

Creating Hybrid Codes with Cray Reveal

Heidi Poxon
Technical Lead
Programming Environment
Cray Inc.

March 2016

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.

Copyright 2016 Cray Inc.

When to Move to a Hybrid Programming Model

- **When code is network bound**
 - Increased MPI collective and point-to-point wait times
- **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**



Approach to Adding Parallelism

1. Identify key high-level loops

- Determine where to add additional levels of parallelism

2. Perform parallel analysis and scoping

- Split loop work among threads

3. Add OpenMP layer of parallelism

- Insert OpenMP directives

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

- We want a performance-portable application at the end

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****

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

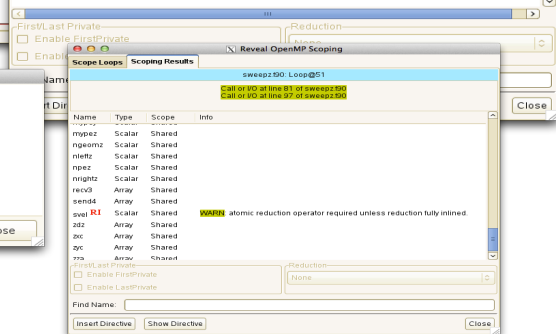
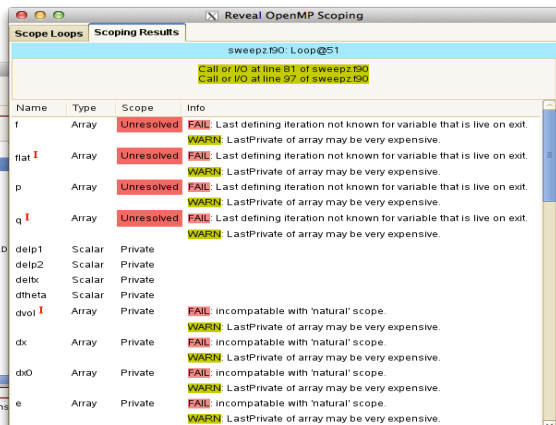
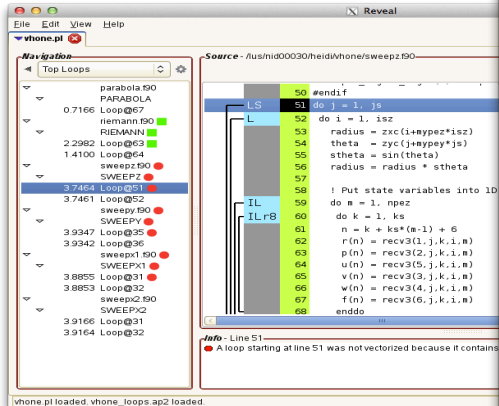
call volume(nmin, nmax, ngeom, radius, xa, dx, dvol)

call remap ← contains more calls

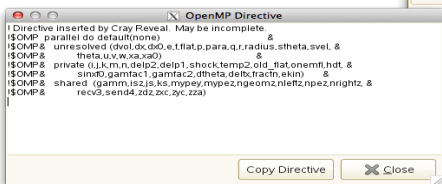
return
end

```

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



Hybridization Step 1: Loop Work Estimates

Gather loop statistics using CCE and 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



perftools-lite-loops

- CrayPat-lite loop work estimates
- Must be used with Cray compiler
- Load before building and running program to get loop work estimates sent to stdout and to .ap2 file for use with Reveal
- Automates loop work experiment by:
 - modifying the compile and link steps to include CCE's `-h profile_generate` option
 - instrumenting the program for tracing (`pat_build -w`)
- **`-h profile_generate` reduces compiler optimization levels**
 - After experiment is complete, unload `perftools-lite-loops` to prevent further program instrumentation.



Collecting Loop Work Estimates

- Load PrgEnv-cray module (must use CCE)
- Load perftools-base module if not already loaded
- Load perftools-lite-loops module

- Build and run application

- Loop work estimates will be available for Reveal in file with .ap2 extension and in text format in file with .rpt extension

- Unload perftools-lite-loops module

Example Loop Work Estimates

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

The CCE Program Library (PL)

- **An application wide repository for compiler and tools information**
 - Allows the user to specify a repository of compiler information for an application build
- **Provides the framework for application analysis**
 - Whole application IPA information for optimization
 - Automatic whole application inlining and cloning
 - Various inter-procedural optimizations
 - Whole application static error detection
- **Provides ability for tools to annotate loops with runtime feedback and other performance hints without source change**
 - Support for the Cray refactoring tool, Reveal.

Generate a Program Library

- `> cc -h pl=himeno.pl -hwp* himeno.c`
- `> ftn -h pl=vhone.pl file1.f90`

* Optionally add whole program analysis for additional inlining.



Launch 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
```

