

# Performance Measurement and Analysis Tools for Cray XE/XK Systems

Heidi Poxon  
Cray Inc.

# Topics

- CrayPat-lite
- Reveal

# CrayPat-lite

# CrayPat-lite Goals

- **Provide automatic application performance statistics at the end of a job**
  - Focus is to offer a simplified interface to basic application performance information for users not familiar with the Cray performance tools and perhaps new to application performance analysis
  - Gives sites the option to enable/disable application performance data collection for all users for a period of time
- **Keep traditional or “classic” perftools working the same as before**
- **Provide a simple way to transition from perftools-lite to perftools to encourage further tool use for performance analysis**

# Steps to Using CrayPat “classic”

## Access performance tools software

```
> module load perftools
```

## Build program, retaining .o files

```
> make
```



```
a.out
```

## Instrument binary

```
> pat_build -O apa a.out
```



```
a.out+pat
```

## Modify batch script and run program

```
aprun a.out+pat
```



```
a.out+pat*.xf
```

## Process raw performance data and create report

```
> pat_report a.out+pat*.xf
```



```
a.out+pat*.ap2
Text report to stdout
a.out+pat*.apa
MPICH_RANK_XXX
```

# Steps to Using 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
```

# Benefits of CrayPat-lite

- Program is automatically relinked to add instrumentation in a.out (pat\_build step done for the user)
- .o files are automatically preserved
- No modifications are needed to a batch script to run instrumented binary, since original binary is replaced with instrumented version
- pat\_report is automatically run before job exits
- Performance statistics are issued to stdout
- User can use “classic” CrayPat for more in-depth performance investigation

# Performance Statistics Available

- **Set of predefined experiments, enabled with the CRAYPAT\_LITE environment variable**
  - Sample\_profile
  - Event\_profile
  - GPU
- **Job information**
  - Number of MPI ranks, ranks per node, number of threads
  - Wallclock
  - High memory water mark
  - Aggregate MFLOPS (CPU only)
- **Profile of top time consuming routines with load balance**
- **Observations**
- **Instructions on how to get more information**



# Sample Output – LAMMPS

```
#####
#
#           CrayPat-lite Performance Statistics
#
#####
```

```
CrayPat/X:  Version 6.1.0.10863 Revision 10863 (xf 10658)  02/13/13
15:23:08
```

```
Number of PEs (MPI ranks):      64
Numbers of PEs per Node:        32  PEs on each of  2  Nodes
Numbers of Threads per PE:      1
Number of Cores per Socket:     16
Execution start time:  Fri Feb 15 14:42:24 2013
System name and speed:  mork 2100 MHz
```

