Applying the “Whack-a-Mole” Method Using Cray’s Perftools for Identifying the Moles

John Levesque, Cray Inc.
Heidi Poxon, Cray Inc.
Outline

- What is the “Whack-a-mole” process?
- Short introduction to perftools-lite
- Applications to be examined
  - UMT
  - Leslie3d
  - VH1
Do Not Assume You Know Your Application Profile

People's minds are changed through observation and not through argument.
Will Rogers
Cray Performance Tools

● Reduce the time investment associated with porting and tuning applications on Cray systems

● Analyze whole-program behavior across many nodes to identify critical performance bottlenecks within a program

● Improve your profiling experience by using simple (lite mode) and/or advanced interfaces for a wealth of capability that targets analyzing large HPC jobs
Interfaces Available

- Simple interface (perftools-lite modes) for convenience
- Advanced interface (perftools) for in-depth performance investigation and tuning assistance as well as data collection control

Both offer:
- Whole program analysis across many nodes
- Indication of causes of problems
- Ability to easily switch between the two interfaces
Used in this Tutorial

- Several simple modes: perftools-lite, perftools-lite-loops, perftools-lite-hbm
  - Simply load the desired module, compile, and execute program

- Additional reports (pat_report –O calltree, etc.)

- perftools advanced interface
  - Load perftools module and build program
  - Instrument program with pat_build -u -g mpi
  - Execute instrumented program
  - Create performance reports with pat_report

- Cray compiler (CCE) listing generated by -hlist=a

- Reveal

- Levesque’s and Heidi’s bag of tricks
Applying a “Whack-a-mole” Method Using Cray’s perftools to Identify the Moles

- Ricky Kendal used the phase extensively

- The *Mole* is the most time consuming process in the application

- *Whacking* it means you optimize the process so it no longer takes the most time
Ricky Kendall 1961 - 2014

- John met Ricky in 1993 when he visited APR to evaluate FORGE
  - Taught him to use “.” in vi
- Worked extensively with Ricky for many years at ORNL (AUG 2005 – 2014)
Whack-a-mole Process

● **Profile your *working* application**
  ● On the problem of interest at the scale of interest

  ● Don’t think you know where the mole is and more importantly why it’s the most important bottleneck in the program

  ● Performance on a single node is *not* necessarily representative of performance on 1000 nodes

  Unless everything scales, routines that scale will not be important at 1000 nodes
Whack-a-mole Process (continued)

1. Profile application (including scaling study)
2. Find Mole-top time consuming process in program
3. Run, checking performance & results
4. Whack Mole-modify code to address problem
5. Identify source of bottleneck

June 2018
Using the Process on Real Applications

UMT
Leslie3d
VH1
The UMT benchmark is a 3D, deterministic, multi-group, photon transport code for unstructured meshes.
## UMT Call Tree on 8 Nodes

### Table 1: Calltree View

<table>
<thead>
<tr>
<th>Nesting Level</th>
<th>Inclusive Percent of Time</th>
<th>Leaf child routine-Exclusive Percent of Time</th>
<th>Called From snflwxyz_</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>100.0%</td>
<td>3,971.6</td>
<td>Total</td>
</tr>
<tr>
<td>9</td>
<td>99.8%</td>
<td>3,961.9</td>
<td>main</td>
</tr>
<tr>
<td>10</td>
<td>98.6%</td>
<td>3,914.8</td>
<td>advance</td>
</tr>
<tr>
<td>11</td>
<td>98.6%</td>
<td>3,914.4</td>
<td>Teton&lt;&gt;::cxxRadtr</td>
</tr>
<tr>
<td>12</td>
<td>95.6%</td>
<td>3,797.9</td>
<td>rtmainsn_</td>
</tr>
<tr>
<td>13</td>
<td>66.9%</td>
<td>2,655.4</td>
<td>rswpmd_</td>
</tr>
<tr>
<td>14</td>
<td>66.8%</td>
<td>2,654.7</td>
<td>snflwxyz_</td>
</tr>
<tr>
<td>15</td>
<td>22.0%</td>
<td>872.3</td>
<td>exchange_</td>
</tr>
<tr>
<td>16</td>
<td>13.5%</td>
<td>535.7</td>
<td>MPI_WAIT</td>
</tr>
<tr>
<td>17</td>
<td>8.5%</td>
<td>336.5</td>
<td>exchange_(exclusive)</td>
</tr>
<tr>
<td>18</td>
<td>15.6%</td>
<td>617.7</td>
<td>snswp3d_</td>
</tr>
<tr>
<td>19</td>
<td>9.7%</td>
<td>383.4</td>
<td>initexchange_</td>
</tr>
<tr>
<td>20</td>
<td>1.0%</td>
<td>40.9</td>
<td>initialize</td>
</tr>
</tbody>
</table>

---

**Nesting Level**

*Inclusive Percent of Time*

*Leaf child routine-Exclusive Percent of Time*
Obtaining a Calltree

- Run `pat_report` with different options after collecting data with `perftools-lite` to get different views of the performance data.

- For a calltree:

  ```
  $ pat_report -O calltree exp_dir > rpt.calltree
  ```
Plan of Attack – Run Weak/Strong Scaling Study

Problem size: X
Run 8 MPI tasks on 8 nodes,
vary OpenMP threads from 1 to 32

Problem size: 2X
Run 16 MPI tasks on 16 nodes,
vary OpenMP threads from 1 to 32

Problem size: 4X
Run 32 MPI tasks on 32 nodes,
vary OpenMP threads from 1 to 32

Problem size: 4X
64 MPI tasks (1-32 OpenMP threads),
128 MPI tasks (1-16 OpenMP threads),
256 MPI tasks (1-8 OpenMP threads),
512 MPI tasks (1-4 OpenMP threads)
UMT Scaling with MPI Tasks and OpenMP Threads

- snswp3d is scaling nicely
- snmoments is not scaling

Percentage on abscissa
Observations

- **OpenMP on the node is not doing well above 4 threads**
  - Need to find out why

- **MPI scaling is good, however**
  - At higher MPI counts threading slows down the application
Using Listing: Existing OpenMP in SNSWP3D

Threaded Region

Threaded loop

Outer Loop of Nest

105. 1 2 M----< !$OMP PARALLEL DO PRIVATE(Angle,mm,thnum)
106. + 1 2 M m---< AngleLoop: do mm=1,NangBin
107. 1 2 M m
108. Angle = QuadSet% AngleOrder(mm,binSend)
109. thnum = 1
110. 1 2 M m
111. 1 2 M m
112. 1 2 M m              Angle = QuadSet% AngleOrder(mm,binSend)
113. thnum = OMP_GET_THREAD_NUM() + 1
114. + 1 2 M m
115. call snreflect(Angle, PSIB)
116. 1 2 M m
117. ! boundary flux array PSIB is also updated here.
118. 1 2 M m
119. 1 2 M m
120. + 1 2 M m              call snswp3d(Groups, Angle, &
121. 1 2 M m                QuadSet%next(1,Angle),QuadSet%nextZ(1,Angle), &
122. 1 2 M m                PSI(1,1,Angle),PSIB(1,1,Angle))
123. 1 2 M m
124. 1 2 M m-->> enddo AngleLoop
SNMOMENTS is Not Threaded

When nesting level contains number – no parallelism or vectorization

V – Vectorized
r2 – unrolled by 2

+ indicates a comment
Obtaining a Listing

● CCE provides loopmark, cross-references, compile options, and optimization messages in easy-to-read text files

● Just add the following CCE flag to Makefile:

   ```
   -h list=a ...
   ```

● Tip: For additional information on restructuring and optimization changes made by CCE, try `–h list=d` for decompiled code
Before Adding Loop-level Parallelism

- **Determine loop lengths by using perftools-lite-loops**
  - Can only use when existing OpenMP in program is disabled
    - perftools-lite disables OpenMP in CCE for the experiment
  - Remember that loops can change with job size (as MPI ranks increase)

Load perftools-lite-loops module
Build and run
View batch job output file or lite report in expdir/rpt-files/RUNTIME.rpt
UMT Loop Profile (perftools-lite-loops)

Table 1: Inclusive and Exclusive Time in Loops (from hprofile_generate)

<table>
<thead>
<tr>
<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>98.9%</td>
<td>99.000645</td>
<td>0.000069</td>
<td>1</td>
<td>7.0</td>
<td>7</td>
</tr>
<tr>
<td>92.4%</td>
<td>52.7448520</td>
<td>0.000154</td>
<td>7</td>
<td>4.7</td>
<td>3</td>
</tr>
<tr>
<td>92.1%</td>
<td>52.728532</td>
<td>0.000114</td>
<td>33</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>92.1%</td>
<td>52.647694</td>
<td>0.000214</td>
<td>33</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>71.6%</td>
<td>40.965850</td>
<td>0.000417</td>
<td>33</td>
<td>1.6</td>
<td>1</td>
</tr>
<tr>
<td>59.3%</td>
<td>33.937160</td>
<td>0.001694</td>
<td>54</td>
<td>6.9</td>
<td>4</td>
</tr>
<tr>
<td>44.0%</td>
<td>25.175652</td>
<td>0.043310</td>
<td>90</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>41.2%</td>
<td>23.541605</td>
<td>0.052865</td>
<td>28</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>28.2%</td>
<td>16.146745</td>
<td>3.884386</td>
<td>8</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>19.1%</td>
<td>10.931600</td>
<td>0.000059</td>
<td>7</td>
<td>5.0</td>
<td>3</td>
</tr>
<tr>
<td>19.1%</td>
<td>10.931541</td>
<td>10.931541</td>
<td>54</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>14.0%</td>
<td>7.993231</td>
<td>1.052804</td>
<td>7</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>11.9%</td>
<td>6.800924</td>
<td>6.800924</td>
<td>10</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>8.2%</td>
<td>4.692389</td>
<td>0.135064</td>
<td>4</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>8.0%</td>
<td>4.557325</td>
<td>4.557325</td>
<td>7,499,520</td>
<td>400.0</td>
<td>40</td>
</tr>
<tr>
<td>6.9%</td>
<td>3.944975</td>
<td>0.000352</td>
<td>405</td>
<td>5.0</td>
<td>3</td>
</tr>
<tr>
<td>6.9%</td>
<td>3.944623</td>
<td>1.409580</td>
<td>2,025</td>
<td>1.6</td>
<td>1</td>
</tr>
<tr>
<td>5.8%</td>
<td>3.312134</td>
<td>3.312134</td>
<td>7,499,520</td>
<td>3.0</td>
<td>3</td>
</tr>
<tr>
<td>5.0%</td>
<td>2.864861</td>
<td>0.000302</td>
<td>40</td>
<td>720.0</td>
<td>270</td>
</tr>
<tr>
<td>5.0%</td>
<td>2.861841</td>
<td>0.051047</td>
<td>28,800</td>
<td>224.0</td>
<td>224</td>
</tr>
<tr>
<td>4.9%</td>
<td>2.810795</td>
<td>2.810795</td>
<td>6,451,200</td>
<td>400.0</td>
<td>400</td>
</tr>
<tr>
<td>4.7%</td>
<td>2.700843</td>
<td>0.000089</td>
<td>405</td>
<td>1.6</td>
<td>1</td>
</tr>
</tbody>
</table>

Remember there is no threading in these samples

Ordered by loops that take most of the time

This is the loop with OpenMP

snmoments has nice loops lengths

Copyright 2018 Cray Inc.
### Table 1: Function Calltree View

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Calls</th>
<th>Calltree</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>57.147586</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>100.0%</td>
<td>57.147550</td>
<td>2.0</td>
<td>main</td>
</tr>
<tr>
<td>98.9%</td>
<td>56.543299</td>
<td>--</td>
<td>main.LOOP.1.li.189</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>advance</td>
</tr>
<tr>
<td>4</td>
<td>98.9%</td>
<td>56.538457</td>
<td>--</td>
</tr>
<tr>
<td>5</td>
<td>98.9%</td>
<td>56.538442</td>
<td>7.0</td>
</tr>
<tr>
<td>6</td>
<td>97.0%</td>
<td>55.459264</td>
<td>7.0</td>
</tr>
<tr>
<td>7</td>
<td>92.5%</td>
<td>52.838008</td>
<td>--</td>
</tr>
<tr>
<td>8</td>
<td>92.3%</td>
<td>52.719389</td>
<td>--</td>
</tr>
<tr>
<td>9</td>
<td>92.1%</td>
<td>52.638677</td>
<td>--</td>
</tr>
<tr>
<td>10</td>
<td>76.7%</td>
<td>43.811029</td>
<td>33.0</td>
</tr>
<tr>
<td>11</td>
<td>76.6%</td>
<td>43.802991</td>
<td>33.0</td>
</tr>
<tr>
<td>12</td>
<td>71.6%</td>
<td>40.911652</td>
<td>--</td>
</tr>
<tr>
<td>13</td>
<td>59.3%</td>
<td>33.883425</td>
<td>--</td>
</tr>
<tr>
<td>14</td>
<td>44.0%</td>
<td>25.123658</td>
<td>--</td>
</tr>
</tbody>
</table>

Remember there is no threading in these samples.

