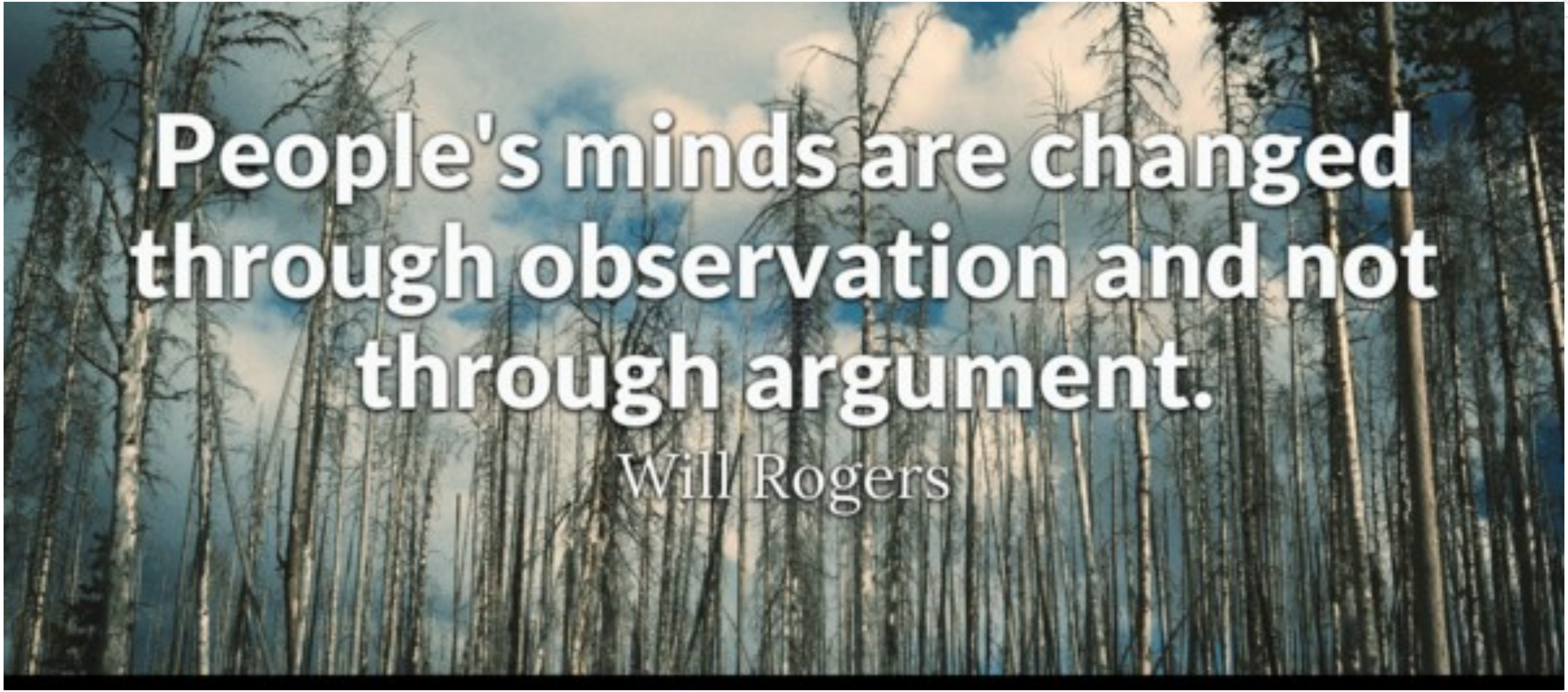
Applying the "Whack-a-Mole" Method Using Cray's Perftools for Identifying the Moles

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



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

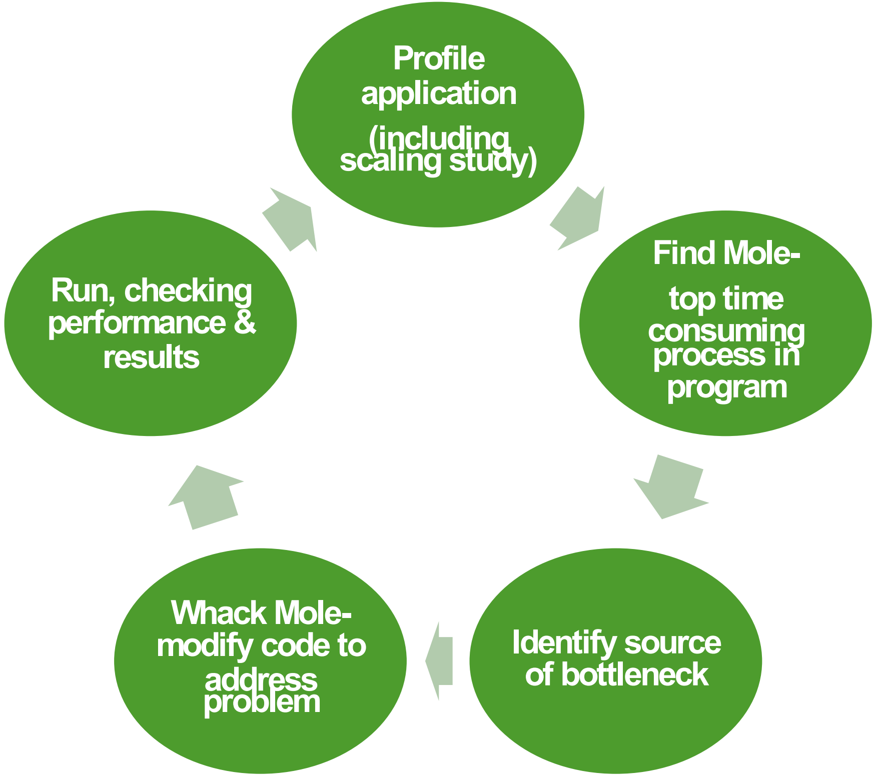
- John met Ricky in 1993 when he visited APR to evaluate FORGE
  - Taught him to use "." in vi
- Worked extensively with Ricky for many years at ORNL (AUG 2005 – 2014)

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)



Using the Process on Real Applications

UMT

Leslie3d

VH1

UMT

The UMT benchmark is a 3D, deterministic, multi-group, photon transport code for unstructured meshes

UMT Call Tree on 8 Nodes

Nesting Level

Inclusive Percent of Time

Leaf child routine- Exclusive Percent of Time

Called From snflwxyz_

```
Table                            ree View

                                  ead=HIDE

   0.0% | 3,971.6 | Total
|------------------------------------------------
| 99.8% | 3,961.9 | main
||-----------------------------------------------
||  9                                  e
3|                                  :cxxRadtr
4|
||-----------------------------------------------
5||||                             insn_
||||||-------------------------------------------
6|||||   66.9% | 2,655.4 | rswpmd_
7|||||   66.8% | 2,654.7 |  snflwxyz_
||||||||-----------------------------------------
8|||||||   22.0% |   872.3 | exchange_
|||||||||----------------------------------------
9||||||||   13.5% |   535.7 | MPI_WAIT
9||||||||    8.5% |   336.5 | exchange_(exclusive)
|||||||||========================================
8|||||||   15.6% |   617.7 | snswp3d_
8|||||||    9.7% |   383.4 | initexchange_
9|||||||         |         | MPI_BARRIER

8|||||||    6.5% |   260.0 | snmoments_
8|||||||    5.8% |   230.6 | __cray_dcopy_HSW
8|||||||    4.0% |   160.7 | testfluxconv_
9|||||||         |         |  mpi_mpiallreducet_r_$mpif90_m
10||||||         |         |   MPI_ALLREDUCE
8|||||||    3.2% |   128.7 | setincidentflux_
||||||||=========================================
6|||||  21.6% |   859.3 | exchange_
|||||||------------------------------------------
7||||||  15.6% |   620.0 | MPI_WAIT
7||||||   6.0% |   239.3 | exchange_(exclusive)
|||||||==========================================
          5.0% |   198.1 | rtstrtsn_
      -------------------------------------------
               |   106.9 | rtstrtsn_(exclusive)
               |    90.1 | __cray_dcopy_HSW
      ===========================================
               |   115.2 | advancert_
      -------------------------------------------
6|||||    1.5% |    60.0 | advancert_(exclusive)
6|||||    1.4% |    55.2 | snmoments_
|||||============================================
||   1.0% |    40.9 | initialize
|================================================
```

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

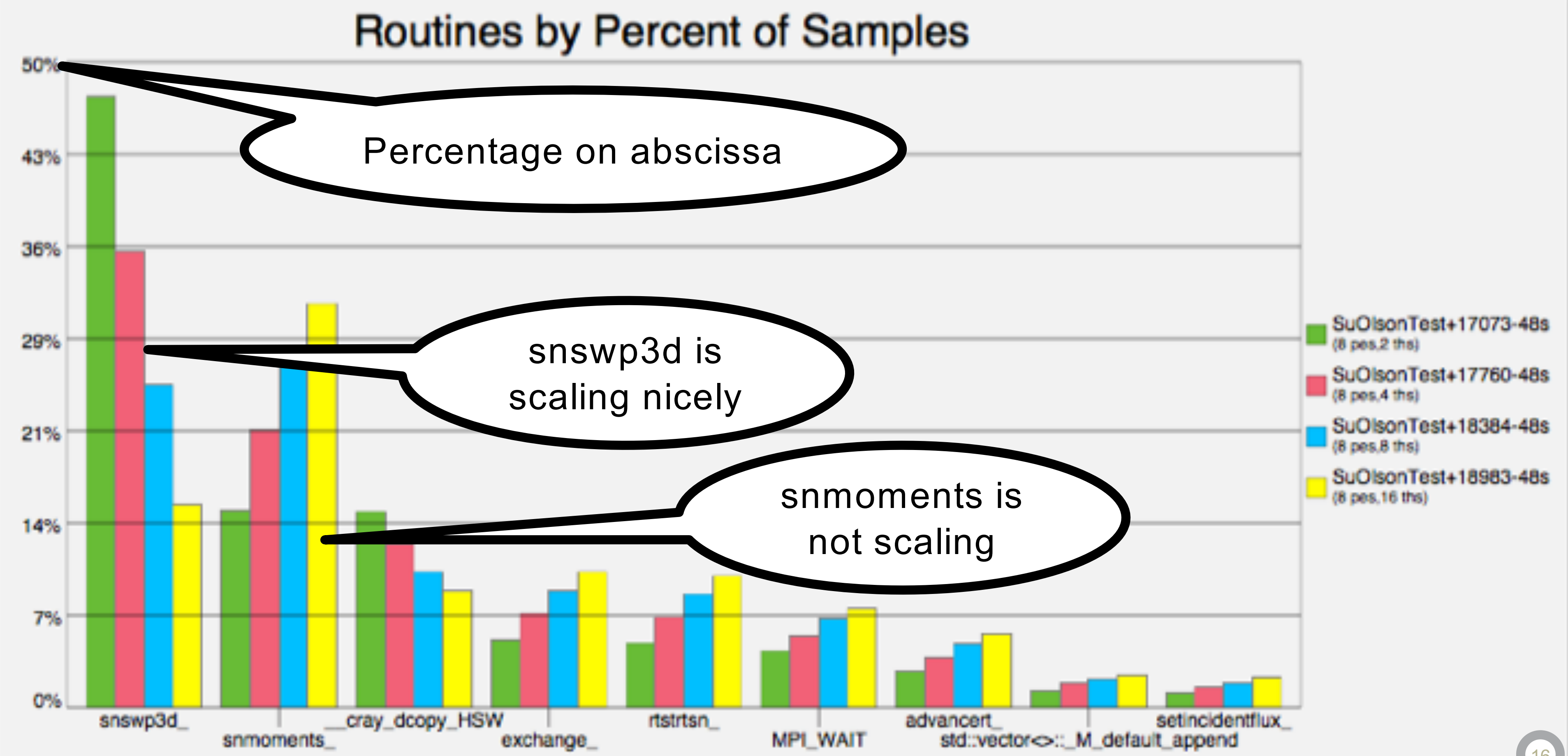
Problem size: X
Run 8 MPI tasks on 8 nodes,
vary OpenMP threads from 1 to 32

