



Addressing Performance and Programmability Challenges in Current and Future Supercomputers

Luiz DeRose
Sr. Principal Engineer
Programming Environments Director
Cray Inc.



Outline

- Computer architecture and applications trends
- Programming Environment mission
- Tools focus for extreme scale computing
- Open issues
- Conclusions



Future Architectural 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
- **As processors get faster, memory bandwidth cannot keep up, resulting in:**
 - More complex caches
 - Non-uniform memory architecture (NUMA) for shared memory on node
 - Operand alignment is becoming more important

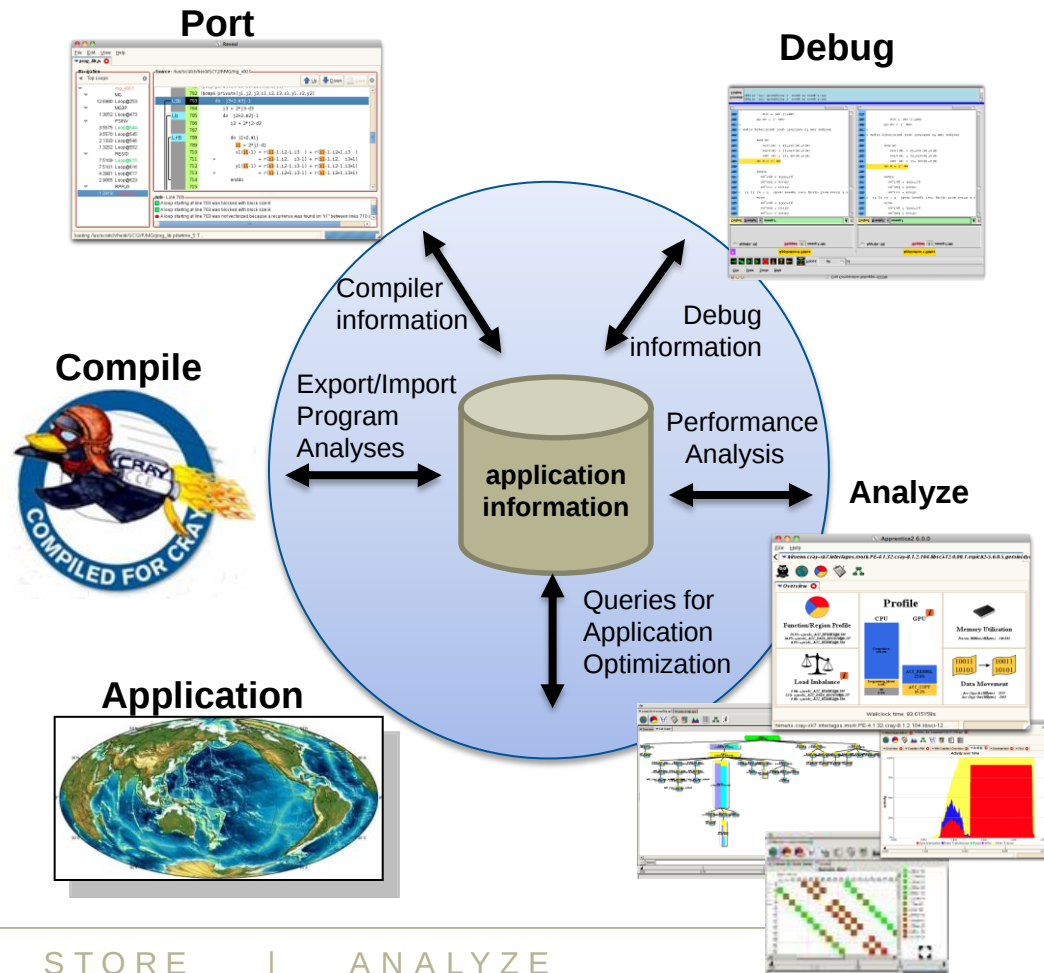


Application Developers: Are you ready for the future?

- Today's petascale applications are not structured to take advantage of these architectures
 - Currently 80-90% of applications use a single level of parallelism
 - Message passing between cores of the MPP system
- Looking forward, application developers are faced with a significant task in preparing their applications for the future
 - You will have to **move to a hybrid** (MPI, threading, & vector) architecture
 - You must start considering how to **manage your arrays** to have them close to the computational engine when you need them
 - We are moving to a more complex memory hierarchy that will require user intervention to achieve good performance.

The Programming Environment Mission

- It is the role of the Programming Environment to **close the gap** between observed performance and achievable performance
- Support the **application development life cycle** by providing a **tightly coupled** environment with compilers, libraries, and tools that will **hide the complexity** of the system
 - Address issues of scale and complexity of HPC systems
 - Target **ease of use** with extended **functionality** and increased **automation**





Extreme Scale Tools Focus

- The current and future generation of tools should focus on
 - Performance
 - **Help users maximize the cycles to the application**
 - Address issues of scale and complexity of HPC systems
 - Programmability
 - How do you get intuitive behavior and best performance with the least amount of effort
 - **Provide the best environment to develop, debug, analyze, and optimize applications for production supercomputing**
 - Power
 - One of the biggest challenges facing extreme scale systems
 - Need to develop power-aware tools and techniques to aid the user to properly direct the system's use of power



What is needed to support the Performance, Programmability, and Power?

- **Transitioning tools** to assist the user in assessing and re-balancing an application for performance and power
- An **environment** that allows the developer **to change applications a little at a time**
- Provide support for **user-directed tuning guidance**
 - for both performance and power
- Provide **analysis and automation** as we learn more

Creating Hybrid Codes, Why Should You Care?



- **For the next decade (at least) all HPC systems will have the same basic architecture:**
 - Communication between nodes
 - Application developers will continue to use MPI between nodes or sockets
 - Maybe SHMEM, UPC, Coarray
 - Shared memory programming paradigm on the node
 - MPI only will not do
 - Vectorization at the low level looping structures
 - SSE, AVX, ...
 - GPU, MIC, ...
 - etc
- **Hybridizing a code gives many performance advantages**
 - Resource contention (network, node memory, ...)
- **While there is a potential acceptance of new languages for addressing multiple levels of parallelism, 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**



Moving to a Hybrid Code: a Three-Task Approach

- **Identify potential loops to accelerate**
 - **Determine where to add additional levels of parallelism**
 - Assumes the MPI application is functioning correctly
 - Find top work-intensive loops
- **Parallelize and vectorize identified loops**
 - **Split loop work among threads**
 - Do parallel analysis and restructuring on targeted high level loops
- **Add OpenMP (and then OpenACC directives)**
 - **Add parallel directives and acceleration extensions**
 - Insert OpenMP directives
 - Verify application and check for performance improvements
 - Convert desired OpenMP directives to OpenACC
- **We want a performance-portable application at the end**