**This is the parallel loop**
Profile Loops with Call Tree (2 of 2)

| 14 | 44.0% | 25.123658 | -- | snflwxyz_LOOP.3.li.106 |
| 15 | 43.4% | 24.805902 | 33,480.0 | snswp3d_ |
| 14 | 15.3% | 8.759766 | 372.0 | exchange_ |

This is the parallel loop

| 13 | 8.0% | 4.592279 | 54.0 | initexchange_ |
| 13 | 2.9% | 1.654016 | 54.0 | testfluxconv_ |
| 13 | 1.4% | 0.781868 | 54.0 | setincidentflux_ |

This is the mole when snflxyz.106 is threaded

| 12 | 4.1% | 2.363264 | 33.0 | snmoments_ |
| 10 | 15.4% | 8.819869 | 33.0 | exchange_ |
| 7 | 3.2% | 1.814788 | 7.0 | rtstrtsn_ |
| 8 | 1.8% | 1.002470 | 7.0 | rtstrtsn_(exclusive) |
| 8 | 1.4% | 0.812278 | 7.0 | setbdy_ |
| 6 | 1.9% | 1.069855 | 7.0 | advancert_ |

---

This is the parallel loop

This is the mole when snflxyz.106 is threaded
Major Loop in SNMOMENTS (From Listing)

56. AC------<>  Phi(:, :) = zero
57. 
58. + 1------<>  AngleLoop: do Angle=1,QuadSet%NumAngles
59.  +1  
60.  1  quadwt = QuadSet% Weight(Angle)
61.  +1 
62.  1  if (quadwt /= zero) then
63.  +1 2------>
64.  +1 2------<  do ic=1,ncornr
65.  1 2 Vr2--<>  do ig=1,Groups
66.  1 2 Vr2  Phi(ig,ic) = Phi(ig,ic) + quadwt*psic(ig,ic,Angle)
67.  1 2 Vr2-->  enddo
68.  1 2------>  enddo
69.  1
70.  1
71.  1
72.  1------->  enddo AngleLoop

Notice that Phi is a reduction.
Questions About This Loop Nest

- Do we want to parallelize on ANGLE?

- How do we handle a reduction operation on an array?
  - This could be inefficient, or at least less efficient than parallelizing on a different loop
Reductions On An Array Are An Issue

This information is obtained from a perftools-lite-loops run:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 5.0% | 2.864861 | 0.003020 | 40 | 720.0 | 720 | 720 | snmoments_.LOOP.1.li.58
| 5.0% | 2.861841 | 0.051047 | 28,800 | 224.0 | 224 | 224 | snmoments_.LOOP.2.li.64
| 4.9% | 2.810795 | 2.810795 | 6,451,200 | 400.0 | 400 | 400 | snmoments_.LOOP.3.li.65

56. AC----------<> Phi(:,:,:) = zero
57. MV----------<> !$OMP PARALLEL DO PRIVATE(angle,quadwt,ic,ig) reduction(+:PHI(:,:,224))
58. + MV m------<> AngleLoop: do Angle=1,QuadSet%NumAngles
59. MV m
60. MV m         quadwt = QuadSet% Weight(Angle)
61. MV m
62. MV m         if (quadwt /= zero) then
63. MV m
64. + MV m 3-----<> do ic=1,ncornr
65. MV m 3 Vr2--<> do ig=1,Groups
66. MV m 3 Vr2          Phi(ig,ic) = Phi(ig,ic) + quadwt*psic(ig,ic,Angle)
67. MV m 3 Vr2-->       enddo
68. MV m 3------<>       enddo
69. MV m
70. MV m         endif
71. MV m
72. MV m--------<>       enddo AngleLoop

These do not need to be constant in new OpenMP standard
Parallelizing a Different Loop

56.   AC--------<>  Phi(:, :) = zero
57.   M----------<  !$OMP PARALLEL DO PRIVATE(ic, angle, quadwt, ig)
58.   + M m--------<  CornerLoop: do ic=1, ncornr
59.   + M m 3------<  AngleLoop: do Angle=1, QuadSet%NumAngles
60.   M m 3
61.   M m 3
62.   M m 3
63.   M m 3  
64.   M m 3
65.   M m 3 Vr2--<  do ig=1, Groups
66.   M m 3 Vr2  
67.   M m 3 Vr2-->
68.   M m 3
69.   M m 3
70.   M m 3
71.   M m 3------->  enddo AngleLoop
72.   M m-------->>  enddo CornerLoop

Pull the ic loop outside the angle loop...
Now `advancert`, `exchange` and `rtstrtsn` are using most of the time on 8 nodes.
Major Loop in RTSTRTSN

51. + F----------------------< ZoneLoop: do zone=1,nzones
52.     F I                      Z => getZoneData(Geom, zone)
53.     F                      nCorner = Z% nCorner
54.     F                      c0 = Z% c0
55. + F 2----------------------< do ia=1,Size%nangSN
56.     F 2 C-------------------< do c=1,nCorner
57.     F 2 C VCr2-------------<> Z% STime(:,c,ia) = tau*psir(:,c0+c,ia)
58.     F 2 C------------------>
59.     F 2------------------>
60.     F-----------------> enddo ZoneLoop
Parallelized Using OpenMP

For the zone loop, the average length is 824 on 8 MPI Tasks and only 14 when using 512 MPI Tasks
One of Two Major Loops in Exchange

88. 1 2 ! Loop over exiting angle, boundary element pairs for this communicator
89. 1 2
90. + 1 2 3----------< do ia=1,NangBin
91. 1 2 3 Angle = QuadSet% AngleOrder(ia,bin)
92. 1 2 3 nsend = Comm% nsend(ia)
93. 1 2 3
94. + 1 2 3 4----------< do i=1,nsend
95. 1 2 3 4 ib = Comm% ListSend(nsend0+i)
96. 1 2 3 4
97. 1 2 3 4
98. + 1 2 3 4 Vpr2------< do ig=1,ngroups
99. 1 2 3 4 Vpr2 Comm% psibsend(message+ig) = psib(ig.ib,Angle)
100. 1 2 3 4 Vpr2------> enddo
101. 1 2 3 4
102. 1 2 3 4 message = message + ngroups
103. 1 2 3 4
104. 1 2 3 4-------> enddo
105. 1 2 3 nsend0 = nsend0 + nsend
106. 1 2 3-------> enddo
Some Major Problems with This Exchange Loop

- *message* and *send0* illustrate loop carried dependencies

- Must restructure to enable parallelization
Exchange Loop Restructured & Parallelized

93. 1 2 ! scalar setup loop
94. + 1 2 r4-------------<
95. 1 2 r4
96. 1 2 r4
97. 1 2 r4
98. 1 2 r4
99. 1 2 r4
100. 1 2 r4------------->
101. 1 2
102. 1 2 M------------------< !$OMP PARALLEL DO PRIVATE(IA,Angle,nsend,i,ib,ig,message,nsend0)
103. + 1 2 M m----------<
104. 1 2 M m
105. 1 2 M m
106. 1 2 M m
107. 1 2 M m
108. 1 2 M m
109. 1 2 M m
110. + 1 2 M m 5----------<
111. 1 2 M m 5
112. 1 2 M m 5
113. 1 2 M m 5
114. + 1 2 M m 5 Vpr2------<
115. 1 2 M m 5 Vpr2
116. 1 2 M m 5 Vpr2------>
117. 1 2 M m 5
118. 1 2 M m 5
119. 1 2 M m 5
120. 1 2 M m 5---------->
121. 1 2 M m
122. 1 2 M m---------->>
Major Loop in AdvanceRT

136. Fr2 I
137. Fr2
138. Fr2
139. Fr2
140. Fr2
141. Fr2
142. + Fr2 2---------------------<
143. Fr2 2
144. Fr2 2
145. Fr2 2
146. + Fr2 2 3---------------------<
147. Fr2 2 3
148. Fr2 2 3--------------------->
149. Fr2 2
150. Fr2 2
151. Fr2 2
152. Fr2 2
153. Fr2 2
154. Fr2 2
155. Fr2 2
156. Fr2 2
157. Fr2 2--------------------->

Z => getZoneData(Geom, zone)

nCorner = Z% nCorner

c0 = Z% c0

PhiAve = zero

do c=1,nCorner

volumeRatio = Z% VolumeOld(c)/Z% Volume(c)

Phi(:,c0+c) = Phi(:,c0+c)*volumeRatio

do ia=1,numAngles

psir(:,c0+c,ia) = psir(:,c0+c,ia)*volumeRatio

enddo

sumRad = zero

do ig=1,ngr

sumRad = sumRad + Phi(ig,c0+c)

enddo

PhiAve = PhiAve + Z% Volume(c)*sumRad

enddo
AdvanceRT Loop Parallelized with OpenMP

134. + F------------------< ZoneLoop2: do zone=1, nzones
135. F
136. F M------------------< !$OMP PARALLEL PRIVATE(c, volumeRatio, ia, sumRad, ig)
137. F M I Z => getZoneData(Geom, zone)
138. F M
139. F M nCorner = Z% nCorner
140. F M c0 = Z% c0
141. F M PhiAve = zero
142. F M !$OMP DO REDUCTION(+: PhiAve)
143. + F M m-------------< do c=1, nCorner
144. F M m volumeRatio = Z% VolumeOld(c)/Z% Volume(c)
145. F M m Vr2-------< Phi(:,c0+c) = Phi(:,c0+c)*volumeRatio
146. F M m
147. + F M m 4----------< do ia=1, numAngles
148. F M m 4 Vr2------< psir(:,c0+c,ia) = psir(:,c0+c,ia)*volumeRatio
149. F M m 4------------ enddo
150. F M m
151. F M m sumRad = zero
152. F M m Vr4-------< do ig=1, ngr
153. F M m Vr4 sumRad = sumRad + Phi(ig,c0+c)
154. F M m Vr4------- enddo
155. F M m
156. F M m PhiAve = PhiAve + Z% Volume(c)*sumRad
157. F M m
158. F M m------------ enddo
159. F M------------------> !$OMP END PARALLEL

Z is a pointer in a module and it must be made threadprivate and set by all threads.
## Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Samp</td>
<td>Samp%</td>
<td>Function=[MAX10]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
<td></td>
</tr>
</tbody>
</table>

100.0% | 27,861.2 | -- | -- | Total |

| -- |  |  |  | USER |

| 79.4% | 22,111.1 | -- | -- | |
| 27.0% | 7,508.6 | 66.4 | 1.0% | snmoments_ |
| 23.2% | 6,453.2 | 73.8 | 1.3% | snswp3d_ |
| 9.5% | 2,636.0 | 23.0 | 1.0% | exchange_ |
| 8.9% | 2,488.8 | 24.2 | 1.1% | rtstrtsn_ |
| 4.9% | 1,378.6 | 6.4 | 0.5% | advancer_ |
| 2.2% | 619.4 | 66.6 | 11.1% | double* std::uninit |
| 2.0% | 561.4 | 8.6 | 1.7% | setincidentflux_ |

| 11.6% | 3,219.2 | -- | -- | ETC |

| 10.8% | 3,011.9 | 107.1 | 3.9% | __cray_dcopy_HSW |
| 9.0% | 2,518.9 | -- | -- | MPI |
| 7.3% | 2,033.1 | 184.9 | 9.5% | MPI_WAIT |
Some Results

