Adding Parallelism to HPC Applications using Reveal

DOE Centers of Excellence Performance Portability Meeting

Heidi Poxon
Technical Lead
Programming Environment
Cray Inc.

April 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 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
    ...
end subroutine sweepz
```

```fortran
subroutine 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
    ...
end subroutine ppmlr
```

April 2016
© Cray Inc. Proprietary
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
## Example Loop Work Estimates

### Table 2: Loop Stats by Function (from `-hprofile_generate`)

<table>
<thead>
<tr>
<th>Loop Incl</th>
<th>Loop Hit</th>
<th>Loop Trips Avg</th>
<th>Loop Trips Min</th>
<th>Loop Trips Max</th>
<th>Function=/&gt;.LOOP[.]</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>
<tr>
<td>1.541299</td>
<td>20062500</td>
<td>12</td>
<td>0</td>
<td>12</td>
<td>riemann_.LOOP.3.1i.64</td>
<td></td>
</tr>
<tr>
<td>0.863656</td>
<td>1687500</td>
<td>104</td>
<td>0</td>
<td>108</td>
<td>parabola_.LOOP.6.1i.67</td>
<td></td>
</tr>
</tbody>
</table>
View Source and Optimization Information
Hybridization Step 2: Scope Selected Loop(s)
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 (2)

'I' identifies variables that reside in functions within the loop.
Review Scoping Results (3)

Reveal identifies calls that prevent parallelization.

Reveal identifies shared reductions down the call chain.
Hybridization Step 3: Generate OpenMP Directives

Hybridization Step 3: Generate OpenMP Directives

```c
! 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,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
  enddo
  do j = 1, jmax
    n = j + 6
    ...
Hybridization Step 4: Performance Analysis

Choose “Compiler Messages” view to access message filtering, then select desired type of message.
Reveal can be used to simplify the task of adding OpenMP to MPI programs

The result is performance portable code
  - 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