Visualize Compiler and Performance Information



The screenshot displays the Reveal IDE interface with several key components:

- Navigation Panel (Left):** Lists performance metrics for various nodes, such as SWEEPY@35 (4.0778) and SWEEPX1@32 (4.0526).
- Source Editor (Center):** Shows Fortran code with colored annotations. A callout bubble labeled "Performance feedback" points to a blue bar on line 31. Another callout labeled "Loopmark and optimization annotations" points to a blue bar on line 32. The code includes a loop structure and a subroutine call "ppmlr".
- Loopmark Legend (Right):** A legend defining various annotations: A (Pattern Matched), C (Collapsed), D (Deleted), E (Cloned), G (Accelerated), I (Inlined), II (Not Inlined), L (Loop), M (Multithreaded), R (Region), S (Scoping Analysis), and V (Vectorized). It also defines letters a through w for specific optimization types.
- Info Panel (Bottom):** A callout bubble labeled "Compiler feedback" points to a red dot in the info panel, which states: "A loop starting at line 32 was not vectorized because it contains a call to subroutine 'ppmlr' on line 53."

Access Cray Compiler Message Information



The screenshot shows the 'Reveal' IDE interface. On the left is a 'Navigation' pane with a 'Program View' tree. The main window displays the 'Source' code for `/lus/sonexion/heidi/reveal/sweepx2.f90`. The code includes a loop starting at line 33: `do i = 1, isy`. A message at the bottom of the source window indicates: `Info - Line 33: A loop starting at line 33 was not vectorized because it does not have a stride of 1. A loop starting at line 33 was unrolled 8 times.`

An 'Explain' dialog box is open, titled 'OPT_INFO: A loop starting at line %s was unrolled.' It contains the following text: `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/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 dialog box also has buttons for 'Explain other message...' and 'Close'.

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

Navigate Loops through Call Chain



The screenshot shows the Reveal IDE interface. On the left, the 'Navigation' pane displays a list of loop performance metrics. The 'Source' pane shows the corresponding Fortran code. Two callouts highlight specific features: 'Loop instances' points to the list of instances for the PARABOLA@67 loop, and 'Loop traceback' points to the traceback list at the bottom.

