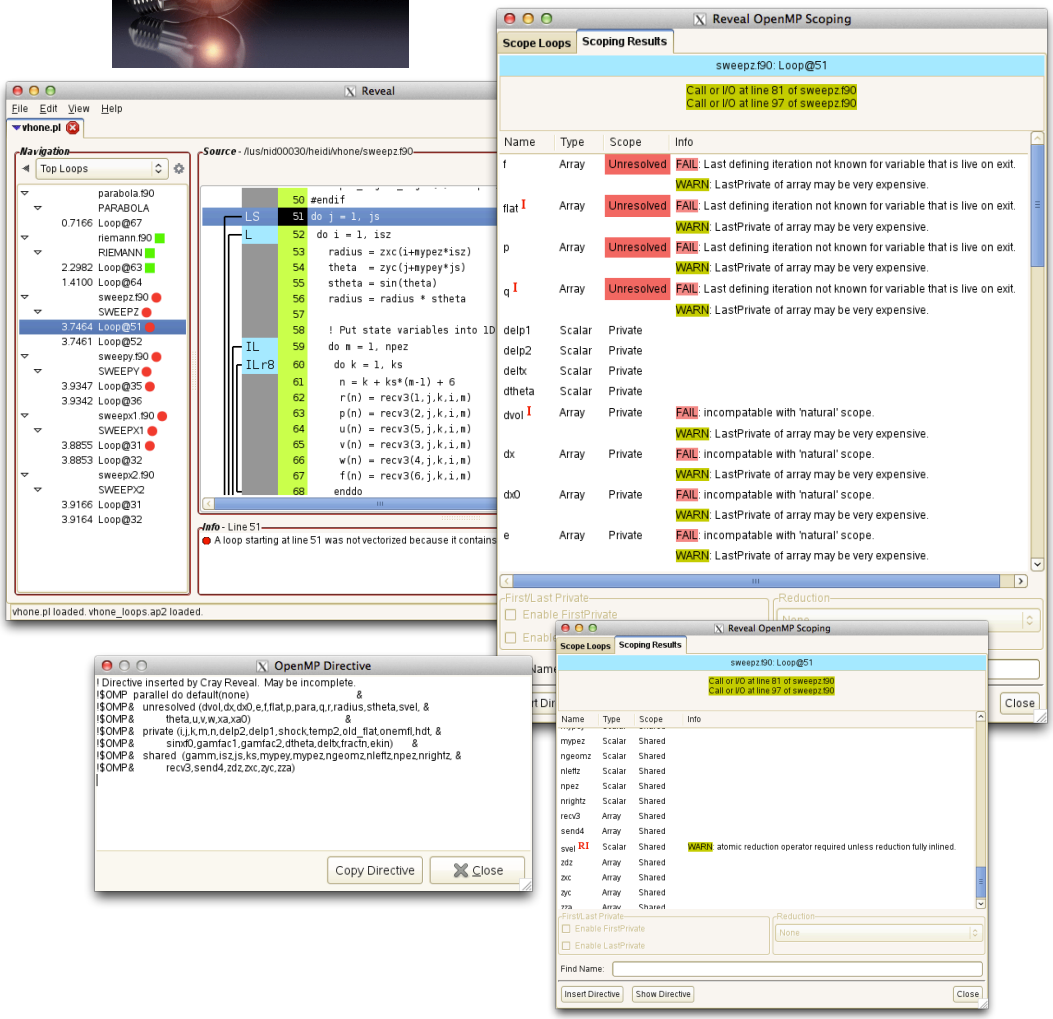
The Problem – How Do I Parallelize This Loop?

- 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



- 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

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 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