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,p1,flat)
call parabola(nmin-4,nmax+4, para,r,dr,r6,r1,flat)
call parabola(nmin-4,nmax+4,para,u,du,u6,ul,flat)

call states(p1,ul,r1,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

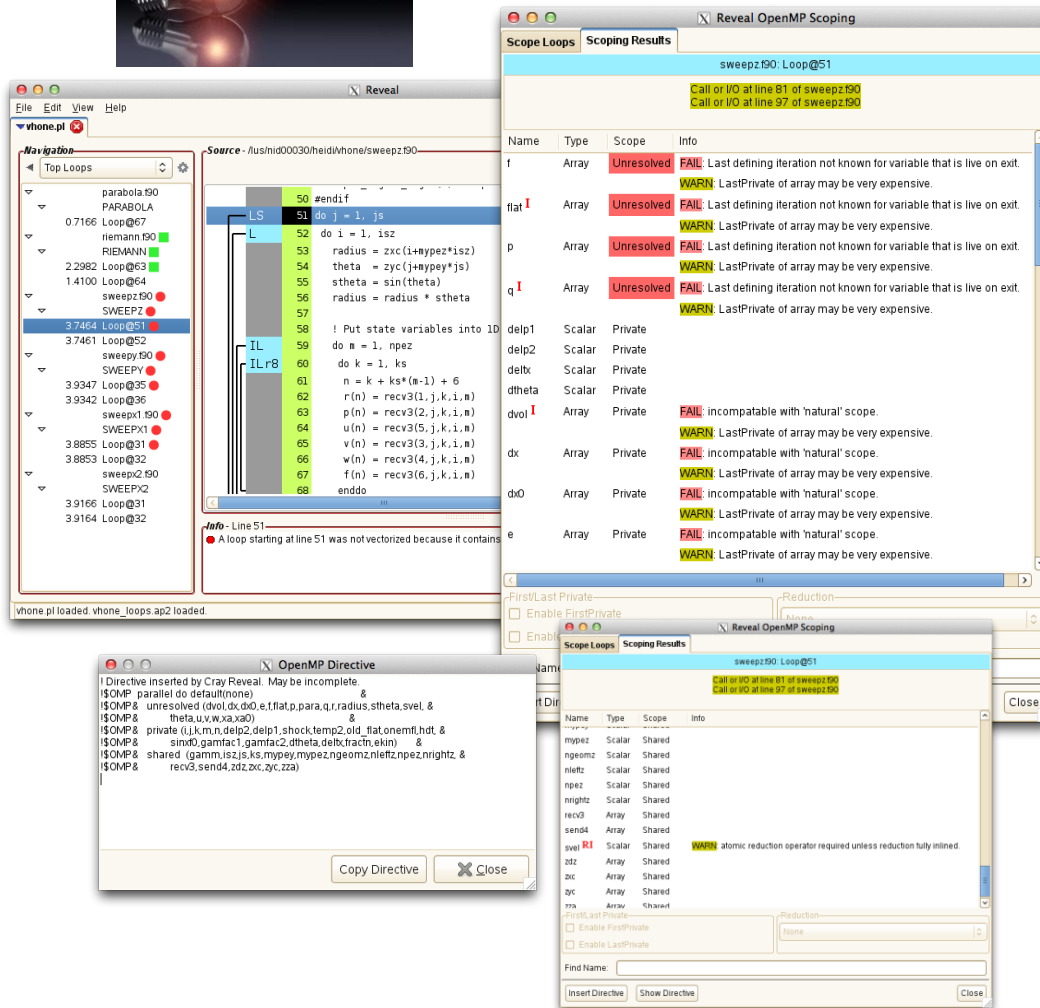
```



Tools Needed When Creating Hybrid Codes

- **Tools to design your application to be performance-portable across a wide range of systems**
 - Application developers want to develop a single code that can run efficiently on multi-core nodes with or without an accelerator
- **Tools to understand your application**
 - Where is the time spent?
 - Where are the key parallel structures (loops)?
 - Where are the important arrays used?
 - What prohibits parallelizing these structures?
- **Tools to help with scoping analysis**
 - Identify shared, private and ambiguous variables

Simplifying the Task with Reveal

Scoping Results Table:

Name	Type	Scope	Info
f	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit. WARN: LastPrivate of array may be very expensive.
flat	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit. WARN: LastPrivate of array may be very expensive.
p	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit. WARN: LastPrivate of array may be very expensive.
q	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit. WARN: LastPrivate of array may be very expensive.
delp1	Scalar	Private	
delp2	Scalar	Private	
deltx	Scalar	Private	
dtheta	Scalar	Private	
dx	Array	Private	FAIL: incompatible with 'natural' scope. WARN: LastPrivate of array may be very expensive.
dx0	Array	Private	FAIL: incompatible with 'natural' scope. WARN: LastPrivate of array may be very expensive.
e	Array	Private	FAIL: incompatible with 'natural' scope. WARN: LastPrivate of array may be very expensive.

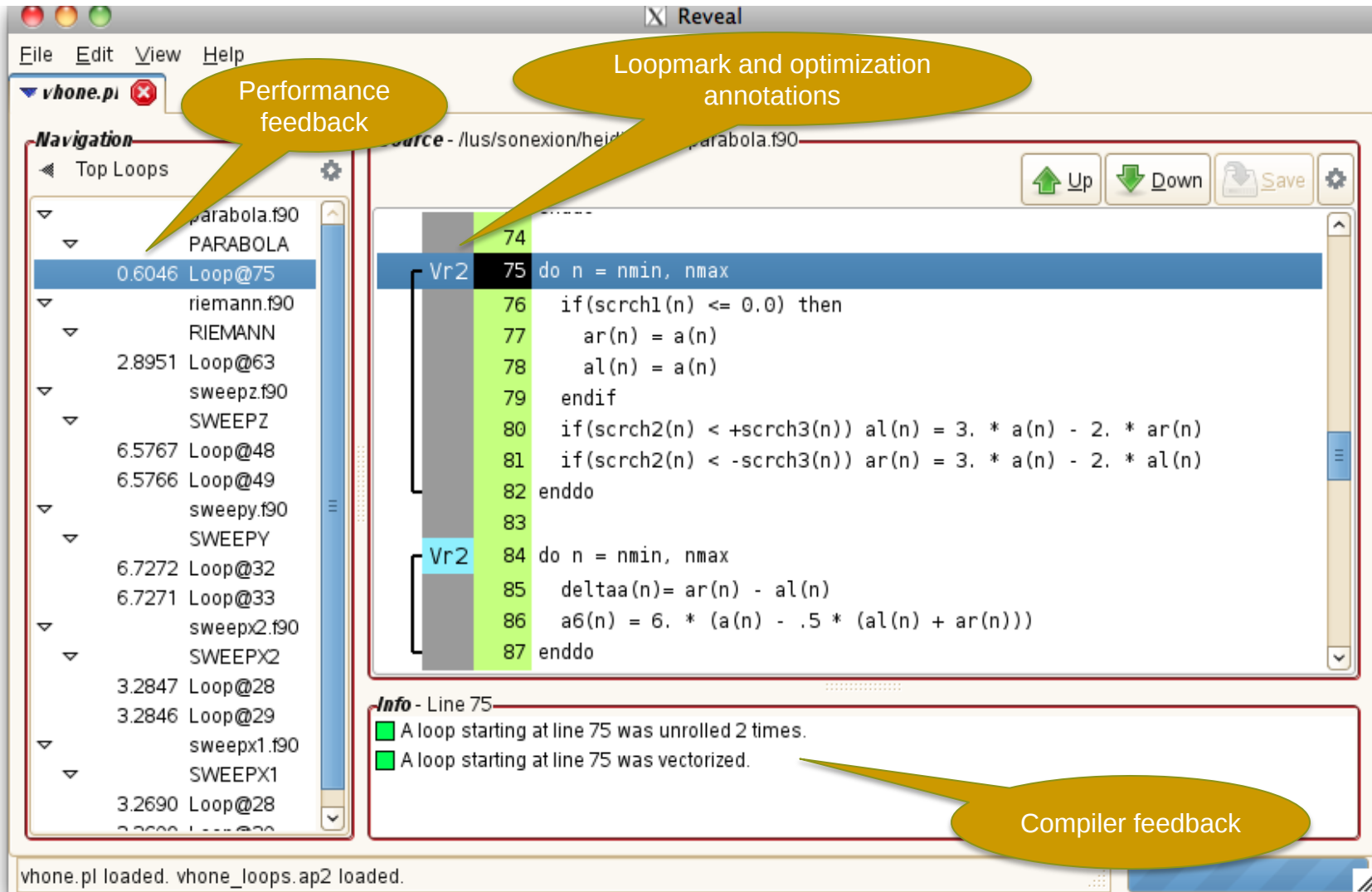
OpenMP Directive:

```

! Directive inserted by Cray Reveal. May be incomplete.
! $OMP parallel do default(none)
! $OMP & unresolved (dvo,dx,dx0,e,flat,p,q,radius,stheta,svel, &
! $OMP & theta,u,v,x,xao)
! $OMP & private (j,k,m,n,delp1,delp2,shock,temp2,old_flat,oneht,hot, &
! $OMP & simo,gamfac1,gamfac2,dtheta,deltx,ekin) &
! $OMP & shared (gamma,sz,sz,sz,mpyez,mpyez,ngomz,nlefft,npeznright, &
! $OMP & recv3,send4,zdz,zc,zc,zza)
  
```

- 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

Visualize Loopmark with Performance Information



Performance feedback

Loopmark and optimization annotations

Compiler feedback

Navigation

- Top Loops
- parabola.f90
 - PARABOLA
 - 0.6046 Loop@75
- riemann.f90
 - RIEMANN
 - 2.8951 Loop@63
- sweepz.f90
 - SWEEPZ
 - 6.5767 Loop@48
 - 6.5766 Loop@49
- sweepy.f90
 - SWEEPY
 - 6.7272 Loop@32
 - 6.7271 Loop@33
- sweepx2.f90
 - SWEEPX2
 - 3.2847 Loop@28
 - 3.2846 Loop@29
- sweepx1.f90
 - SWEEPX1
 - 3.2690 Loop@28

Source - /lus/sonexion/heid... parabola.f90

```

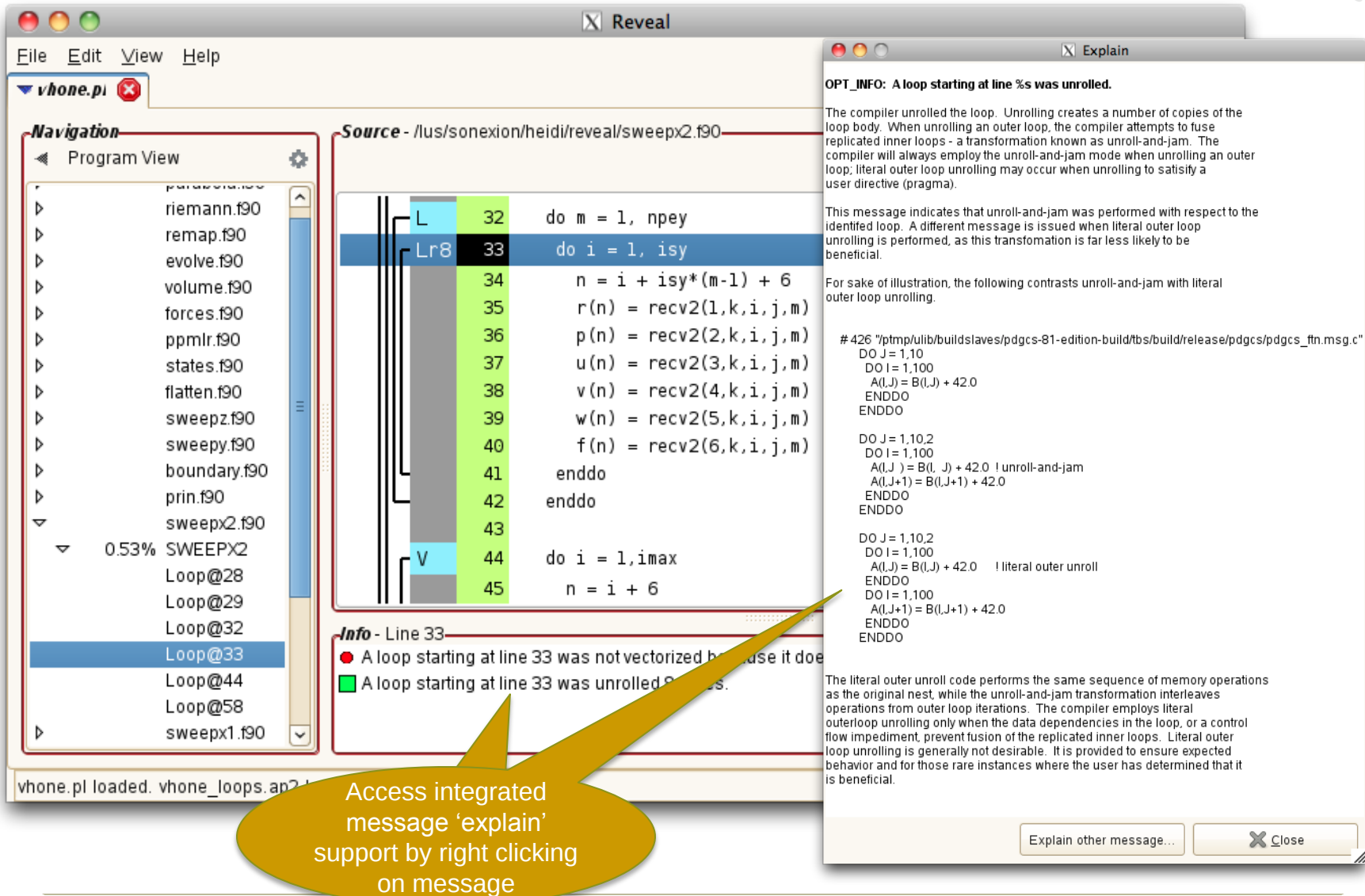
74
Vr2 75 do n = nmin, nmax
76   if(scrch1(n) <= 0.0) then
77     ar(n) = a(n)
78     al(n) = a(n)
79   endif
80   if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
81   if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
82 enddo
83
Vr2 84 do n = nmin, nmax
85   deltaa(n)= ar(n) - al(n)
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 Cray Compiler Message Information



The screenshot shows the Cray compiler interface with two main windows: 'Reveal' and 'Explain'.

Reveal Window:

- Navigation:** A tree view on the left showing the project structure. The file 'sweepx2.f90' is selected under the '0.53% SWEEPX2' folder.
- Source:** The main window displays the source code for 'sweepx2.f90'. The code is as follows:


```

      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
      
```
- Info:** A message box at the bottom of the source window indicates:
 - A loop starting at line 33 was not vectorized because it does not have a constant stride.
 - A loop starting at line 33 was unrolled 8 times.

Explain Window:

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/pdgc81-edition-build/tbs/build/release/pdgc81/pdgc81_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