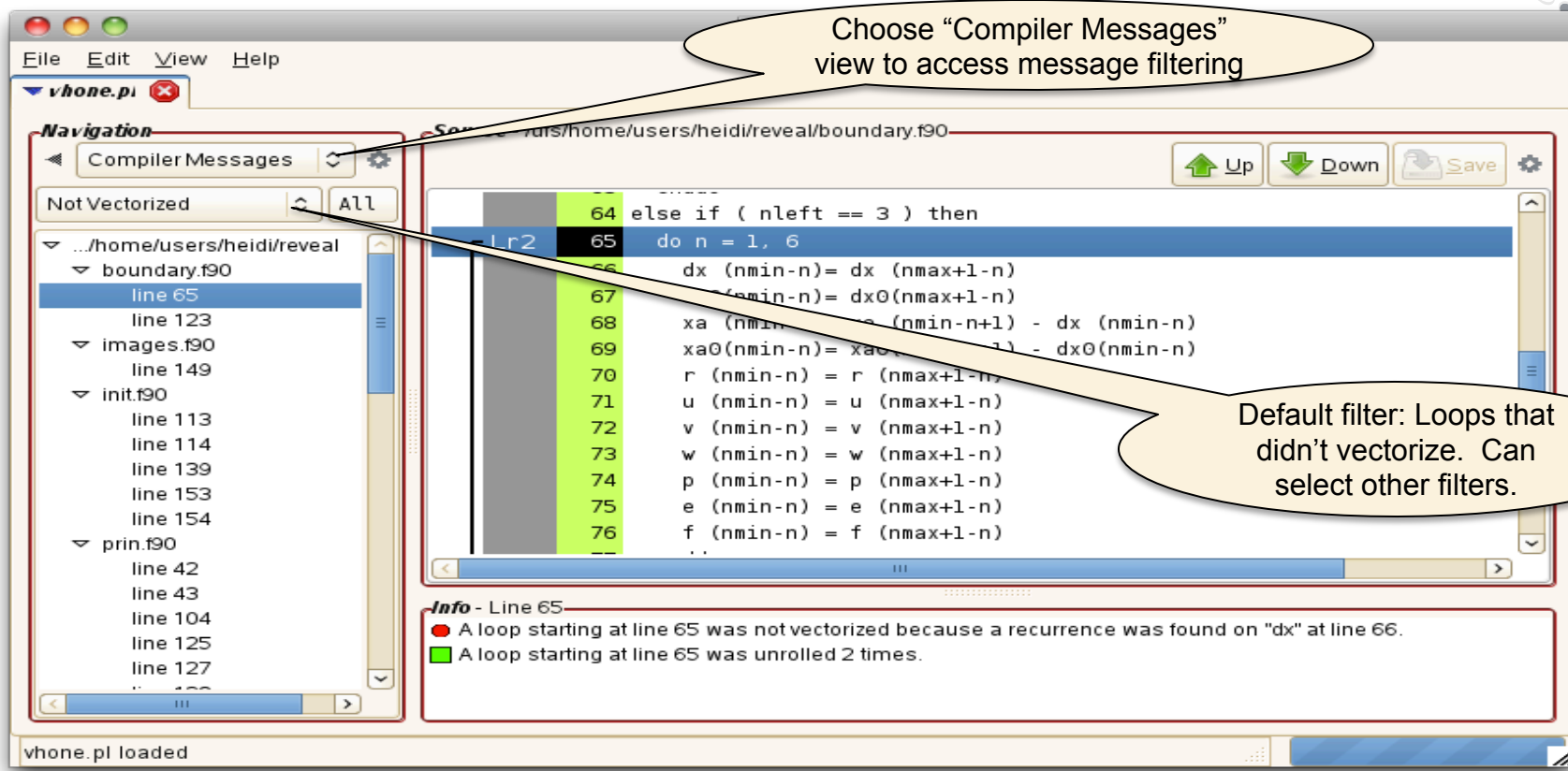
Time	Loop Name	Instances
4.0423	SWEEPX2@32	1
3.8576	SWEEPZ@51	1
3.8573	SWEEPZ@52	1
2.2068	RIEMANN@63	1
1.2299	RIEMANN@64	1
0.8068	PARABOLA@67	7
0.0146	Instance #1	
0.0156	Instance #2	
0.0156	Instance #3	
0.0163	Instance #4	
0.0163	Instance #5	
0.0174	Instance #6	
0.0167	Instance #7	

```
Source - /home/users/heidi/reveal/parabola.f90
66
67 do n = nmin, nmax
68   deltaa(n) = ar(n) - al(n)
69   a6(n)      = 6. * (a(n) - .5 * (al(n) + ar(n)))
70   scrch1(n)  = (ar(n) - a(n)) * (a(n)-al(n))
71   scrch2(n)  = deltaa(n) * deltaa(n)
72   scrch3(n)  = deltaa(n) * a6(n)
73 enddo
74
75 do n = nmin, nmax
76   if(scrch1(n) <= 0.0) then
77     ar(n) = a(n)
78     al(n) = a(n)
```

Info - Loop
A loop starting at line 67 was fused with the loop starting at line 53.

Traceback
PARABOLA@67
PPMLR@51
sweep1__LOOP2.li.32@53
sweep1__LOOP1.li.31@32
SWEEPX1@31
VHONE@232

Navigate Code via Compiler Messages



The screenshot shows a code editor window with a menu bar (File, Edit, View, Help) and a toolbar (Up, Down, Save). A "Navigation" sidebar on the left lists files like boundary.f90, images.f90, init.f90, and prin.f90. The main editor displays Fortran code with line numbers 64-76. A callout bubble points to the "Compiler Messages" dropdown in the sidebar, stating: "Choose 'Compiler Messages' view to access message filtering". Another callout bubble points to the "Not Vectorized" filter, stating: "Default filter: Loops that didn't vectorize. Can select other filters." Below the code, an "Info" panel for line 65 provides details: a red dot indicates a loop not vectorized due to a recurrence on "dx" at line 66, and a green dot indicates a loop unrolled 2 times.

Choose "Compiler Messages" view to access message filtering

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

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.