```
Wall Clock Time:  122.608994 secs
High Memory:    45.70 MBytes
MFLOPS (aggregate):  15763.16 M/sec
```

# Sample Output (cont'd)

Table 1: Profile by Function Group and Function (top 7 functions shown)

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE
100.0%	101.961423	--	--	5315211.9	Total
-----					
<b>92.5%</b>	<b>94.267451</b>	<b>--</b>	<b>--</b>	<b>5272245.9</b>	<b>USER</b>
-----					
75.8%	77.248585	2.356249	3.0%	1001.0	LAMMPS_NS::PairLJCut::compute
6.5%	6.644545	0.105246	1.6%	51.0	LAMMPS_NS::Neighbor::half_bin_newton
4.1%	4.131842	0.634032	13.5%	1.0	LAMMPS_NS::Verlet::run
3.8%	3.841349	1.241434	24.8%	5262868.9	LAMMPS_NS::Pair::ev_tally
1.3%	1.288463	0.181268	12.5%	1000.0	LAMMPS_NS::FixNVE::final_integrate
=====					
<b>7.0%</b>	<b>7.110931</b>	<b>--</b>	<b>--</b>	<b>42637.0</b>	<b>MPI</b>
-----					
4.8%	4.851309	3.371093	41.6%	12267.0	MPI_Send
1.5%	1.536106	2.592504	63.8%	12267.0	MPI_Wait
=====					

# Sample Output (cont'd)

===== Observations and suggestions =====

## MPI Grid Detection:

There appears to be point-to-point MPI communication in a **4 X 2 X 8 grid** pattern. The execution time spent in MPI functions might be reduced with a rank order that maximizes communication between ranks on the same node. The effect of several rank orders is estimated below.

A file named `MPICH_RANK_ORDER.Grid` was generated along with this report and contains usage instructions and the Hilbert rank order from the following table.

Rank Order	On-Node Bytes/PE	On-Node Bytes/PE% of Total Bytes/PE	MPICH_RANK_REORDER_METHOD
<b>Hilbert</b>	<b>5.533e+10</b>	<b>90.66%</b>	<b>3</b>
Fold	4.907e+10	80.42%	2
SMP	4.883e+10	80.02%	1
RoundRobin	3.740e+10	61.28%	0

# Reveal

# Porting to a Hybrid or Many-core System

# 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**

***Cray performance tools and Reveal can help***

# Tools needed to Create Hybrid Codes

- **A good Programming Environment closes the gap between peak performance and possible performance**
  - A lot more than just a compiler
- **Specific tools needed for identifying the parallelism in an application**
  - Fine-grained profiling: loop level rather than routine
  - Profiling and character looping structures in a complex application
  - Scoping tools for investigating parallelisability of high-level looping structures
  - Tools for maintaining performance-portable applications
    - Application developers want to develop a single core that can run efficiently on multi-core nodes with or without an accelerator

# 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****



# Optimizations for Multi-core Systems

- Reduce number of MPI ranks per node