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



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

```
    1 2 M-----< !$OMP PARALLEL DO  PRIVATE(Angle,mm,thnum)
  + 1 2 M m---<        AngleLoop: do mm=1,NangBin
    1 2 M m
                          Angle = QuadSet% AngleOrder(mm,binSend)
                          thnum = 1
    1 2                   thnum = OMP_GET_THREAD_NUM() + 1
    1 2 M m
    1 2 M m     !         Set angular fluxes for reflected angles
    1
  + 1                     call snreflect(Angle, PSIB)
    1 2 M m
    1 2 M m     !         Sweep the mesh, calculating PSI for each corner; the
    1 2 M m     !         boundary flux array PSIB is also updated here.
    1 2 M m     !         Mesh cycles are fixed automatically.
    1 2   m
  + 1 2 M                 call snswp3d(Groups, Angle,                          &
    1 2 M m                             QuadSet%next(1,Angle),QuadSet%nextZ(1,Angle),   &
    1 2 M m                             PSI(1,1,Angle),PSIB(1,1,Angle))
    1 2 M m
    1 2 M m-->>        enddo AngleLoop
```

Threaded Region

Threaded loop

Outer Loop of Nest

SNMOMENTS is Not Threaded

```
   AC------<>    Phi(:,:) = zero

 + 1--------<    Angl          ngle=1,QuadSet%NumAngles
   1
   1               quadwt = QuadSet% Weight(Angle)
   1
   1               if (quadwt /= zero) then
   1
 + 1 2------<        do ic=1,ncornr
   1 2 Vr2--<          do ig=1,Groups
   1 2 Vr2               Phi(ig,ic) = Phi(ig,ic) + quadwt*psic(ig,ic,Angle)
   1 2 Vr2-->          enddo
   1 2------>        ddo
   1
   1               endif

    -------->    enddo AngleLoop


                 return
                 end subroutine snmoments
```

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)

Remember there is no threading in these samples

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

Ordered by loops that take most of the time

This is the loop with OpenMP

snmoments has nice loops lengths

[illegible]	[illegible]	[illegible]	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
98.9%	[illegible]	0.000069	1	7.0	7	7	main.LOOP.1.l[illegible]
92.4%	[illegible]48820	0.000154	7	4.7	3	5	rtmainsn_.LOOP.1[illegible]
92.2%	52.728532	0.000114	33	1.0	1	1	rtmainsn_.LOOP[illegible].159
92.1%	52.647694	0.000214	33	1.0	1	1	rtmainsn_.LO[illegible].li.167
71.6%	40.965850	0.000417	33	1.6	1	3	snflwxyz_.[illegible].1.li.87
59.3%	33.937160	0.001694	54	6.9	4	8	snflwxyz[illegible]LOOP.2.li.95
44.0%	25.175652	0.043310	[illegible]	[illegible]	[illegible]	90	snflwxyz_.LOOP.3.li.106
41.2%	23.541605	0.052865	[illegible]	[illegible]	[illegible]	28	snswp3d_.LOOP.01.li.78
28.2%	16.146745	3.884386	[illegible]	[illegible]	[illegible]	8	snswp3d_.LOOP.04.li.99
19.1%	10.931600	0.000059	[illegible]	[illegible]	[illegible]	8	exchange_.LOOP.06.li.120
19.1%	10.931541	10.931541	[illegible]	[illegible]	[illegible]	6	exchange_.LOOP.07.li.121
14.0%	7.993231	1.052804	7,[illegible]	[illegible]	[illegible]	3	snswp3d[illegible]
11.9%	6.800924	6.800924	10,[illegible]	[illegible]	[illegible]	400	[illegible]
8.2%	4.692389	0.135064	[illegible]	[illegible]	[illegible]	8	[illegible]
8.0%	4.557325	4.557325	7,499,520	400.0	4[illegible]0	400	[illegible]
6.9%	3.944975	0.000352	405	5.0	3	6	exchan[illegible]OP.08.li.132
6.9%	3.944623	1.409580	2,025	1.6	1	8	excha[illegible]LOOP.09.li.134
5.8%	3.312134	3.312134	7,499,520	3.0	3	[illegible]	snsw[illegible]d_.LOOP.05.li.110
5.0%	2.864861	0.003020	40	720.0	720	[illegible]	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
4.7%	2.700843	0.000089	405	1.6	1	8	exchange_.LOOP.01.li.71

Profile Loops with Call Tree (1 of 2)

```
Table 1:  Function Calltree View
  Time% |       Time |     Calls | Calltree
        |            |           |  PE=HIDE

 100.0% | 57.147586 |        -- | Total
|--------------------------------------------------------
| 100.0% | 57.147550 |       2.0 | main
|  98.9% | 56.543299 |        -- |  main.LOOP.1.li.189
3        |           |           |   advance
4  98.9% | 56.538457 |        -- |    Teton<>::cxxRadtr
5  98.9% | 56.538442 |       7.0 |     radtr_
|||||-----------------------------------------------------
6|||||  97.0% | 55.459264 |       7.0 | rtmainsn_
|||||||---------------------------------------------------
7||||||  92.5% | 52.838008 |        -- | rtmainsn_.LOOP.1.li.149
8||||||  92.3% | 52.719389 |        -- |  rtmainsn_.LOOP.2.li.159
9||||||  92.1% | 52.638677 |        -- |   rtmainsn_.LOOP.3.li.167
|||||||||-------------------------------------------------
10||||||||  76.7% | 43.811029 |      33.0 | rswpmd_
11||||||||  76.6% | 43.802991 |      33.0 |  snflwxyz_
||||||||||||--------------------------------------------
12||||||||||  71.6% | 40.911652 |        -- | snflwxyz_.LOOP.1.li.87
|||||||||||||-------------------------------------------
13|||||||||||  59.3% | 33.883425 |        -- | snflwxyz_.LOOP.2.li.95
||||||||||||||------------------------------------------
14||||||||||||  44.0% | 25.123658 |        -- | snflwxyz_.LOOP.3.li.106
```

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

```
      AC------<>     Phi(:,:) = zero

    + 1--------<     AngleLoop: do Angle=1,QuadSet%NumAngles
      1
      1                quadwt = QuadSet% Weight(Angle)
      1
      1                if (quadwt /= zero) then
      1
    + 1 2------<         do ic=1,ncornr
      1 2 Vr2--<           do ig=1,Groups
      1 2 Vr2                Phi(ig,ic) = Phi(ig,ic) + quadwt*psic(ig,ic,Angle)
      1 2 Vr2-->           enddo
      1 2------>         enddo
      1
      1                endif
      1
      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 informtation is obtained from a perftools-lite-loops run:

|  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(:400,: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
```

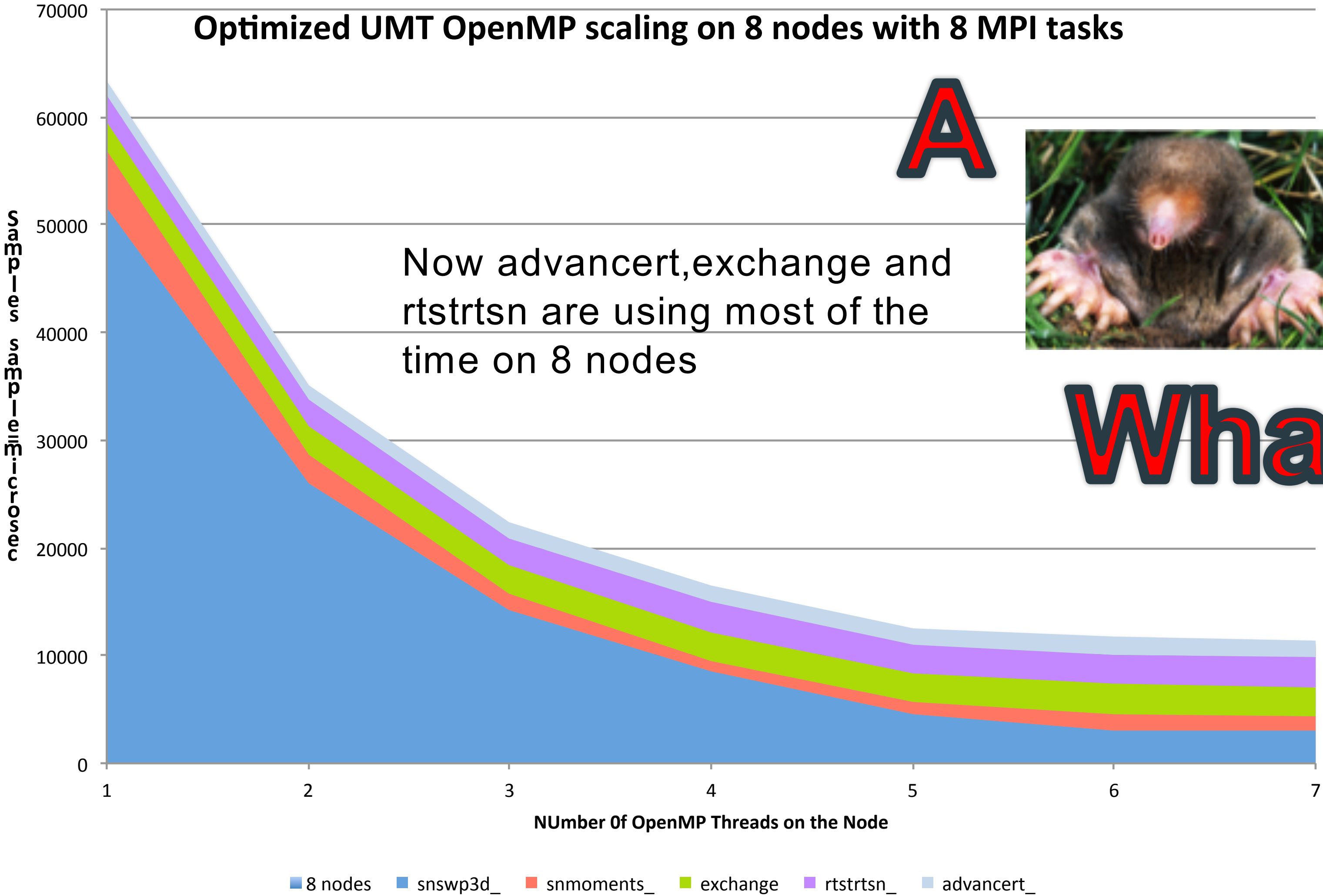
These do not need to be constant in new OpenMP standard