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 I Vw 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-2 * (1.5625e-2 * $I_L
88          $I_L88_102 = 1 + $I_L88_102
89 AIVfr4 call grid(jmax,ymin,
90 AIf     call grid(kmax,zmin,
```

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.

Search

grid

Whole Symbol Ignore Case Previous Next Cancel

Callouts:

- Expand to see pseudo code
- Search code with Ctrl-F
- Inlined call sites marked

Hybridization Step 2: Scope Selected Loop(s)



Reveal OpenMP Scoping

Scope Loops | Scoping Results

Edit List | List of Loops to be Scoped

Scope?	Line #	File or Source Line
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/evolve.f90
<input type="checkbox"/>		/home/users/heidi/reveal/flatten.f90
<input type="checkbox"/>		/home/users/heidi/reveal/forces.f90
<input type="checkbox"/>		/home/users/heidi/reveal/images.f90
<input type="checkbox"/>		/home/users/heidi/reveal/init.f90
<input type="checkbox"/>		/home/users/heidi/reveal/parabola.f90
<input type="checkbox"/>		/home/users/heidi/reveal/ppmlr.f90
<input type="checkbox"/>		/home/users/heidi/reveal/prin.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/remap.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/riemann.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/states.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/sweepx1.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/sweepx2.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/sweepy.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/sweepz.f90

Apply Filter | Time: 0.000 | Trips: 2 | Threads: 4 | Speedup: 0.010

Start Scoping | Cancel | 26 Loops selected | Close

Review Scoping Results



Navigation

- 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
- parabola.f90
- ppmlr.f90
- prin.f90
- remap.f90
- riemann.f90
- states.f90
- sweepx1.f90
- sweepx2.f90
- sweepx.f90

```
21
22 if ( nleft == 0 ) then
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     (nmin-n) = p (nmin+n-1)
33     (nmin-n) = e (nmin+n-1)
34     f (nmin-n) = f (nmin+n-1)
35   enddo
36 else if ( nleft == 1 ) then
37   do n = 1, 6
38     dx (nmin-n) = dx (nmin)
39     dx0(nmin-n) = dx0(nmin)
```

Info - Line 23

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

Scoping Results

Name	Type	Scope	Info
xa	Array	Unresolved	FAIL: Possible recurrence involving this object
xa0	Array	Unresolved	FAIL: Possible recurrence involving this object
n	Scalar	Private	
dx	Array	Shared	
dx0	Array	Shared	
e	Array	Shared	
f	Array	Shared	
nmin	Scalar	Shared	
p	Array	Shared	
r	Array	Shared	
u	Array	Shared	
v	Array	Shared	
w	Array	Shared	

Loops with scoping information are flagged. Red needs user assistance

Parallelization inhibitor messages are provided to assist user with analysis

User addresses issues for variables with FAIL status



Review Scoping Results (2)

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

Name	Type	Scope	Info
ar@parabola_	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
da@parabola_	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
delta@remap_	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
dvol	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.
e	Array	Unresolved	FAIL: Possible recurrence involving this object.

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

First/Last Private: Enable FirstPrivate Enable LastPrivate

Reduction:

Find Name:

See where variable
came from
(@function_name)

Review Scoping Results (3)



Reveal identifies calls that prevent parallelization

Name	Type	Scope	Info
ks	Scalar	Shared	
mypey	Scalar	Shared	
ndim	Scalar	Shared	
npey	Scalar	Shared	
recv1	Array	Shared	
send2	Array	Shared	
svel RI	Scalar	Shared	WARN: atomic reduction operator required unless reduction fully
zdy	Array	Shared	
zxc	Array	Shared	
zya	Array	Shared	

Reveal identifies shared reductions down the call chain

Hybridization Step 3: Generate OpenMP Directives



```
! 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,zdy,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

Or Validate User Inserted Directives

The screenshot shows the Cray Reveal IDE interface. The main window displays a source file with the following code:

```
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)
69 do l = lmin, lmax
70 do n = 1, 12
71 pmold(l
72 wlft(l
73 wrgh(l
74 wlft(l
75 wrgh(l
76 zlft(l
```

A dialog box titled "Scope Loops" is open, showing the "Scoping Results" for the loop "riemann.f90: Loop@69". The table below shows the results:

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	

The "Info" panel at the bottom left shows a warning: "A loop starting at line 69 was not..." and a note: "A loop starting at line 69 was pa...".

A callout bubble points to the variable 'n' in the code and the warning in the dialog, with the text: "User inserted directive with mis-scoped variable 'n'".