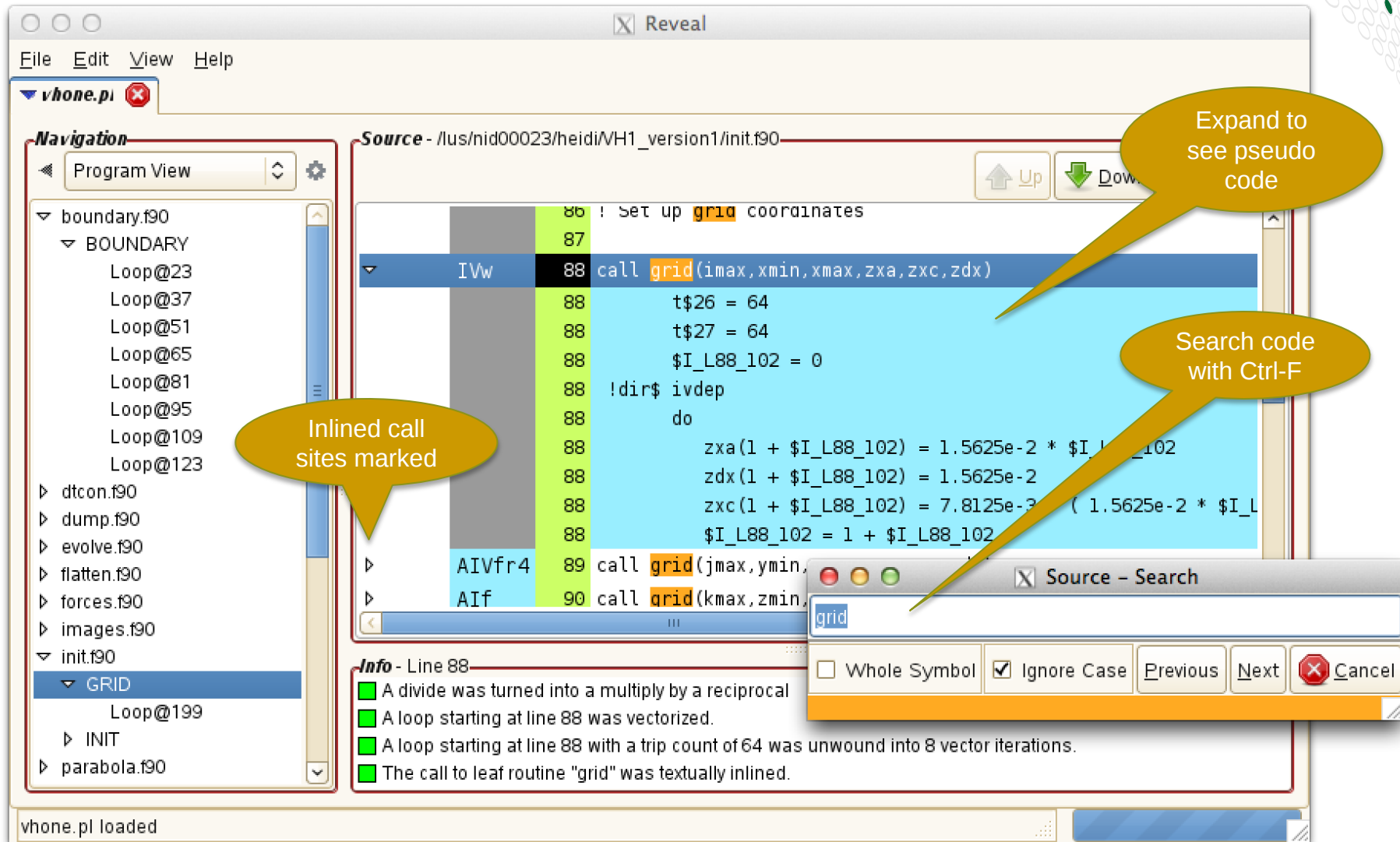
DO J = 1,10,2
DO I = 1,100
A(I,J) = B(I,J) + 42.0 ! literal outer unroll
ENDDO
DO I = 1,100
A(I,J+1) = B(I,J+1) + 42.0
ENDDO
ENDDO

```

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 outer loop 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

View Pseudo Code for Inlined Functions



The screenshot shows the 'Reveal' IDE interface with the file 'vhone.pl' open. The 'Navigation' pane on the left shows a tree view of the code structure, with 'init.f90' and 'GRID' selected. The 'Source' pane displays the pseudo code for the 'grid' function, which has been inlined. The code is color-coded, and call sites are marked with 'IVw', 'AIVfr4', and 'AIf'. A callout bubble points to the 'grid' function name in the code, stating 'Expand to see pseudo code'. Another callout bubble points to the 'grid' function name in the search box, stating 'Search code with Ctrl-F'. A third callout bubble points to the 'IVw' marker, stating 'Inlined call sites marked'. A search window titled 'Source - Search' is open, showing the search results for 'grid'.

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,
90 call grid(kmax,zmin,

```

Info - Line 88

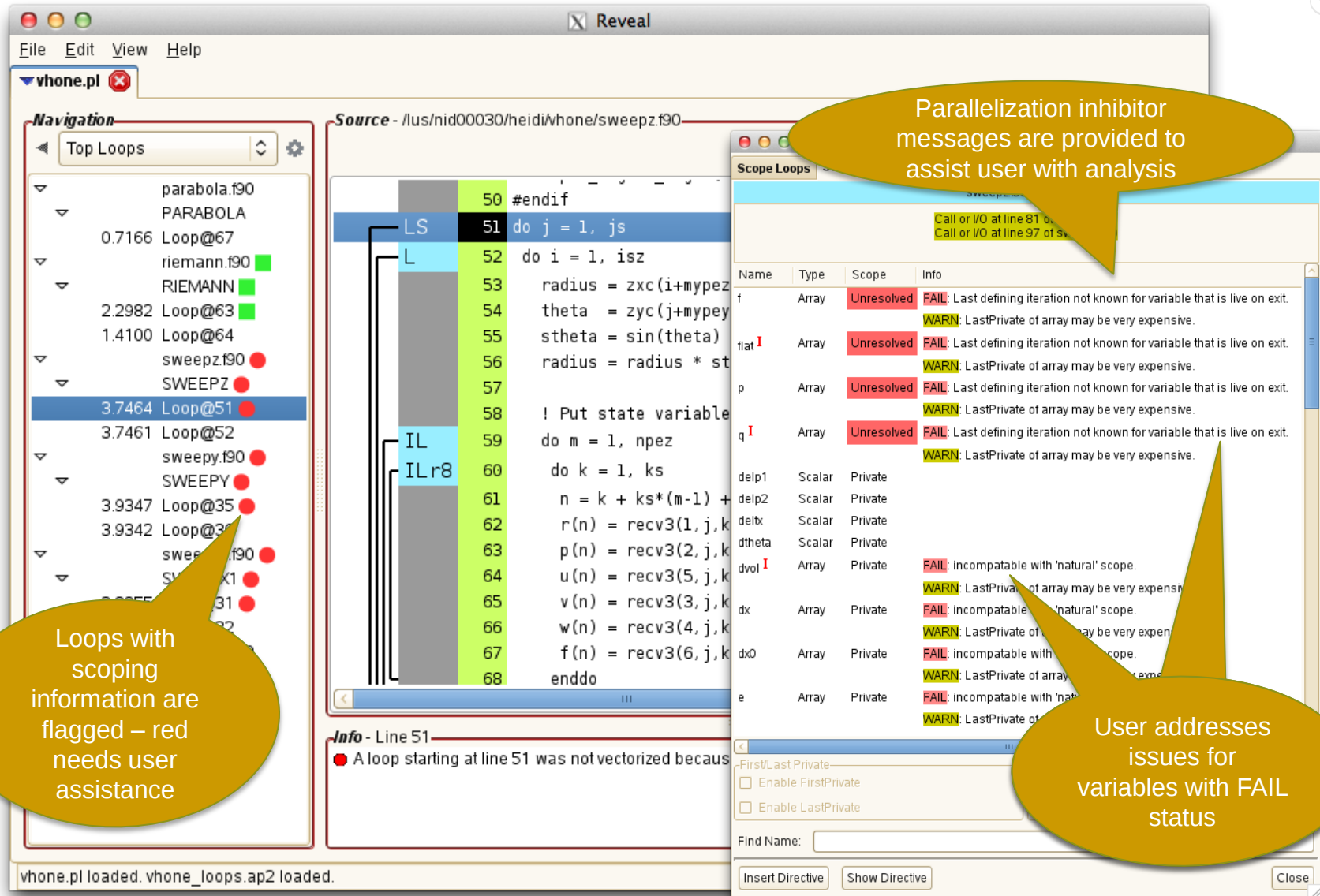
- A divide was turned into a multiply by a reciprocal
- A loop starting at line 88 was vectorized.
- A loop starting at line 88 with a trip count of 64 was unwound into 8 vector iterations.
- The call to leaf routine "grid" was textually inlined.