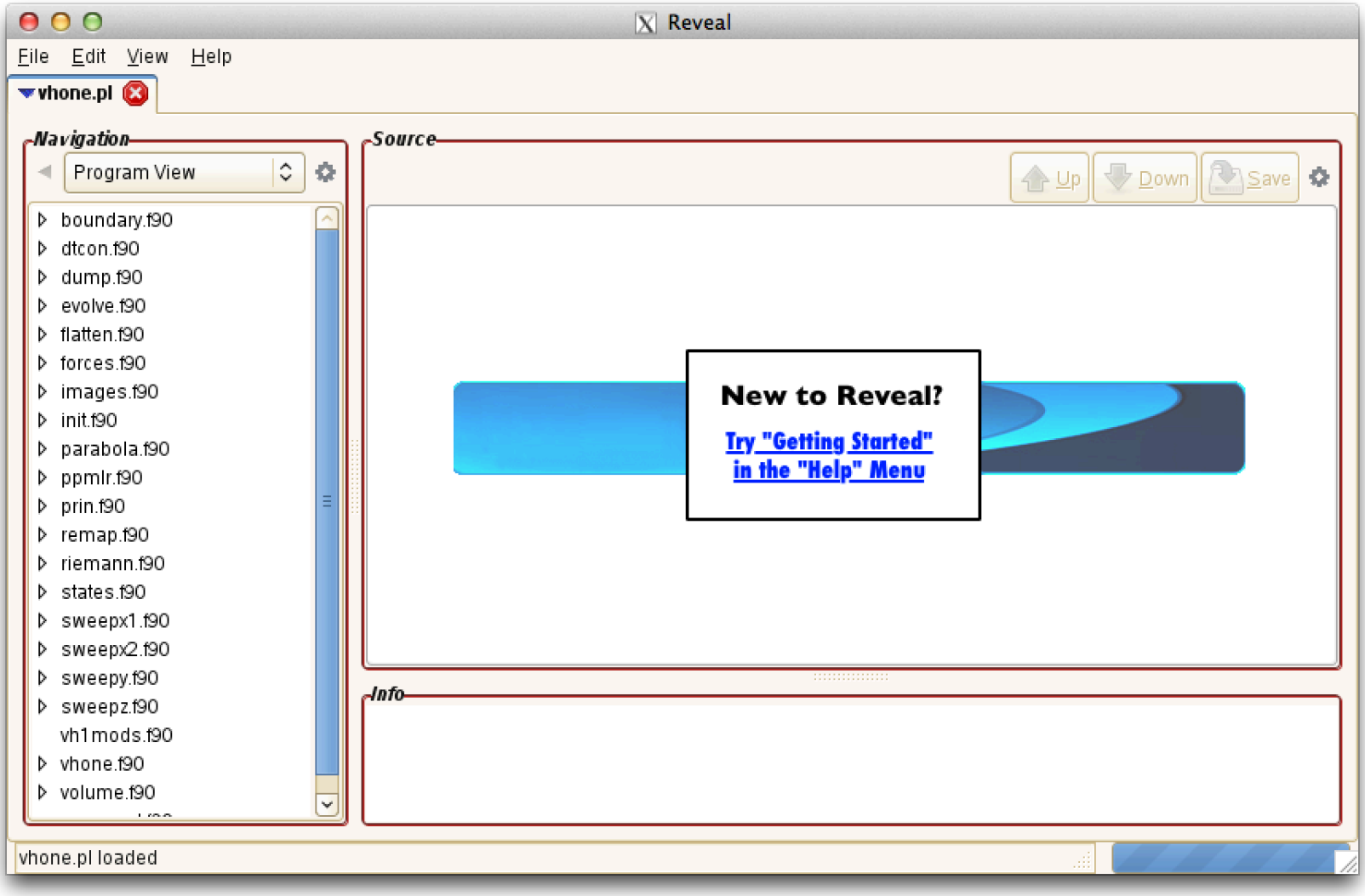
How to Use Reveal

- **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`

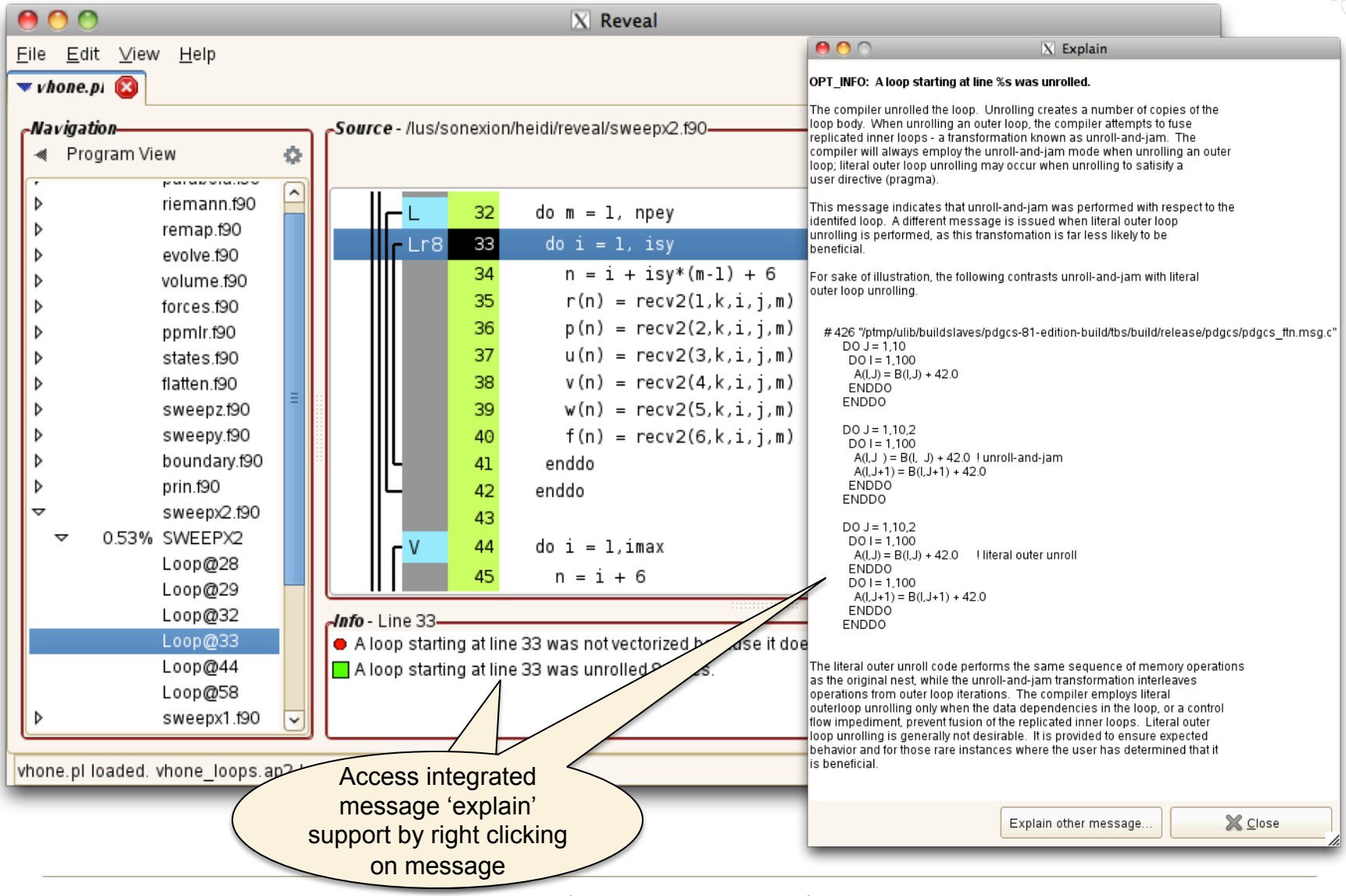
Optionally add whole program analysis for more aggressive inlining

- **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`

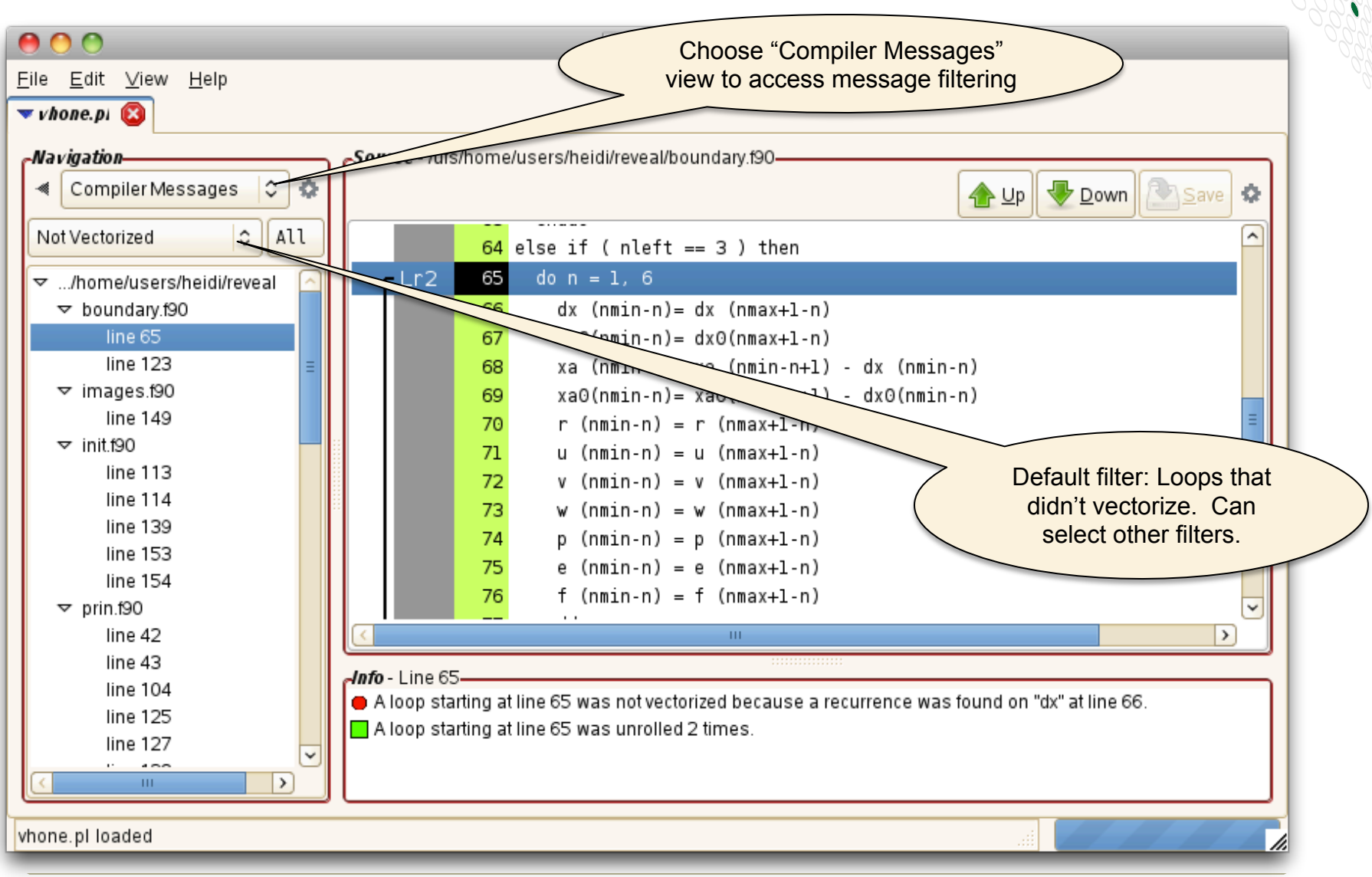
Browse Source and Compiler Optimizations