Parallelizing a Different Loop

```
      AC--------<>     Phi(:,:) = zero
      M----------< !$OMP PARALLEL DO PRIVATE(ic,angle,quadwt,ig)
    + M m--------<     CornerLoop: do ic=1,ncornr
    + M m 3------<     AngleLoop: do Angle=1,QuadSet%NumAngles
      M m 3
      M m 3              quadwt = QuadSet% Weight(Angle)
      M m 3
      M m 3              if (quadwt /= zero) then
      M m 3
      M m 3 Vr2--<           do ig=1,Groups
      M m 3 Vr2                Phi(ig,ic) = Phi(ig,ic) + quadwt*psic(ig,ic,Angle)
      M m 3 Vr2-->           enddo
      M m 3
      M m 3              endif
      M m 3
      M m 3------>     enddo AngleLoop
      M m------->>     enddo CornerLoop
```

Pull the ic loop outside
the angle loop…

Optimized UMT OpenMP scaling on 8 nodes with 8 MPI tasks



Now advancert,exchange and rtstrtsn are using most of the time on 8 nodes

A

Whacked

Major Loop in RTSTRTSN

```
+ F---------------<       ZoneLoop: do zone=1,nzones
  F I                       Z => getZoneData(Geom, zone)
  F                         nCorner = Z% nCorner
  F                         c0      = Z% c0
+ F 2-------------<         do ia=1,Size%nangSN
  F 2 C-----------<           do c=1,nCorner
  F 2 C VCr2------<>            Z% STime(:,c,ia) = tau*psir(:,c0+c,ia)
  F 2 C----------->           enddo
  F 2------------->         enddo
  F--------------->       enddo ZoneLoop
```

Parallelized Using OpenMP

```
     M-----------------<  !$OMP PARALLEL DO PRIVATE(zone,nCorner,c0,ia,c)
   + M mF--------------<        ZoneLoop: do zone=1,nzones
     M mF I                       Z => getZoneData(Geom, zone)
     M mF                         nCorner = Z% nCorner
     M mF                         c0      = Z% c0
   + M mF 3------------<          do ia=1,Size%nangSN
     M mF 3 C----------<            do c=1,nCorner
     M mF 3 C VCr2-----<>             Z% STime(:,c,ia) = tau*psir(:,c0+c,ia)
     M mF 3 C---------->            enddo
     M mF 3------------>          enddo
     M mF-------------->>       enddo ZoneLoop
```

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 communicat
 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
 96.      1 2 3 4                           ib = Comm% ListSend(nsend0+i)
 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

```
      1 2                     ! scalar setup loop
    + 1 2 r4-------------<             do ia=1,NangBin
      1 2 r4                             vmessage(ia) = message
      1 2 r4                             vnsend0(ia) = nsend0
      1 2 r4                             nsend = Comm% nsend(ia)
      1 2 r4                             message = message + nsend * ngroups
      1 2 r4                             nsend0 = nsend0 + nsend
      1 2 r4------------->             enddo
      1 2
      1 2 M--------------<    !$OMP PARALLEL DO PRIVATE(IA,Angle,nsend,i,ib,ig,message,nsend0)
    + 1 2 M m------------<             do ia=1,NangBin
      1 2 M m                            message = vmessage(ia)
      1 2 M m                            nsend0 = vnsend0(ia)
      1 2 M m
      1 2 M m                            Angle = QuadSet% AngleOrder(ia,bin)
      1 2 M m                            nsend = Comm% nsend(ia)
      1 2 M m
    + 1 2 M m 5----------<               do i=1,nsend
      1 2 M m 5
      1 2 M m 5                            ib = Comm% ListSend(nsend0+i)
      1 2 M m 5
    + 1 2 M m 5 Vpr2-----<                 do ig=1,ngroups
      1 2 M m 5 Vpr2                         Comm% psibsend(message+ig) = psib(ig,ib,Angle)
      1 2 M m 5 Vpr2----->                 enddo
      1 2 M m 5
      1 2 M m 5                            message = message + ngroups
      1 2 M m 5
      1 2 M m 5---------->               enddo
      1 2 M m                            nsend0 = nsend0 + nsend
      1 2 M m----------->>             enddo
```

Major Loop in AdvanceRT

```
136.     Fr2 I                Z        => getZoneData(Geom, zone)
137.     Fr2
138.     Fr2                  nCorner = Z% nCorner
139.     Fr2                  c0      = Z% c0
140.     Fr2                  PhiAve  = zero
141.     Fr2
142.   + Fr2 2-----------<    do c=1,nCorner
143.     Fr2 2                  volumeRatio = Z% VolumeOld(c)/Z% Volume(c)
144.     Fr2 2 Vr2-------<>     Phi(:,c0+c) = Phi(:,c0+c)*volumeRatio
145.     Fr2 2
146.   + Fr2 2 3---------<      do ia=1,numAngles
147.     Fr2 2 3 Vr2-----<>       psir(:,c0+c,ia) = psir(:,c0+c,ia)*volumeRatio
148.     Fr2 2 3--------->      enddo
149.     Fr2 2
150.     Fr2 2                  sumRad = zero
151.     Fr2 2 Vr4--------<     do ig=1,ngr
152.     Fr2 2 Vr4                sumRad = sumRad + Phi(ig,c0+c)
153.     Fr2 2 Vr4-------->     enddo
154.     Fr2 2
155.     Fr2 2                  PhiAve = PhiAve + Z% Volume(c)*sumRad
156.     Fr2 2
157.     Fr2 2----------->    enddo
```

AdvanceRT Loop Parallelized with OpenMP

```
 + F----------------<    ZoneLoop2: do zone=1,nzones
   F
   F M--------------< !$OMP PARALLEL PRIVATE(c,volumeRatio,ia,sumRad,ig)
   F M I                      Z        => getZoneData(Geom, zone)
   F M
   F M                        nCorner = Z% nCorner
   F M                        c0      = Z% c0
   F M                        PhiAve  = zero
   F M                !$OMP DO  REDUCTION(+:PhiAve)
 + F M m------------<         do c=1,nCorner
   F M m                        volumeRatio = Z% VolumeOld(c)/Z% Volume(c)
   F M m Vr2-------<>           Phi(:,c0+c) = Phi(:,c0+c)*volumeRatio
   F M m
 + F M m 4----------<           do ia=1,numAngles
   F M m 4 Vr2-----<>             psir(:,c0+c,ia) = psir(:,c0+c,ia)*volumeRatio
   F M m 4---------->           enddo
   F M m
   F M m                        sumRad = zero
   F M m Vr4--------<           do ig=1,ngr
   F M m Vr4                      sumRad = sumRad + Phi(ig,c0+c)
   F M m Vr4-------->           enddo
   F M m
   F M m                        PhiAve = PhiAve + Z% Volume(c)*sumRad
   F M m
   F M m------------>         enddo
   F M--------------> !$OMP END PARALLEL
```

Z is a pointer in a module and it must be made threadprivate

and set by all threads

8 Node Configuration

Original

```
Table 1:  Profile by Function

  Samp% |      Samp |  Imb. |  Imb. | Group
        |           |  Samp | Samp% |  Function=[MAX10]
        |           |       |       |   PE=HIDE
        |           |       |       |    Thread=HIDE

 100.0% | 27,861.2 |    -- |    -- | Total
|-------------------------------------------------------
|  79.4% | 22,111.1 |    -- |    -- | USER
||------------------------------------------------------
||  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% | advancert_
||   2.2% |    619.4 |  66.6 | 11.1% | double* std::__un
||   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
|=======================================================
```

Optimized

```
Table 1:  Profile by Function

  Samp% |      Samp |  Imb. |  Imb. | Group
        |           |  Samp | Samp% |  Function=[MAX10]
        |           |       |       |   PE=HIDE
        |           |       |       |    Thread=HIDE

 100.0% | 15,762.0 |    -- |    -- | Total
|-------------------------------------------------------
|  67.6% | 10,658.9 |    -- |    -- | USER
||------------------------------------------------------
||  41.3% |  6,509.0 |  37.0 |  0.6% | snswp3d_
||   5.8% |    917.6 |  10.4 |  1.3% | snmoments_.LOOP@li.58
||   3.9% |    620.6 |  65.4 | 10.9% | double* std::__uninit
||   3.5% |    557.4 |   8.6 |  1.7% | setincidentflux_
||   3.5% |    551.4 |   2.6 |  0.5% | rtstrtsn_.LOOP@li.51
||   2.2% |    350.4 |   5.6 |  1.8% | advancert_.LOOP@li.142
||   2.1% |    338.5 |  16.5 |  5.3% | exchange_.LOOP@li.103
||   2.1% |    334.6 |   9.4 |  3.1% | exchange_.LOOP@li.178
||======================================================
|  20.1% |  3,168.6 |    -- |    -- | ETC
||------------------------------------------------------
||  18.7% |  2,949.2 |  94.8 |  3.6% | __cray_dcopy_HSW
||======================================================
|  12.1% |  1,904.5 |    -- |    -- | MPI
||------------------------------------------------------
||  10.5% |  1,653.0 |  66.0 |  4.4% | 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

32 Node Configuration

Original

```
Table 1:  Profile by Function

  Samp% |     Samp |  Imb. |  Imb. | Group
        |          |  Samp | Samp% |  Function=[MAX10]
        |          |       |       |   PE=HIDE
        |          |       |       |    Thread=HIDE

 100.0% | 4,015.1 |    -- |    -- | Total
|-----------------------------------------------------
|  47.9% | 1,924.6 |    -- |    -- | MPI
||----------------------------------------------------
||  29.6% | 1,190.3 | 452.7 | 27.7% | MPI_WAIT
||  13.4% |   537.8 | 616.2 | 53.6% | MPI_BARRIER
||   4.7% |   188.6 | 239.4 | 56.2% | MPI_ALLREDUCE
||====================================================
|  43.8% | 1,760.5 |    -- |    -- | USER
||----------------------------------------------------
||  14.1% |   567.9 |  46.1 |  7.5% | snswp3d_
||  13.9% |   558.1 | 166.9 | 23.1% | exchange_
||   7.5% |   300.1 |  11.9 |  3.8% | snmoments_
||   3.1% |   124.6 |  41.4 | 25.1% | setincidentflux_
||   2.6% |   103.6 |   3.4 |  3.2% | rtstrtsn_
||   1.4% |    58.1 |   2.9 |  4.8% | advancert_
||====================================================
|   8.0% |   320.6 |    -- |    -- | ETC
||----------------------------------------------------
||   7.7% |   310.6 |  51.4 | 14.2% | __cray_dcopy_HSW
|=====================================================
```

Optimized

```
Table 1:  Profile by Function

  Samp% |     Samp |  Imb. |  Imb. | Group
        |          |  Samp | Samp% |  Function=[MAX10]
        |          |       |       |   PE=HIDE
        |          |       |       |    Thread=HIDE