Source - Search

grid

☐ Whole Symbol ☒ Ignore Case Previous Next Cancel

Scoping Assistance – Review Scoping Results



Navigation

- Top Loops
 - parabola.f90
 - PARABOLA
 - 0.7166 Loop@67
 - riemann.f90
 - RIEMANN
 - 2.2982 Loop@63
 - 1.4100 Loop@64
 - sweepz.f90
 - SWEEPZ
 - 3.7464 Loop@51
 - 3.7461 Loop@52
 - sweepy.f90
 - SWEEPY
 - 3.9347 Loop@35
 - 3.9342 Loop@36
 - sweepx.f90
 - SWEEPX
 - 3.9347 Loop@35
 - 3.9342 Loop@36
 - sweepz.f90
 - SWEEPZ
 - 3.7464 Loop@51
 - 3.7461 Loop@52
 - sweepy.f90
 - SWEEPY
 - 3.9347 Loop@35
 - 3.9342 Loop@36
 - sweepx.f90
 - SWEEPX

Source - /lus/nid00030/heidi/vhone/sweepz.f90

```

50 #endif
51 do j = 1, js
52   do i = 1, isz
53     radius = zxc(i+mypex, j)
54     theta = zyc(j+mypey, i)
55     stheta = sin(theta)
56     radius = radius * stheta
57   enddo
58   ! Put state variable
59   do m = 1, npez
60     do k = 1, ks
61       n = k + ks*(m-1) +
62       r(n) = recv3(1, j, k)
63       p(n) = recv3(2, j, k)
64       u(n) = recv3(5, j, k)
65       v(n) = recv3(3, j, k)
66       w(n) = recv3(4, j, k)
67       f(n) = recv3(6, j, k)
68     enddo

```

Scope Loops

Name	Type	Scope	Info
f	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit. WARN: LastPrivate of array may be very expensive.
flat	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit. WARN: LastPrivate of array may be very expensive.
p	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit. WARN: LastPrivate of array may be very expensive.
q	Array	Unresolved	FAIL: Last defining iteration not known for variable that is live on exit. WARN: LastPrivate of array may be very expensive.
delp1	Scalar	Private	
delp2	Scalar	Private	
delbx	Scalar	Private	
dtheta	Scalar	Private	
dvol	Array	Private	FAIL: incompatible with 'natural' scope. WARN: LastPrivate of array may be very expensive.
dx	Array	Private	FAIL: incompatible with 'natural' scope. WARN: LastPrivate of array may be very expensive.
dx0	Array	Private	FAIL: incompatible with 'natural' scope. WARN: LastPrivate of array may be very expensive.
e	Array	Private	FAIL: incompatible with 'natural' scope. WARN: LastPrivate of array may be very expensive.

Info - Line 51

- A loop starting at line 51 was not vectorized because...

vhone.pl loaded. vhone_loops.ap2 loaded.

Parallelization inhibitor messages are provided to assist user with analysis

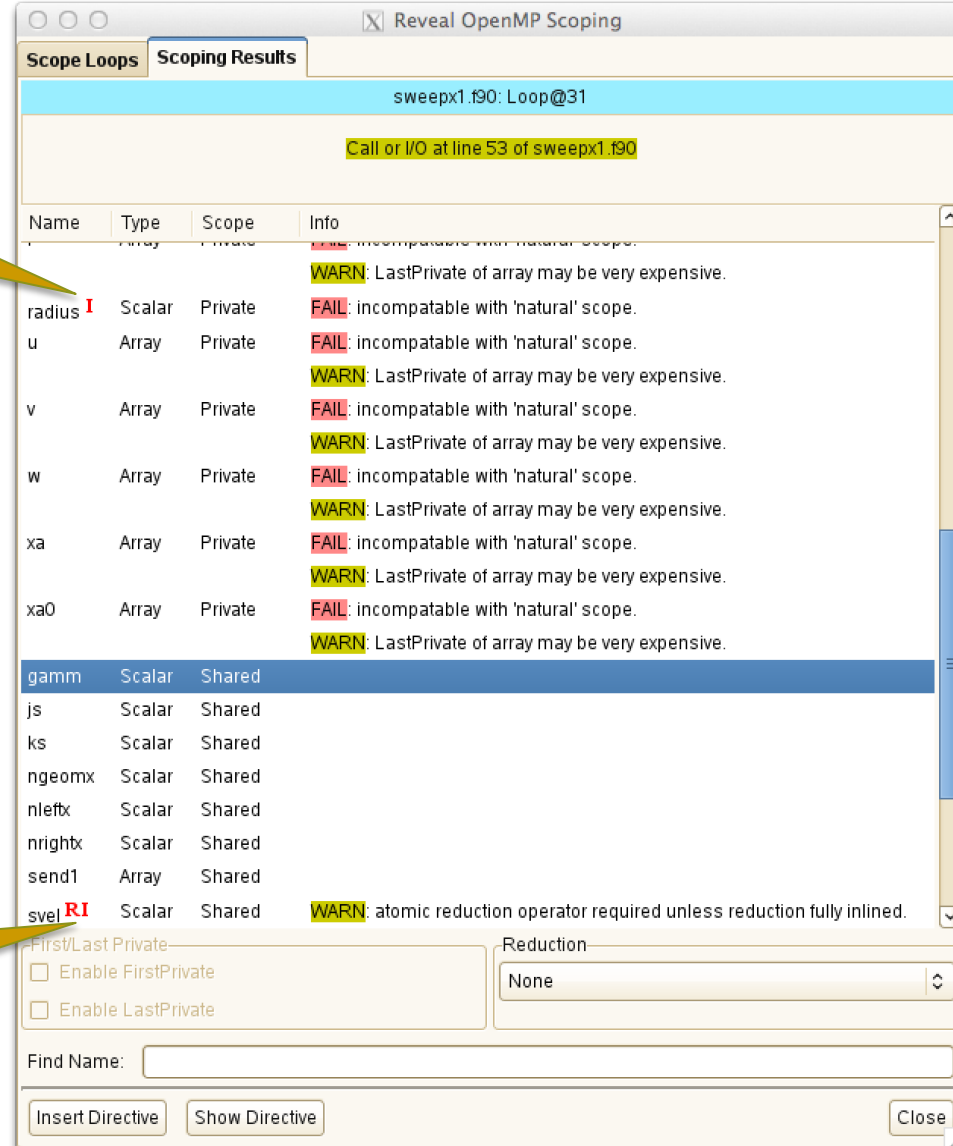
Loops with scoping information are flagged – red needs user assistance

User addresses issues for variables with FAIL status

Reveal Gives Feedback on Scoping Results

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

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



Name	Type	Scope	Info
radius	Scalar	Private	I WARN: LastPrivate of array may be very expensive. FAIL: incompatible with 'natural' scope.
u	Array	Private	FAIL: incompatible with 'natural' scope.
v	Array	Private	WARN: LastPrivate of array may be very expensive. FAIL: incompatible with 'natural' scope.
w	Array	Private	WARN: LastPrivate of array may be very expensive. FAIL: incompatible with 'natural' scope.
xa	Array	Private	WARN: LastPrivate of array may be very expensive. FAIL: incompatible with 'natural' scope.
xa0	Array	Private	WARN: LastPrivate of array may be very expensive. FAIL: incompatible with 'natural' scope.
gamm	Scalar	Shared	
js	Scalar	Shared	
ks	Scalar	Shared	
ngeomx	Scalar	Shared	
nleftx	Scalar	Shared	
nrightx	Scalar	Shared	
send1	Array	Shared	
sycl	Scalar	Shared	RI WARN: atomic reduction operator required unless reduction fully inlined.

