



# Reveal

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

# 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 for further optimizations

# WARNING!!!

- **Nothing comes for free, nothing is automatic**
  - Hybridization of an application is difficult
  - Efficient code requires interaction with the compiler to generate
    - High level OpenMP structures
    - Low level vectorization of major computational areas
- **Performance is also dependent upon the location of the data**
  - CPU: NUMA, first-touch
  - Accelerator: resident or data-sloshing
- **Software such as Cray's Hybrid Programming Environment provides tools to help, but cannot replace the developer's inside knowledge**

# 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 Reveal

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

# Loop Work Estimates

- 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 that can be used to approximate work and help carve up loop on GPU
  - 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 text 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

# How to Use Reveal (Continued)

- **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

- **Run Reveal**

- 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



New to Reveal? Click [here](#) for Getting Started Info

# Visualize Loopmark & Compiler Optimizations

Loopmark and optimization annotations

The screenshot shows a software interface for visualizing loopmarks and compiler optimizations. On the left is a navigation pane titled "Program View" showing a tree of files and loops. A yellow callout points to the "Loop@23" entry under "boundary.f90". The main area is titled "Source - /lus/nid00023/hej...\_version1/boundary.f90". It displays a snippet of Fortran code:

```
22 if ( nleft == 0 ) then
Vpr4 23   do n = 1, 6
24     dx (nmin-n)= dx (nmin+n-1)
25     dx0(nmin-n)= dx0(nmin+n-1)
26     xa (nmin-n)= xa (nmin-n+1) - dx (nmin-n)
27     xa0(nmin-n)= xa0(nmin-n+1) - dx0(nmin-n)
28     r (nmin-n) = r (nmin+n-1)
29     u (nmin-n) = -u(nmin+n-1)
30     v (nmin-n) = v (nmin+n-1)
31     w (nmin-n) = w (nmin+n-1)
32     p (nmin-n) = p (nmin+n-1)
33     e (nmin-n) = e (nmin+n-1)
34     f (nmin-n) = f (nmin+n-1)
35   ..
```

A yellow callout points to the "Vpr4" annotation on line 23. At the bottom, there is an "Info - Line 23" panel with the following content:

- A loop starting at line 23 was partially vectorized.
- A loop starting at line 23 was unrolled 4 times.

Compiler feedback

File Edit View Help  
vhone.pl

Navigation

Program View

boundary.f90

  └ BOUNDARY

    └ Loop@23

      Loop@37

      Loop@51

      Loop@65

      Loop@81

      Loop@95

      Loop@109

      Loop@123

  dtcon.f90

  dump.f90

  evolve.f90

  flatten.f90

  forces.f90

  images.f90

  init.f90

    └ GRID

      └ Loop@199

    INIT

  parabola.f90

Source - /lus/nid00023/hej...\_version1/boundary.f90

Up Down Save

Info - Line 23

■ A loop starting at line 23 was partially vectorized.

■ A loop starting at line 23 was unrolled 4 times.

vhone.pl loaded

# Navigate Source with Performance Information

Performance feedback

The screenshot shows the Reveal tool interface. On the left is a navigation tree titled "Navigation" under "Top Loops". It lists several loops with their execution times and names, such as "0.6046 Loop@75", "2.8951 Loop@63", and "3.2847 Loop@28". A yellow arrow points from the text "Performance feedback" to the "Info" section at the bottom right. The main area displays the source code for "parabola.f90". The code includes loops labeled "Vr2" and "Vr2" with line numbers 74 through 87. The "Info" section at the bottom right provides details about the loop at line 75: "A loop starting at line 75 was unrolled 2 times." and "A loop starting at line 75 was vectorized." The status bar at the bottom indicates "vhone.pl loaded. vhone\_loops.ap2 loaded."

```
source - /us/sonexion/heidi/reveal/parabola.f90
```

```
74
Vr2 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
Vr2 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.

# Access CCE's Message Explanations

**Navigation**

- Program View
- riemann.f90
- remap.f90
- evolve.f90
- volume.f90
- forces.f90
- ppmlr.f90
- states.f90
- flatten.f90
- sweepx.f90
- sweeppy.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**