 100.0% | 3,651.9 |    -- |    -- | Total
|-----------------------------------------------------
|  54.7% | 1,998.8 |    -- |    -- | MPI
||----------------------------------------------------
||  37.2% | 1,360.0 | 444.0 | 24.7% | MPI_WAIT
||  12.3% |   447.6 | 668.4 | 60.1% | MPI_BARRIER
||   5.1% |   184.8 | 269.2 | 59.5% | MPI_ALLREDUCE
||====================================================
|  35.6% | 1,299.8 |    -- |    -- | USER
||----------------------------------------------------
||  15.9% |   581.5 |  44.5 |  7.1% | snswp3d_
||   4.4% |   161.0 |  60.0 | 27.3% | exchange_.LOOP@li.103
||   4.2% |   154.4 |   3.6 |  2.3% | snmoments_.LOOP@li.58
||   3.5% |   127.8 |  50.2 | 28.3% | setincidentflux_
||   3.0% |   110.6 |  86.4 | 44.1% | exchange_.LOOP@li.178
||   1.8% |    67.0 |   2.0 |  2.9% | rtstrtsn_.LOOP@li.51
||====================================================
|   9.0% |   328.6 |    -- |    -- | ETC
||----------------------------------------------------
||   8.6% |   314.5 |  39.5 | 11.2% | __cray_dcopy_HSW
|=====================================================
```

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 |   Local |  Remote |     Thread |  Memory |  Memory | Numanode
 Traffic |  Memory |  Memory |       Time | Traffic | Traffic |  PE=HIDE
  GBytes | Traffic | Traffic |            |  GBytes |       / |
         |  GBytes |  GBytes |            |   / Sec | Nominal |
         |         |         |            |         |    Peak |
|--------------------------------------------------------------------------
|   2,164 |   2,162 |    2.36 | 83.914890 |   25.79 |   33.6% | numanode.0
|  752.57 |  752.39 |    0.18 | 83.916253 |    8.97 |   11.7% | numanode.1
|==========================================================================
```

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

  Samp% |  Samp | Imb.  |  Imb. |  MEM_LOAD_UOPS_RETIRED |   RESOURCE_STALLS | Group
        |       | Samp  | Samp% |     :HIT_LFB:precise=2 |              :ANY |  Function=[MAX10]
        |       |       |       |                        |                   |   Source
        |       |       |       |                        |                   |    Line
        |       |       |       |                        |                   |     PE=HIDE

 100.0% | 862.5 |    -- |    -- | 89,509,042,630,218,624 | 1,722,469,802,124 | Total
|-----------------------------------------------------------------------------
|  62.4% | 537.9 |    -- |    -- | 56,224,626,618,220,104 |   991,877,756,035 | USER
||----------------------------------------------------------------------------
||  31.2% | 269.1 |    -- |    -- | 28,464,157,033,148,192 |   608,254,074,506 | snswp3d_
3|        |       |       |       |                        |                   |  snswp3d.F90
||||--------------------------------------------------------------------------
4|||    9.1% |  78.2 |   9.8 | 12.7% |   8,549,802,421,201,201 |   176,888,912,461 | line.92
4|||    1.2% |  10.0 |   6.0 | 42.9% |     879,609,303,122,411 |    22,751,971,753 | line.93
4|||    1.0% |   9.0 |   3.0 | 28.6% |     879,609,302,723,331 |    20,451,617,059 | line.95
4|||    1.1% |   9.9 |   7.1 | 47.9% |     879,609,302,703,468 |    22,293,446,105 | line.112
4|||    7.8% |  67.6 |  12.4 | 17.7% |   7,599,824,374,166,244 |   152,716,084,911 | line.128
4|||    1.4% |  12.2 |   2.8 | 21.0% |   1,337,006,139,902,300 |    27,724,424,302 | line.138
4|||    1.6% |  14.1 |   3.9 | 24.6% |   1,266,637,396,229,511 |    32,065,385,985 | line.206
4|||    4.6% |  39.2 |   8.8 | 20.8% |   4,151,755,908,199,040 |    88,460,785,924 | line.229
||||==========================================================================
||  15.3% | 132.2 |    -- |    -- | 14,179,301,951,169,120 |   150,304,652,642 | snmoments_
3|        |       |       |       |                        |                   |  snmoments.F90
4|  15.3% | 132.1 |   5.9 |  4.9% | 14,144,117,579,091,356 |   150,152,590,310 |   line.66
||   7.0% |  60.0 |    -- |    -- |  5,207,287,073,618,042 |   105,802,278,732 | exchange_
3|        |       |       |       |                        |                   |  exchange.F90
||||--------------------------------------------------------------------------
4|||    5.2% |  44.9 |   4.1 |  9.6% |   3,588,805,956,844,644 |    77,368,597,030 | line.115
4|||    1.8% |  15.1 |   2.9 | 18.3% |   1,618,481,116,773,398 |    28,433,681,701 | line.190
||||==========================================================================
```

OMG – ARRAY SYNTAX

```
  + F 2 3-------------<         do icface=1,ncfaces
    F 2 3
    F 2 3                         afpm(icface) = omega(1)*Z%A_fp(1,icface,c) + &
    F 2 3                                        omega(2)*Z%A_fp(2,icface,c) + &
    F 2 3                                        omega(3)*Z%A_fp(3,icface,c)
    F 2 3
    F 2 3                         icfp     = Z%Connect(1,icface,c)
    F 2 3                         ib       = Z%Connect(2,icface,c)
    F 2 3
    F 2 3                         if ( afpm(icface) >= zero ) then
    F 2 3                           sumArea = sumArea + afpm(icface)
    F 2 3                         else
    F 2 3                           if (icfp == 0) then
    F 2 3 A----------<>               psifp(:,icface) = psib(:,ib)
    F 2 3                           else
    F 2 3 A----------<>               psifp(:,icface) = psic(:,icfp)
    F 2 3                           endif
    F 2 3
    F 2 3 Vr4--------<>             src(:,c)  = src(:,c) - afpm(icface)*psifp(:,icface)
    F 2 3                         endif
    F 2 3-------------->        enddo
```

ARRAY SYNTAX: Good for the Cyber 205 and CM5, Not Good for Cache Based Systems

A

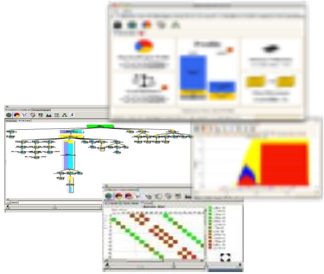
Whacked

```
F 2
F 2 r6------------<          do icface=1,ncfaces
F 2 r6
F 2 r6                         afpm(icface)  = omega(1)*Z%A_fp
F 2 r6                                         omega(2)*Z%A_fp
F 2 r6                                         omega(3)*Z%A_fp
F 2 r6
F 2 r6                         icfp     = Z%Connect(1,icface,c)
F 2 r6                         ib       = Z%Connect(2,icface,c)
F 2 r6
F 2 r6                         if ( afpm(icface) >= zero ) then
F 2 r6                           sumArea = sumArea + afpm(icface)
F 2 r6                         else
F 2 r6 Vr2--------<             do ig = 1, Groups
F 2 r6 Vr2                       if (icfp == 0) then
F 2 r6 Vr2                         psifp(ig,icface) =
F 2 r6 Vr2                       else
F 2 r6 Vr2                         psifp(ig,icface) =
F 2 r6 Vr2                       endif
F 2 r6 Vr2
F 2 r6 Vr2                       src(ig,c)   = src(ig,c
F 2 r6 Vr2-------->             enddo
F 2 r6                         endif
F 2 r6------------>          enddo
```

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

```
 Samp% |        Samp |       Imb. |    Imb. |Group
       |             |       Samp |   Samp% | Function
       |             |            |         |  PE=HIDE