- **Fastest 8 node configuration:**
  - 8 MPI ranks and 8 threads

- **Fastest 32 node configuration:**
  - 256 MPI tasks and 4 threads on 32 nodes
Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>4,015.1</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>47.9%</td>
<td>1,924.6</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>29.6%</td>
<td>1,190.3</td>
<td>452.7</td>
<td>27.7%</td>
<td>MPI_WAIT</td>
</tr>
<tr>
<td>13.4%</td>
<td>537.8</td>
<td>616.2</td>
<td>53.6%</td>
<td>MPI_BARRIER</td>
</tr>
<tr>
<td>4.7%</td>
<td>188.6</td>
<td>239.4</td>
<td>56.2%</td>
<td>MPI_ALLREDUCE</td>
</tr>
<tr>
<td>43.8%</td>
<td>1,760.5</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>14.1%</td>
<td>567.9</td>
<td>46.1</td>
<td>7.5%</td>
<td>snswp3d_</td>
</tr>
<tr>
<td>13.9%</td>
<td>558.1</td>
<td>166.9</td>
<td>23.1%</td>
<td>exchange_</td>
</tr>
<tr>
<td>7.5%</td>
<td>300.1</td>
<td>11.9</td>
<td>3.8%</td>
<td>snmoments_</td>
</tr>
<tr>
<td>3.1%</td>
<td>124.6</td>
<td>41.4</td>
<td>25.1%</td>
<td>setincidentflux_</td>
</tr>
<tr>
<td>2.6%</td>
<td>103.6</td>
<td>3.4</td>
<td>3.2%</td>
<td>rtstrtsn_</td>
</tr>
<tr>
<td>1.4%</td>
<td>58.1</td>
<td>2.9</td>
<td>4.8%</td>
<td>advancert_</td>
</tr>
<tr>
<td>8.0%</td>
<td>320.6</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td>7.7%</td>
<td>310.6</td>
<td>51.4</td>
<td>14.2%</td>
<td>_cray_dcopy_HSW</td>
</tr>
</tbody>
</table>

Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>3,651.9</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>54.7%</td>
<td>1,998.8</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>37.2%</td>
<td>1,360.0</td>
<td>444.0</td>
<td>24.7%</td>
<td>MPI_WAIT</td>
</tr>
<tr>
<td>12.3%</td>
<td>447.6</td>
<td>668.4</td>
<td>60.1%</td>
<td>MPI_BARRIER</td>
</tr>
<tr>
<td>5.1%</td>
<td>184.8</td>
<td>269.2</td>
<td>59.5%</td>
<td>MPI_ALLREDUCE</td>
</tr>
<tr>
<td>35.6%</td>
<td>1,299.8</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>15.9%</td>
<td>581.5</td>
<td>44.5</td>
<td>7.1%</td>
<td>snswp3d_</td>
</tr>
<tr>
<td>4.4%</td>
<td>161.0</td>
<td>60.0</td>
<td>27.3%</td>
<td>exchange_.LOOP@li.103</td>
</tr>
<tr>
<td>4.2%</td>
<td>154.4</td>
<td>3.6</td>
<td>2.3%</td>
<td>snmoments_.LOOP@li.58</td>
</tr>
<tr>
<td>3.5%</td>
<td>127.8</td>
<td>50.2</td>
<td>28.3%</td>
<td>setincidentflux_</td>
</tr>
<tr>
<td>3.0%</td>
<td>110.6</td>
<td>86.4</td>
<td>44.1%</td>
<td>exchange_.LOOP@li.178</td>
</tr>
<tr>
<td>1.8%</td>
<td>67.0</td>
<td>2.0</td>
<td>2.9%</td>
<td>rtstrtsn_.LOOP@li.51</td>
</tr>
<tr>
<td>9.0%</td>
<td>328.6</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td>8.6%</td>
<td>314.5</td>
<td>39.5</td>
<td>11.2%</td>
<td>_cray_dcopy_HSW</td>
</tr>
</tbody>
</table>
Not Real Exciting Speedup at 32 Nodes; However, ??

● Have we looked at everything?
  ● Vectorization ✓
  ● Parallelization ✓
  ● Memory Utilization ✗
Memory Bandwidth Metrics with perftools-lite

Avg Process Time: 83.99 secs
High Memory: 15,821.3 MiBytes 659.2 MiBytes per PE
Observed CPU clock boost: 118.1 %
Percent cycles stalled: 76.0 %
Vector intensity (packed instr): 14.7 %
Instr per Cycle: 0.38
I/O Write Rate: 2.407172 MiBytes/sec

Table 3: Memory Bandwidth by Numanode

| Memory Traffic GBytes | Local Memory Traffic GBytes | Remote Memory Traffic GBytes | Thread Time | Memory Traffic GBytes / Sec | Memory Traffic Traffic PE=HIDE | Numanode
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2,164</td>
<td>2,162</td>
<td>2.36</td>
<td>83.914890</td>
<td>25.79</td>
<td>33.6%</td>
<td>numanode.0</td>
</tr>
<tr>
<td>752.57</td>
<td>752.39</td>
<td>0.18</td>
<td>83.916253</td>
<td>8.97</td>
<td>11.7%</td>
<td>numanode.1</td>
</tr>
</tbody>
</table>
Using perftools-lite-hbm for Memory Traffic Info

Load perftools-lite-hbm module

Build and run

View batch job output file or lite report in expdir/rpt-files/RUNTIME.rpt
# Perftools-lite-hbm Results

## Table 5: Profile by Group, Function, and Line

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>MEM_LOAD_UOPS_RETIRED</th>
<th>RESOURCE_STALLS</th>
<th>Group</th>
<th>Function=[MAX10]</th>
<th>Source</th>
<th>Line</th>
<th>PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>862.5</td>
<td>--</td>
<td>--</td>
<td>89,509,042,630,218,624</td>
<td>1,722,469,802,124</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62.4%</td>
<td>537.9</td>
<td>--</td>
<td>--</td>
<td>56,224,626,618,220,104</td>
<td>991,877,756,035</td>
<td>USER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31.2%</td>
<td>269.1</td>
<td>--</td>
<td>--</td>
<td>28,464,157,033,148,192</td>
<td>608,254,074,506</td>
<td>snswp3d_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.6%</td>
<td>39.2</td>
<td>8.8</td>
<td>20.8</td>
<td>4,151,755,908,199,040</td>
<td>88,460,785,924</td>
<td>line.129</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.3%</td>
<td>132.2</td>
<td>--</td>
<td>--</td>
<td>14,179,301,951,169,120</td>
<td>150,304,652,642</td>
<td>snmoments_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.3%</td>
<td>132.1</td>
<td>5.9</td>
<td>4.9</td>
<td>14,144,117,579,091,356</td>
<td>150,152,590,310</td>
<td>line.66</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.0%</td>
<td>60.0</td>
<td>--</td>
<td>--</td>
<td>5,207,287,073,618,042</td>
<td>105,802,278,732</td>
<td>exchange_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.2%</td>
<td>44.9</td>
<td>4.1</td>
<td>9.6</td>
<td>3,588,805,956,844,644</td>
<td>77,368,597,030</td>
<td>line.115</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.8%</td>
<td>15.1</td>
<td>2.9</td>
<td>18.3</td>
<td>1,618,481,116,773,398</td>
<td>28,433,681,701</td>
<td>line.190</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
do icface=1,ncfaces
afpm(icface) = omega(1)*Z%A_fp(1,icface,c) + &
omega(2)*Z%A_fp(2,icface,c) + &
omega(3)*Z%A_fp(3,icface,c)
icfp = Z%Connect(1,icface,c)
ib = Z%Connect(2,icface,c)
if ( afpm(icface) >= zero ) then
  sumArea = sumArea + afpm(icface)
else
  if ( icfp == 0) then
    psifp(:,icface) = psib(:,ib)
  else
    psifp(:,icface) = psic(:,icfp)
  endif
endif
src(:,c) = src(:,c) - afpm(icface)*psifp(:,icface)
enddo
ARRAY SYNTAX: Good for the Cyber 205 and CM5, Not Good for Cache Based Systems

This simple change improved snswp3d by 50%
Tips When Dealing with Big Applications

- Getting a Preview
- Investigating Load imbalance
- Speeding up Report Generation
- Controlling Amount of Data
Perftool’s Traditional Sampling Approach

- Perftools profiles every MPI task on routine and line level
- In the resulting report
  - Gives table of heavy hitters where the routine is evaluated on the average number of samples across all MPI tasks.
  - Gives load imbalance figure showing how minimum and maximum deviates from average.
  - Typically does not show routines whose average is less than 1% of the total time.
  - Sampling can not distinguish between MPI_SYNC time; that is, the time waiting at the MPI function for all other MPI tasks to arrive and the actual MPI routine time. Therefore a lot of time for a MPI routine may be load imbalance.
## Typical Profile at Scale

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Samp</td>
<td>Samp%</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>100.0%</th>
<th>299,066.1</th>
<th>--</th>
<th>--</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>56.8%</td>
<td>169,977.1</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>------</td>
</tr>
<tr>
<td>28.5%</td>
<td>85,307.3</td>
<td>37,316.7</td>
<td>30.5%</td>
<td>MPI_ALLREDUCE</td>
</tr>
<tr>
<td>26.6%</td>
<td>79,700.7</td>
<td>51,912.3</td>
<td>39.5%</td>
<td>mpi_waitall</td>
</tr>
<tr>
<td>43.0%</td>
<td>128,473.8</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>------</td>
</tr>
<tr>
<td>4.8%</td>
<td>14,233.0</td>
<td>1,410.0</td>
<td>9.0%</td>
<td>routine1</td>
</tr>
<tr>
<td>2.9%</td>
<td>8,607.9</td>
<td>11,569.1</td>
<td>57.4%</td>
<td>routine2</td>
</tr>
<tr>
<td>2.6%</td>
<td>7,818.9</td>
<td>476.1</td>
<td>5.7%</td>
<td>routine3</td>
</tr>
<tr>
<td>2.4%</td>
<td>7,041.9</td>
<td>4,443.1</td>
<td>38.7%</td>
<td>routine4</td>
</tr>
<tr>
<td>2.3%</td>
<td>6,911.4</td>
<td>771.6</td>
<td>10.1%</td>
<td>routine5</td>
</tr>
<tr>
<td>2.1%</td>
<td>6,284.8</td>
<td>6,075.2</td>
<td>49.2%</td>
<td>__intel_mic_avx512f_memset</td>
</tr>
<tr>
<td>2.1%</td>
<td>6,220.8</td>
<td>10,186.2</td>
<td>62.1%</td>
<td>routine6</td>
</tr>
<tr>
<td>1.7%</td>
<td>4,992.0</td>
<td>2,480.0</td>
<td>33.2%</td>
<td>routine7</td>
</tr>
<tr>
<td>1.4%</td>
<td>4,277.9</td>
<td>1,902.1</td>
<td>30.8%</td>
<td>routine8</td>
</tr>
<tr>
<td>1.3%</td>
<td>3,869.4</td>
<td>294.6</td>
<td>7.1%</td>
<td>routine9</td>
</tr>
<tr>
<td>1.2%</td>
<td>3,583.1</td>
<td>5,093.9</td>
<td>58.7%</td>
<td>routine10</td>
</tr>
<tr>
<td>1.0%</td>
<td>2,992.9</td>
<td>83.1</td>
<td>2.7%</td>
<td>routine11</td>
</tr>
<tr>
<td>1.0%</td>
<td>2,909.3</td>
<td>1,848.7</td>
<td>38.9%</td>
<td>routine12</td>
</tr>
</tbody>
</table>
Analysis of xRage at LANL

- This approach was taken on xRage running on over 256,000 MPI tasks

- A large amount of time > 75% was attributed to MPI routines. Why? Was it load imbalance or actual time to do MPI operation?
How to Find the Offending Moles

- Ran both sampling and tracing experiments which generated tremendous amount of data
  - The post processing step took longer than the computation

- Choose subset of nodes to analyze

- Needed tracing to identify where the MPI_SYNC time occurred, giving us a clue to location of offending routine

- Use pat_regions to tunnel down to the offending code
How to Reduce Performance Data

- Choose subset of nodes
  - `user@login> export PAT_RT_EXPFILE_PES=0,128,512`

- Trace MPI routines to see if there is MPI_SYNC time
  - `user@login> pat_build --g mpi ./my_program`

- Use `pat_region` API to drill down to finer granularity
  - See `pat_help` utility for API details
  - Example: `user@login> pat_help regions C`
And the results

- Identified two separate areas that contributed to the load imbalance
  - One was in the AMR section where each MPI task needed to figure out who was going to see them data
  - The other was in interpolating from one grid to another
Significant Scalability Improvements in AMR via RMA Alltoall and Comm Caching

- The results are outstanding!
- ~3x performance improvement at scale
  - Total application runtime!