- Add parallelism to MPI ranks to take advantage of cores within a node while minimizing network injection contention
- Maximize on-node communication between MPI ranks
- Relieve on-node shared resource contention by pairing threads or processes that perform different work (for example computation with off-node communication) on the same node
- **Accelerate work intensive parallel loops**

# 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 from optimizations

# Step 1 - Identify possible accelerator kernels

# Loop Work Estimates

- **Helps identify high-level serial loops to parallelize**
  - Based on runtime analysis, approximates how much work exists within a loop
  - Provides min, max and average trip counts that can be used to approximate work and help carve up loop on GPU

# Collecting Loop Work Estimates

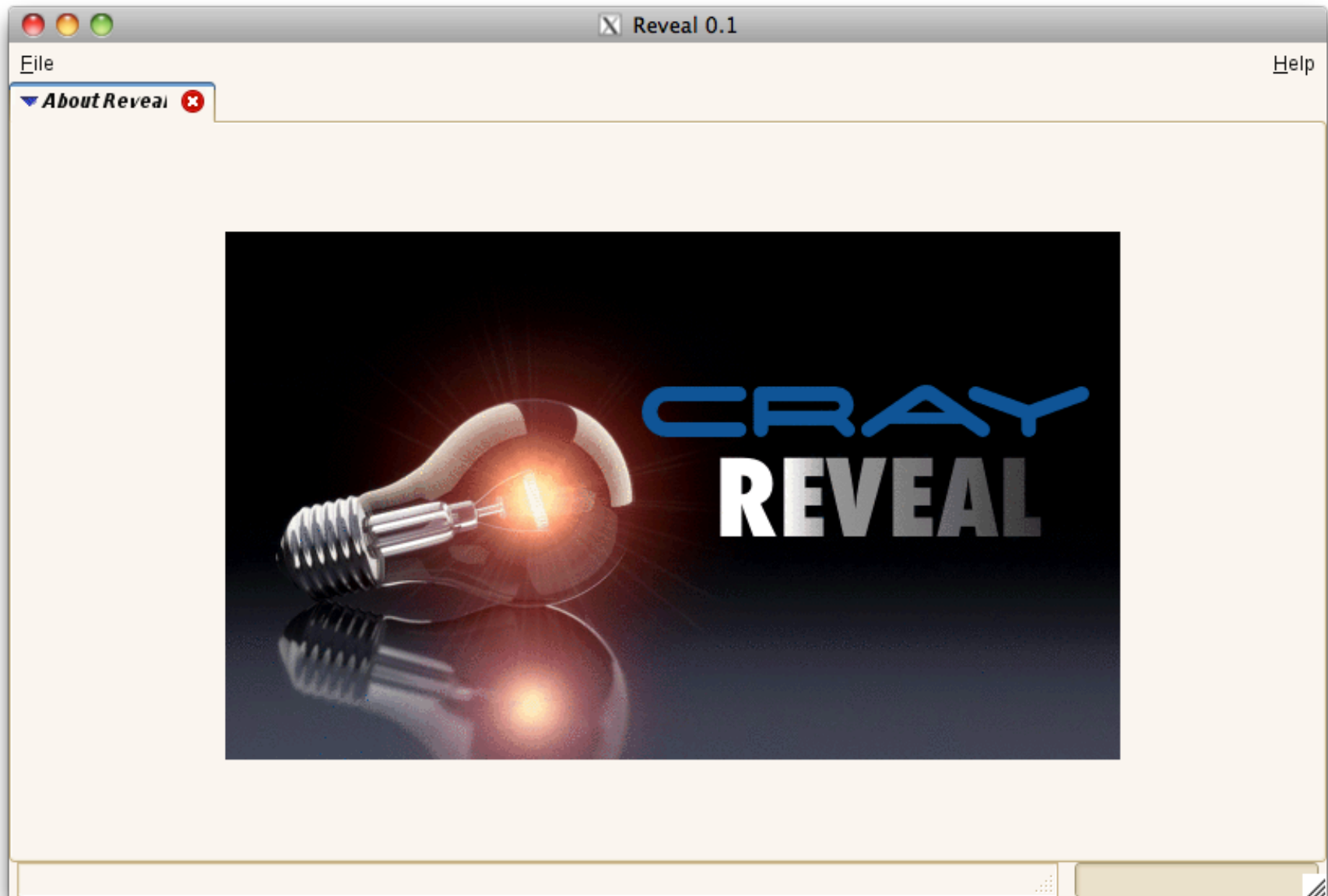
- Load PrgEnv-cray module
- Load perftools module
- Compile **AND** link with `-h profile_generate`
- Instrument binary for tracing
  - `pat_build -w my_program`
- Run application
- Create report with loop statistics
  - `pat_report my_program.xf > loops_report`

# 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

**Step 2 - Perform parallel  
analysis, scoping and  
vectorization  
&  
Step 3 - Move to  
OpenMP and then to  
OpenACC**





# Reveal

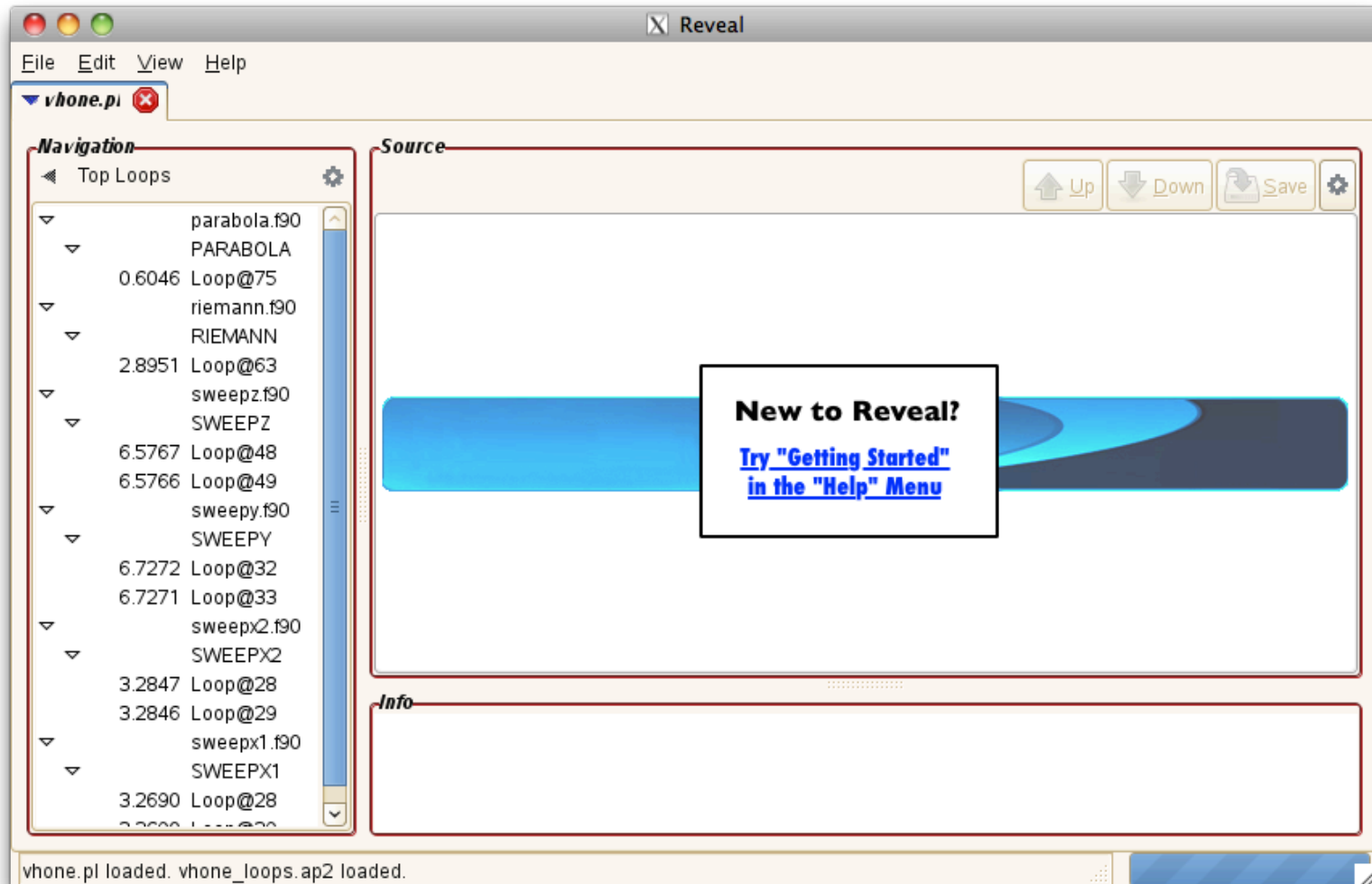
## New code analysis and restructuring assistant...

- 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

- **Optionally create loop statistics using the Cray performance tools to determine which loops have the most work**
- **Compile your application with Cray CCE to generate a program library**
  - > `ftn -h pl=vhone.pl -c file1.f90`
- **Run reveal**
  - **Compiler information only:**
    - > `reveal vhone.pl`
  - **Compiler + loop work estimates**
    - > `reveal vhone.pl vhone_loops.ap2`

# Reveal with Loop Work Estimates



# Visualize Loopmark with Performance Information

The screenshot shows the 'Reveal' tool interface with a file named 'vhone.pl'. The 'Navigation' pane on the left lists various loops, with '0.6046 Loop@75' selected. The main window displays the source code for 'parabola.f90' with a loopmark visualization. The loopmark shows a loop starting at line 75, which is unrolled 2 times and vectorized. The code includes a 'do' loop with nested 'if' statements and another 'do' loop below it. The 'Info' pane at the bottom provides compiler feedback: 'A loop starting at line 75 was unrolled 2 times.' and 'A loop starting at line 75 was vectorized.'

**Performance feedback**

**Loopmark and optimization annotations**