Hybridization Step 4: Performance Analysis



The screenshot shows a software interface with a menu bar (File, Edit, View, Help) and a window titled 'vhone.pl'. On the left is a 'Navigation' pane with a tree view showing folders like 'boundary.f90', 'images.f90', 'init.f90', and 'rinemann.f90'. The main area displays source code with line numbers 41-55. Line 42 is highlighted in blue with 'F' and '41' in the margin. Lines 43-44 are highlighted in green with 'F' and '42' in the margin. Lines 44-53 are highlighted in light green with 'FVw' and '44' in the margin. Line 54 is highlighted in blue with '!I' and '53' in the margin. A callout bubble points to the 'Compiler Messages' dropdown menu. Another callout bubble points to the code block from line 42 to 53. A third callout bubble points to the 'Info' pane at the bottom.

Choose "Compiler Messages" view to access message filtering

See loops that didn't vectorize. Can select other filters.

See all compiler messages for a loop nest

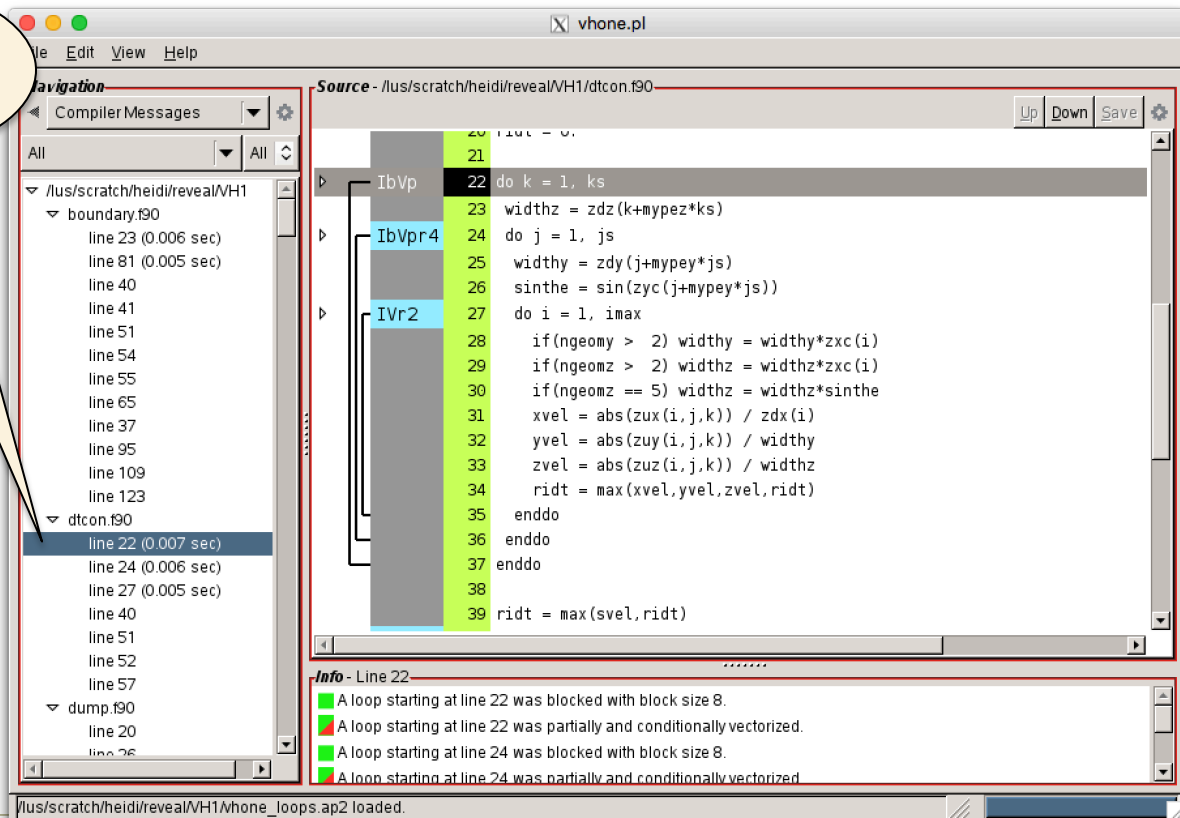
```
41
42 do k = 1, ks
43   do j = 1, js
44     do i = 1, imax
45       send_buff(i,1,j,k) = zro(i,j,k)
46       send_buff(i,2,j,k) = zpr(i,j,k)
47       send_buff(i,3,j,k) = zux(i,j,k)
48       send_buff(i,4,j,k) = zuy(i,j,k)
49       send_buff(i,5,j,k) = zuz(i,j,k)
50     enddo
51   enddo
52 enddo
53
54 gathbuffer_size = imax * js * ks * nvarout
55 call MPI_GATHER(send_buff, gathbuffer_size, MPI_REAL, recv, gat
```