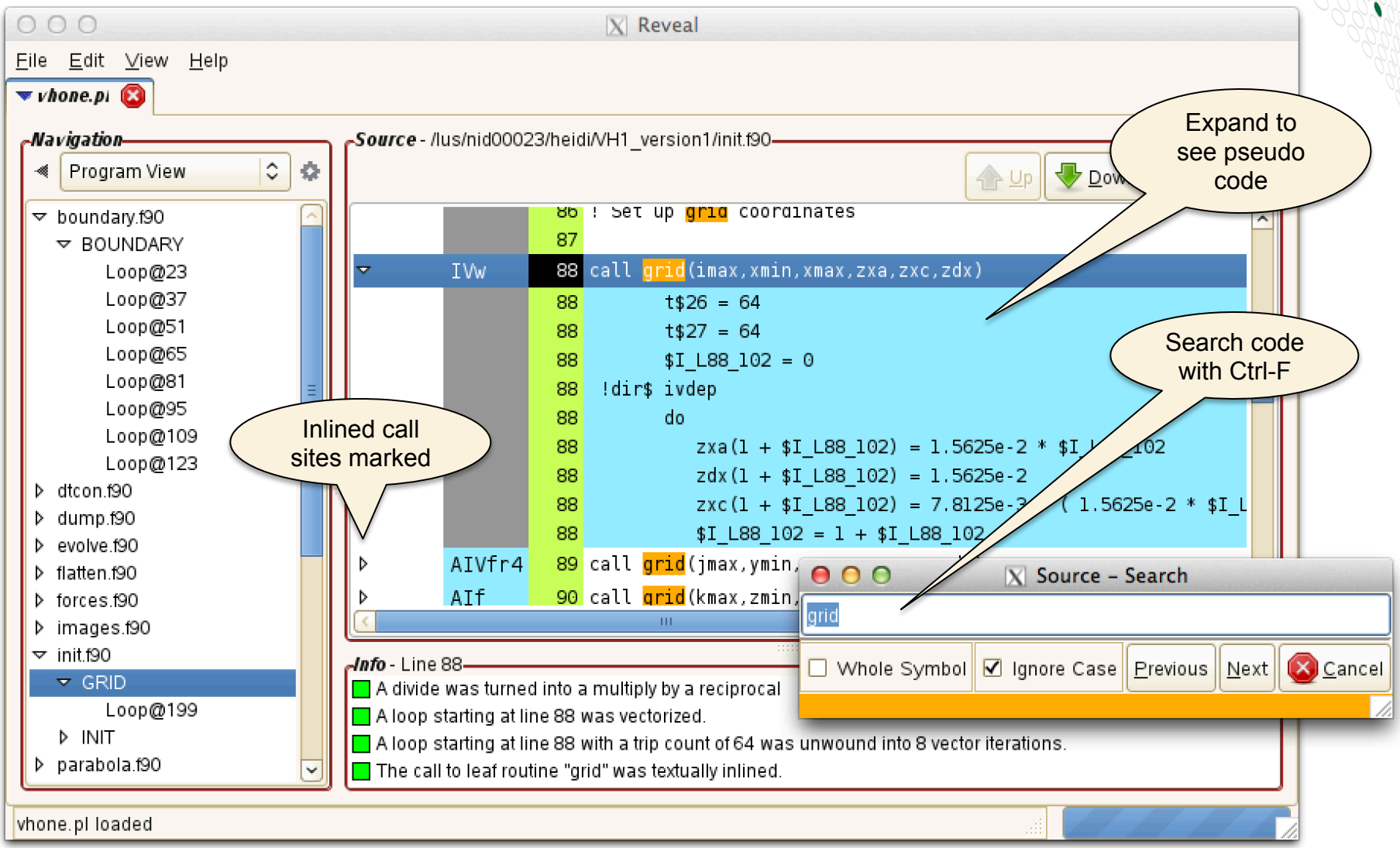
Access Cray Compiler Message Information