```

L 32 do m = 1, npey
| Lr8 33 do i = 1, isy
|   n = i + isy*(m-1) + 6
|   r(n) = recv2(1,k,i,j,m)
|   p(n) = recv2(2,k,i,j,m)
|   u(n) = recv2(3,k,i,j,m)
|   v(n) = recv2(4,k,i,j,m)
|   w(n) = recv2(5,k,i,j,m)
|   f(n) = recv2(6,k,i,j,m)
| enddo
| enddo
V 44 do i = 1,imax
  n = i + 6
  
```

**Info - Line 33**

- A loop starting at line 33 was not vectorized because it does not have enough iterations.
- A loop starting at line 33 was unrolled 8 times.

**Reveal**

**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 "/ptmp/ulib/buildslaves/pdgcs-81-edition-build/bbs/build/release/pdgcs/pdgcs_ftr.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

Explain other message... Close

# View Pseudo Code for Inlined Functions

**Navigation**

- Program View
- boundary.f90
- BOUNDARY
  - Loop@23
  - Loop@37
  - Loop@51
  - Loop@65
  - Loop@81
  - Loop@95
  - Loop@109
  - Loop@123
- dtcon.f90
- dump.f90
- evolve.f90
- flatten.f90
- forces.f90
- images.f90
- init.f90
  - GRID
    - Loop@199
  - INIT
  - parabola.f90

**Source - /lus/nid00023/heidi/VH1\_version1/init.f90**