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

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

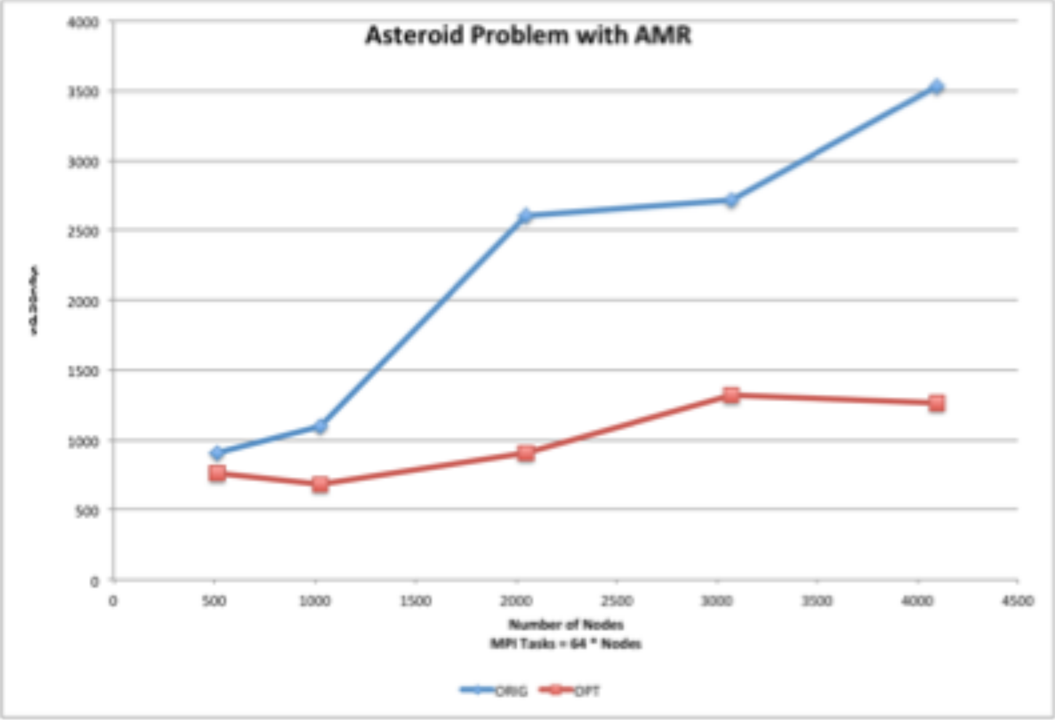
How to Reduce Performance Data

- 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**

Leslie3D

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

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

Three Moles

Call Tree with Loops

They are all called
from high level loops

```
 Time% |         Time |          Calls | Calltree
       |              |                |  PE=HIDE

 100.0% | 242.649260 |             -- | Total

|-----------------------------------------------------------
| 100.0% | 242.649223 |            2.0 | les3d_
|  99.2% | 240.654228 |             -- |  les3d_.LOOP.3.li.216
|||----------------------------------------------------------
3||  97.9% | 237.444979 |            -- | les3d_.LOOP.4.li.272
||||---------------------------------------------------------
4|||  29.1% |  70.519335 |       4,000.0 | fluxk_
|||||--------------------------------------------------------
5||||  17.7% |  42.941276 |            -- | fluxk_.LOOP.1.li.28
6||||        |            |               |  fluxk_.LOOP.2.li.29
7||||  17.7% |  42.941276 |   9,408,000.0 |   visck_
5||||   6.6% |  16.130702 |       4,000.0 | fluxk_(exclusive)
5||||   4.7% |  11.447357 |       4,000.0 | extrapk_
|||||========================================================
4|||  28.1% |  68.288330 |       4,000.0 | fluxj_
|||||--------------------------------------------------------
5||||  18.3% |  44.519098 |            -- | fluxj_.LOOP.1.li.28
6||||        |            |               |  fluxj_.LOOP.2.li.29
7||||  18.3% |  44.519098 |   9,408,000.0 |   viscj_
5||||   5.8% |  14.084979 |       4,000.0 | fluxj_(exclusive)
5||||   4.0% |   9.684254 |       4,000.0 | extrapj_
============
```

```
          |         Time |          Calls | Calltree
          |              |                |  PE=HIDE

4|||  27.2% |  65.960873 |       4,000.0 | fluxi_
|||||--------------------------------------------------------
5||||  15.0% |  36.464645 |            -- | fluxi_.LOOP.1.li.21
6||||        |            |               |  fluxi_.LOOP.2.li.22
7||||  15.0% |  36.464645 |   9,216,000.0 |   visci_
5||||   6.3% |  15.328236 |       4,000.0 | extrapi_
||||||-------------------------------------------------------
6|||||   3.5% |   8.424708 |       4,000.0 | extrapi_(exclusive)
6|||||   2.8% |   6.903528 |            -- | extrapi_.LOOP.1.li.128
7|||||        |            |               |  extrapi_.LOOP.2.li.129
8|||||   2.8% |   6.903528 |  10,609,000.0 |   exi4_
||||||=======================================================
5||||   5.8% |  14.167993 |       4,000.0 | fluxi_(exclusive)
|||||========================================================
4|||   9.6% |  23.356145 |       4,000.0 | parallel_
|||||--------------------------------------------------------
5||||   7.2% |  17.457089 |            -- | parallel_.LOOP.1.li.16
6||||   7.2% |  17.457089 |      20,000.0 |  mpicx_
5||||   2.4% |   5.843909 |      20,000.0 | ghost_
|||||========================================================
4|||   3.6% |   8.788456 |       4,000.0 | update_
||||=========================================================
3||   1.1% |   2.557943 |         401.0 | tmstep_
|============================================================
```

Do Loop Table

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

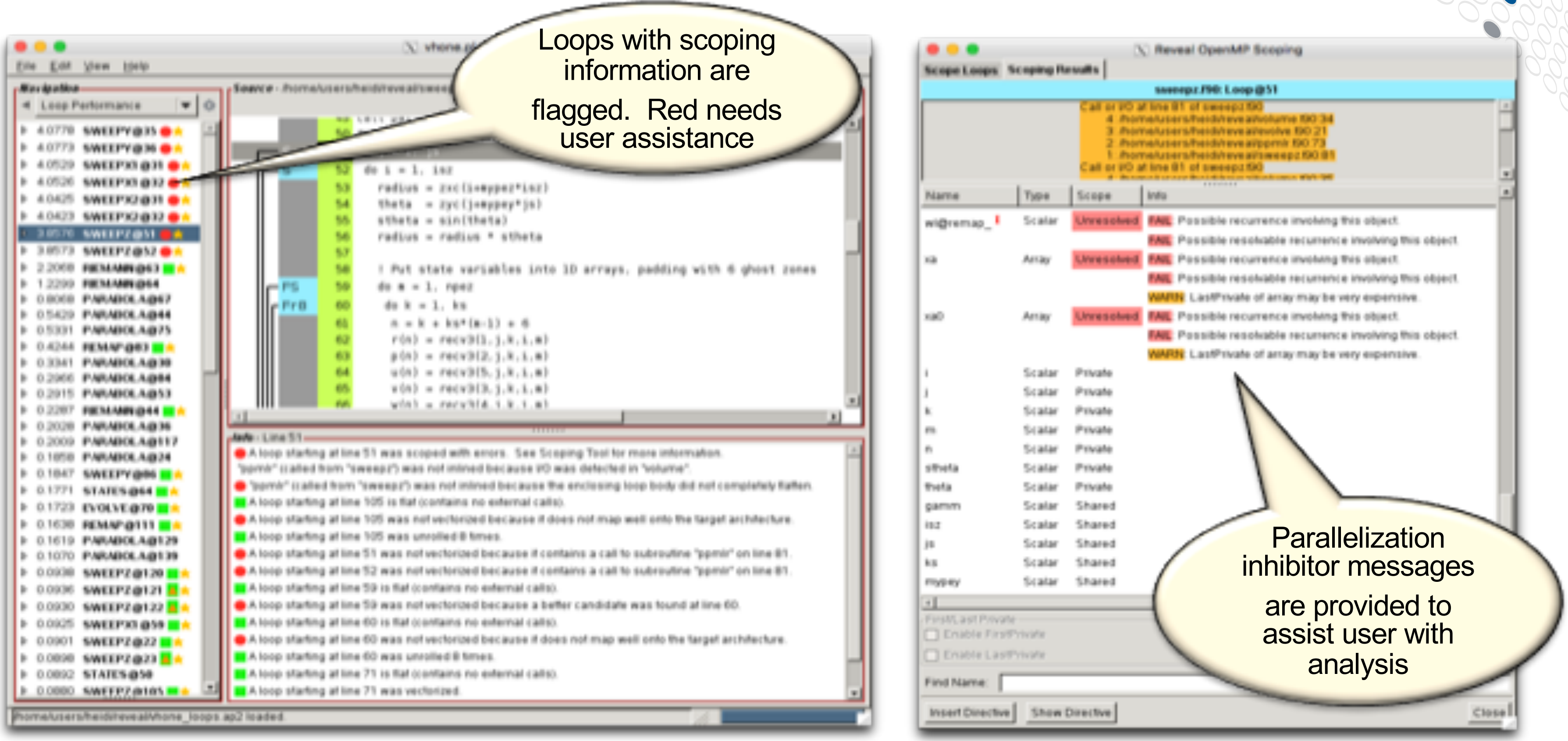
Loop Incl Time%	Loop Incl Time	Loop Hit	Loop Trips Avg	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
99.2%	246.816563	1	2,000.0	2,000	2,000	les3d_.LOOP.3.li.216
97.9%	243.603685	2,000	2.0	2	2	les3d_.LOOP.4.li.272
24.3%	60.541874	4,000	48.0	48	48	fluxk_.LOOP.1.li.28
24.1%	60.069820	4,000	48.0	48	48	fluxj_.LOOP.1.li.28
22.9%	57.005766	192,000	49.0	49	49	fluxj_.LOOP.2.li.29
22.4%	55.615468	192,000	49.0	49	49	fluxk_.LOOP.2.li.29
20.9%	52.076589	4,000	48.0	48	48	fluxi_.LOOP.1.li.21
20.9%	52.070251	192,000	48.0	48	48	fluxi_.LOOP.2.li.22
17.1%	42.629610	9,408,000	48.0	48	48	viscj_.LOOP.1.li.340
16.5%	40.972257	9,408,000	48.0	48	48	visck_.LOOP.1.li.344
13.9%	34.612953	9,216,000	49.0	49	49	visci_.LOOP.1.li.782
7.0%	17.507682	4,002	5.0	5	5	parallel_.LOOP.1.li.16
5.0%	12.451170	4,000	51.5	51	52	extrapi_.LOOP.1.li.128
5.0%	12.448703	206,000	51.5	51	52	extrapi_.LOOP.2.li.129
4.6%	11.434859	4,000	49.0	49	49	extrapk_.LOOP.1.li.138
4.6%	11.432312	196,000	51.5	51	52	extrapk_.LOOP.2.li.140
3.9%	9.672461	4,000	51.5	51	52	extrapj_.LOOP.1.li.135
3.9%	9.668853	206,000	49.0	49	49	extrapj_.LOOP.2.li.136

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



CCE Listing: FLUXI.lst (-h list=a)

```
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
29.    M m 3 fV-------<>             QS(0:ICMAX) = UAV(0:ICMAX,J,K) * SIX(0:ICMAX,J,K) +
30.    M m 3                  >                    VAV(0:ICMAX,J,K) * SIY(0:ICMAX,J,K) +
31.    M m 3                  >                    WAV(0:ICMAX,J,K) * SIZ(0:ICMAX,J,K)
32.    M m 3
```

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.

FLUXI.lst (1 of 2)

```
   +                            CALL EXTRAPI ( FSI )
                          ! Directive inserted by Cray Reveal.  May be incomplete.
     M---------------< !$OMP  parallel do default(none)
     M                 !$OMP&   private (i,j,k,l,qs,qsp,qspi)
     M                 !$OMP&   shared  (dq,dtv,iadd,icmax,jcmax,kcmax,pav,qav,six,siy,siz,u,
     M                 !$OMP&           uav,v,vav,w,wav)
     M                 !$OMP&   firstprivate(fsi)
   + M m-------------<          DO K = 1, KCMAX
   + M m 3-----------<             DO J = 1, JCMAX
     M m 3
     M m 3 fV-------<>                QS(0:ICMAX) = UAV(0:ICMAX,J,K) * SIX(0:ICMAX,J,
     M m 3                  >                       VAV(0:ICMAX,J,K) * SIY(0:ICMAX,J,K) +
     M m 3                  >                       WAV(0:ICMAX,J,K) * SIZ(0:ICMAX,J,K)
     M m 3
     M m 3                            IF ( NSCHEME .EQ. 2 ) THEN
     M m 3 D---------<                   DO I = 0, ICMAX
     M m 3 D                                L = I + 1 - IADD
     M m 3 D                                QSP = U(L,J,K) * SIX(I,J,K)
     M m 3 D                >                     V(L,J,K) * SIY(I,J,K) +
     M m 3 D                >                     W(L,J,K) * SIZ(I,J,K)
     M m 3 D                                QSPI = (QSP - QS(I)) * DBLE(1 - 2 *
     M m 3 D                                IF ( QSPI .GT. 0.0D0 ) QS(I) = 0.5D0 *          + QSP)
     M m 3 D--------->                   ENDDO
     M m 3                            ENDIF
     M m 3
```