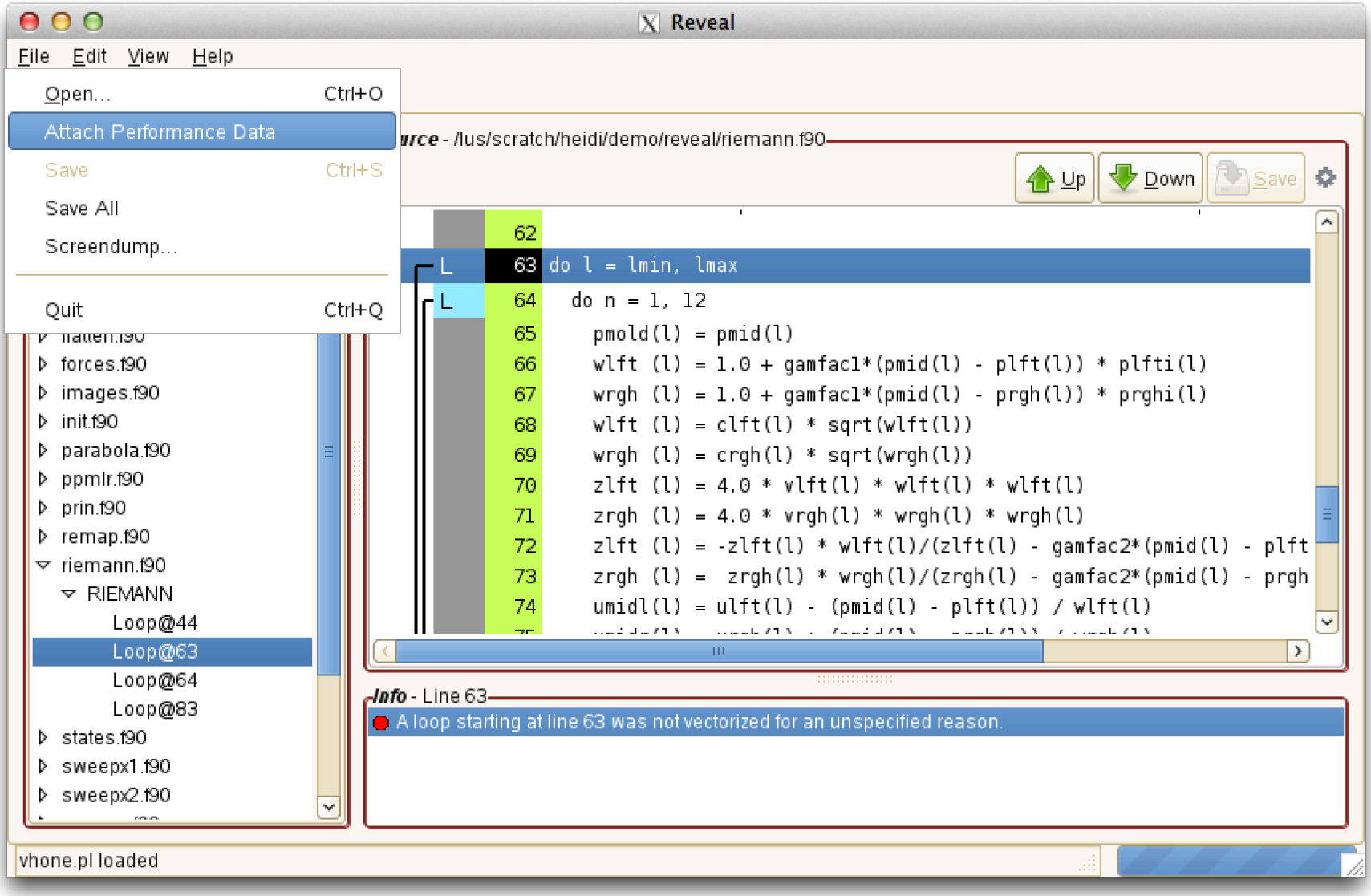
Navigate Code via Compiler Messages



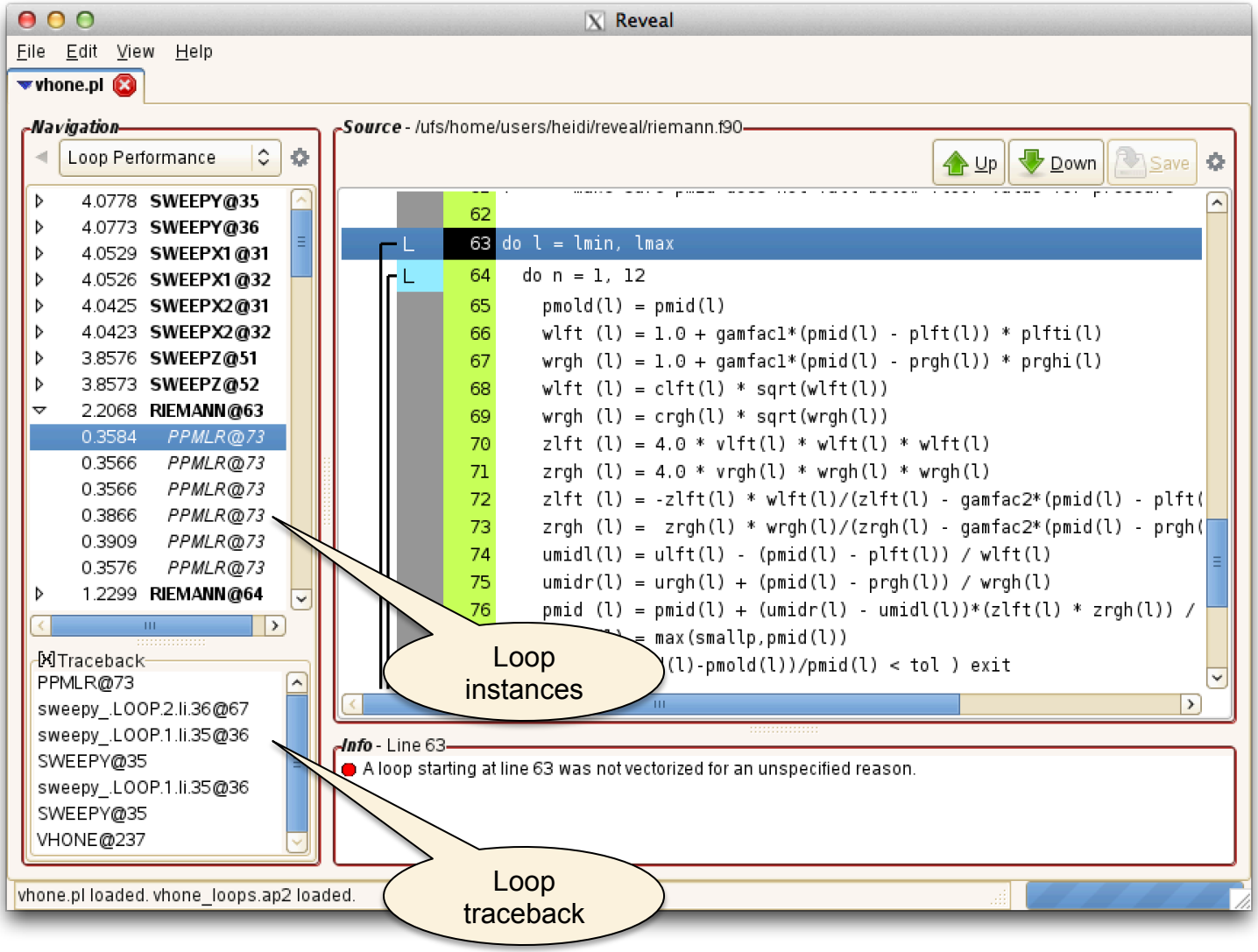
View Pseudo Code for Inlined Functions



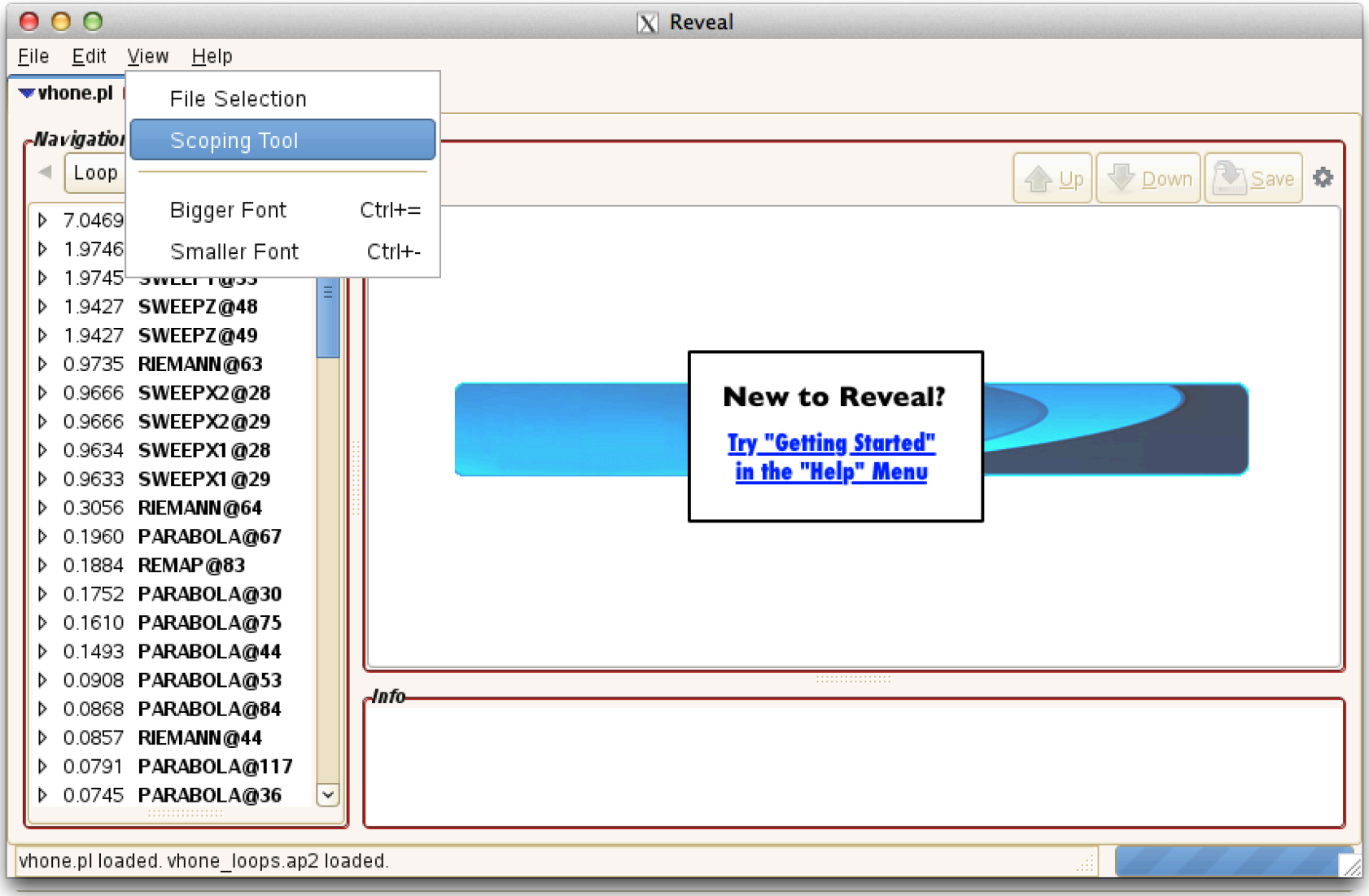