- RMA based Alltoall
- Caching and reuse of AMR communication pattern
The main purpose of this code is to model chemically reacting (i.e., burning) turbulent flows. Various different physical models are available in this algorithm but for this benchmark, only the basic conservation equations will be solved. Many, many different configurations can be modeled; however, for the sake of scalability and easy of use, only a rather simple configuration will be considered.
Leslie3D

- MPI only
- Some vectorization
### Sampling Profile 64 MPI Tasks on 8 Nodes

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>21,119.9</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>94.1%</td>
<td>19,868.2</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>19.3%</td>
<td>4,083.9</td>
<td>117.1</td>
<td>2.8%</td>
<td>visck_</td>
</tr>
<tr>
<td>18.3%</td>
<td>3,859.1</td>
<td>80.9</td>
<td>2.1%</td>
<td>viscj_</td>
</tr>
<tr>
<td>18.0%</td>
<td>3,805.6</td>
<td>151.4</td>
<td>3.9%</td>
<td>visci_</td>
</tr>
<tr>
<td>6.0%</td>
<td>1,273.0</td>
<td>82.0</td>
<td>6.1%</td>
<td>fluxk_</td>
</tr>
<tr>
<td>5.1%</td>
<td>1,076.7</td>
<td>57.3</td>
<td>5.1%</td>
<td>extrapk_</td>
</tr>
<tr>
<td>5.1%</td>
<td>1,075.2</td>
<td>84.8</td>
<td>7.4%</td>
<td>fluxi_</td>
</tr>
<tr>
<td>5.1%</td>
<td>1,074.0</td>
<td>65.0</td>
<td>5.8%</td>
<td>extrapi_</td>
</tr>
<tr>
<td>5.0%</td>
<td>1,050.8</td>
<td>65.2</td>
<td>5.9%</td>
<td>fluxj_</td>
</tr>
<tr>
<td>4.3%</td>
<td>902.8</td>
<td>38.2</td>
<td>4.1%</td>
<td>update_</td>
</tr>
<tr>
<td>4.1%</td>
<td>862.0</td>
<td>40.0</td>
<td>4.5%</td>
<td>extrapj_</td>
</tr>
<tr>
<td>1.3%</td>
<td>278.9</td>
<td>77.1</td>
<td>22.0%</td>
<td>mpicx_</td>
</tr>
<tr>
<td>5.6%</td>
<td>1,178.1</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>3.2%</td>
<td>676.2</td>
<td>312.8</td>
<td>32.1%</td>
<td>MPI_SEND</td>
</tr>
<tr>
<td>1.0%</td>
<td>209.6</td>
<td>64.4</td>
<td>23.9%</td>
<td>MPI_ALLREDUCE</td>
</tr>
</tbody>
</table>

---

Three Moles

June 2018

Copyright 2018 Cray Inc.
Call Tree with Loops

They are all called from high level loops

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Calls</th>
<th>Calltree</th>
<th>PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>242.649260</td>
<td>--</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>242.649223</td>
<td>2.0</td>
<td>les3d_LOOP.3.11.216</td>
<td></td>
</tr>
<tr>
<td>99.2%</td>
<td>240.654228</td>
<td>--</td>
<td>les3d_LOOP.4.11.272</td>
<td></td>
</tr>
<tr>
<td>29.1%</td>
<td>70.519335</td>
<td>4,000.0</td>
<td>fluxk_</td>
<td></td>
</tr>
<tr>
<td>17.7%</td>
<td>42.941276</td>
<td>--</td>
<td>fluxk_LOOP.1.11.28</td>
<td></td>
</tr>
<tr>
<td>6.6%</td>
<td>16.130702</td>
<td>4,000.0</td>
<td>fluxk(exclusive)</td>
<td></td>
</tr>
<tr>
<td>4.7%</td>
<td>11.447357</td>
<td>4,000.0</td>
<td>extrapK_</td>
<td></td>
</tr>
<tr>
<td>21.7%</td>
<td>54.688330</td>
<td>4,000.0</td>
<td>fluxj_</td>
<td></td>
</tr>
<tr>
<td>18.3%</td>
<td>44.519098</td>
<td>--</td>
<td>fluxj_LOOP.1.11.28</td>
<td></td>
</tr>
<tr>
<td>5.8%</td>
<td>14.0884979</td>
<td>4,000.0</td>
<td>fluxj(exclusive)</td>
<td></td>
</tr>
<tr>
<td>4.0%</td>
<td>9.684254</td>
<td>4,000.0</td>
<td>extrapJ_</td>
<td></td>
</tr>
<tr>
<td>17.7%</td>
<td>42.941276</td>
<td>9,408,000.0</td>
<td>visck_</td>
<td></td>
</tr>
<tr>
<td>6.3%</td>
<td>15.328236</td>
<td>4,000.0</td>
<td>extrapi_</td>
<td></td>
</tr>
<tr>
<td>5.8%</td>
<td>14.167993</td>
<td>4,000.0</td>
<td>fluxi_</td>
<td></td>
</tr>
<tr>
<td>9.6%</td>
<td>23.356145</td>
<td>4,000.0</td>
<td>parallel_</td>
<td></td>
</tr>
<tr>
<td>7.2%</td>
<td>17.457089</td>
<td>--</td>
<td>parallel_LOOP.1.11.16</td>
<td></td>
</tr>
<tr>
<td>7.2%</td>
<td>17.457089</td>
<td>20,000.0</td>
<td>mpiCX_</td>
<td></td>
</tr>
<tr>
<td>2.4%</td>
<td>5.843909</td>
<td>20,000.0</td>
<td>ghost_</td>
<td></td>
</tr>
<tr>
<td>3.6%</td>
<td>8.788456</td>
<td>4,000.0</td>
<td>update_</td>
<td></td>
</tr>
<tr>
<td>1.1%</td>
<td>2.557943</td>
<td>401.0</td>
<td>tmstep_</td>
<td></td>
</tr>
</tbody>
</table>
## Do Loop Table

**Table 1: Inclusive and Exclusive Time in Loops (from -hprofile_generate)**

<table>
<thead>
<tr>
<th>Loop</th>
<th>Loop Incl</th>
<th>Loop Hit</th>
<th>Loop</th>
<th>Loop</th>
<th>Loop</th>
<th>Function=/&gt;.LOOP[.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incl</td>
<td>Time%</td>
<td>Trips</td>
<td>Trips</td>
<td>Trips</td>
<td>Trips</td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>Time%</td>
<td>Avg</td>
<td>Min</td>
<td>Max</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>----------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>---------------------</td>
</tr>
<tr>
<td>99.2%</td>
<td>246.816563</td>
<td>1</td>
<td>2,000.0</td>
<td>2,000</td>
<td>2,000</td>
<td>les3d_.LOOP.3.li.216</td>
</tr>
<tr>
<td>97.9%</td>
<td>243.603685</td>
<td>2,000</td>
<td>2.0</td>
<td>2</td>
<td>2</td>
<td>les3d_.LOOP.4.li.272</td>
</tr>
<tr>
<td>24.3%</td>
<td>60.541874</td>
<td>4,000</td>
<td>48.0</td>
<td>48</td>
<td>48</td>
<td>fluxk_.LOOP.1.li.28</td>
</tr>
<tr>
<td>24.1%</td>
<td>60.069820</td>
<td>4,000</td>
<td>48.0</td>
<td>48</td>
<td>48</td>
<td>fluxj_.LOOP.1.li.28</td>
</tr>
<tr>
<td>22.9%</td>
<td>57.005766</td>
<td>192,000</td>
<td>49.0</td>
<td>49</td>
<td>49</td>
<td>fluxj_.LOOP.2.li.29</td>
</tr>
<tr>
<td>22.4%</td>
<td>55.615468</td>
<td>192,000</td>
<td>49.0</td>
<td>49</td>
<td>49</td>
<td>fluxk_.LOOP.2.li.29</td>
</tr>
<tr>
<td>20.9%</td>
<td>52.076589</td>
<td>4,000</td>
<td>48.0</td>
<td>48</td>
<td>48</td>
<td>fluxi_.LOOP.1.li.21</td>
</tr>
<tr>
<td>20.9%</td>
<td>52.070251</td>
<td>192,000</td>
<td>48.0</td>
<td>48</td>
<td>48</td>
<td>fluxi_.LOOP.2.li.22</td>
</tr>
<tr>
<td>17.1%</td>
<td>42.629610</td>
<td>9,408,000</td>
<td>48.0</td>
<td>48</td>
<td>48</td>
<td>viscj_.LOOP.1.li.340</td>
</tr>
<tr>
<td>16.5%</td>
<td>40.972257</td>
<td>9,408,000</td>
<td>48.0</td>
<td>48</td>
<td>48</td>
<td>visck_.LOOP.1.li.344</td>
</tr>
<tr>
<td>13.9%</td>
<td>36.129533</td>
<td>9,216,000</td>
<td>49.0</td>
<td>49</td>
<td>49</td>
<td>visci_.LOOP.1.li.782</td>
</tr>
<tr>
<td>7.0%</td>
<td>17.507682</td>
<td>4,002</td>
<td>5.0</td>
<td>5</td>
<td>5</td>
<td>parallel_.LOOP.1.li.16</td>
</tr>
<tr>
<td>5.0%</td>
<td>12.451170</td>
<td>4,000</td>
<td>51.5</td>
<td>51</td>
<td>52</td>
<td>extrapi_.LOOP.1.li.128</td>
</tr>
<tr>
<td>5.0%</td>
<td>12.448703</td>
<td>206,000</td>
<td>51.5</td>
<td>51</td>
<td>52</td>
<td>extrapi_.LOOP.2.li.129</td>
</tr>
<tr>
<td>4.6%</td>
<td>11.434859</td>
<td>4,000</td>
<td>49.0</td>
<td>49</td>
<td>49</td>
<td>extrapk_.LOOP.1.li.138</td>
</tr>
<tr>
<td>4.6%</td>
<td>11.432312</td>
<td>196,000</td>
<td>51.5</td>
<td>51</td>
<td>52</td>
<td>extrapk_.LOOP.2.li.140</td>
</tr>
<tr>
<td>3.9%</td>
<td>9.672461</td>
<td>4,000</td>
<td>51.5</td>
<td>51</td>
<td>52</td>
<td>extrapj_.LOOP.1.li.135</td>
</tr>
<tr>
<td>3.9%</td>
<td>9.668853</td>
<td>206,000</td>
<td>49.0</td>
<td>49</td>
<td>49</td>
<td>extrapj_.LOOP.2.li.136</td>
</tr>
</tbody>
</table>

Mesh Loops of Reasonable Length
Look at Threading Using OpenMP

- First examine the loops in FLUX routines using Reveal
  - Create program library `-hwp -hpl=leslie.pl` at build time

- Gather DO loop statistics with `perftools-lite-loops`

- `user@login>` `reveal my_program library my_perftools statistics`
  - Select one of the three high level loops in fluxk, flukj, fluxi
Reveal Example Scoping Results

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

Parallelization inhibitor messages are provided to assist user with analysis.
CALL EXTRAPI ( FSI )

! Directive inserted by Cray Reveal. May be incomplete.