Info - Line 42

- A loop starting at line 42 is flat (contains no external calls).
- A loop starting at line 42 was not vectorized because a better candidate was found at line 44.
- A loop starting at line 43 is flat (contains no external calls).
- A loop starting at line 43 was not vectorized because a better candidate was found at line 44.
- A loop starting at line 44 is flat (contains no external calls).
- A loop starting at line 44 was vector pipelined.

Focus on Relevant Loops (June'16)

Compiler messages sorted by time



The screenshot shows a compiler interface with a source code editor and a navigation pane. The source code is for a file named `vhone.pl` and contains a loop structure with nested loops `IbVp`, `IbVpr4`, and `IvR2`. The compiler messages pane on the left shows a list of messages sorted by time, with the message for line 22 (0.007 sec) selected. The `Info` pane at the bottom right provides details for the selected message, indicating that a loop starting at line 22 was blocked with a block size of 8 and partially and conditionally vectorized.

```
Source - /lus/scratch/heidi/reveal/VH1/dtcon.f90
20  ...
21  ...
22  do k = 1, ks
23    widthz = zdz(k+mypez*ks)
24    do j = 1, js
25      widthy = zdy(j+mypey*js)
26      sinthe = sin(zyc(j+mypey*js))
27      do i = 1, imax
28        if(ngeomy > 2) widthy = widthy*zxc(i)
29        if(ngeomz > 2) widthz = widthz*zxc(i)
30        if(ngeomz == 5) widthz = widthz*sinthe
31        xvel = abs(zux(i,j,k)) / zdx(i)
32        yvel = abs(zuy(i,j,k)) / widthy
33        zvel = abs(zuz(i,j,k)) / widthz
34        ridt = max(xvel,yvel,zvel,ridt)
35      enddo
36    enddo
37  enddo
38  ...
39  ridt = max(svel,ridt)
```

Navigation - Compiler Messages

- All
- /lus/scratch/heidi/reveal/VH1
 - boundary.f90
 - line 23 (0.006 sec)
 - line 81 (0.005 sec)
 - line 40
 - line 41
 - line 51
 - line 54
 - line 55
 - line 65
 - line 37
 - line 95
 - line 109
 - line 123
 - dtcon.f90
 - line 22 (0.007 sec)
 - line 24 (0.006 sec)
 - line 27 (0.005 sec)
 - line 40
 - line 51
 - line 52
 - line 57
 - dump.f90
 - line 20
 - line 26

Info - Line 22

- A loop starting at line 22 was blocked with block size 8.
- A loop starting at line 22 was partially and conditionally vectorized.
- A loop starting at line 24 was blocked with block size 8.
- A loop starting at line 24 was partially and conditionally vectorized.

Hybridization Step 4: Performance Analysis



```
===== Observations and suggestions =====
D1 cache utilization:
61.7% of total execution time was spent in 1 functions with D1 cache
hit ratios below the desirable minimum of 90.0%. Cache utilization
might be improved by modifying the alignment or stride of references
to data arrays in these functions.

      D1      Time%  Function
cache
hit
ratio

      74.3%    61.7%  calc3_

D1 + D2 cache utilization:
61.7% of total execution time was spent in 1 functions with combined
D1 and D2 cache hit ratios below the desirable minimum of 97.0%.
Cache utilization might be improved by modifying the alignment or
stride of references to data arrays in these functions.

D1+D2      Time%  Function
cache
hit
ratio

      96.6%    61.7%  calc3_
...

```

Summary

- **Reveal can be used to simplify the task of adding OpenMP to MPI programs.**
- **The result is performance portable code: OpenMP directives (programs can be built with any compiler that supports OpenMP)**
- **Can be used as a stepping stone for codes targeted for nodes with higher core counts and as the first step in adding directives to applications to target GPUs**
- **Moving to OpenMP 4.0 accelerator directives or OpenACC via OpenMP is a good idea**
 - Same work required
 - Can have both (conditionally compile one or other or none)
 - First level of debugging on multicore CPU