Add Performance Data to Find Top Loops



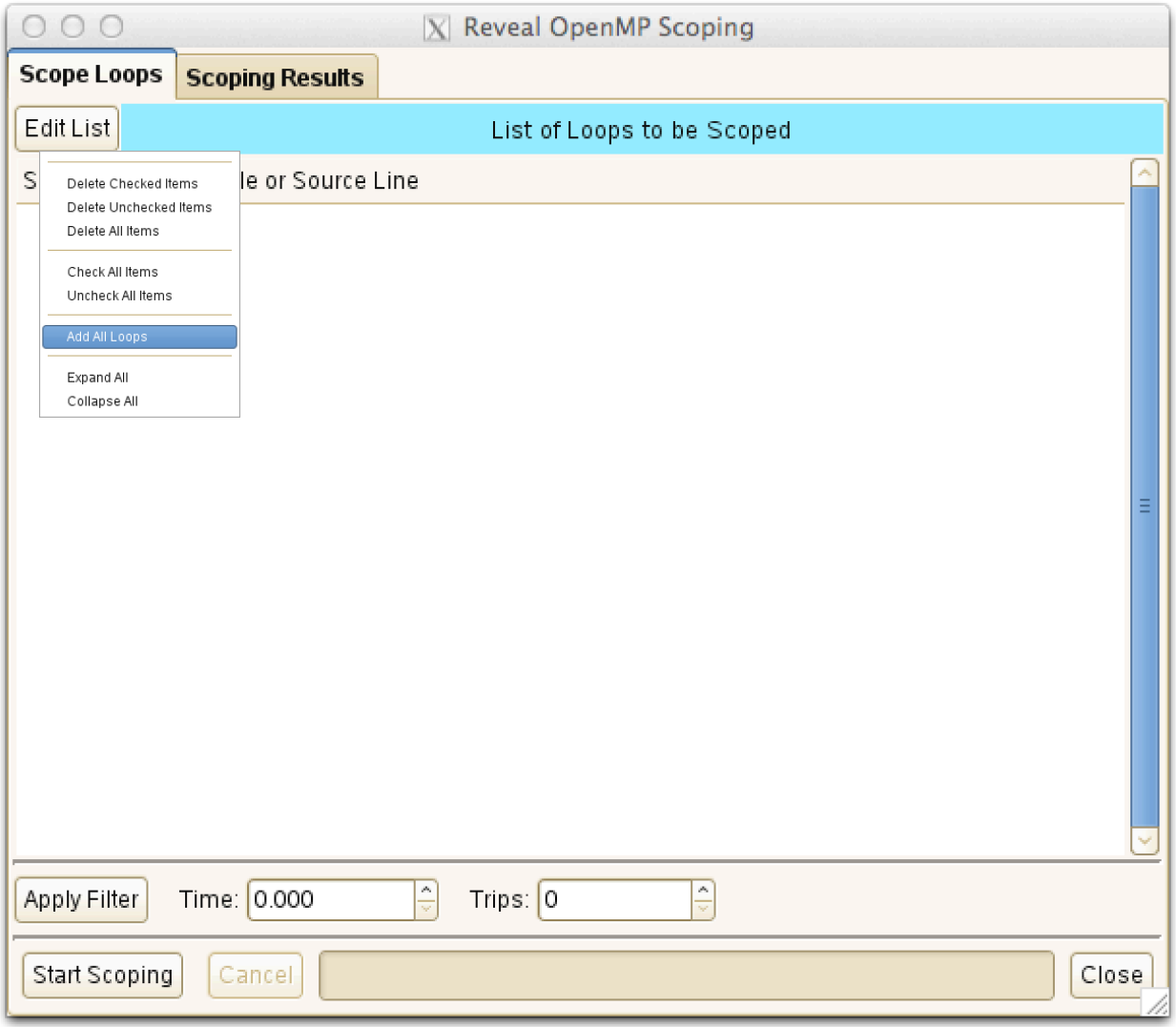
View Loops through Call Chain



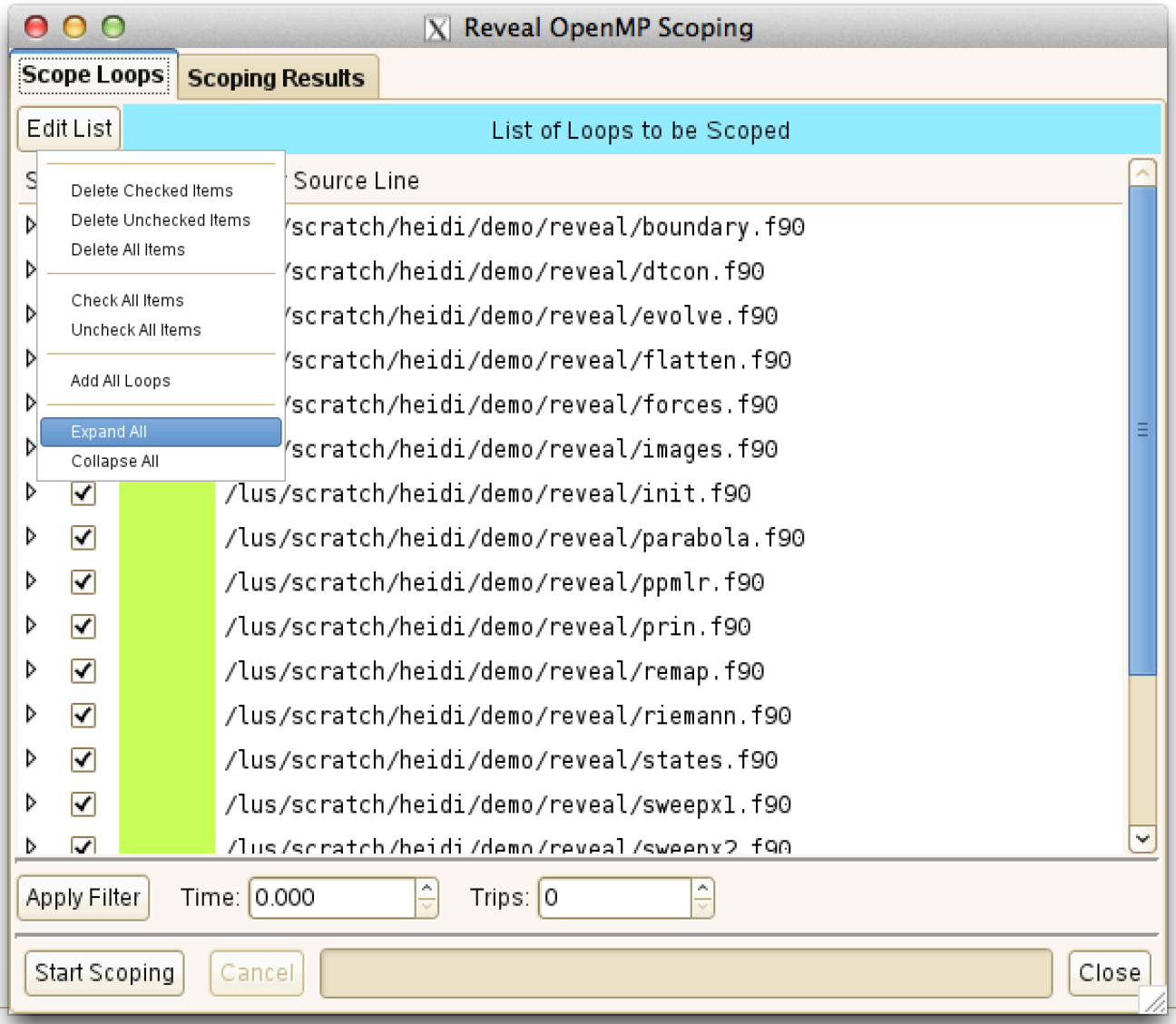