```

source - /lus/sonexion/h... 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.

**Compiler feedback**

vhone.pl loaded. vhone\_loops.ap2 loaded.

# Visualize CCE's Loopmark with Performance Profile (2)



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`. A performance profile is overlaid on the code, showing execution time for various loops. A yellow callout bubble points to the 'Info' section, which contains the following messages:

- 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 on the right, 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 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.

Buttons at the bottom of the dialog include 'Explain other message...' and 'Close'.

Integrated message 'explain support'

# View Pseudo Code for Inlined Functions

**Navigation**

- Program View
- parabola.f90
- riemann.f90
- remap.f90
- evolve.f90
- volume.f90
- for...
- p...
- sta...
- flatten.f90
- sweepz.f90
- sweepy.f90
- boundary.f90
- prin.f90
- sweepx2.f90
- sweepx1.f90
- dtcon.f90
- vhone.f90
- init.f90
- 0.00% GRID
- Loop@199
- 0.01% INIT

**Source - /lus/sonexion/heidi/reveal/init.f90**

```

87
IVr2 88 call grid(imax, xmin, xmax, zxa, zxc, zdx)
      88   t$26 = 100
      88   t$27 = 100
      88   $I_L88_100 = 0
      88   !dir$ ivdep
      88   do
      88     zxa(1 + $I_L88_100) = 9.9999998e-3 * $I_L88_100
      88     zdx(1 + $I_L88_100) = 9.9999998e-3
      88     zxc(1 + $I_L88_100) = 4.9999999e-3 + ( 9.9999998e-3 *
      88     $I_L88_100 = 1 + $I_L88_100
      88     if ( $I_L88_100 >= 100 ) exit
      88   enddo
If    89 call grid(jmax, ymin, ymax, zya, zyc, zdy)
Tf   90 call grid(kmax, zmin, zmax, zza, zzc, zdz)
  
```

**Info - Line 89**

- A divide was turned into a multiply by a reciprocal
- A loop starting at line 89 was fused with the loop starting at line 88.
- The call to leaf routine "grid" was textually inlined.

vhone.pl loaded. vhone\_loops.ap2 loaded.

