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.: 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 2017 Cray Inc.
Cray Compiler Optimization Feedback

OpenMP Assistance

MCDRAM Allocation Assistance
Reveal Overview

- Reduce effort associated with adding OpenMP to MPI programs when a pure MPI program no longer scales
- Produce performance portable code through OpenMP directives
- Get insight into optimizations performed by the Cray compiler
- Use 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
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
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?

```fortran
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
    enddo
  enddo
  call ppmlr
  do k = 1, kmax
    n = k + 6
    xa(n) = zza(k)
    dx(n) = zdx(k)
    xa0(n) = zza(k)
    dx0(n) = zdx(k)
    e(n) = p(n)/r(n)*gamm + 0.5 * (u(n)**2 + v(n)**2 + w(n)**2)
  enddo
  call ppmlr
enddo
```

```fortran
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,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
```
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
  - $ 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</th>
<th>Loop</th>
<th>Loop</th>
<th>Loop</th>
<th>Loop</th>
<th>Function=/.LOOP[.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incl</td>
<td>Hit</td>
<td>Trips</td>
<td>Trips</td>
<td>Trips</td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td>Avg</td>
<td>Min</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loop</th>
<th>Incl</th>
<th>Hit</th>
<th>Trips</th>
<th>Trips</th>
<th>Trips</th>
<th>PE=HIDE</th>
</tr>
</thead>
<tbody>
<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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</tr>
</tbody>
</table>
View Source and Optimization Information
Scope Selected Loop(s)

- Trigger dependence analysis
- scope loops above given threshold
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scope</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ks</td>
<td>Scalar</td>
<td>Shared</td>
<td></td>
</tr>
<tr>
<td>mypey</td>
<td>Scalar</td>
<td>Shared</td>
<td></td>
</tr>
<tr>
<td>ndim</td>
<td>Scalar</td>
<td>Shared</td>
<td></td>
</tr>
<tr>
<td>npey</td>
<td>Scalar</td>
<td>Shared</td>
<td></td>
</tr>
<tr>
<td>recv1</td>
<td>Array</td>
<td>Shared</td>
<td></td>
</tr>
<tr>
<td>send2</td>
<td>Array</td>
<td>Shared</td>
<td></td>
</tr>
<tr>
<td>svez</td>
<td>Scalar</td>
<td>Shared</td>
<td>WARN</td>
</tr>
<tr>
<td>zdy</td>
<td>Array</td>
<td>Shared</td>
<td></td>
</tr>
<tr>
<td>zxc</td>
<td>Array</td>
<td>Shared</td>
<td></td>
</tr>
<tr>
<td>zya</td>
<td>Array</td>
<td>Shared</td>
<td></td>
</tr>
</tbody>
</table>

WARN: atomic reduction operator required unless reduction fully.
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[*][*]

---

Copyright 2017 Cray Inc.
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,
& xa,xa0)
!$OMP& private (i,j,k,m,n,$$_n,delp2,delp1,shock,temp2,old_flat,
& onemfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn,
& ekin)
!$OMP& shared (gamm,isy,js,ks,mypey,ndim,ngeomy,nlefty,npey,nrighty, & recv1,send2,zdy,zxc,zya)
do k = 1, ks
  do i = 1, isy
    radius = zxc(i+mpey*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
  do j = 1, jmax
    n = j + 6

Reveal generates OpenMP directive with illegal clause marking variables that need addressing
Reveal Auto-Parallelization

- Minimal user time investment includes time to set up and run optimization experiment
  - Collect loop work estimates
  - Build program library
  - Click button in Reveal
  - Run experimental binary,
  - Compare against original program

- Even if experiment does not yield a performance improvement, Reveal will provide insight into parallelization issues

- Use on X86 hardware or Armv8 hardware
Validate User Inserted Directives

User inserted directive with mis-scoped variable ‘n’
Look For Vectorization Opportunities

Choose “Compiler Messages” view to access message filtering, then select desired type of message.
Thank You!