```

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_L88_102
88   zdx(1 + $I_L88_102) = 1.5625e-2
88   zxc(1 + $I_L88_102) = 7.8125e-3 * ( 1.5625e-2 * $I_L88_102
88   $I_L88_102 = 1 + $I_L88_102
89 call grid(jmax,ymin,
89 call grid(kmax,zmin,

```

**Inlined call sites marked**

**Info - Line 88**

- 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.

**Reveal**

**Up** **Down**

**Search code with Ctrl-F**

**grid**

**Source - Search**

Whole Symbol  Ignore Case Previous Next Cancel

vhone.pl loaded

# Navigate Code via Compiler Messages

Choose “Compiler Messages” view to access message filtering

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

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

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.

vphone.f90 loaded

# Navigate Code via Compiler Messages (2)

The screenshot shows a software interface for navigating code via compiler messages. On the left, a panel titled "Compiler Messages" displays a list of "Not Inlined" functions from two files: "evolve.f90" and "images.f90". The function "volume" is listed under "evolve.f90" at line 50. A yellow callout bubble points to this entry with the text: "See functions that didn't get inlined (tip: use -hwp so compiler can find files to inline from)". The main window on the right shows the source code for "evolve.f90" with line numbers 49 to 61. The code includes a call to "volume" at line 50 and several case statements for "ngeom". At the bottom, an info box states: "Info - Line 50" and "volume" was not inlined because it is not in the body of a loop.

File Edit View Help

vphone.pl

Navigation

Compiler Messages

Not Inlined All

evolve.f90

- line 21
- line 45
- line 46
- line 50

images.f90

- line 42
- line 49
- line 54
- line 55
- line 58
- line 59
- line 60
- line 61
- line 64
- line 65
- line 66
- line 67

Source - /ufs/home/meridi/reveal/evolve.f90

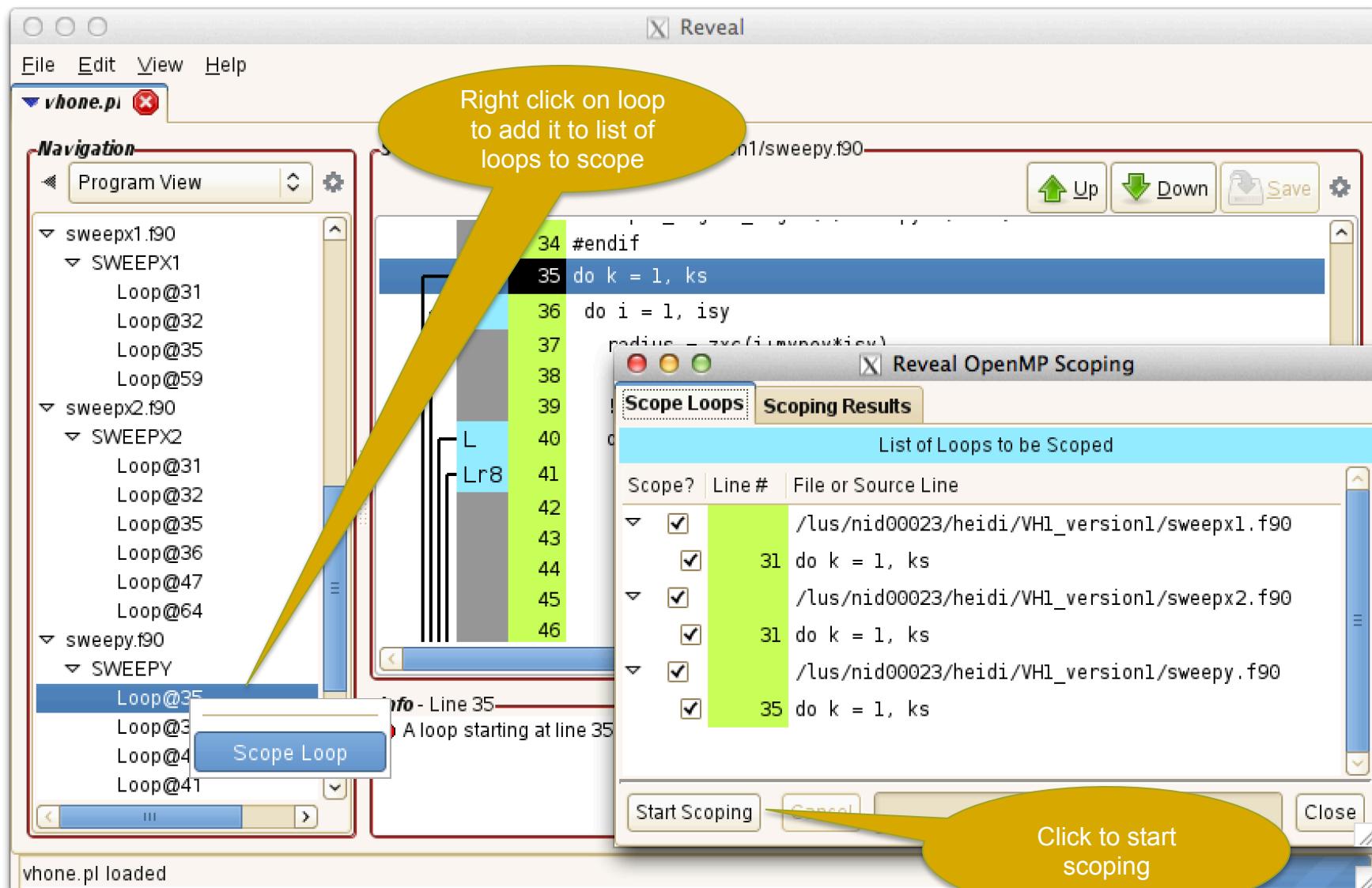
! I 50 call volume (nmin, nmax, ngeom, radius, xa , dx , dvol )  
51  
52 select case (ngeom)  
53 case(1)  
Vr4 54 amid = 0.5\*(xa + xal)  
55 case (2)  
Vr4 56 amid = (xa-xal)\*(third\*(xa-xal)+xal)\*\*2  