!$OMP parallel do default(none)

!$OMP& private (i,j,k,l,qs,qsp,qspi)

!$OMP& shared (dq,dtv,iadd,icmax,jcmax,kcmax,pav,qav,six,siy,siz,u,

!$OMP& uav,v,vav,w,wav)

!$OMP& firstprivate(fsi)

DO K = 1, KCMAX

DO J = 1, JCMAX

QS(0:ICMAX) = UAV(0:ICMAX,J,K) * SIX(0:ICMAX,J,K) +

VAV(0:ICMAX,J,K) * SIY(0:ICMAX,J,K) +

WAV(0:ICMAX,J,K) * SIZ(0:ICMAX,J,K)

Reveal did not auto-magically do this. It pointed out several unresolved variables. In this case, the user had to scope the questionable variable as private or shared.
19. + CALL EXTRAPI ( FSI )
20.                      ! Directive inserted by Cray Reveal. May be incomplete.
21. M------------------< !$OMP parallel do default(none)
22. M                       !$OMP& private (i,j,k,l,qs,qsp,qspi)
23. M                       !$OMP& shared (dq,dtv,iadd,icmax,jcmax,kcmax,pav,qav,six,siy,siz,u,
24. M                       !$OMP& uav,v,vav,w,wav)
25. M                           !$OMP& firstprivate(fsi)
26. + M m------------------< DO K = 1, KCMAX
27. + M m 3----------------< DO J = 1, JCMAX
28. M m 3 fV-----------< QS(0:ICMAX) = UAV(0:ICMAX,J,K) * SIX(0:ICMAX,J,K) +
29. M m 3                  > VAV(0:ICMAX,J,K) * SIY(0:ICMAX,J,K) +
30. M m 3                  > WAV(0:ICMAX,J,K) * SIZ(0:ICMAX,J,K)
31. M m 3
32. M m 3
33. M m 3  IF ( NSCHEME .EQ. 2 ) THEN
34. M m 3 D--------------< DO I = 0, ICMAX
35. M m 3 D
36. M m 3 D                L = I + 1 - IADD
37. M m 3 D                QSP = U(L,J,K) * SIX(I,J,K)
38. M m 3 D                > V(L,J,K) * SIY(I,J,K) +
39. M m 3 D                > W(L,J,K) * SIZ(I,J,K)
40. M m 3 D                QSPI = (QSP - QS(I)) * DBLE(1 - 2 * IADD)
41. M m 3 D--------------< IF ( QSPI .GT. 0.0D0 ) QS(I) = 0.5D0 * (QS(I) + QSP)
42. M m 3 D--------------< ENDDO
43. M m 3
44. M m 3

Array syntax – vectorized and fused with other loops

Compiler knows NSCHEME is not equal to 2
44.  M  m  f........<>  FSI(0:ICMAX,1) = QAV(0:ICMAX,J,K,1) * QS(0:ICMAX)
45.  M  m  f........<>  FSI(0:ICMAX,2) = QAV(0:ICMAX,J,K,2) * QS(0:ICMAX) +
46.  M  m  >           PAV(0:ICMAX,J,K) * SIX(0:ICMAX,J,K)
47.  M  m  f........<>  FSI(0:ICMAX,3) = QAV(0:ICMAX,J,K,3) * QS(0:ICMAX) +
48.  M  m  >           PAV(0:ICMAX,J,K) * SIY(0:ICMAX,J,K)
49.  M  m  fVR2.......<>  FSI(0:ICMAX,4) = QAV(0:ICMAX,J,K,4) * QS(0:ICMAX) +
50.  M  m  >           PAV(0:ICMAX,J,K) * SIZ(0:ICMAX,J,K)
51.  M  m  f........<>  FSI(0:ICMAX,5) = (QAV(0:ICMAX,J,K,5) + PAV(0:ICMAX,J,K)) *
52.  M  m  >           QS(0:ICMAX)
53.  M  m  
54.  M  m  
55.  M  m  IF ( ISGSK .EQ. 1 ) THEN
56.  M  m  FSI(0:ICMAX,7) = QAV(0:ICMAX,J,K,7)
57.  M  m  ENDIF
58.  M  m  
59.  M  m  D----------<  IF ( ICHEM .GT. 0 ) THEN
60.  M  m  D  DO L = 8, 7 + NSPECI
61.  M  m  D           FSI(0:ICMAX,L) = QAV(0:ICMAX,J,K,L) * QS(0:ICMAX)
62.  M  m  D--------->
63.  M  m  
64.  + M  m  
65.  M  m  IF ( VISCOUS ) CALL VISCI ( 0, ICMAX, J, K, FSI )

VISCI is called from OpenMP loop
64 MPI Tasks on 8 Nodes

Routines by Percent of Samples

- visck_
- viscj_
- visci_
- extrapol
- extrapol
- update
- extrapolj
- fluxk
- MPI_SEND

Included Hyper-threads

June 2018

Copyright 2018 Cray Inc.
Observations From Timings

- VISC(I,J,K) still using significant amount of time
- EXTRAP(I,J,K) needs to be addressed
- UPDATE needs to be addressed
Concern with the VISCs

ftn-6383 ftn: VECTOR VISCK, File = fluxk.f, Line = 344
A loop starting at line 344 requires an estimated 25 vector registers at line 485; 2 of these have been preemptively forced to memory.

ftn-6204 ftn: VECTOR VISCK, File = fluxk.f, Line = 344
A loop starting at line 344 was vectorized.
Modifications

- Optimization: promoted 12 scalars to arrays to enable the lengthy loop to be split into three separate loops

- Result: all three routines improved by 25%
Modifications (continued)

\[
\begin{align*}
\text{DUDX} &= T11(I,J,K,3) \times DUDXI \\
\text{DVDX} &= T11(I,J,K,3) \times DVDXI \\
\text{DWDX} &= T11(I,J,K,3) \times DWDXI \\
\text{DTDX} &= T11(I,J,K,3) \times DTDXI \\
\text{DUDY} &= T21(I,J,K,3) \times DUDXI \\
\text{DV DY} &= T21(I,J,K,3) \times DVDXI \\
\text{DWDY} &= T21(I,J,K,3) \times DWDXI \\
\text{DTDY} &= T21(I,J,K,3) \times DTDXI \\
\text{DUDZ} &= T31(I,J,K,3) \times DUDXI \\
\text{DV DZ} &= T31(I,J,K,3) \times DVDXI \\
\text{DW DZ} &= T31(I,J,K,3) \times DWDXI \\
\text{DTDZ} &= T31(I,J,K,3) \times DTDXI
\end{align*}
\]
Reveal Scoping of EXTRAP Routines

142.
143.                      ! Directive inserted by Cray Reveal. May be incomplete.
144.    M---------------< !$OMP parallel do default(none)
145.    M                !$OMP& private (rwrk,j,k,kg)
146.    M                !$OMP& shared (i1,i2,j1,j2,kcmax,kmax,kstart,pav,qav,tav,
147.    M                !$OMP&                       uav,vav,wav)
148.  + M m---------------< DO K = 0, KCMAX
149.    M m                 KG = K + KSTART - 1
150.  + M m 3--------------< DO J = J1, J2
151.    M m 3
152.    M m 3               IF ( NSCHEME .EQ. 2 .OR. (.NOT. KPERIODIC .AND.
153.    M m 3                     (KG .EQ. 0 .OR. KG .EQ. KMAX-1))) THEN
154.  + M m 3               CALL EXK2 ( I1, I2, J, K )
155.    M m 3                 ELSE
156.  + M m 3                CALL EXK4 ( I1, I2, J, K,
157.    M m 3                      RWRK(I1,1), RWRK(I1,2), RWRK(I1,3) )
158.    M m 3

Once again Reveal needed help scoping some of the arrays that had questionable usage
Reveal Scoping of UPDATE Routine

9. ! Directive inserted by Cray Reveal. May be incomplete.
10. M------< !$OMP parallel do default(none)
11. M !$OMP& private (i,j,k,ke)
12. M !$OMP& shared (dq,icmax,jcmax,kcmax,m,n,p,q,t,u,v,w)
13. + M m------< DO K = 1, KCMAX
14. + M m 3----< DO J = 1, JCMAX
15. M m 3 V--< DO I = 1, ICMAX
16. M m 3 V
17. M m 3 V IF ( N .EQ. 1 ) THEN
18. M m 3 V Q(I,J,K,1,M) = Q(I,J,K,1,N) + DQ(I,J,K,1)
23. M m 3 V ELSE
24. M m 3 V Q(I,J,K,1,M) = 0.5D+00 * (Q(I,J,K,1,M) +
25. M m 3 V > Q(I,J,K,1,N) + DQ(I,J,K,1))
26. M m 3 V Q(I,J,K,2,M) = 0.5D+00 * (Q(I,J,K,2,M) +
27. M m 3 V > Q(I,J,K,2,N) + DQ(I,J,K,2))
28. M m 3 V Q(I,J,K,3,M) = 0.5D+00 * (Q(I,J,K,3,M) +

This one Reveal did without help
Another Modification to UPDATE

```fortran
  10.  + 1----------------->  DO K = 1, KCMAX
  11.  + 1 2----------------->  DO J = 1, JCMAX
  12.  1 2 iVbr4<--<<  DO I = 1, ICMAX
  13.  1 2 iVbr4
  14.  1 2 iVbr4
  15.  + 1 2 iVbr4 ib--<>  IF ( N .EQ. 1 ) THEN
  16.  1 2 iVbr4
  17.  + 1 2 iVbr4 ib--<>  ELSE
          Q(I,J,K,1:5,M) = 0.5D+00 * (Q(I,J,K,1:5,M) +
  18.  1 2 iVbr4  Q(I,J,K,1:5,N) + DQ(I,J,K,1:5))
  19.  1 2 iVbr4
  20.  1 2 iVbr4
  21.  1 2 iVbr4  ENDIF
  22.  1 2 iVbr4
  23.  1 2 iVbr4
  24.  1 2 iVbr4
  25.  1 2 iVbr4  U(I,J,K) = Q(I,J,K,2,M) / Q(I,J,K,1,M)
  26.  1 2 iVbr4  V(I,J,K) = Q(I,J,K,3,M) / Q(I,J,K,1,M)
  27.  1 2 iVbr4  W(I,J,K) = Q(I,J,K,4,M) / Q(I,J,K,1,M)
  28.  1 2 iVbr4
  29.  1 2 iVbr4  KE = 0.5D+00 * (U(I,J,K) * U(I,J,K) +
  30.  1 2 iVbr4  V(I,J,K) * V(I,J,K) +
  31.  1 2 iVbr4  W(I,J,K) * W(I,J,K))
  32.  1 2 iVbr4
  33.  1---------------------  T(I,J,K) = (Q(I,J,K,5,M) / Q(I,J,K,1,M) - KE) / CVAIR
  34.  1 2 iVbr4  P(I,J,K) = Q(I,J,K,1,M) * RGAIR * T(I,J,K)
  35.  1 2 iVbr4<--<  ENDDO
  36.  1---------------------  ENDDO
  37.  1---------------------  ENDDO
```

Copyright 2018 Cray Inc.
Another Modification to UPDATE

15c18,22
<
---
Q(I,J,K,1:5,M) = Q(I,J,K,1:5,N) + DQ(I,J,K)
>Q(I,J,K,1,M) = Q(I,J,K,1,N) + DQ(I,J,K)
>Q(I,J,K,2,M) = Q(I,J,K,2,N) + DQ(I,J,K)
>Q(I,J,K,3,M) = Q(I,J,K,3,N) + DQ(I,J,K)
>Q(I,J,K,4,M) = Q(I,J,K,4,N) + DQ(I,J,K)
>Q(I,J,K,5,M) = Q(I,J,K,5,N) + DQ(I,J,K)
17,18c24,33
<
< >
---
Q(I,J,K,1:5,M) = 0.5D+00 * (Q(I,J,K,1:5,N) + DQ(I,J,K))
>Q(I,J,K,1,M) = 0.5D+00 * (Q(I,J,K,1,M) + DQ(I,J,K))
>Q(I,J,K,2,M) = 0.5D+00 * (Q(I,J,K,2,M) + DQ(I,J,K))
>Q(I,J,K,3,M) = 0.5D+00 * (Q(I,J,K,3,M) + DQ(I,J,K))
>Q(I,J,K,4,M) = 0.5D+00 * (Q(I,J,K,4,M) + DQ(I,J,K))
>Q(I,J,K,5,M) = 0.5D+00 * (Q(I,J,K,5,M) + DQ(I,J,K))
Optimization Results for 64 MPI Tasks on 8 Nodes

Routines by Percent of Samples

- **visck**
- **viscj**
- **visci**
- **fluxk**
- **update**
- **exk4**
- **MPI SEND**
- **fluxi**
- **fluxj**

June 2018

**Optimization Results for 64 MPI Tasks on 8 Nodes**

**Routines by Percent of Samples**

- **visck**
- **viscj**
- **visci**
- **fluxk**
- **update**
- **exk4**
- **MPI SEND**
- **fluxi**
- **fluxj**

June 2018

Copyright 2018 Cray Inc.

