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 publicly 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.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. 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 on this website are the property of their respective owners.
• Reduce effort associated with adding OpenMP to MPI programs

• Get insight into optimizations performed by the Cray compiler

• Add OpenMP as a first step to parallelize loops that will target GPUs

• Track requests to memory and evaluate the bandwidth contribution of objects within a program for loop tuning
Approach to Adding Parallelism

1. Identify key high-level loops

2. Perform parallel analysis and scoping

3. Add OpenMP directive layer of parallelism

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

5. Port parallel loops to GPU with OpenMP target directives
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 = zxci+mypeziisz
      theta = zyc(j+mpeyijs)
      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
    enddo
  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,plf,ulft,rlf,prgh,urgh,rrgh)
  call riemann(nmin-3, nmax+4, gam, prgh, urgh, rrgh, &
               plf, ulft, rlf, 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
```
Loop Work Estimates

Gather loop statistics using the **Cray performance tools and the Cray Compiling Environment (CCE)** 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
Collect Loop Work Estimates

• Set up loop work estimates experiment with Cray compiler and Cray performance tools
  • user@login> module load PrgEnv-cray perftools-lite-loops

• Build program with Cray program library
  • –h pl=/full_path/program.pl

• Run program to get loop work estimates
## Example Loop Statistics

### Table 2: Loop Stats by Function

<table>
<thead>
<tr>
<th>Loop Incl</th>
<th>Loop Hit</th>
<th>Loop Trips</th>
<th>Loop Trips Avg</th>
<th>Loop Trips Min</th>
<th>Loop Trips Max</th>
<th>Function=/.LOOP[.]</th>
<th>PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.995914</td>
<td>100</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>sweepy_.LOOP.1.1i.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.995604</td>
<td>2500</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>sweepy_.LOOP.2.1i.34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.894750</td>
<td>50</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>sweepz_.LOOP.05.1i.49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.894637</td>
<td>1250</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>sweepz_.LOOP.06.1i.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.420629</td>
<td>50</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>sweepx2_.LOOP.1.1i.29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.420536</td>
<td>1250</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>sweepx2_.LOOP.2.1i.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.387534</td>
<td>50</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>sweepx1_.LOOP.1.1i.29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.387457</td>
<td>1250</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>sweepx1_.LOOP.2.1i.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.523214</td>
<td>187500</td>
<td>107</td>
<td>0</td>
<td>107</td>
<td>riemann_.LOOP.2.1i.63</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
View Source and Optimization Information

A loop starting at line 67 was fused with the loop starting at line 53.
Scope Selected Loop(s)

• Trigger dependence analysis
• scope loops above given threshold
Review Scoping Results

Loops with scoping information are flagged. Red needs user assistance.

Parallelization inhibitor messages are provided to assist user with analysis.
Review Scoping Results (continued)

Reveal identifies shared reductions down the call chain

Reveal identifies calls that prevent parallelization
Review Scoping Results (continued)

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scope</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0i</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>a0r</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>a1i</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>a1r</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>a2i</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>a2r</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>b0i</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>b0r</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>b1i</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>b1r</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>b2i</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>b2r</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
</tbody>
</table>

```

**Scoping Footnote**

Assume no overlap between `lattice[*].mom[*]` and `tempmom[*][*]`
## Review Scoping Results (continued)

### Scope Loops and Scoping Results

#### fluxk.f: Loop@28

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scope</th>
<th>Info</th>
</tr>
</thead>
</table>
| fsk  | Array| Private| WARN: LastPrivate of array may be very expensive.  
|      |      |       | FAIL: FirstPrivate/Shared Scope Conflict. |
| i    | Scalar| Private| FAIL: FirstPrivate/Shared Scope Conflict. |
| j    | Scalar| Private|     |
| k    | Scalar| Private|     |
| l    | Scalar| Private|     |
| qsp  | Scalar| Private|     |
| qspk | Scalar| Private|     |
| dq   | Array| Shared|     |
| dtv  | Array| Shared|     |
| ind  | Scalar| Shared|     |
| jmax | Scalar| Shared|     |
| kadd | Scalar| Shared|     |
|      | First/Last Private| | Reduction: None |

- **WARN:** LastPrivate of array may be very expensive.
- **FAIL:** FirstPrivate/Shared Scope Conflict.
- **FAIL:** Ambiguous store conflict.
- **FAIL:** Last defining iteration not known for variable that may be live on exit.
- **FAIL:** Conflicting requirements, unable to scope.
View Loops through Call Chain

Loop instances

Loop traceback
! Directive inserted by Cray Reveal. May be incomplete.
!$OMP parallel do default(none) &
!$OMP& unresolved (dvol, dx0, e, f, flat, 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, deltax, 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
  enddo
enddo

do j = 1, jmax
  n = j + 6

Reveal generates OpenMP directive with illegal clause marking variables that need addressing
Validate User Inserted Directives

User inserted directive with mis-scoped variable 'n'
Choose “Compiler Messages” view to access message filtering, then select desired type of message.
QUESTIONS?