First/Last Private:

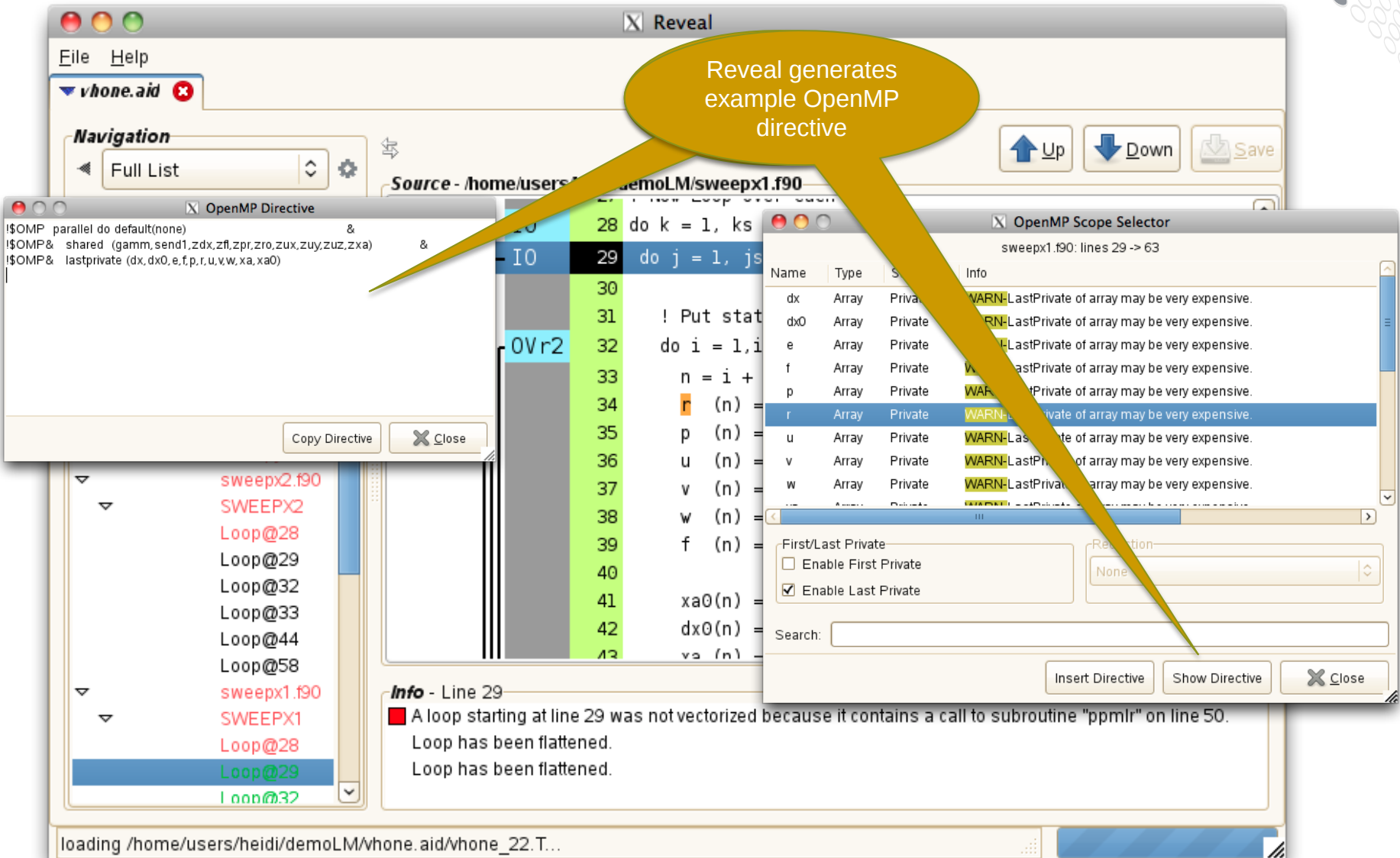
☐ Enable FirstPrivate

☐ Enable LastPrivate

Reduction: None

Find Name:

Scoping Assistance – Reveal Generates Directive



The screenshot shows the Reveal IDE interface with several windows open:

- Reveal (Main Window):** Displays the source code for `sweepx1.f90`. A yellow callout bubble points to the code, stating: "Reveal generates example OpenMP directive". The code includes a loop starting at line 29:


```
do k = 1, ks
  do j = 1, js
    ! Put stat
    do i = 1, i
      n = i +
      r (n)
      p (n)
      u (n)
      v (n)
      w (n)
      f (n)
      xa0(n)
      dx0(n)
      va (n)
```
- OpenMP Directive:** A window showing the generated OpenMP directive:


```
!$OMP parallel do default(none) &
!$OMP shared (gamm,send1,zdx,zfl,zpr,zro,zux,zuy,zuz,zxa) &
!$OMP lastprivate (dx,dx0,e,f,p,r,u,v,w,xa,xa0)
```
- OpenMP Scope Selector:** A window titled "sweepx1.f90: lines 29 -> 63" showing a table of variables and their types. The table has columns: Name, Type, Scope, and Info.

Name	Type	Scope	Info
dx	Array	Private	WARN: LastPrivate of array may be very expensive.
dx0	Array	Private	WARN: LastPrivate of array may be very expensive.
e	Array	Private	WARN: LastPrivate of array may be very expensive.
f	Array	Private	WARN: LastPrivate of array may be very expensive.
p	Array	Private	WARN: LastPrivate of array may be very expensive.
r	Array	Private	WARN: LastPrivate of array may be very expensive.
u	Array	Private	WARN: LastPrivate of array may be very expensive.
v	Array	Private	WARN: LastPrivate of array may be very expensive.
w	Array	Private	WARN: LastPrivate of array may be very expensive.

 Below the table, there are checkboxes for "First/Last Private":
 - ☐ Enable First Private
 - ☒ Enable Last Private
 A "Search:" field is also present.
- Info - Line 29:** A window showing a message:


```
■ A loop starting at line 29 was not vectorized because it contains a call to subroutine "ppmlr" on line 50.
Loop has been flattened.
Loop has been flattened.
```
- Navigation:** A window showing a list of files and loops, including `sweepx2.f90`, `SWEEPX2`, `Loop@28`, `Loop@29`, `Loop@32`, `Loop@33`, `Loop@44`, `Loop@58`, `sweepx1.f90`, `SWEEPX1`, `Loop@28`, `Loop@29`, and `Loop@32`.

Use Reveal to Validate User Inserted Directives



The screenshot shows the Reveal IDE interface. The main window displays a Fortran source file named `riemann.f90` with the following code:

```
64 !$OMP parallel do default(none)
65 !$OMP private (n)
66 !$OMP shared (lmin,lmax,prgh,urgh,vrgh,plft,ulft,vlft,pmid,clft,
67 !$OMP crgh,l,plfti,pmold,prghi,umidl,umidr,wlft,wrgh,zlft,
68 !$OMP zrgh,gamfac1,gamfac2)
69 do l = lmin, lmax
70 do n = 1, 12
71 pmold(l) = pmid(l)
72 wlft(l) = 1.0 + gamfac1*(pmid(l) - plft(l))
73 wrgh(l) = 1.0 + gamfac1*(pmid(l) - plft(l))
74 wlft(l) = clft(l) * sqrt(wlft(l))
75 wrgh(l) = crgh(l) * sqrt(wrgh(l))
76 zlft(l) = 4.0 * vlft(l) * wlft(l) * w
77 zrgh(l) = 4.0 * vrgh(l) * wrgh(l) * w
78 zlft(l) = -zlft(l) * wlft(l)/(zlft(l)
79 zrgh(l) = zrgh(l) * wrgh(l)/(zrgh(l)
80 umidl(l) = ulft(l) - (pmid(l) - plft(l)
81 umidr(l) = urgh(l) + (pmid(l) - prgh(l)
82 pmid(l) = pmid(l) + (umidr(l) - umidl(l)
83 pmid(l) = max(smallp,pmid(l))
84 if (abs(pmid(l)-pmold(l))/pmid(l) < to
85 enddo
86 enddo
```

A yellow callout bubble points to the `do l = lmin, lmax` line, stating: "User inserted directive with mis-scoped variable 'l'".

The left sidebar shows a navigation tree with the following structure:

- boundary.f90
 - BOUNDARY
- dtcon.f90
- dump.f90
- evolve.f90
- flatten.f90
- forces.f90
- 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

The bottom status bar shows "vhone.pl loaded".

The right sidebar shows the "Scope Loops" and "Scoping Results" panels. The "Scoping Results" panel displays a table of variables and their scopes:

Name	Type	Scope	Info
l	Scalar	Private	WARN: Scope does not agree with user OMP directive.
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	

The "Info - Line 69" panel shows the following messages:

- A loop starting at line 69 was not vectorized for an unspecified reason.
- A loop starting at line 69 was partitioned.