57 case (4)  
V 58 do n = nmin-3, nmax+4  
59 dtheta = xa(n) - xal(n)  
60 if(dtheta == 0.0) then  
61 amid(n) = sin(xa(n))

Info - Line 50

"volume" (called from "evolve") was not inlined because it is not in the body of a loop.

vphone.pl loaded

# Scoping Assistance - Scope Loop(s)



# Scoping Assistance – Review Scoping Results

The screenshot shows the Reveal tool interface with three main windows:

- Navigation:** Shows a tree view of loops in the file. Loops flagged with scoping information (red) include "sweepz.f90" (SWEEPZ), "Loop@48", "Loop@49", "sweeypy.f90" (SWEEPY), "Loop@32", "Loop@33", "x2.f90" (X2), "Loop@28", "Loop@29", "sweepx1.f90" (SWEEPX1), "Loop@28", "Loop@29", "riemann.f90" (RIEMANN), "Loop@63", "parabola.f90" (PARABOLA), and "Loop@77". A yellow callout points to "Loop@48" with the text: "Loops with scoping information are flagged— red needs user assistance".
- Source:** Displays the code for "sweepz.f90" with annotations. Lines 48-60 are highlighted in green, indicating they are part of a loop. A yellow callout points to the "Info" section at the bottom left with the text: "A loop starting at line 48 was not vectorized. Loop has been flattened."
- Parallelization inhibitor selector:** A dialog box for "sweepz.f90: line 48 > 107" showing variable information. It lists variables f, flat, q, zyc, i, j, k, m, isz, js, ks, mypey, mypez, ngeomz, nleftz, npez, and nrightz. The "Scope" column shows "Unresolved" for f, flat, and q, and "Conflict" for zyc. The "Info" column provides error/warning messages for each. A yellow callout points to this window with the text: "Parallelization inhibitor messages are provided to assist user with analysis". Another yellow callout points to the "Reduction" section with the text: "User addresses parallelization issues for unresolved variables".

# Scoping Assistance – User Resolves Issues

**Reveal – 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.

Loop@33  
Loop@37  
Loop@38  
Loop@  
Loop@

Use Reveal's OpenMP parallelization tips

Loop@32  
Loop@33  
Loop@44  
Loop@58  
sweepx1.f90  
volume.f90  
states.f90  
riemann.f90

**!sweepx2.f90**