# Scoping Assistance – Review Scoping Results

**Navigation**

- ▼ Top Loops
  - ▼ sweepz.f90
    - SWEEPZ
      - 6.5767 Loop@48 (highlighted in red)
      - 6.5766 Loop@49
    - ▼ sweepy.f90
      - SWEEPY
        - Loop@32
        - Loop@33
        - Loop@34
        - Loop@35
        - Loop@36
        - Loop@37
        - Loop@38
        - Loop@39
        - Loop@40
        - Loop@41
        - Loop@42
        - Loop@43
        - Loop@44
        - Loop@45
        - Loop@46
        - Loop@47
        - Loop@48
        - Loop@49
        - Loop@50
        - Loop@51
        - Loop@52
        - Loop@53
        - Loop@54
        - Loop@55
        - Loop@56
        - Loop@57
        - Loop@58
        - Loop@59
        - Loop@60
  - ▼ riemann.f90
    - RIEMANN
      - 2.8951 Loop@63
    - ▼ parabola.f90
      - PARABOLA

**Source - /lus/sonexion/heidi/reveal/sweepz.f90**

```

47 ! Now Loop over j
48 do j = 1, js
49 do i = 1, isz
50 radius = zxc
51 theta = zyc
52 stheta = sin(theta)
53 radius = radius * stheta
54
55 ! Put state in array
56 do m = 1, n
57 do k = 1, n
58 n = k + k
59 r(n) = radius
60 p(n) = radius

```

**Info - Line 48**

- A loop starting at line 48 was not vectorized. Loop has been flattened.

**Open Module Selector**

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.
q	Array	Unresolved	FAIL-Last defining iteration not known for variable that is live on exit. WARN-LastPrivate of array may be very expensive.
zyc	Array	Conflict	
i	Scalar	Private	
j	Scalar	Private	
k	Scalar	Private	
m	Scalar	Private	
isz	Scalar	Shared	
js	Scalar	Shared	
ks	Scalar	Shared	
mypez	Scalar	Shared	
mypez	Scalar	Shared	
ngeomz	Scalar	Shared	
nleftz	Scalar	Shared	
npez	Scalar	Shared	
nrightz	Scalar	Shared	

**Annotations:**

- Loops with scoping information are highlighted – red needs user assistance
- Parallelization inhibitor messages are provided to assist user with analysis
- User addresses parallelization issues for unresolved variables

loading /lus/sonexion/heidi/reveal/vhone.pl/vhone\_22.T...

# Scoping Assistance – User Resolves Issues

**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.

**Code Editor:**

```

Loop@33
Loop@37
Loop@38
Loop@
Loop@
1250
31250
125000
r8
30
31
32
33
34
35
36
37
38
39
40
41
! Put state variables i
do m = 1, npey
do i = 1, isy
n = i + isy*(m-1) +
r(n) = recv2(1,k,i,j)
p(n) = recv2(2,k,i,j)
u(n) = recv2(3,k,i,j)
v(n) = recv2(4,k,i,j)
w(n) = recv2(5,k,i,j)
f(n) = recv2(6,k,i,j)
enddo

```

**OpenMP Scope Selector (sweep2.f90: lines 28 -> 69)**

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	
nietx	Scalar	Shared	
npey	Scalar	Shared	
nrightx	Scalar	Shared	
recv2	Array	Shared	
zdx	Array	Shared	
zll	Array	Shared	
zpr	Array	Shared	
zro	Array	Shared	
zux	Array	Shared	
zuy	Array	Shared	
zuz	Array	Shared	

**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.

**Callouts:**

- Use Reveal's OpenMP parallelization tips
- Click on variable to view all occurrences in loop



# Scoping Assistance – Generate Directive

Reveal generates example OpenMP directive

```

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) =
      x0(n)
      dx0(n)
      va (n)

```

**OpenMP Directive**

```

!$OMP parallel do default(none) &
!$OMP shared (gamm,send1,zdx,zff,zpr,zro,zux,zuy,zuz,zxa) &
!$OMP lastprivate (dx,dx0,e,f,p,r,u,v,w,xa,x0)

```

**OpenMP Scope Selector**  
sweepx1.f90: lines 29 -> 63

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.

**Info - Line 29**

- 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.

loading /home/users/heidi/demoLM/vhone.aid/vhone\_22.T...

# Questions ?