Scope Top Time Consuming Loops



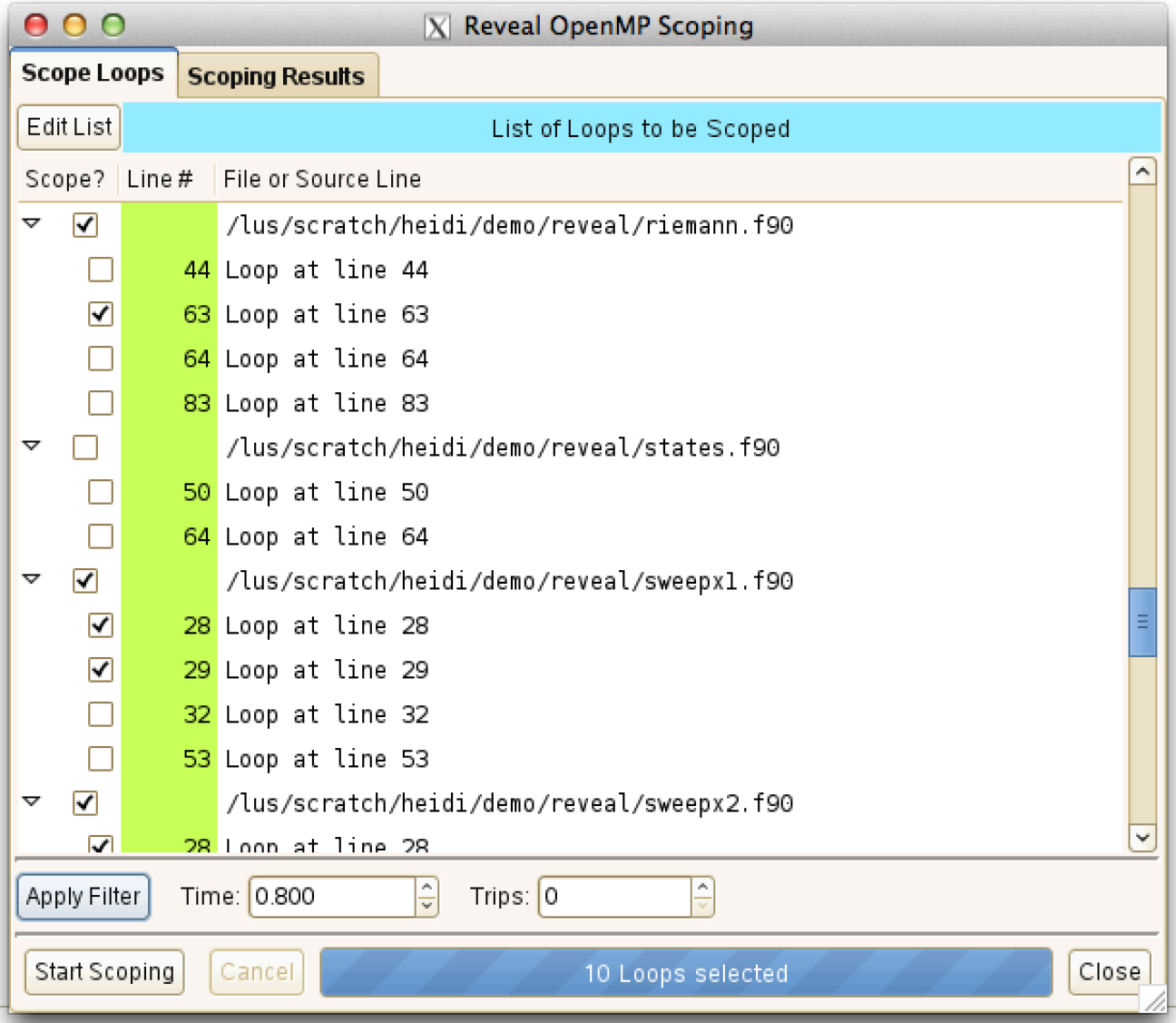
Include All Loops as Initial Candidates



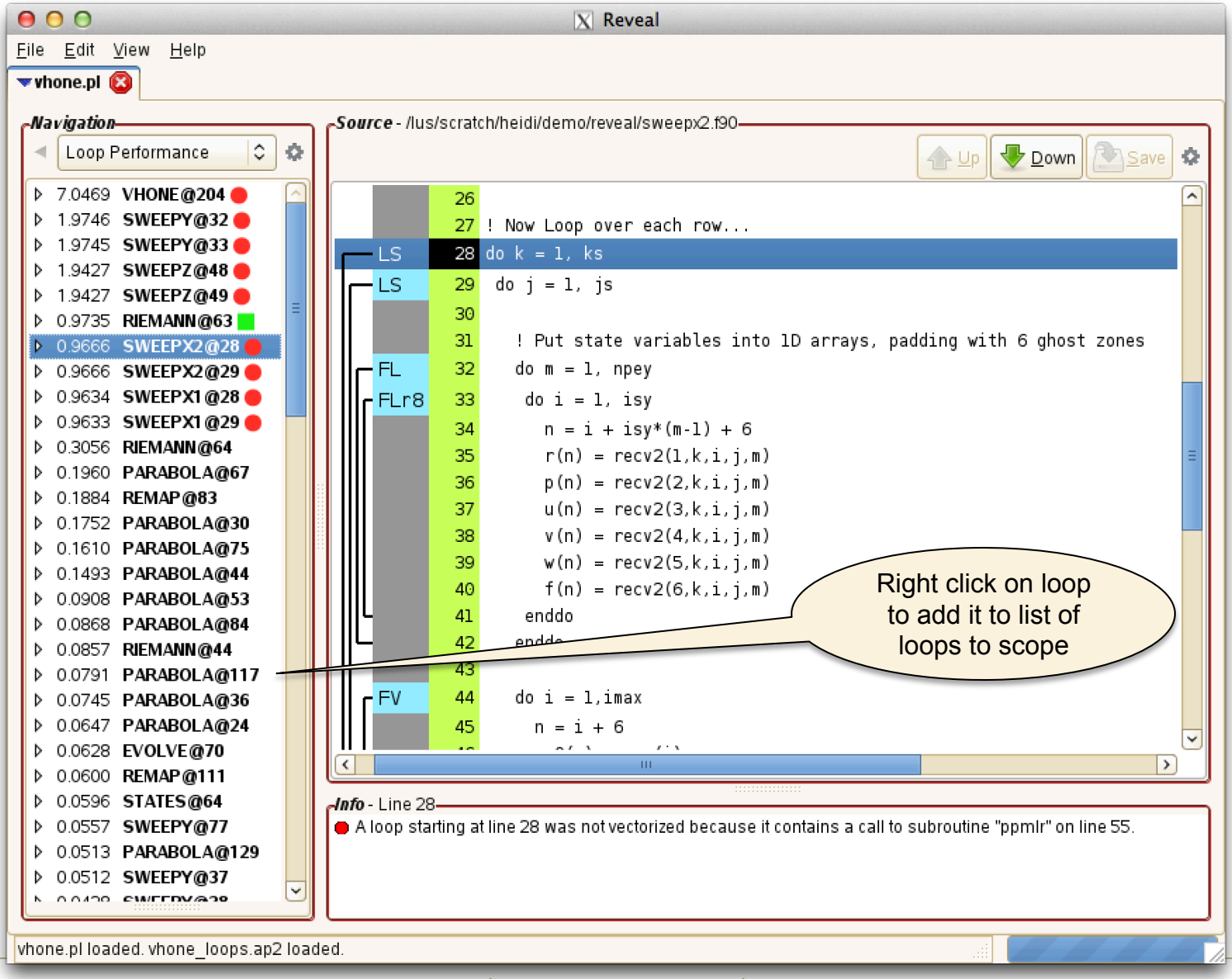
Include All Loops as Initial Candidates (2)