**Optimization Results for 64 MPI Tasks on 8 Nodes**

**Routines by Percent of Samples**

- **visck**
- **viscj**
- **visci**
- **fluxk**
- **update**
- **exk4**
- **MPI SEND**
- **fluxi**
- **fluxj**

June 2018

Copyright 2018 Cray Inc.
Final Sampling Profile

Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Samp</td>
<td>Samp</td>
<td></td>
<td>Function=[MAX10]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
</tr>
</tbody>
</table>

100.0% | 9,268.0 | -- | -- | Total
|-------|-------|----|----|-------|
| 86.4% | 8,003.3 | -- | -- | USER
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12.0%</td>
<td>1,114.2</td>
<td>84.8</td>
<td>7.2%</td>
</tr>
<tr>
<td></td>
<td>11.9%</td>
<td>1,106.2</td>
<td>58.8</td>
<td>5.1%</td>
</tr>
<tr>
<td></td>
<td>11.4%</td>
<td>1,054.9</td>
<td>49.1</td>
<td>4.5%</td>
</tr>
<tr>
<td></td>
<td>5.7%</td>
<td>531.1</td>
<td>46.9</td>
<td>8.2%</td>
</tr>
<tr>
<td></td>
<td>5.4%</td>
<td>499.0</td>
<td>45.0</td>
<td>8.4%</td>
</tr>
<tr>
<td></td>
<td>5.3%</td>
<td>490.8</td>
<td>65.2</td>
<td>11.9%</td>
</tr>
<tr>
<td></td>
<td>5.2%</td>
<td>484.0</td>
<td>41.0</td>
<td>7.9%</td>
</tr>
<tr>
<td></td>
<td>4.6%</td>
<td>424.7</td>
<td>35.3</td>
<td>7.8%</td>
</tr>
<tr>
<td></td>
<td>4.1%</td>
<td>382.4</td>
<td>50.6</td>
<td>11.9%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-------</td>
<td>-------</td>
<td>----</td>
<td>----</td>
<td>------</td>
</tr>
</tbody>
</table>
| 11.6% | 1,078.5 | -- | -- | MPI
|-------|-------|----|----|------|
| 5.5%  | 507.0  | 205.0 | 29.3% | MPI_SEND
|-------|-------|----|----|------|
| 1.9%  | 178.7  | -- | -- | ETC

Best time on 8 nodes is with 64 MPI tasks and 4 Threads per MPI task

2.3 times faster than original!
Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Samp</td>
<td>Samp%</td>
<td>Function</td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>100.0%</td>
<td>8,919.6</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>84.4%</td>
<td>7,532.4</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>12.4%</td>
<td>1,108.2</td>
<td>56.8</td>
<td>4.9%</td>
<td>viscj_</td>
</tr>
<tr>
<td>11.8%</td>
<td>1,048.1</td>
<td>93.9</td>
<td>8.3%</td>
<td>visci_</td>
</tr>
<tr>
<td>11.7%</td>
<td>1,047.8</td>
<td>74.2</td>
<td>6.6%</td>
<td>visck_</td>
</tr>
<tr>
<td>6.0%</td>
<td>530.9</td>
<td>37.1</td>
<td>6.6%</td>
<td>update_</td>
</tr>
<tr>
<td>5.8%</td>
<td>518.4</td>
<td>49.6</td>
<td>8.8%</td>
<td>exk4_</td>
</tr>
<tr>
<td>5.8%</td>
<td>513.7</td>
<td>36.3</td>
<td>6.6%</td>
<td>exi4_</td>
</tr>
<tr>
<td>4.9%</td>
<td>439.1</td>
<td>53.9</td>
<td>11.0%</td>
<td>exj4_</td>
</tr>
<tr>
<td>4.4%</td>
<td>392.1</td>
<td>41.9</td>
<td>9.7%</td>
<td>fluxk_</td>
</tr>
<tr>
<td>4.3%</td>
<td>382.1</td>
<td>53.9</td>
<td>12.4%</td>
<td>fluxi_</td>
</tr>
<tr>
<td>4.2%</td>
<td>373.2</td>
<td>42.8</td>
<td>10.3%</td>
<td>fluxj_</td>
</tr>
<tr>
<td>4.0%</td>
<td>359.9</td>
<td>61.1</td>
<td>14.6%</td>
<td>extrapi_</td>
</tr>
<tr>
<td>2.1%</td>
<td>184.0</td>
<td>35.0</td>
<td>16.0%</td>
<td>extrapj_</td>
</tr>
<tr>
<td>1.8%</td>
<td>159.8</td>
<td>39.2</td>
<td>19.8%</td>
<td>extrapk_</td>
</tr>
<tr>
<td>1.4%</td>
<td>129.1</td>
<td>67.9</td>
<td>34.6%</td>
<td>mpicx_</td>
</tr>
<tr>
<td>1.4%</td>
<td>120.8</td>
<td>46.2</td>
<td>27.8%</td>
<td>parallel_</td>
</tr>
<tr>
<td>1.3%</td>
<td>114.9</td>
<td>57.1</td>
<td>33.3%</td>
<td>ghost_</td>
</tr>
</tbody>
</table>

Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Samp</td>
<td>Samp%</td>
<td>Function</td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>100.0%</td>
<td>9,206.9</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>85.8%</td>
<td>7,899.0</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>18.1%</td>
<td>1,662.2</td>
<td>90.8</td>
<td>5.2%</td>
<td>fluxj_</td>
</tr>
<tr>
<td>17.4%</td>
<td>1,602.9</td>
<td>225.1</td>
<td>12.4%</td>
<td>fluxk_</td>
</tr>
<tr>
<td>16.8%</td>
<td>1,542.4</td>
<td>109.6</td>
<td>6.7%</td>
<td>fluxi_</td>
</tr>
<tr>
<td>9.3%</td>
<td>855.7</td>
<td>72.3</td>
<td>7.8%</td>
<td>extrapi_</td>
</tr>
<tr>
<td>7.1%</td>
<td>652.3</td>
<td>53.7</td>
<td>7.6%</td>
<td>extrapk_</td>
</tr>
<tr>
<td>6.5%</td>
<td>602.6</td>
<td>62.4</td>
<td>9.4%</td>
<td>extrapj_</td>
</tr>
<tr>
<td>5.7%</td>
<td>528.0</td>
<td>60.0</td>
<td>10.2%</td>
<td>update_</td>
</tr>
<tr>
<td>1.4%</td>
<td>130.4</td>
<td>43.6</td>
<td>25.1%</td>
<td>mpicx_</td>
</tr>
<tr>
<td>1.3%</td>
<td>123.0</td>
<td>70.0</td>
<td>36.4%</td>
<td>parallel_</td>
</tr>
<tr>
<td>1.1%</td>
<td>100.7</td>
<td>81.3</td>
<td>44.8%</td>
<td>ghost_</td>
</tr>
<tr>
<td>13.7%</td>
<td>1,263.9</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>8.1%</td>
<td>749.8</td>
<td>270.2</td>
<td>26.6%</td>
<td>MPI_SEND</td>
</tr>
<tr>
<td>2.3%</td>
<td>209.7</td>
<td>101.3</td>
<td>32.7%</td>
<td>MPI_ALLREDUCE</td>
</tr>
<tr>
<td>1.5%</td>
<td>136.1</td>
<td>269.9</td>
<td>66.7%</td>
<td>MPI_WAIT</td>
</tr>
<tr>
<td>1.5%</td>
<td>135.8</td>
<td>119.2</td>
<td>46.9%</td>
<td>MPI_REDUCE</td>
</tr>
</tbody>
</table>
All That Work for Nothing?

- We could do a lot more to optimize for cache in the VISC(I,J,K) routines

- But, a pure MPI solution runs pretty well on multicore node like haswell, broadwell, skylake and even KNL
### Memory Analysis of the Optimized Code

#### Table 5: Profile by Group, Function, and Line

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>MEM_LOAD_UOPS_RETIRED</th>
<th>RESOURCE_STALLS</th>
<th>Group</th>
<th>Source</th>
<th>Function=[MAX10]</th>
<th>Line</th>
<th>PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>359.1</td>
<td>--</td>
<td>--</td>
<td>37,453,764,107,885,936</td>
<td>586,696,060,451</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90.3%</td>
<td>324.2</td>
<td>--</td>
<td>--</td>
<td>33,675,842,152,965,068</td>
<td>538,408,160,657</td>
<td>USER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.6%</td>
<td>59.7</td>
<td>--</td>
<td>--</td>
<td>6,020,925,677,307,978</td>
<td>98,381,410,309</td>
<td>fluxk_3</td>
<td></td>
<td>fluxk.f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.7%</td>
<td>52.8</td>
<td>--</td>
<td>--</td>
<td>5,770,237,024,852,115</td>
<td>88,570,460,626</td>
<td>visci_3</td>
<td></td>
<td>visci.f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.8%</td>
<td>49.7</td>
<td>--</td>
<td>--</td>
<td>5,246,869,490,559,427</td>
<td>82,777,122,645</td>
<td>fluxj_3</td>
<td></td>
<td>fluxj.f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0%</td>
<td>38.8</td>
<td>--</td>
<td>--</td>
<td>4,728,626,552,108,359</td>
<td>76,281,840,808</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.7%</td>
<td>41.2</td>
<td>--</td>
<td>--</td>
<td>4,603,602,008,685,437</td>
<td>74,201,128,360</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.8%</td>
<td>34.6</td>
<td>--</td>
<td>--</td>
<td>3,535,410,628,332,144</td>
<td>50,100,086,904</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9%</td>
<td>41.1</td>
<td>--</td>
<td>--</td>
<td>2,685,621,724,485,968</td>
<td>44,200,000,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

June 2018

Copyright 2018 Cray Inc.
DO K = 1, KCMAX
   DO L = 1, 5
   ENDDO
   IF (ISGSK .EQ. 1) THEN
      DQ(1:IND,J,K,7) = DQ(1:IND,J,K,7) - DTV(1:IND,J,K) * (FSK(1:IND,K,7) - FSK(1:IND,K-1,7))
   ENDIF
   IF (ICHEM .GT. 0) THEN
      DO L = 8, 7 + NSPECI
      ENDDO
      ENDIF
ENDDO
Array Syntax and Automatic Blocking by the Compiler

73. M m !dir$ noblocking
74. + M m 3------< DO K = 1, KCMAX
75. M m 3 V------< DO I = 1, IND
76. M m 3 V !dir$ unroll(5)
77. M m 3 V w------< DO L = 1, 5
78. M m 3 V w DQ(I,J,K,L) = DQ(I,J,K,L) -
79. M m 3 V w > DTV(I,J,K) * (FSK(I,K,L) - FSK(I,K-1,L))
80. M m 3 V w------> ENDDO
81. M m 3 V
82. M m 3 V IF (ISGSK .EQ. 1) THEN
83. M m 3 V DQ(I,J,K,7) = DQ(I,J,K,7) -
84. M m 3 V > DTV(I,J,K) * (FSK(I,K,7) - FSK(I,K-1,7))
85. M m 3 V ENDIF
86. M m 3 V
87. M m 3 V IF (ICHEM .GT. 0) THEN
88. M m 3 V D------< DO L = 8, 7 + NSPECI
89. M m 3 V D DQ(I,J,K,L) = DQ(I,J,K,L) -
90. M m 3 V D > DTV(I,J,K) * (FSK(I,K,L) - FSK(I,K-1,L))
91. M m 3 V D------> ENDDO
92. M m 3 V ENDIF
93. M m 3 V
94. M m 3 V------> ENDDO
95. M m 3-------> ENDDO
96. M m-------> ENDDO

Unfortunately there is a lot more array syntax...
VH1
VH1 Already Parallelized Using Reveal

- Need to investigate additional optimization
- Vectorization and Memory utilization optimization
### Table 1: Profile by Function 1 Thread

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>39,072.1</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>77.5%</td>
<td>30,298.1</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>18.8%</td>
<td>7,340.2</td>
<td>254.8</td>
<td>3.4</td>
<td>parabola_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>14.2%</td>
<td>5,533.8</td>
<td>362.2</td>
<td>6.2</td>
<td>riemann_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>10.7%</td>
<td>4,173.6</td>
<td>221.4</td>
<td>5.1</td>
<td>sweepy_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>8.5%</td>
<td>3,340.0</td>
<td>247.0</td>
<td>6.9</td>
<td>sweepz_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>8.3%</td>
<td>3,226.1</td>
<td>173.9</td>
<td>5.2</td>
<td>remap_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>4.2%</td>
<td>1,653.7</td>
<td>151.3</td>
<td>8.4</td>
<td>paraset_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>3.0%</td>
<td>1,184.3</td>
<td>76.7</td>
<td>6.1</td>
<td>sweepx1_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>2.9%</td>
<td>1,123.3</td>
<td>68.7</td>
<td>5.8</td>
<td>sweepx2_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>16.9%</td>
<td>6,600.0</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>16.3%</td>
<td>6,373.7</td>
<td>670.3</td>
<td>9.6</td>
<td>mpi_alltoall</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>5.5%</td>
<td>2,160.6</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>3.4%</td>
<td>1,328.0</td>
<td>110.0</td>
<td>7.7</td>
<td>__cray_sset_SNB</td>
</tr>
</tbody>
</table>