Array syntax – vectorized and fused with other loops

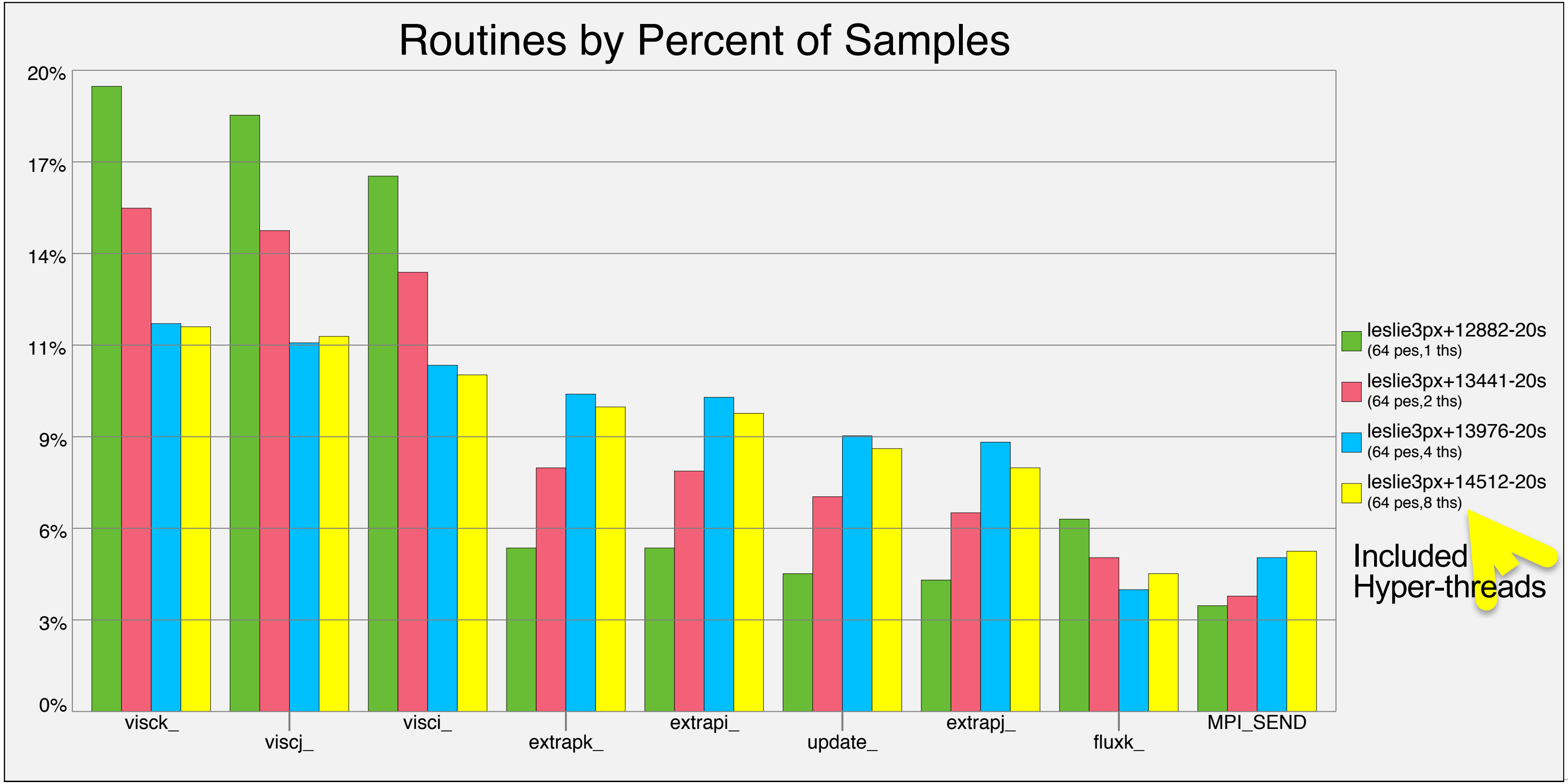
Compiler knows NSCHEME is not equal to 2

FLUXI.lst (2 of 2)

```
44.       M m 3 f--------<>          FSI(0:ICMAX,1) =  QAV(0:ICMAX,J,K,1) * QS(0:ICMAX)
45.       M m 3 f--------<>          FSI(0:ICMAX,2) =  QAV(0:ICMAX,J,K,2) * QS(0:ICMAX) +
46.       M m 3                >                             PAV(0:ICMAX,J,K) * SIX(0:ICMAX,J,K)
47.       M m 3 f--------<>          FSI(0:ICMAX,3) =  QAV(0:ICMAX,J,K,3) * QS(0:ICMAX) +
48.       M m 3                >                             PAV(0:ICMAX,J,K) * SIY(0:ICMAX,J,K)
49.       M m 3 fVr2-----<>          FSI(0:ICMAX,4) =  QAV(0:ICMAX,J,K,4) * QS(0:ICMAX) +
50.       M m 3                >                             PAV(0:ICMAX,J,K) * SIZ(0:ICMAX,J,K)
51.       M m 3 f--------<>          FSI(0:ICMAX,5) = (QAV(0:ICMAX,J,K,5) + PAV(0:ICMAX,J,K)) *
52.       M m 3                >                                          QS(0:ICMAX)
53.       M m 3
54.       M m 3                      IF ( ISGSK .EQ. 1 ) THEN
55.       M m 3                         FSI(0:ICMAX,7) = QAV(0:ICMAX,J,K,7)
56.       M m 3                      ENDIF
57.       M m 3
58.       M m 3                      IF ( ICHEM .GT. 0 ) THEN
59.       M m 3 D---------<             DO L = 8, 7 + NSPECI
60.       M m 3 D                          FSI(0:ICMAX,L) = QAV(0:ICMAX,J,K,L) * QS(0:ICMAX)
61.       M m 3 D--------->             ENDDO
62.       M m 3                      ENDIF
63.       M m 3
64.     + M m 3                      IF ( VISCOUS ) CALL VISCI ( 0, ICMAX, J, K, FSI )
65.       M m 3
```

VISCI is called
from OpenMP loop

64 MPI Tasks on 8 Nodes



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)

A

```
<          DUDX = T11(I,J,K,3) * DUDXI
<          DVDX = T11(I,J,K,3) * DVDXI
<          DWDX = T11(I,J,K,3) * DWDXI
<          DTDX = T11(I,J,K,3) * DTDXI
<
<          DUDY = T21(I,J,K,3) * DUDXI
<          DVDY = T21(I,J,K,3) * DVDXI
<          DWDY = T21(I,J,K,3) * DWDXI
<          DTDY = T21(I,J,K,3) * DTDXI
<
<          DUDZ = T31(I,J,K,3) * DUDXI
<          DVDZ = T31(I,J,K,3) * DVDXI
<          DWDZ = T31(I,J,K,3) * DWDXI
<          DTDZ = T31(I,J,K,3) * DTDXI
<
---
>                 ,K,3) * DUDXI
>                 ,K,3) * DVDXI
>                 ,K,3) * DWDXI
>                 ,K,3) * DTDXI
>
>          DUDY(I) = T21(I,J,K,3) * DUDXI
>          D   Y   )   T21(I,J,K,3) * DVDXI
>          DW
>          DT    (I
>
>          DUDZ(I) = T31(I,J,K,3) * DUDXI
>          DVDZ(I) = T31(I,J,K,3) * DVDXI
>          DWDZ(I) = T31(I,J,K,3) * DWDXI
>          DTDZ(I) = T31(I,J,K,3) * DTDXI
```

Whacked

Reveal Scoping of EXTRAP Routines

```

                                  ! Directive inserted by Cray Reveal.  May be incomplete.
        M---------------< !$OMP  parallel do default(none)
        M                 !$OMP&    private (rwrk,j,k,kg)
        M                 !$OMP&    shared  (i1,i2,j1,j2,kcmax,kmax,kstart,pav,qav,tav,
        M                 !$OMP&             uav,vav,wav)
      + M m-------------<           DO K = 0, KCMAX
        M m                            KG = K + KSTART - 1
      + M m 3-----------<              DO J = J1, J2
        M m 3
        M m 3                             IF ( NSCHEME .EQ. 2 .OR. (.NOT. KPERIODIC .AND.
        M m 3                  >                       (KG .EQ. 0 .OR. KG .EQ. KMAX-1))) THEN
      + M m 3                                CALL EXK2 ( I1, I2, J, K )
        M m 3                             ELSE
      + M m 3                                CALL EXK4 ( I1, I2, J, K,
        M m 3                  >                         RWRK(I1,1), RWRK(I1,2), RWRK(I1,3) )
        M m 3                             ENDIF
```

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

Reveal Scoping of UPDATE Routine

```
                      ! Directive inserted by Cray Reveal.  May be incomplete.
        M--------< !$OMP  parallel do default(none)
        M          !$OMP&   private (i,j,k,ke)
        M          !$OMP&   shared  (dq,icmax,jcmax,kcmax,m,n,p,q,t,u,v,w)
      + M m------<       DO K = 1, KCMAX
      + M m 3----<          DO J = 1, JCMAX
        M m 3 V--<             DO I = 1, ICMAX
        M m 3 V
        M m 3 V                   IF ( N .EQ. 1 ) THEN
        M m 3 V                      Q(I,J,K,1,M) = Q(I,J,K,1,N) + DQ(I,J,K,1)
        M m 3 V                      Q(I,J,K,2,M) = Q(I,J,K,2,N) + DQ(I,J,K,2)
        M m 3 V                      Q(I,J,K,3,M) = Q(I,J,K,3,N) + DQ(I,J,K,3)
        M m 3 V                      Q(I,J,K,4,M) = Q(I,J,K,4,N) + DQ(I,J,K,4)
        M m 3 V                      Q(I,J,K,5,M) = Q(I,J,K,5,N) + DQ(I,J,K,5)
        M m 3 V                   ELSE
        M m 3 V                      Q(I,J,K,1,M) = 0.5D+00 * (Q(I,J,K,1,M) +
        M m 3 V      >                                Q(I,J,K,1,N) + DQ(I,J,K,1))
        M m 3 V                      Q(I,J,K,2,M) = 0.5D+00 * (Q(I,J,K,2,M) +
        M m 3 V      >                                Q(I,J,K,2,N) + DQ(I,J,K,2))
        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

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

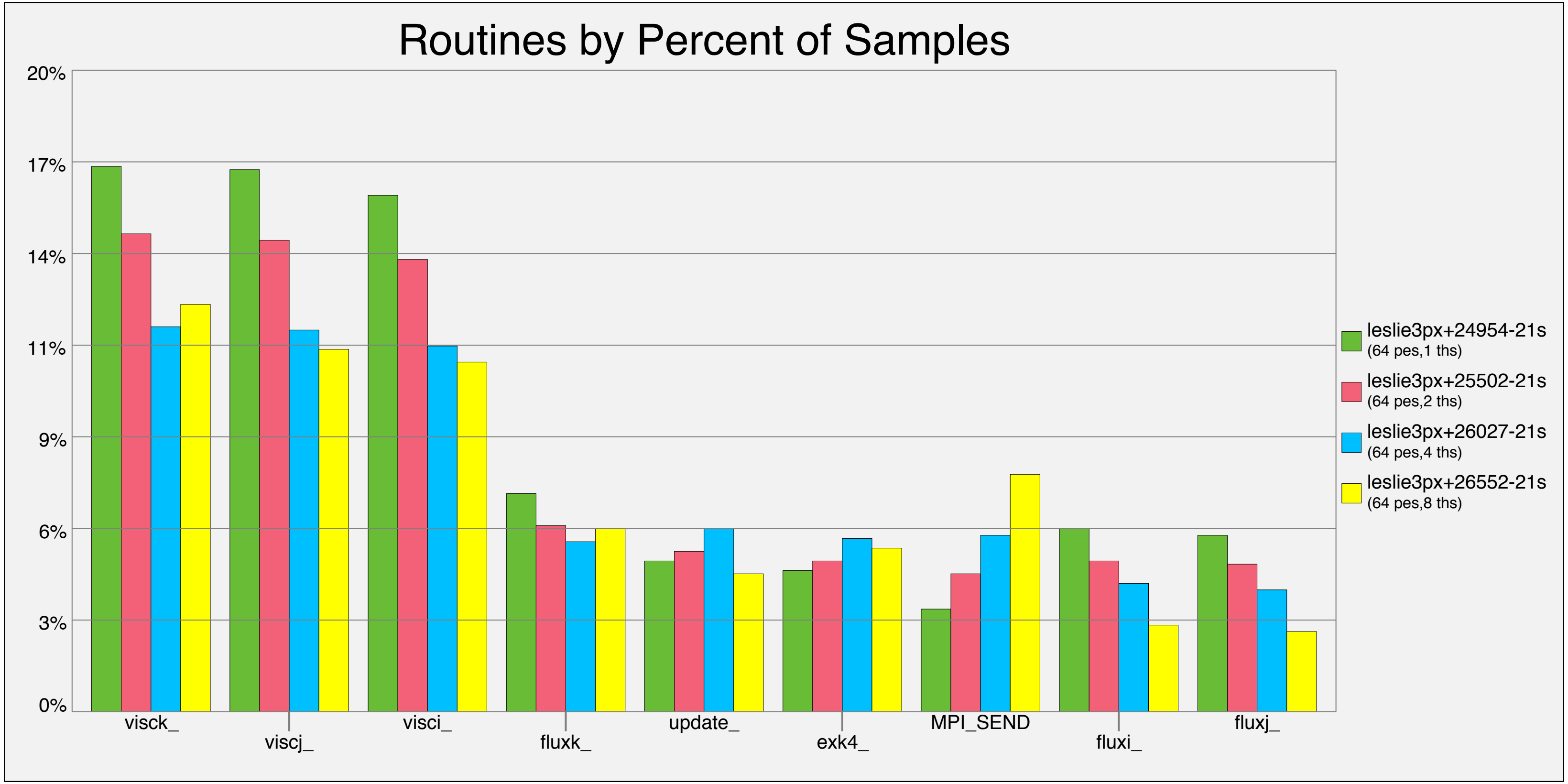
Another Modification to UPDATE

A

Whacked