The bottom right panel shows the "First/Last Private" and "Reduction" settings, with a search bar and buttons for "Insert Directive", "Show Directive", and "Close".

COMPUTE | STORE | ANALYZE

Moving to a Hybrid Code: a Three-Task Approach

- **Identify potential loops to accelerate**
 - **Determine where to add additional levels of parallelism**
 - Assumes the MPI application is functioning correctly
 - Find top work-intensive loops
 - **Performance tools + compiler loop work estimates**
- **Parallelize and vectorize identified loops**
 - **Split loop work among threads**
 - Do parallel analysis and restructuring on targeted high level loops
 - **Reveal with compiler feedback and source code browsing**
- **Add OpenMP (and then OpenACC directives)**
 - **Add parallel directives and acceleration extensions**
 - Insert OpenMP directives
 - **Reveal with compiler scoping assistance**
 - Verify application and check for performance improvements
 - Convert desired OpenMP directives to OpenACC
- **We want a performance-portable application at the end**

Debugging on Extreme Scale Systems

- Systems with thousands of threads of execution need a new debugging paradigm
- Need to build tools around traditional debuggers with innovative techniques for productivity and **scalability**
 - Support for traditional control-centric debugging mechanism
- **STAT - Stack Trace Analysis Tool**
 - MRNet based scalable generation of a single, merged, stack backtrace tree
- **ATP - Abnormal Termination Processing**
 - Scalable analysis of a sick application, delivering a STAT tree and a minimal, comprehensive, core file set.
- **Comparative debugging**
 - Collaboration with University Queensland
 - A **data-centric paradigm**
 - Ability to see data from multiple program executions in the same instance



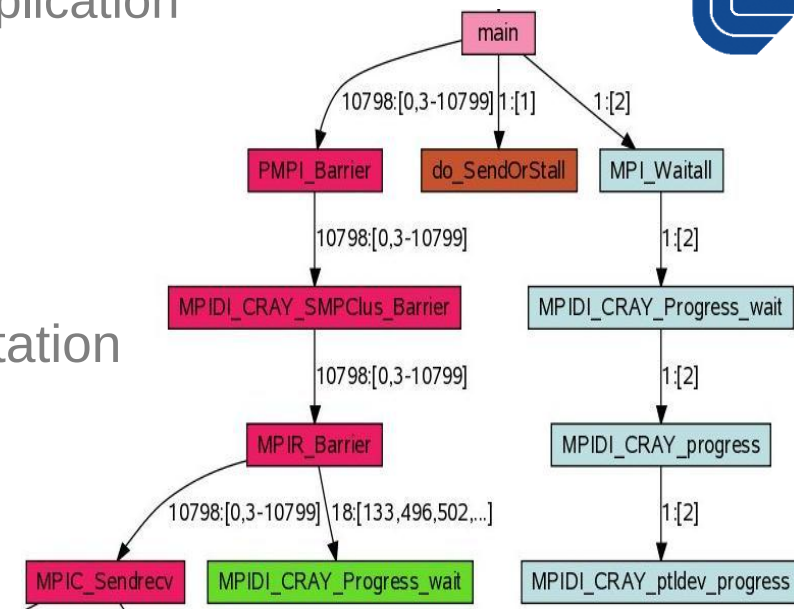
Stack Trace Analysis Tool (STAT)

- **Stack trace sampling and analysis for large scale applications**

- Sample application stack traces
- Scalable generation of a single, merged, stack backtrace tree
 - A comprehensible view of the entire application
 - Discover equivalent process behavior
 - Group similar processes
 - Reduce number of tasks to debug

- **Merge/analyze traces:**

- Facilitate scalable analysis/data presentation
- Multiple traces over space or time
- Create call graph prefix tree
 - Compressed representation
 - Scalable visualization & analysis



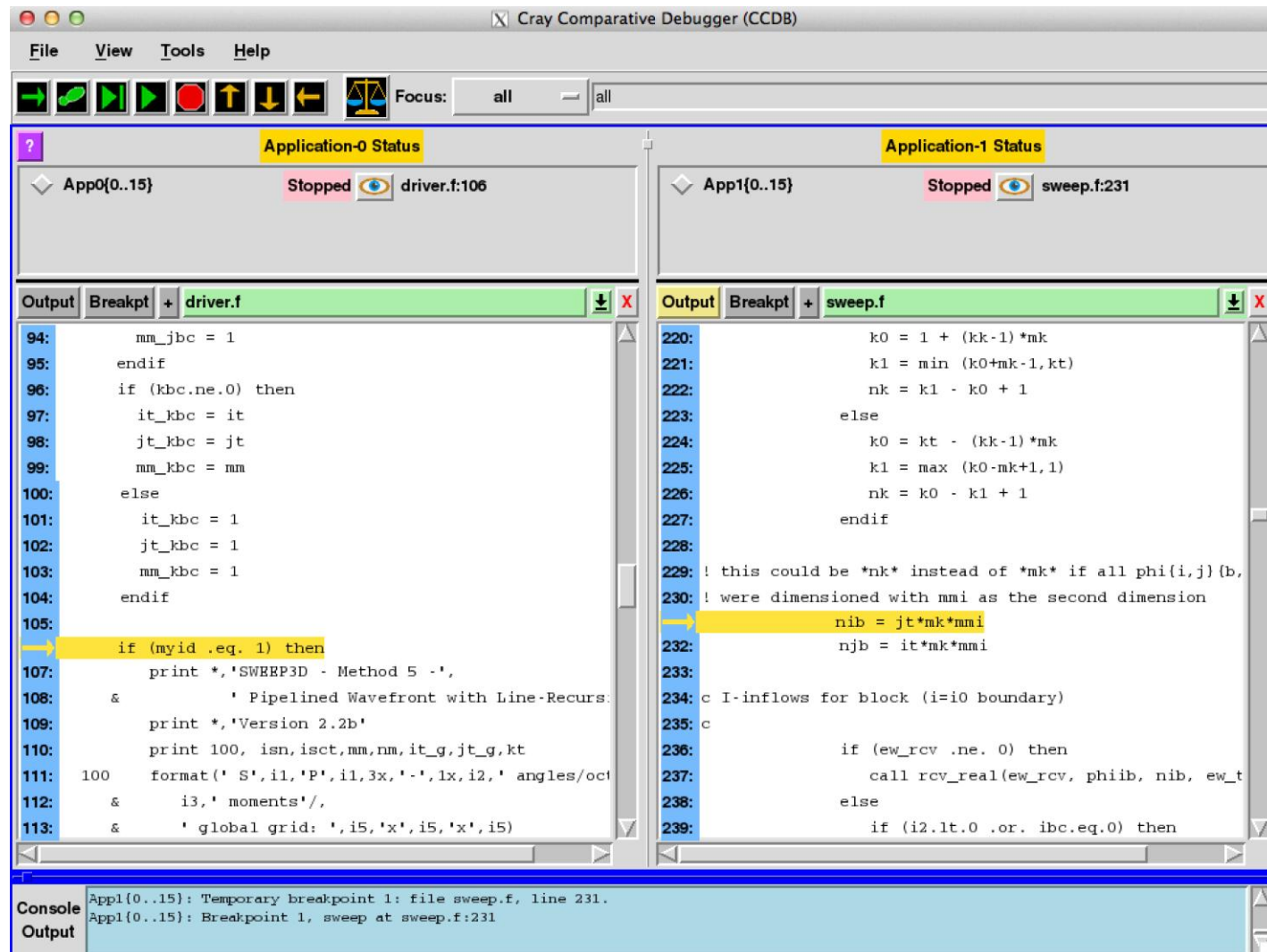
- **ATP - Automatic Termination Processing**

- System of light weight back-end monitor processes on compute nodes
- Leap into action on any application process trapping

Comparative Debugger

- Helps the programmer locate errors in the program by observing the divergence in key data structures as the programs are executing
- Allows comparison of a “suspect” program against a “reference” code using assertions
 - Simultaneous execution of both
 - Ability to assert the match of data at given points in execution
 - Focus on data – not state and internal operations
 - Narrow down problem without massive thread study
- Data comparison
 - Tolerance control – nobody expect it to be perfect
 - Array subsets – correlate serial to parallel bits
 - Array index permutation – loops rearranged
 - Automated asserts – let it run until a problem is found
 - Forcing correct values – continue on with correct data

Cray Comparative Debugger



The screenshot shows the Cray Comparative Debugger (CCDB) interface. The title bar reads "Cray Comparative Debugger (CCDB)". The menu bar includes "File", "View", "Tools", and "Help". Below the menu bar is a toolbar with various icons for navigation and execution. A "Focus:" dropdown menu is set to "all".