```

1 Loop over each row...
2   = l, ks
3   = l, js
4
5   ! Put state variables i
6   do m = 1, npey
7     do i = 1, isy
8       n = i + isy*(m-1) +
9       r(n) = recv2(1,k,i,j)
10      p(n) = recv2(2,k,i,j)
11      u(n) = recv2(3,k,i,j)
12      v(n) = recv2(4,k,i,j)
13      w(n) = recv2(5,k,i,j)
14      f(n) = recv2(6,k,i,j)
15    enddo
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

```

**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.

**OpenMP 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	

First/Last Private  
 Enable First Private  
 Enable Last Private

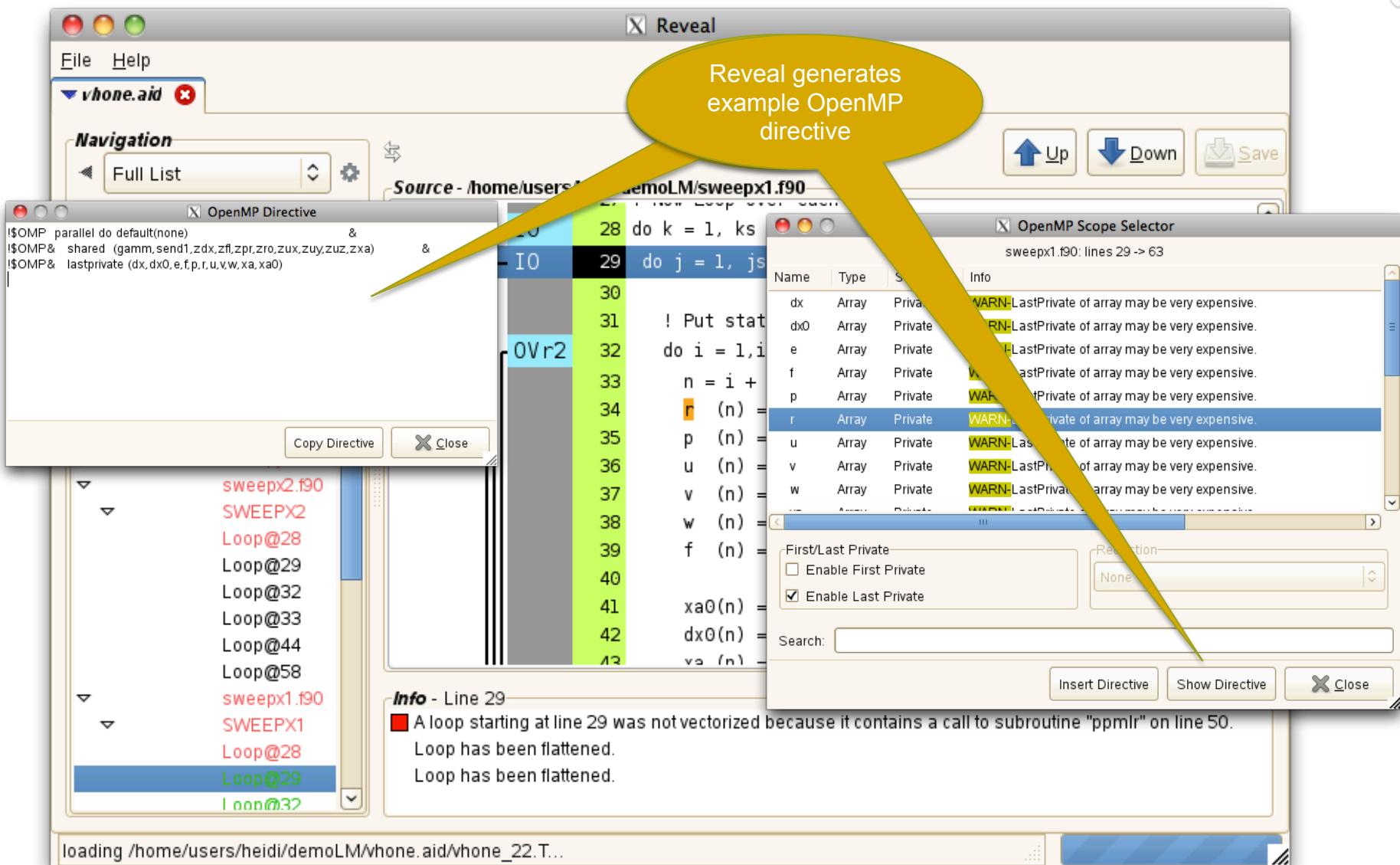
None

Search:

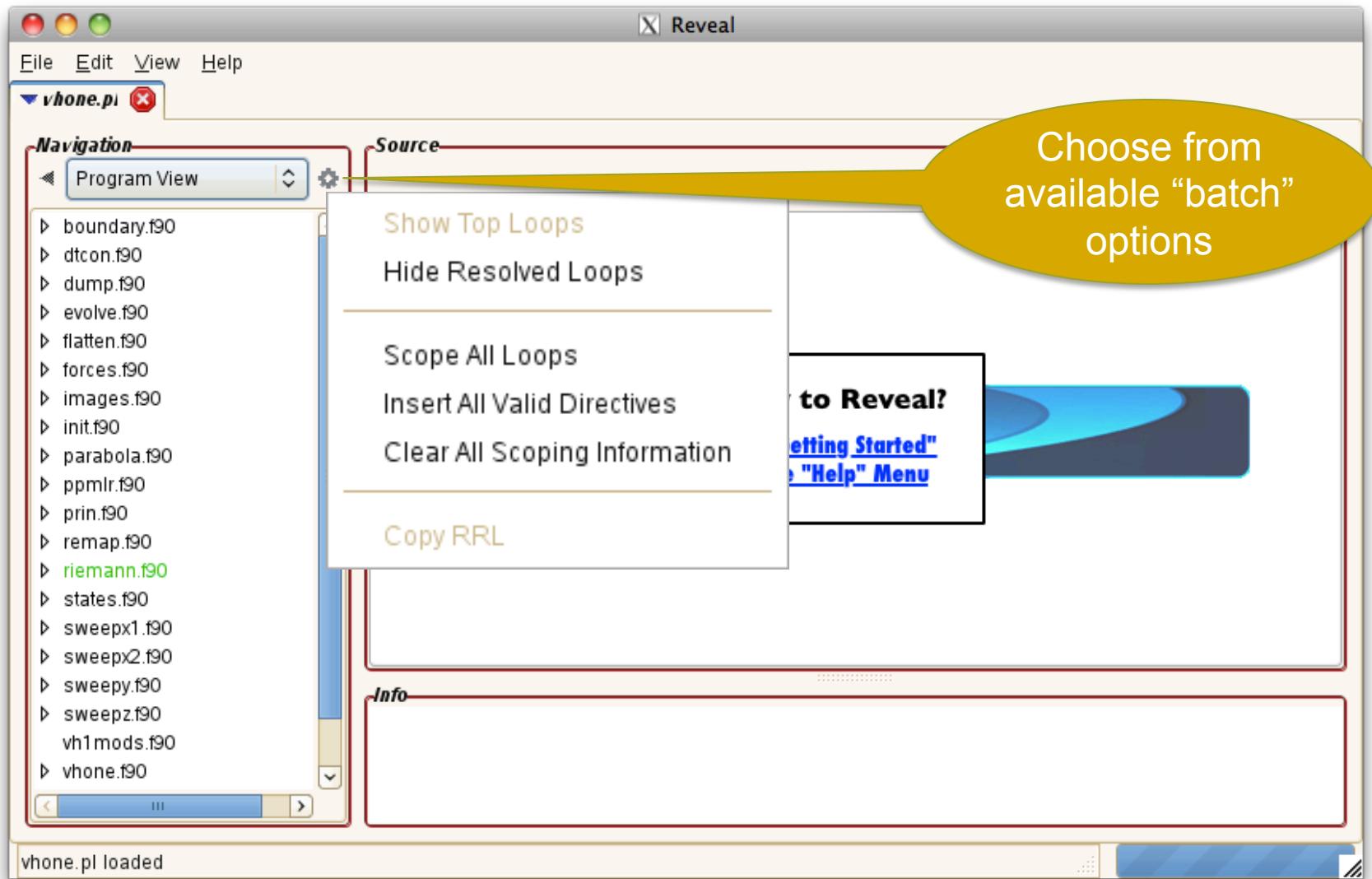
Insert Directive Show Directive Close

Click on variable to view all occurrences in loop

# Scoping Assistance – Reveal Generates Directive



# Batch Options Available



Choose from  
available “batch”  
options

# Use Reveal to Validate User Inserted Directives

User inserted directive with mis-scoped variable 'l'

The screenshot shows the Reveal application window with the following details:

- Navigation:** Shows a tree view of files and loops. The 'riemann.f90' section is expanded, showing 'Loop@44', 'Loop@69' (selected), 'Loop@70', and 'Loop@89'.
- Source:** Displays the Fortran code for 'riemann.f90' with syntax highlighting. Line 69 is highlighted in blue, indicating a loop scopability analysis.
- Info - Line 69:**
  - A loop starting at line 69 was not vectorized for an unspecified reason.
  - A loop starting at line 69 was partitioned.