```
15c18,22
<             Q(I,J,K,1:5,M) = Q(I,J,K,1:5,N) +
---
>             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,5)
17,18c24,33
<             Q(I,J,K,1:5,M) = 0.5D+00 * (Q(I,J,K,1:5,N
<      >                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) +
>      >                Q(I,J,K,1,N) + DQ(I,J,K,1))
>             Q(I,J,K,2,M) = 0.5D+00 * (Q(I,J,K,2,M) +
>      >                Q(I,J,K,2,N) + DQ(I,J,K,2))
>             Q(I,J,K,3,M) = 0.5D+00 * (Q(I,J,K,3,M) +
>      >                Q(I,J,K,3,N) + DQ(I,J,K,3))
>             Q(I,J,K,4,M) = 0.5D+00 * (Q(I,J,K,4,M) +
>      >                Q(I,J,K,4,N) + DQ(I,J,K,4))
>             Q(I,J,K,5,M) = 0.5D+00 * (Q(I,J,K,5,M) +
>      >                Q(I,J,K,5,N) + DQ(I,J,K,5))
```

Optimization Results for 64 MPI Tasks on 8 Nodes



Final Sampling Profile

```
Table 1:  Profile by Function

  Samp% |     Samp |  Imb. |  Imb. | Group
        |          |  Samp | Samp% |  Function=[MAX10]
        |          |       |       |   PE=HIDE
        |          |       |       |    Thread=HIDE

 100.0% | 9,268.0 |    -- |    -- | Total
|-----------------------------------------------------
|  86.4% | 8,003.3 |    -- |    -- | USER
||----------------------------------------------------
||  12.0% | 1,114.2 |  84.8 |  7.2% | visck_
||  11.9% | 1,106.2 |  58.8 |  5.1% | viscj_
||  11.4% | 1,054.9 |  49.1 |  4.5% | visci_
||   5.7% |   531.1 |  46.9 |  8.2% | update_
||   5.4% |   499.0 |  45.0 |  8.4% | exk4_
||   5.3% |   490.8 |  65.2 | 11.9% | fluxk_
||   5.2% |   484.0 |  41.0 |  7.9% | extrapi_
||   4.6% |   424.7 |  35.3 |  7.8% | exi4_
||   4.1% |   382.4 |  50.6 | 11.9% | exj4_
||====================================================
|  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!

Pure MPI Profile

ORIGINAL

Table 1: Profile by Function

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

Table 1: Profile by Function

Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function PE=HIDE
100.0%	9,206.9	--	--	Total
85.8%	7,899.0	--	--	USER
18.1%	1,662.2	90.8	5.2%	fluxj_
17.4%	1,602.9	225.1	12.4%	fluxk_
16.8%	1,542.4	109.6	6.7%	fluxi_
9.3%	855.7	72.3	7.8%	extrapi_
7.1%	652.3	53.7	7.6%	extrapk_
6.5%	602.6	62.4	9.4%	extrapj_
5.7%	528.0	60.0	10.2%	update_
1.4%	130.4	43.6	25.1%	mpicx_
1.3%	123.0	70.0	36.4%	parallel_
1.1%	100.7	81.3	44.8%	ghost_
13.7%	1,263.9	--	--	MPI
8.1%	749.8	270.2	26.6%	MPI_SEND
2.3%	209.7	101.3	32.7%	MPI_ALLREDUCE
1.5%	136.1	269.9	66.7%	MPI_WAIT
1.5%	135.8	119.2	46.9%	MPI_REDUCE

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

  Samp% |  Samp | Imb. |  Imb. |  MEM_LOAD_UOPS_RETIRED | RESOURCE_STALLS | Group
        |       | Samp | Samp% |     :HIT_LFB:precise=2 |            :ANY |  Function=[MAX10]
        |       |      |       |                        |                 |   Source
        |       |      |       |                        |                 |    Line
        |       |      |       |                        |                 |     PE=HIDE

 100.0% | 359.1 |   -- |    -- | 37,453,764,107,885,936 | 586,696,060,451 | Total
|-----------------------------------------------------------------------------
|  90.3% | 324.2 |   -- |    -- | 33,675,842,152,965,068 | 538,408,160,657 | USER
||----------------------------------------------------------------------------
||  16.6% |  59.7 |   -- |    -- |  6,020,925,677,307,978 |  98,381,410,309 | fluxk_
3|        |       |      |       |                        |                 |  fluxk.f
||||--------------------------------------------------------------------------
4|||    3.7% |  13.4 |  9.6 | 42.2% |  1,411,772,930,846,618 |  22,424,492,033 | line.36
4|||    1.1% |   4.0 |  7.0 | 64.9% |    369,435,907,219,731 |   6,575,931,964 | line.54
4|||    9.1% |  32.6 | 37.4 | 54.3% |  3,294,136,838,755,624 |  53,681,888,808 | line.76
||||==========================================================================
||  14.7% |  52.8 |   -- |    -- |  5,770,237,024,852,115 |  88,570,460,626 | visci_
3|        |       |      |       |                        |                 |  fluxi.f
||||--------------------------------------------------------------------------
4|||    1.1% |   4.1 |  4.9 | 55.7% |    452,998,790,837,684 |   6,904,792,416 | line.807
4|||    2.1% |   7.5 |  9.5 | 57.0% |    818,036,651,385,400 |  12,609,245,898 | line.822
||||==========================================================================
||  13.8% |  49.7 |   -- |    -- |  5,246,869,490,559,427 |  82,777,122,645 | fluxj_
3|        |       |      |       |                        |                 |  fluxj.f
||||--------------------------------------------------------------------------
4|||    2.8% |  10.2 | 18.8 | 66.0% |  1,077,521,395,774,529 |  16,750,162,451 | line.36
4|||    8.2% |  29.3 | 40.7 | 59.0% |  3,118,214,977,991,204 |  48,910,720,877 | line.76
||||==========================================================================
```

Code in FLUXK Responsible for Memory Bandwidth Utilization

```
    M m
  + M m ib----------<          DO K = 1, KCMAX
  + M m ib i--------<             DO L = 1, 5
    M m ib i Vbr4--<>                DQ(1:IND,J,K,L) = DQ(1:IND
    M m ib i               >              DTV(1:IND,J,K) * (FSK            ,K-1,L))
    M m ib i-------->             ENDDO
    M m ib
    M m ib                        IF (ISGSK .EQ. 1) THEN
    M m ib                           DQ(1:IND,J,K,7) = DQ(1:IND,J,K,7) -
    M m ib                 >              DTV(1:IND,J,K) * (FSK(1:IND,K,7) - FSK(1:IND,K-1,7))
    M m ib                        ENDIF
    M m ib
    M m ib                        IF ( ICHEM .GT. 0 ) THEN
    M m ib D--------<                DO L = 8, 7 + NSPECI
    M m ib D                            DQ(1:IND,J,K,L) = DQ(1:IND,J,K,L) -
    M m ib D               >              DTV(1:IND,J,K) * (FSK(1:IND,K,L) - FSK(1:IND,K-1,L))
    M m ib D-------->                ENDDO
    M m ib                        ENDIF
    M m ib
    M m ib---------->          ENDDO
```

Array Syntax and Automatic Blocking by the Compiler

A

y 33% -
ere is a lot
x

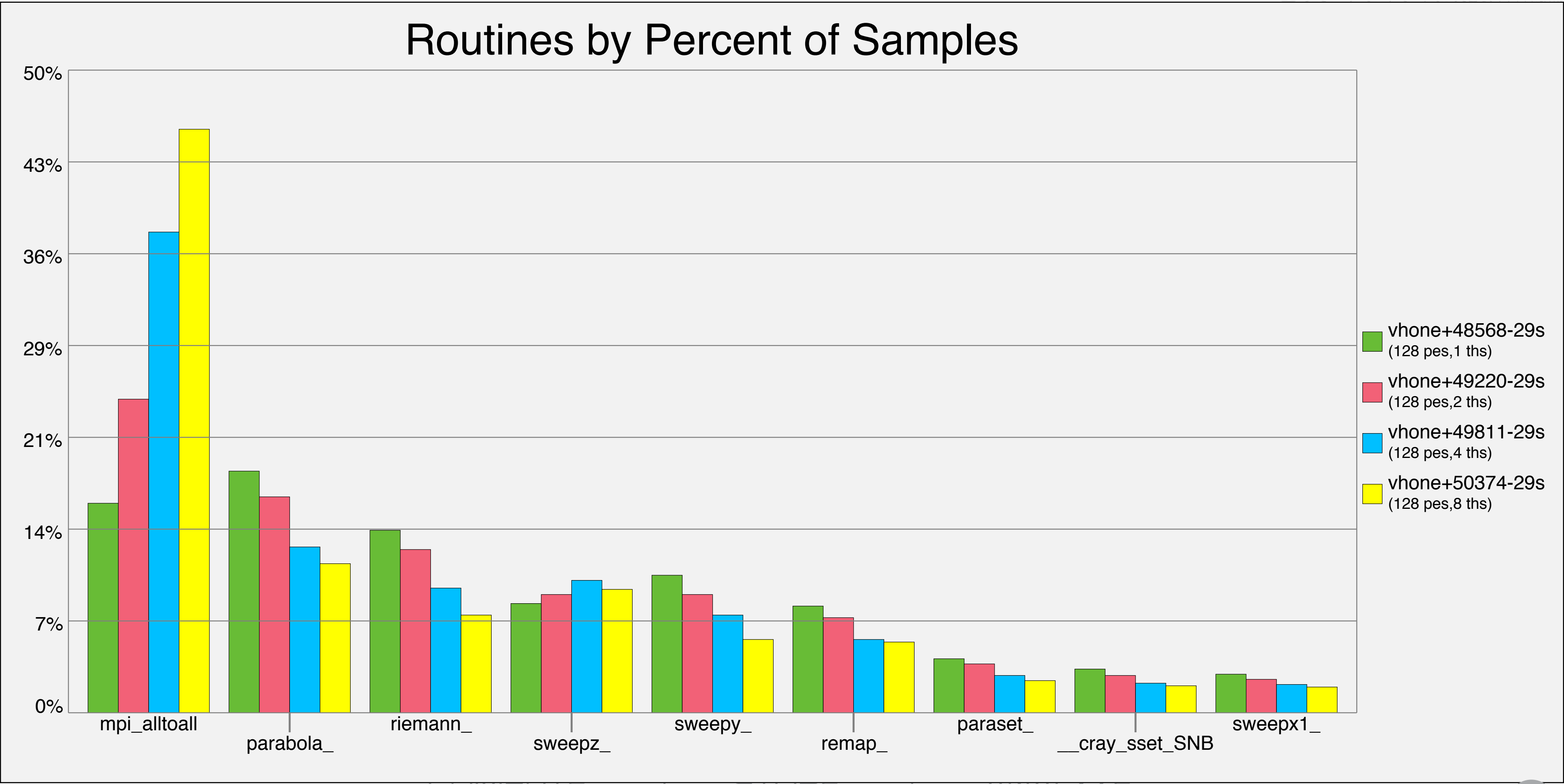
```
    M m               !dir$ noblocking
  + M m 3--------<          DO K = 1, KCMAX
    M m 3 V------<           DO I = 1, IND
    M m 3 V           !dir$ unroll(5)
    M m 3 V w----<              DO L = 1, 5
    M m 3 V w                      DQ(I,J,K,L) = DQ(I,J,K,L) -
    M m 3 V w           >               DTV(I,J,K) * (FSK(I,K,L)
    M m 3 V w---->              ENDDO
    M m 3 V
    M m 3 V                     IF (ISGSK .EQ. 1) THEN
    M m 3 V                        DQ(I,J,K,7) = DQ(I,J,K,7) -
    M m 3 V             >               DTV(I,J,K) * (FSK(I,K,7) - F
    M m 3 V                     ENDIF
    M m 3 V
    M m 3 V                     IF ( ICHEM .GT. 0 ) THEN
    M m 3 V D----<                 DO L = 8, 7 + NSPECI
    M m 3 V D                         DQ(I,J,K,L) = DQ(I,J,K,L) -
    M m 3 V D           >               DTV(I,J,K) * (FSK(I,K,L) - FSK(I,K-1,L))
    M m 3 V D---->                 ENDDO
    M m 3 V                     ENDIF
    M m 3 V
    M m 3 V------>           ENDDO
    M m 3-------->          ENDDO
    M m--------->>        ENDDO
```

Whacked

VH1

VH1 Already Parallelized Using Reveal

- Need to investigate additional optimization
- Vectorization and Memory utilization optimization



Routines by Percent of Samples
50%
43%
36%
29%
21%
14%
7%
0%
mpi_alltoall
parabola_
riemann_
sweepz_
sweepy_
remap_
paraset_
__cray_sset_SNB
sweepx1_
vhone+48568-29s
(128 pes,1 ths)
vhone+49220-29s
(128 pes,2 ths)
vhone+49811-29s
(128 pes,4 ths)
vhone+50374-29s
(128 pes,8 ths)

Profile for 1 Thread and 4 Threads