### Table 1: Profile by Function 4 Threads

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>16,290.4</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>58.4%</td>
<td>9,508.0</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>12.9%</td>
<td>2,105.5</td>
<td>167.5</td>
<td>7.4</td>
<td>parabola_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>10.3%</td>
<td>1,679.0</td>
<td>107.0</td>
<td>6.0</td>
<td>sweepy_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>9.7%</td>
<td>1,585.0</td>
<td>151.0</td>
<td>8.8</td>
<td>riemann_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>7.6%</td>
<td>1,242.2</td>
<td>72.8</td>
<td>5.6</td>
<td>sweepz_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>5.7%</td>
<td>926.7</td>
<td>77.3</td>
<td>7.8</td>
<td>remap_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>2.9%</td>
<td>476.5</td>
<td>50.5</td>
<td>9.7</td>
<td>paraset_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>2.2%</td>
<td>355.4</td>
<td>64.6</td>
<td>15.5</td>
<td>sweepx1_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>2.0%</td>
<td>323.2</td>
<td>33.8</td>
<td>9.5</td>
<td>sweepx2_</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>37.7%</td>
<td>6,135.4</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>37.4%</td>
<td>6,095.4</td>
<td>100.6</td>
<td>1.6</td>
<td>mpi_alltoall</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>3.8%</td>
<td>619.9</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>2.3%</td>
<td>378.2</td>
<td>65.8</td>
<td>14.9</td>
<td>__cray_sset_SNB</td>
</tr>
</tbody>
</table>

Copyright 2018 Cray Inc.
Table 1: Inclusive and Exclusive Time in Loops (from -hprofile_generate)

<table>
<thead>
<tr>
<th>Loop</th>
<th>Loop Incl</th>
<th>Loop Hit</th>
<th>Loop</th>
<th>Loop</th>
<th>Loop</th>
<th>Function=/.LOOP[.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incl</td>
<td>Time</td>
<td>Time%</td>
<td>Trips</td>
<td>Trips</td>
<td>Trips</td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>Time%</td>
<td>Avg</td>
<td>Min</td>
<td>Max</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>100.0%</td>
<td>631.108612</td>
<td>1</td>
<td>50.0</td>
<td>50</td>
<td>50</td>
<td>vhone_.LOOP.2.li.205</td>
</tr>
<tr>
<td>30.6%</td>
<td>193.127659</td>
<td>100</td>
<td>64.0</td>
<td>64</td>
<td>64</td>
<td>sweepy_.LOOP.1.li.38</td>
</tr>
<tr>
<td>30.6%</td>
<td>193.127263</td>
<td>6,400</td>
<td>128.0</td>
<td>128</td>
<td>128</td>
<td>sweepy_.LOOP.2.li.39</td>
</tr>
<tr>
<td>27.8%</td>
<td>175.704223</td>
<td>50</td>
<td>128.0</td>
<td>128</td>
<td>128</td>
<td>sweepz_.LOOP.05.li.53</td>
</tr>
<tr>
<td>27.8%</td>
<td>175.703983</td>
<td>6,400</td>
<td>64.0</td>
<td>64</td>
<td>64</td>
<td>sweepz_.LOOP.06.li.54</td>
</tr>
<tr>
<td>20.6%</td>
<td>130.290184</td>
<td>2,457,600</td>
<td>1,031.0</td>
<td>1,031</td>
<td>1,031</td>
<td>riemann_.LOOP.2.li.63</td>
</tr>
<tr>
<td>14.0%</td>
<td>88.344946</td>
<td>50</td>
<td>64.0</td>
<td>64</td>
<td>64</td>
<td>sweepx2_.LOOP.1.li.33</td>
</tr>
<tr>
<td>14.0%</td>
<td>88.344688</td>
<td>3,200</td>
<td>128.0</td>
<td>128</td>
<td>128</td>
<td>sweepx2_.LOOP.2.li.34</td>
</tr>
<tr>
<td>13.9%</td>
<td>87.468636</td>
<td>50</td>
<td>64.0</td>
<td>64</td>
<td>64</td>
<td>sweepx1_.LOOP.1.li.33</td>
</tr>
<tr>
<td>13.9%</td>
<td>87.468481</td>
<td>3,200</td>
<td>128.0</td>
<td>128</td>
<td>128</td>
<td>sweepx1_.LOOP.2.li.34</td>
</tr>
<tr>
<td>10.3%</td>
<td>64.960442</td>
<td>22,118,400</td>
<td>1,028.0</td>
<td>1,026</td>
<td>1,032</td>
<td>parabola_.LOOP.7.li.75</td>
</tr>
<tr>
<td>5.6%</td>
<td>35.320759</td>
<td>2,533,785,600</td>
<td>12.0</td>
<td>12</td>
<td>12</td>
<td>riemann_.LOOP.3.li.64</td>
</tr>
<tr>
<td>1.5%</td>
<td>9.708508</td>
<td>409,600</td>
<td>16.0</td>
<td>16</td>
<td>16</td>
<td>sweepz_.LOOP.07.li.61</td>
</tr>
<tr>
<td>1.5%</td>
<td>9.202399</td>
<td>50</td>
<td>128.0</td>
<td>128</td>
<td>128</td>
<td>sweepz_.LOOP.01.li.22</td>
</tr>
<tr>
<td>1.5%</td>
<td>9.201736</td>
<td>6,400</td>
<td>8.0</td>
<td>8</td>
<td>8</td>
<td>sweepz_.LOOP.02.li.23</td>
</tr>
<tr>
<td>1.5%</td>
<td>9.198684</td>
<td>51,200</td>
<td>128.0</td>
<td>128</td>
<td>128</td>
<td>sweepz_.LOOP.03.li.24</td>
</tr>
<tr>
<td>1.3%</td>
<td>7.941815</td>
<td>50</td>
<td>64.0</td>
<td>64</td>
<td>64</td>
<td>sweepz_.LOOP.12.li.116</td>
</tr>
<tr>
<td>1.3%</td>
<td>7.941276</td>
<td>3,200</td>
<td>64.0</td>
<td>64</td>
<td>64</td>
<td>sweepz_.LOOP.13.li.117</td>
</tr>
<tr>
<td>1.3%</td>
<td>7.935293</td>
<td>204,800</td>
<td>16.0</td>
<td>16</td>
<td>16</td>
<td>sweepz_.LOOP.14.li.118</td>
</tr>
</tbody>
</table>
# Call Tree with Loops

## Table 1: Function Call Tree View

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Calls</th>
<th>Calltree</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>623.115520</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>100.0%</td>
<td>623.115490</td>
<td>2.0</td>
<td>vhone_</td>
</tr>
<tr>
<td>100.0%</td>
<td>622.866674</td>
<td>--</td>
<td>vhone_.LOOP.2.1i.205</td>
</tr>
<tr>
<td>37.5%</td>
<td>233.441869</td>
<td>50.0</td>
<td>sweepz_</td>
</tr>
<tr>
<td>23.6%</td>
<td>147.324792</td>
<td>50.0</td>
<td>sweepz_.LOOP.05.1i.53</td>
</tr>
<tr>
<td>5.8%</td>
<td>35.940023</td>
<td>4,915,200.0</td>
<td>parabola_ (exclusive)</td>
</tr>
<tr>
<td>9.5%</td>
<td>58.970451</td>
<td>819,200.0</td>
<td>remap_</td>
</tr>
<tr>
<td>5.8%</td>
<td>35.940023</td>
<td>4,915,200.0</td>
<td>parabola_</td>
</tr>
<tr>
<td>2.9%</td>
<td>18.356531</td>
<td>819,200.0</td>
<td>remap_ (exclusive)</td>
</tr>
<tr>
<td>7.7%</td>
<td>47.983811</td>
<td>819,200.0</td>
<td>riemann_</td>
</tr>
<tr>
<td>2.9%</td>
<td>17.979033</td>
<td>2,457,600.0</td>
<td>parabola_</td>
</tr>
<tr>
<td>1.7%</td>
<td>10.292367</td>
<td>819,200.0</td>
<td>evolve_</td>
</tr>
<tr>
<td>13.8%</td>
<td>86.117076</td>
<td>50.0</td>
<td>sweepz_ (exclusive)</td>
</tr>
</tbody>
</table>

---

**Parallized Loop**
PARABOLA

If it Vectorizes, Check Memory Utilization

44. V----< do n = nmin-1, nmax
45. V ar(n) = a(n) + para(n,1)*diffa(n) + para(n,2)*da(n+1) + para(n,3)*da(n)
46. V al(n+1) = ar(n)
47. V----> enddo
49. V----< do n = nmin, nmax
50. V onemfl= 1.0 - flat(n)
51. V ar(n) = flat(n) * a(n) + onemfl * ar(n)
52. V al(n) = flat(n) * a(n) + onemfl * al(n)
53. V----> enddo
55. fV----< do n = nmin, nmax
56. fV ar(n) = flat(n) * a(n) + onemfl * ar(n)
57. fV al(n) = flat(n) * a(n) + onemfl * al(n)
58. fV----> enddo
60. fV al(n+1) = ar(n)
61. fV----> enddo
63. fV ar(n) = a(n)
64. fV al(n) = a(n)
65. fV----> enddo
67. f----< do n = nmin, nmax
68. f deltaa(n) = ar(n) - al(n)
69. f a6(n) = 6. * (a(n) - 0.5 * (al(n) + ar(n)))
70. f scrch1(n) = (ar(n) - a(n)) * (a(n) - al(n))
71. f scrch2(n) = deltaa(n) * deltaa(n)
72. f scrch3(n) = deltaa(n) * a6(n)
73. f----> enddo
75. V r2----< do n = nmin, nmax
76. V r2 if(scrch1(n) <= 0.0) then
77. V r2 ar(n) = a(n)
78. V r2 al(n) = a(n)
79. V r2 endif
80. V r2 if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
81. V r2 if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
82. V r2----> enddo
84. V r2----< do n = nmin, nmax
85. V r2 deltaa(n)= ar(n) - al(n)
86. V r2 a6(n) = 6. * (a(n) - 0.5 * (al(n) + ar(n)))
87. V r2----> enddo

June 2018
Copyright 2018 Cray Inc.
**If it Vectorizes, Check Memory Utilization**

If it vectorizes, check memory utilization.

44. \( V \quad \text{do } n = \text{nmin-1, nmax} \)
45. \( V \quad a(n) = a(n) + \text{para}(n,1) * \text{diffa}(n) + \text{para}(n,2) * \text{da}(n+1) + \text{para}(n,3) * \text{da}(n) \)
46. \( V \quad a(n+1) = a(n) \)
47. \( V \quad \text{endo} \)
48. \( V\text{--}> \quad \text{endo} \)
49. \( fV \quad \text{do } n = \text{nmin, nmax} \)
50. \( fV \quad \text{onemfl } = 1.0 - \text{flat}(n) \)
51. \( fV \quad \text{al}(n) = \text{flat}(n) * a(n) + \text{onemfl} * a(n) \)
52. \( fV \quad \text{al}(n) = \text{flat}(n) * a(n) + \text{onemfl} * a(n) \)
53. \( fV \quad \text{endo} \)
54. \( fV \quad \text{endo} \)
55. \( fV \quad \text{endo} \)
56. \( fV \quad \text{endo} \)
57. \( fV \quad \text{endo} \)
58. \( fV \quad \text{endo} \)
59. \( fV \quad \text{endo} \)
60. \( fV \quad \text{endo} \)
61. \( fV \quad \text{endo} \)
62. \( fV \quad \text{endo} \)
63. \( fV \quad \text{endo} \)
64. \( fV \quad \text{endo} \)
65. \( fV \quad \text{endo} \)
66. \( fV \quad \text{endo} \)
67. \( fV \quad \text{endo} \)
68. \( fV \quad \text{endo} \)
69. \( fV \quad \text{endo} \)
70. \( fV \quad \text{endo} \)
71. \( fV \quad \text{endo} \)
72. \( fV \quad \text{endo} \)
73. \( fV \quad \text{endo} \)
74. \( fV \quad \text{endo} \)
75. \( fV \quad \text{endo} \)
76. \( fV \quad \text{endo} \)
77. \( fV \quad \text{endo} \)
78. \( fV \quad \text{endo} \)
79. \( fV \quad \text{endo} \)
80. \( fV \quad \text{endo} \)
81. \( fV \quad \text{endo} \)
82. \( fV \quad \text{endo} \)
83. \( fV \quad \text{endo} \)
84. \( fV \quad \text{endo} \)
85. \( fV \quad \text{endo} \)
86. \( fV \quad \text{endo} \)
87. \( fV \quad \text{endo} \)