- Reveal OpenMP Scoping:** A modal dialog box shows the 'Scoping Results' tab for the loop from line 69 to 86. It lists variables with their types and scopes, and highlights a warning for variable 'l'.

Name	Type	Scope	Info
l	Scalar	Private	WARN Scope does not agree with user OMP directive.
n	Scalar	Private	
cfft	Array	Shared	
zrgh	Scalar	Shared	
umidl	Scalar	Shared	
umidr	Scalar	Shared	
pmid	Scalar	Shared	
pmid	Scalar	Shared	
lmax	Scalar	Shared	
lmin	Scalar	Shared	
plft	Array	Shared	
plfti	Array	Shared	
pmid	Array	Shared	
pmold	Array	Shared	

- Buttons:** 'Insert Directive', 'Show Directive', 'Close'.

# What's Coming Next...

- **More information on scoped variables**
  - Call out variables included in scoping results due to inlining (shouldn't be included on OMP directive)
  - Call out reductions
- **Navigation panel coloring improvements**
  - Follow Info panel style when highlighting loops that are scoped
- **OpenACC support**
  - Scoping for OpenACC loops
  - OpenACC directive generation
  - Data region building support

# Highlighting Scoped Loops...

The screenshot shows the Cray CS (Code Surveyor) interface. The window title is "Reveal" and the file path is "Source - /lus/nid00023/heidi/WH1\_version1/sweepx1.f90". The left panel, titled "Navigation", lists various F90 files. The "Program View" tab is selected, showing a tree structure with nodes like "dtcon.f90", "dump.f90", etc., and two expanded sections under "sweepx1.f90": "SWEEPX1" and "SWEEPX2", each containing several loop definitions. A red dot icon is next to each node. The right panel displays the source code for "sweepx1.f90". Lines 31 through 49 are highlighted in green, indicating they belong to a scopable loop. The code itself is as follows:

```
      n = t + o
  37  r (n) = zro(i,j,k)
  38  p (n) = zpr(i,j,k)
  39  u (n) = zux(i,j,k)
  40  v (n) = zuy(i,j,k)
  41  w (n) = zuz(i,j,k)
  42  f (n) = zfl(i,j,k)
  43
  44  xa0(n) = zxa(i)
  45  dx0(n) = zdx(i)
  46  xa (n) = zxa(i)
  47  dx (n) = zdx(i)
  48  p (n) = max(smallp,p(n))
  49  e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
```

The status bar at the bottom left says "vhone.pl loaded". The bottom right corner contains the Cray logo.

**Info - Line 31**

- A loop starting at line 31 was not vectorized because it contains a call to subroutine "ppmlr" on line 53.

# More Information on Scoped Variables

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

Reveal OpenMP Scoping

Scope Loops Scoping Results

sweepx1.f90: Loop@31

Call or I/O at line 53 of sweepx1.f90

Name	Type	Scope	Info
radius <b>I</b>	Scalar	Private	<b>WARN</b> : LastPrivate of array may be very expensive. <b>FAIL</b> : incompatible with 'natural' scope.
u	Array	Private	<b>FAIL</b> : incompatible with 'natural' scope. <b>WARN</b> : LastPrivate of array may be very expensive.
v	Array	Private	<b>FAIL</b> : incompatible with 'natural' scope. <b>WARN</b> : LastPrivate of array may be very expensive.
w	Array	Private	<b>FAIL</b> : incompatible with 'natural' scope. <b>WARN</b> : LastPrivate of array may be very expensive.
xa	Array	Private	<b>FAIL</b> : incompatible with 'natural' scope. <b>WARN</b> : LastPrivate of array may be very expensive.
xa0	Array	Private	<b>FAIL</b> : incompatible with 'natural' scope. <b>WARN</b> : LastPrivate of array may be very expensive.
gamm	Scalar	Shared	
js	Scalar	Shared	
ks	Scalar	Shared	
ngeomx	Scalar	Shared	
nleftx	Scalar	Shared	
nrightx	Scalar	Shared	
send1	Array	Shared	
svel <b>RI</b>	Scalar	Shared	<b>WARN</b> : atomic reduction operator required unless reduction fully inlined.
<b>First/Last Private</b> <input type="checkbox"/> Enable FirstPrivate <input type="checkbox"/> Enable LastPrivate			
<b>Reduction</b> <input type="button" value="None"/>			
Find Name: <input type="text"/>			
<input type="button" value="Insert Directive"/>		<input type="button" value="Show Directive"/>	
<input type="button" value="Close"/>			

Reduction variable from down the call stack - hover over 'RI' to see what symbol means



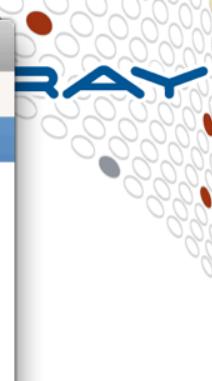
# Questions

?



## Reveal - Getting Started

# Getting Started



### Source Navigation

- Selecting a file name in the navigation panel brings the source into the source panel for viewing.
- The source panel contains a column of loopmark information.  
See 'Help -> Loopmark Legend' for an explanation of the symbols used.
- The 'File Selection' option in the 'View' menu can be used to select which files are shown in the navigation panel views.

### Scoping Loop Selection

- Right-clicking on a file, function or loop in the navigation panel shows a menu. The 'Scope Loops' option will add loops to the list of loops to be scoped.
- You can also select a loop to be scoped by double-clicking on the first line of a loop in the source panel.
- You may queue up loops from multiple files before you start the scoping process.

### OpenMP Directive Generation

- After a loop has been scoped, you may view the computed scoping information by selecting the loop in the source panel. A window will be opened with the computed scoping information.
- You can find more detailed information on addressing scoping problem under 'Help -> OpenMP Tips'
- You may also have Reveal insert an OpenMP directive into your source view using the 'Insert Directive' button.
- Inserted directives are maintained separately from your source file until you request a save with the 'Save' button in the source panel.

Close