Apply Filter to Select Only Top Loops



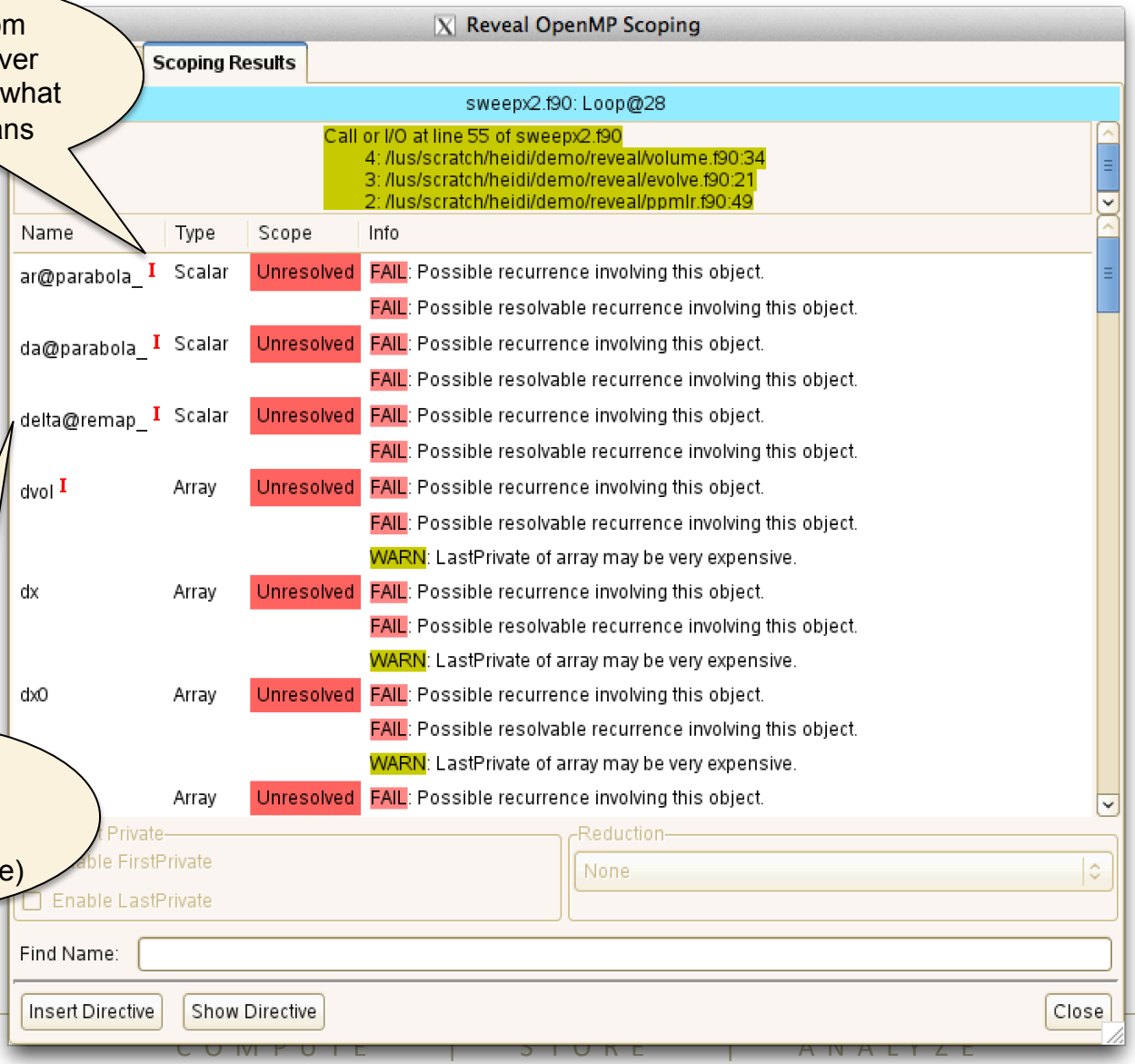
View Scoping Results



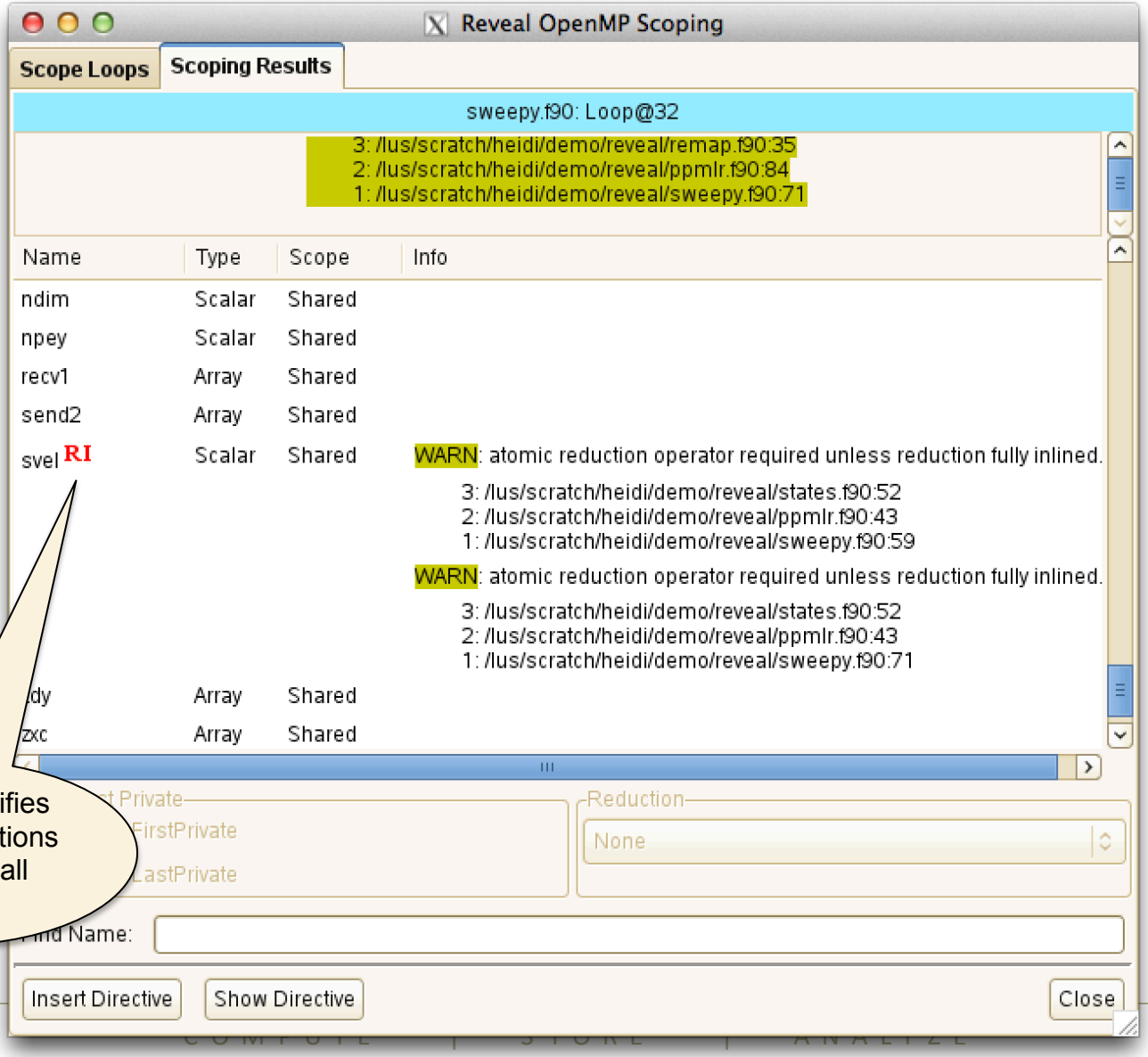
Reveal Gives Feedback on Scoping Results

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

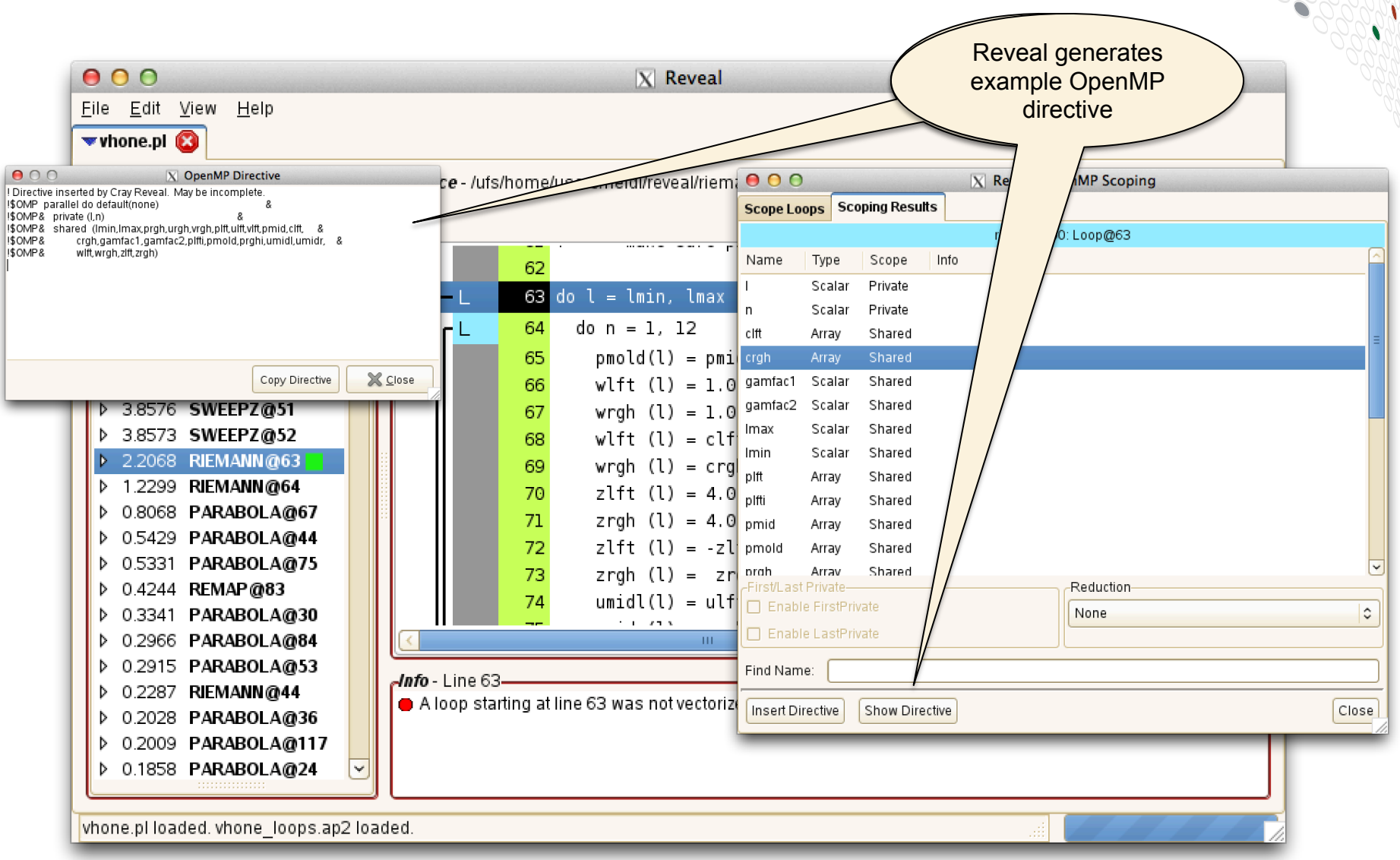
See where variable came from (@function_name)