**Remove unnecessary arrays and fuse loops**

52. \( Vr2 \quad \text{do } n = \text{nmin, nmax} \)
53. \( Vr2 \quad \text{onemfl } = 1.0 - \text{flat}(n) \)
54. \( Vr2 \quad \text{ar}(n) = \text{flat}(n) * a(n) + \text{onemfl} * a(n) \)
55. \( Vr2 \quad \text{al}(n) = \text{flat}(n) * a(n) + \text{onemfl} * a(n) \)
56. \( Vr2 \quad \text{delta}(n) = \text{ar}(n) - \text{al}(n) \)
57. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
58. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
59. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
60. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
61. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
62. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
63. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
64. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
65. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
66. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
67. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
68. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)
69. \( Vr2 \quad \text{al}(n) = 3.0 * \text{a}(n) - 2.0 * \text{a}(n) \)

---

June 2018

Copyright 2018 Cray Inc.
A loop starting at line 64 was not vectorized because a recurrence was found on "pmid" at line 77.
If it Doesn’t Vectorize – Fix It

63. + 1--------- do l = lmin, lmax
64. + 1 2--< do n = 1, 12
65. 1 2 pmold(l) = pmid(l)
66. 1 2 wlf(l) = 1.0 + gamfac1*(pmid(l) - pflt(l)) * pflt(l)
67. 1 2 wrgh(l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prgh(l)
68. 1 2 wlf(l) = clft(l) * sqrt(wlf(l))
69. 1 2 wrgh(l) = crgh(l) * sqrt(wrgh(l))
70. 1 2 zlft(l) = 4.0 * vlft(l) * wlf(l) * wlf(l)
71. 1 2 zrgh(l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72. 1 2 umid(l) = ulft(l) - (pmid(l) - plft(l)) / wlf(l)
73. 1 2 umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
74. 1 2 pmid(l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (zrgh(l) - zlft(l))
75. 1 2 pmid(l) = max(smallp, pmid(l))
76. 1 2 if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
77. 1 2--> enddo
78. 1----> enddo

62. A----< converged = .F.
63. + 1------< do n = 1, 12
64. 1 Vr2--< do l = lmin, lmax
65. 1 Vr2 if(.not.converged(l))then
66. 1 Vr2 pmold(l) = pmid(l)
67. 1 Vr2 wlf(l) = 1.0 + gamfac1*(pmid(l) - pflt(l)) * pflt(l)
68. 1 Vr2 wrgh(l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prgh(l)
69. 1 Vr2 wlf(l) = clft(l) * sqrt(wlf(l))
70. 1 Vr2 wrgh(l) = crgh(l) * sqrt(wrgh(l))
71. 1 Vr2 zlft(l) = 4.0 * vlft(l) * wlf(l) * wlf(l)
72. 1 Vr2 zrgh(l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
73. 1 Vr2 umid(l) = ulft(l) - (pmid(l) - plft(l)) / wlf(l)
74. 1 Vr2 umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
75. 1 Vr2 pmid(l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (zrgh(l) - zlft(l))
76. 1 Vr2 pmid(l) = max(smallp, pmid(l))
77. 1 Vr2 if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) then
78. 1 Vr2 converged(l) = .T.
79. 1 Vr2 endif
80. 1 Vr2 endif
81. 1 Vr2----> enddo
82. 1 Vr2----> enddo
83. 1 Vr2 if(all(converged(lmin:lmax)))exit
84. 1 Vr2---> enddo
85. + 1
86. 1------> enddo
<table>
<thead>
<tr>
<th>%</th>
<th>Time</th>
<th>Line</th>
<th>%</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.3</td>
<td>1,679.0</td>
<td>---</td>
<td>---</td>
<td>sweepz_3</td>
</tr>
<tr>
<td>3</td>
<td>leveque/CUG_2018/VH1_version1/sweepz.f90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.0</td>
<td>489.4</td>
<td>21.6</td>
<td>4.3</td>
</tr>
<tr>
<td>4</td>
<td>1.3</td>
<td>211.9</td>
<td>38.1</td>
<td>15.4</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>165.0</td>
<td>43.0</td>
<td>20.8</td>
</tr>
<tr>
<td>4</td>
<td>2.7</td>
<td>447.2</td>
<td>36.8</td>
<td>7.7</td>
</tr>
</tbody>
</table>
22.  + i---------------< do j = 1, js
23.  + i i---------------< do m = 1, npey
24.  + i i i---------------< do i = 1, isy
25.  i i i                   n = i + isy*(m-1)
26.  + i i i i 4-----------< do k = 1, ks
27.  i i i 4  VR--<>       send3(1,j,k,n) = recv2(1,k,i,j,m)
28.  i i i 4                 send3(2,j,k,n) = recv2(2,k,i,j,m)
29.  i i i 4                 send3(3,j,k,n) = recv2(3,k,i,j,m)
30.  i i i 4                 send3(4,j,k,n) = recv2(4,k,i,j,m)
31.  i i i 4                 send3(5,j,k,n) = recv2(5,k,i,j,m)
32.  i i i 4                 send3(6,j,k,n) = recv2(6,k,i,j,m)
33.  i i i 4---------->      enddo
34.  i i i---------->       enddo
35.  i i i---------->       enddo
36.  i----------->         enddo
SWEEPZ (continued)

21. !-----------------------------------------------
22. !$OMP PARALLEL DO
23.  do j = 1, js
24.  do m = 1, npey
25.  do i = 1, isy
26.  n = i + isy*(m-1)
27.  do k = 1, ks
28.  send3(1,j,k,n) = recv2(1,k,i,j,m)
29.  send3(2,j,k,n) = recv2(2,k,i,j,m)
30.  send3(3,j,k,n) = recv2(3,k,i,j,m)
31.  send3(4,j,k,n) = recv2(4,k,i,j,m)
32.  send3(5,j,k,n) = recv2(5,k,i,j,m)
33.  send3(6,j,k,n) = recv2(6,k,i,j,m)
34.  enddo
35.  enddo
36.  enddo
37.  enddo
Routines by Percent of Samples

- mpi_alltoall
- parabola
- sweep
- sweepz
- remap
- riemann
- paraset
- __cray_sset_SNB
- sweepx

Routines by Percent of Samples:

- vhone+69323-16s (128 pes, 1 ths)
- vhone+69969-16s (128 pes, 2 ths)
- vhone+70555-16s (128 pes, 4 ths)
- vhone+71112-16s (128 pes, 8 ths)

June 2018

Copyright 2018 Cray Inc.
# Profile for 1 Thread and 4 Threads

## Table 1: Profile by Function (1 Thread)

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Samp%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

100.0% | 38,034.0 | -- | -- | Total

| 73.5 | 27,936.5 | -- | -- | USER
|------|----------|----|----|------|
| 19.2 | 7,320.6  | 202.4 | 2.7 | parabola_
| 11.1 | 4,206.8  | 179.2 | 4.1 | sweepy_
| 8.8  | 3,350.8  | 233.2 | 6.6 | sweepz_
| 8.5  | 3,231.5  | 138.5 | 4.1 | remap_
| 8.2  | 3,106.3  | 2,321.7 | 43.1 | riemann_
| 4.4  | 1,660.7  | 105.3 | 6.0 | paraset_
| 3.1  | 1,190.9  | 72.1  | 5.8 | sweepx1_
| 3.0  | 1,136.4  | 68.6  | 5.7 | sweepx2_

| 20.5 | 7,814.0  | -- | -- | MPI
|------|----------|----|----|------|
| 18.8 | 7,159.8  | 835.2 | 10.5 | mpi_alltoall

## Table 1: Profile by Function (4 threads)

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Samp%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

100.0% | 15,265.8 | -- | -- | Total

| 54.1 | 8,260.7 | -- | -- | USER
|------|--------|----|----|------|
| 13.8 | 2,101.9 | 84.1 | 3.9 | parabola_
| 8.2  | 1,252.8 | 62.2 | 4.8 | sweepy_
| 7.1  | 1,090.2 | 63.8 | 5.6 | sweepz_
| 6.1  | 926.8   | 62.2 | 6.3 | remap_
| 5.9  | 901.4   | 712.6 | 44.5 | riemann_
| 3.1  | 480.2   | 71.8 | 13.1 | paraset_
| 2.4  | 359.3   | 49.7 | 12.3 | sweepx1_
| 2.2  | 329.7   | 46.3 | 12.4 | sweepx2_

| 41.4 | 6,326.9 | -- | -- | MPI
|------|--------|----|----|------|
| 40.1 | 6,120.0 | 209.0 | 3.3 | mpi_alltoall

---

June 2018

Copyright 2018 Cray Inc.
## Table 5: Profile by Group, Function, and Line

<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Mem LOAD UOPS RETIRED</th>
<th>Resource Stalls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>506.4</td>
<td>--</td>
<td>--</td>
<td>52,444,505,648,539,880</td>
<td>810,810,111,305</td>
</tr>
<tr>
<td>67.0%</td>
<td>339.1</td>
<td>--</td>
<td>--</td>
<td>35,351,497,873,955,312</td>
<td>551,686,946,473</td>
</tr>
<tr>
<td>25.6%</td>
<td>129.5</td>
<td>--</td>
<td>--</td>
<td>13,561,376,423,707,960</td>
<td>207,864,527,154</td>
</tr>
<tr>
<td>14.0%</td>
<td>71.1</td>
<td>77.9</td>
<td>52.7%</td>
<td>7,540,450,746,779,697</td>
<td>113,994,086,387</td>
</tr>
<tr>
<td>1.0%</td>
<td>5.2</td>
<td>8.8</td>
<td>63.0%</td>
<td>510,173,395,620,708</td>
<td>8,400,427,348</td>
</tr>
<tr>
<td>8.1%</td>
<td>41.0</td>
<td>41.0</td>
<td>50.4%</td>
<td>4,241,915,862,200,189</td>
<td>65,974,673,046</td>
</tr>
<tr>
<td>1.3%</td>
<td>6.8</td>
<td>9.2</td>
<td>58.0%</td>
<td>714,682,558,419,997</td>
<td>10,882,073,895</td>
</tr>
<tr>
<td>12.1%</td>
<td>61.2</td>
<td>--</td>
<td>--</td>
<td>6,449,735,211,652,689</td>
<td>103,694,034,743</td>
</tr>
<tr>
<td>3.2%</td>
<td>16.2</td>
<td>19.8</td>
<td>55.6%</td>
<td>1,653,665,489,088,910</td>
<td>27,467,001,609</td>
</tr>
<tr>
<td>6.4%</td>
<td>32.2</td>
<td>34.8</td>
<td>52.4%</td>
<td>3,465,660,652,265,039</td>
<td>54,562,573,915</td>
</tr>
<tr>
<td>10.2%</td>
<td>51.9</td>
<td>--</td>
<td>--</td>
<td>5,479,965,955,430,281</td>
<td>81,379,551,948</td>
</tr>
<tr>
<td>2.4%</td>
<td>12.3</td>
<td>19.7</td>
<td>62.0%</td>
<td>1,356,797,349,204,156</td>
<td>21,098,054,630</td>
</tr>
<tr>
<td>1.8%</td>
<td>8.9</td>
<td>13.1</td>
<td>60.0%</td>
<td>945,580,000,330,081</td>
<td>13,804,851,073</td>
</tr>
<tr>
<td>3.7%</td>
<td>18.8</td>
<td>28.2</td>
<td>60.4%</td>
<td>1,961,528,744,947,444</td>
<td>29,502,970,062</td>
</tr>
</tbody>
</table>

---

June 2018

Copyright 2018 Cray Inc.
Code Responsible for Memory Bandwidth

Difficult to merge these loops – compiler did merge two
Summary

● The Whack-a-mole approach was shown to be an effective recipe for using the tools to obtain results.

● Cray compiler and performance tools gives customers the capability they need to identify performance issues at scale in important production codes.

● Simple interfaces coupled with a wealth of capability offer the most flexibility for users who need to scale applications to larger job and problem sizes.
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.