The main workspace is divided into two panels:

- Application-0 Status:** Shows "App0{0..15}" in a "Stopped" state at "driver.f:106". Below this is a tabbed view with "Output", "Breakpt", and "driver.f" selected. The code for "driver.f" is displayed, with line 105 highlighted: `if (myid .eq. 1) then`.
- Application-1 Status:** Shows "App1{0..15}" in a "Stopped" state at "sweep.f:231". Below this is a tabbed view with "Output", "Breakpt", and "sweep.f" selected. The code for "sweep.f" is displayed, with line 230 highlighted: `nib = jt*mk*mmi`.

At the bottom, a "Console Output" pane shows the following messages:

```
App1{0..15}: Temporary breakpoint 1: file sweep.f, line 231.
App1{0..15}: Breakpoint 1, sweep at sweep.f:231
```

CCDB - Comparison



CCDB Comparison

Application-0

App0{0..15}

sweep.f:160

Application-1

App1{0..15}

sweep.f:160

Up

Down

	Name or Expression	Type	Results	Type Template	App-0 Decomp	App-1 Decomp	Op	Eps
☐	jkm	INTEGER*4	Click to see results				==	e
☐	mio	INTEGER*4	Click to see results				==	e
☐	nk	INTEGER*4	Click to see results				==	e
☐	ti	REAL*8	Click to see results				==	e
☐	tj	REAL*8	Click to see results				==	e
☐	weta	REAL*8 (6)	Click to see results		None	None	==	e
☐	wmu	REAL*8 (6)	Click to see results		None	None	==	e

Compare

Add Comparison

Result Filter:

all

fail

warn

pass

todo

Close

CCDB – Comparison



CCDB Comparison

Application-0

App0{0..15}

sweep.f:160

Application-1

App1{0..15}

sweep.f:160

Up

Down

	Name or Expression	Type	Results	Type Template	App-0 Decomp	App-1 Decomp	Op	Eps
☐	jkm	INTEGER*4	Click to see results				==	e
☐	mio	INTEGER*4	Click to see results				==	e
☐	nk	INTEGER*4	Click to see results				==	e
☐	ti	REAL*8	Click to see results				==	e
☐	tj	REAL*8	Click to see results				==	e
☐	weta	REAL*8 (6)	Click to see results		None		==	e
☐	wmu	REAL*8 (6)	Click to see results		None		==	e

Compare

Add Comparison

Result Filter: ☐ all ☒ fail ☐ warn ☐ pass ☐ todo

Close

App0{0}: Comparison is true.

App0{1..7}: Comparison is false.

App0{8..15}: Comparison is false.

The difference delta is:

App0{1..7}: 64

App0{8..15}: 64



Why Application Level Power Management?

- **Power consumption is one of the biggest challenges facing extreme scale systems**
 - Scaling up from today's requirements for a petaflop computer is not a viable solution
 - Sites are already increasingly constrained by power & cooling limitations as well as cost of system power and cooling
- **Hardware could attempt to manage power automatically**
 - This strategy would most likely be reactive and certainly be sub-optimal
- **Power management must be proactive**
 - Software can help by monitoring and proactively managing how power is being used in the system
 - The programmer must understand the tradeoffs between power and performance
- **More research and development is needed for**
 - Power measurement and monitoring tools
 - Power analysis
 - Management of power consumption



Power Analysis

- **Ultimate goal is to develop power-aware tools and techniques**
 - Provide suggestions of possible transformations or implementations of the application with different performance/power profiles
 - Aid the user to properly direct the system's use of power
 - Using transparent automation where possible
 - Transparently adopting environmental settings to achieve a desired power reduction and energy savings with a predetermined impact to performance
 - Can leverage the work done in automatic performance analysis
- **Challenge**
 - Ability to identify and correctly attribute the power weights to the individual “operation” components and map back to the application



Power Measurement Wish List

- **Per H/W core or memory component**
 - How much does each H/W attribute (L1 cache, L2, cache, TLB, etc) draw?
 - Get relative power usage of components
 - Get information on joules per operator or per access
 - Example: how many joules does a DCACHE hit or miss use per reference?
 - Provide core utilization metrics
- **Overall program power measurement**
 - Similar to wallclock (total watts or joules)
 - Multiple power meters
 - 1 for core
 - 1 for memory system
- **Correlating HWPCs to power usage in performance tools**
 - Can be directly or indirectly
 - Roughly measured with existing H/W performance counters



Adaptive Power Management

- **Statically managed**

- Compiler support
 - Compile-time options to direct optimization for maximum or minimum
 - e.g., -P 1, 2, 3 like current -O 1, 2, 3
 - Compiler directives
 - Put this data there and keep it there until I tell you to release it
- Application-selected power modes/algorithms

- **Runtime managed**

- Based on compiler hints
 - Which components are about to be utilized more or less heavily
 - Integer algorithm coming up
 - Intensive floating point region coming up
 - Poor locality coming up, turn off cache use for this next segment
- Auto-tuning
 - Auto-tuning for performance and for power (and a combination of both)
- User Guided
 - Runtime assertions/options/hints



Outline

- Computer architecture and applications trends
- Programming Environment mission
- Tools focus for extreme scale computing
- **Open Issues**
- **Conclusions**



How About Resilience?

- On extreme scale systems the probability of a node failure is substantial
- Extreme-scale applications will have built-in resilience support
 - There will be support for detection of failure and information about which images failed
 - e.g., error return for synchronization and collective operations can indicate that an image has failed
- Will tools be ready for resilient applications?
- Will tools be resilient themselves?

Other Issues to Think About

- **Running on an extreme-scale environment**
 - Target **scalability** issues in all areas of tool development
- **Volume of data**
 - Need to rethink the performance measurement approach
 - Can we focus only on things that are different?
 - What kind of runtime aggregation will be possible?
 - How about the “Heisenbug” effect?
- What can you do to prep the tools?
 - Availability of extreme-scale systems will be limited!
- Cost/benefit estimates (porting apps)
 - Can you provide an estimate to the user of the improvements that certain transformations will bring?
- Ease of use
 - Automatic program instrumentation
 - Automatic analysis



CrayPat-lite

Access light version of performance tools software

```
> module load perftools-lite
```

Build program

```
> make
```



a.out (instrumented program)

Run program (no modification to batch script)

```
aprun a.out
```



Condensed report to stdout
a.out*.rpt (same as stdout)
a.out*.ap2
MPICH_RANK_XXX files

Conclusions

- **Extreme scale systems will be more parallel, with more processors per node, more threads per processor, longer vector lengths, more complex memory hierarchies, and ...**
- **Application developers will need sophisticated tools and adaptive runtime systems to help them address issues of scale and complexity of these systems**
- **Extreme scale tools should address issues of**
 - Performance
 - Help users maximize the cycles to the application
 - Programmability
 - Ease developing, porting, and tuning efforts
 - Power
 - Help the user understand the tradeoffs between power and performance, and tune for both