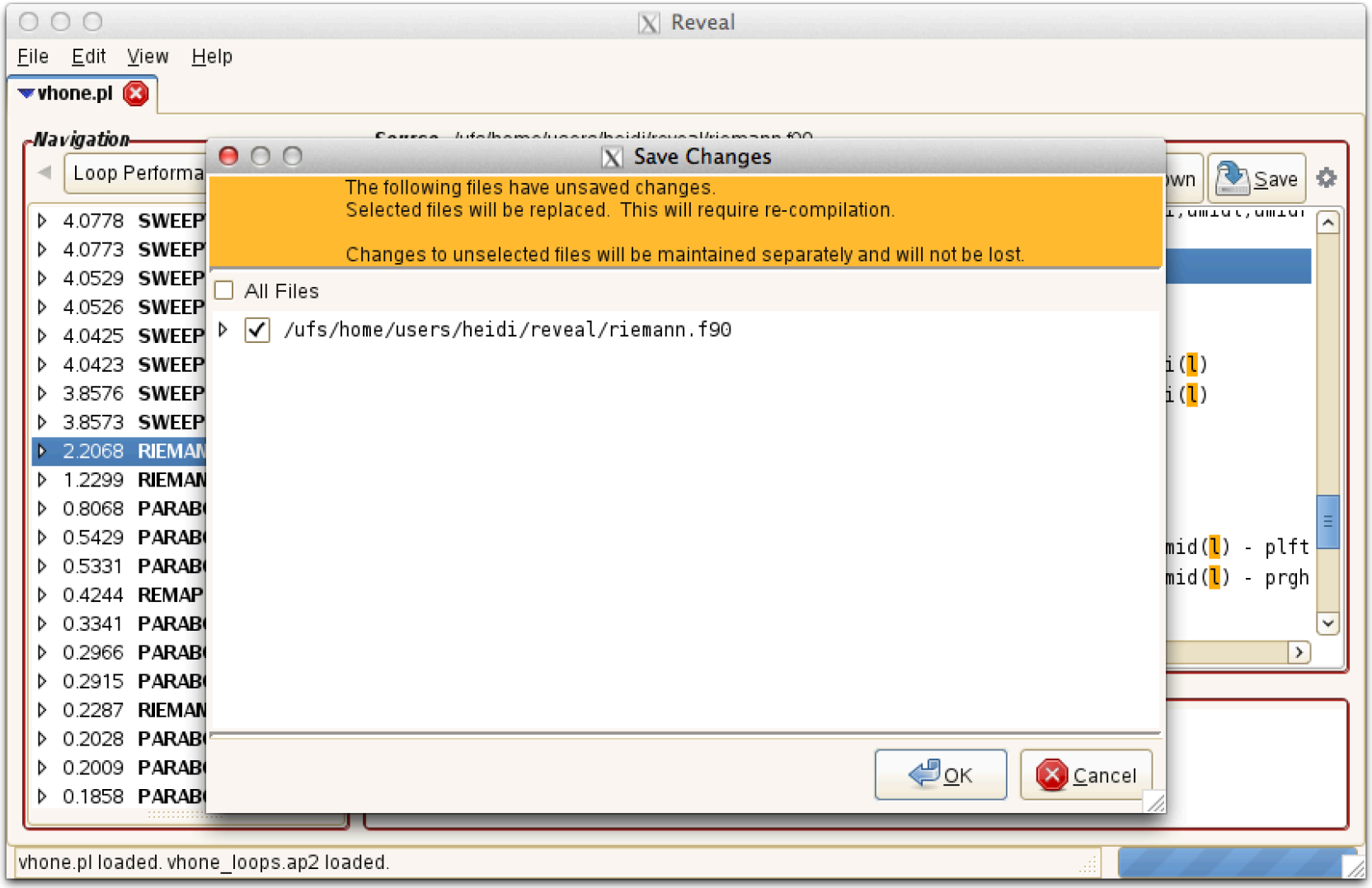
Reveal Points Out Parallelization Issues



Generate Directive



Optionally Insert Directive Into Source



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,       &
!$OMP&             o  mfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn, &
!$OMP&             ekin                                                  &
!$OMP&   shared   (gamm,i  js,ks,mypey,ndim,ngeomy,nlefty,npey,nrighty,  &
!$OMP&             recv1,sen  zdy,zxc,zya)
do k = 1, ks
 do i = 1, isy
   radius = zxc(i+mypey*isy)

   ! Put state variables into 1D arrays,  ing 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)

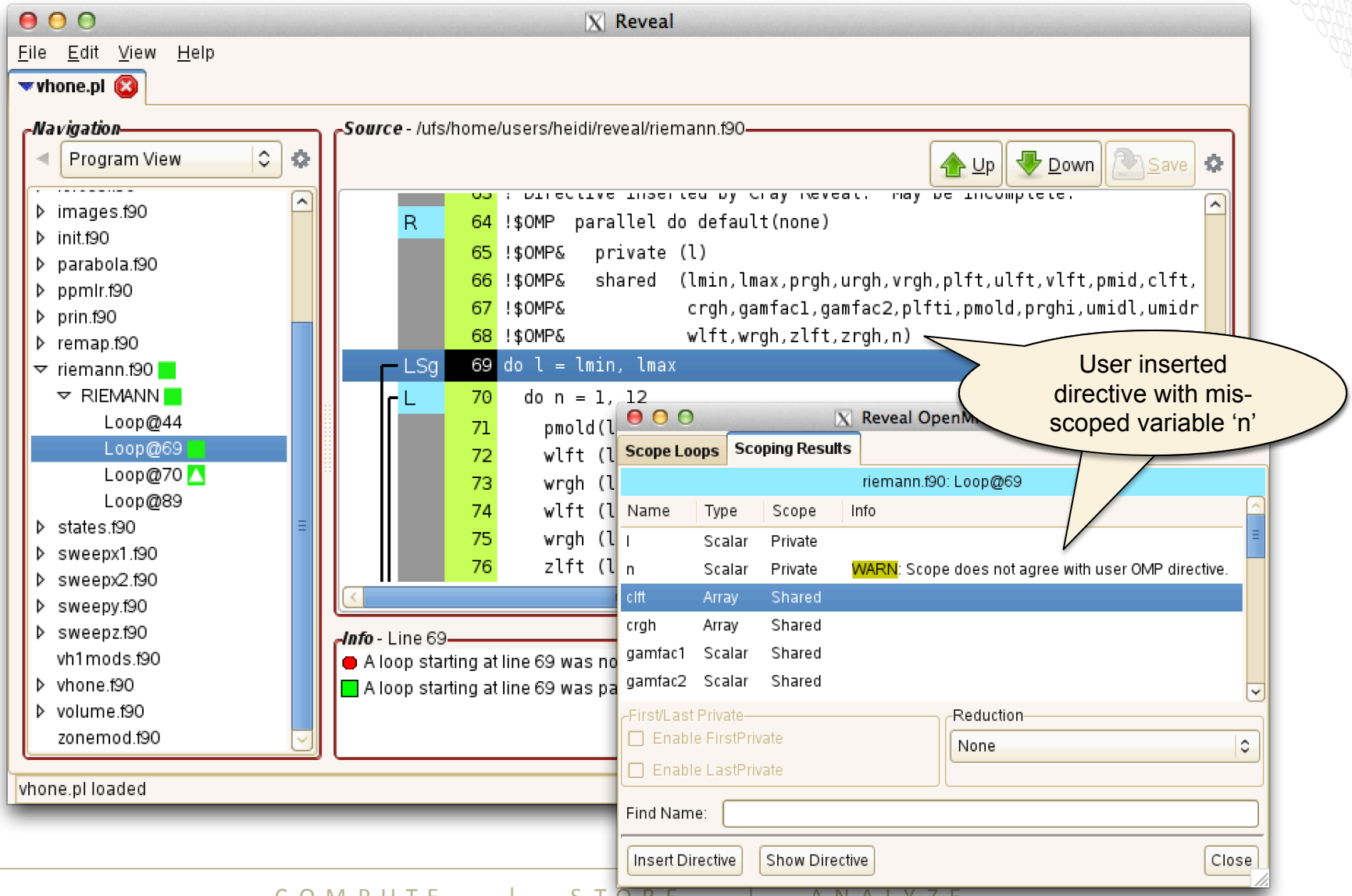
Original

```
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



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.