```
Table 1:  Profile by Function      1 Thread

  Samp% |      Samp | Imb. |  Imb. | Group
        |           | Samp | Samp% |  Function=[MAX10]
        |           |      |       |   PE=HIDE

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

```
Table 1:  Profile by Function    4 Threads

  Samp% |      Samp | Imb. |  Imb. | Group
        |           | Samp | Samp% |  Function=[MAX10]
        |           |      |       |   PE=HIDE
        |           |      |       |    Thread=HIDE

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

Do Loop Table

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

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

Call Tree with Loops

```
Table 1:  Function Calltree View

  Time% |        Time |        Calls | Calltree
        |             |              |  PE=HIDE

 100.0% | 623.115520 |           -- | Total
|----------------------------------------------------------
| 100.0% | 623.115490 |          2.0 | vhone_
| 100.0% | 622.866674 |           -- |  vhone_.LOOP.2.li.205
|||--------------------------------------------------------
3||   37.5% | 233.441869 |         50.0 | sweepz_
||||-------------------------------------------------------
4|||   23.6% | 147.324792 |           -- | sweepz_.LOOP.05.li.53
5|||         |            |              |  sweepz_.LOOP.06.li.54
6|||         |            |              |   ppmlr_
|||||||----------------------------------------------------
7||||||    9.5% |  58.970451 |    819,200.0 | remap_
||||||||---------------------------------------------------
8|||||||    5.8% |  35.940023 | 4,915,200.0 | parabola_
8|||||||    2.9% |  18.356531 |    819,200.0 | remap_(exclusive)
||||||||===================================================
7||||||    7.7% |  47.983811 |    819,200.0 | riemann_
7||||||    2.9% |  17.979033 | 2,457,600.0 | parabola_
7||||||    1.7% |  10.292367 |    819,200.0 | evolve_
|||||||====================================================
4|||   13.8% |   86.117076 |         50.0 | sweepz_(exclusive)
||||=======================================================
```

Parallized
Loop

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)*d
48.  V          al(n+1) = ar(n)
49.  V---->   enddo
53.  fV---<   do n = nmin, nmax
54.  fV         onemfl= 1.0 - flat(n)
55.  fV         ar(n) = flat(n) * a(n) + onemfl * ar(n)
56.  fV         al(n) = flat(n) * a(n) + onemfl * al(n)
57.  fV--->   enddo
67.  f----< do n = nmin, nmax
68.  f        deltaa(n) = ar(n) - al(n)
69.  f        a6(n)     = 6. * (a(n) - .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
74.
75.  Vr2--< do n = nmin, nmax
76.  Vr2      if(scrch1(n) <= 0.0) then
77.  Vr2        ar(n) = a(n)
78.  Vr2        al(n) = a(n)
79.  Vr2      endif
80.  Vr2      if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
81.  Vr2      if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
82.  Vr2--> enddo
83.
84.  Vr2--< do n = nmin, nmax
85.  Vr2      deltaa(n)= ar(n) - al(n)
86.  Vr2      a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
87.  Vr2--> enddo
```

PARABOLA

```
4|||   0.1% |   3.4 |   5.6 |  62.0% |line.44
4|||   2.5% |  88.5 |  27.5 |  23.8% |line.45
4|||   0.2% |   7.1 |   8.9 |  56.0% |line.48
4|||   0.0% |   0.1 |   1.9 |  94.9% |line.49
4|||   0.3% |   9.6 |  10.4 |  52.3% |line.53
4|||   0.3% |  10.1 |  10.9 |  51.9% |line.54
        .9% |  31.3 |  14.7 |  32.0% |line.55
        .2% |   7.3 |  11.7 |  61.6% |line.56
        .0% |   0.2 |   1.8 |  90.6% |line.57
        .4% |  13.6 |  10.4 |  43.5% |line.68
        .8% |  27.2 |  13.8 |  33.9% |line.69
        .4% |  15.4 |  11.6 |  43.2% |line.70
        .4% |  13.4 |  10.6 |  44.3% |line.71
        .4% |  14.0 |  12.0 |  46.2% |line.72
4|||   0.1% |   4.8 |   8.2 |  63.5% |line.75
4|||   0.3% |  10.2 |   9.8 |  49.1% |line.76
4|||   0.6% |  19.4 |  14.6 |  43.1% |line.77
4|||   0.5% |  17.9 |  11.1 |  38.4% |line.78
4|||   1.2% |  40.6 |  16.4 |  28.8% |line.80
4|||   1.8% |  63.9 |  24.1 |  27.5% |line.81
4|||   0.0% |   0.2 |   2.8 |  92.7% |line.82
4|||   0.1% |   4.3 |   7.7 |  64.4% |line.84
4|||   0.5% |  16.0 |  12.0 |  43.2% |line.85
4|||   0.6% |  21.3 |  13.7 |  39.4% |line.86
4|||   0.0% |   0.0 |   1.0 |  95.7% |line.87
```

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)
48.   V         al(n+1) = ar(n)
49.   V---->  enddo
53.   fV---<  do n = nmin, nmax
54.   fV        onemfl= 1.0 - flat(n)
55.   fV        ar(n) = flat(n) * a(n) + onemfl * ar(n)
56.   fV        al(n) = flat(n) * a(n) + onemfl * al(n)
57.   fV--->  enddo
67.   f----< do n = nmin, nmax
68.   f        deltaa(n) = ar(n) - al(n)
69.   f        a6(n)     = 6. * (a(n) - .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
74.
75.   Vr2--< do n = nmin, nmax
76.   Vr2      if(scrch1(n) <= 0.0) then
77.   Vr2        ar(n) = a(n)
78.   Vr2        al(n) = a(n)
79.   Vr2      endif
80.   Vr2      if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
81.   Vr2      if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
82.   Vr2--> enddo
83.
84.   Vr2--< do n = nmin, nmax
85.   Vr2      deltaa(n)= ar(n) - al(n)
86.   Vr2      a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
87.   Vr2--> enddo
```

Remove unnecessary arrays and fuse loops

```
52.   Vr2--<   do n = nmin, nmax
53.   Vr2        onemfl= 1.0 - flat(n)
54.   Vr2        ar(n) = flat(n) * a(n) + onemfl * ar(n)
55.   Vr2        al(n) = flat(n) * a(n) + onemfl * al(n)
56.   Vr2      deltaa(n) = ar(n) - al(n)
57.   Vr2      a6(n)     = 6. * (a(n) - .5 * (al(n) + ar(n)))
58.   Vr2      scrch1s = (ar(n) - a(n)) * (a(n)-al(n))
59.   Vr2      scrch2s = deltaa(n) * deltaa(n)
60.   Vr2      scrch3s = deltaa(n) * a6(n)
61.   Vr2      if(scrch1s <= 0.0) then
62.   Vr2        ar(n) = a(n)
63.   Vr2        al(n) = a(n)
64.   Vr2      endif
65.   Vr2      if(scrch2s < +scrch3s) al(n) = 3. * a(n) - 2. * ar(n)
66.   Vr2      if(scrch2s < -scrch3s) ar(n) = 3. * a(n) - 2. * al(n)
67.   Vr2      deltaa(n)= ar(n) - al(n)
68.   Vr2      a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
69.   Vr2--> enddo
```

If it Doesn't Vectorize – Fix It

```
||||===========================================
||  10.8% |   376.9 |    -- |     -- |riemann_
3|        |         |       |        | riemann.f90
||||-------------------------------------------
4|||   1.4% |    47.4 |  32.6 | 41.0% |line.77
4|||   3.9% |   135.8 |  28.2 | 17.3% |line.78
```

```
63.  + 1----< do l = lmin, lmax
64.  + 1 2--<   do n = 1, 12
65.    1 2        pmold(l) = pmid(l)
66.    1 2        wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67.    1 2        wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68.    1 2        wlft (l) = clft(l) * sqrt(wlft(l))
69.    1 2        wrgh (l) = crgh(l) * sqrt(wrgh(l))
70.    1 2        zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71.    1 2        zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72.    1 2        zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
73.    1 2        zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
74.    1 2        umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75.    1 2        umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76.    1 2        pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (zrgh(l) - zlft(l))
77.    1 2        pmid (l) = max(smallp,pmid(l))
78.    1 2        if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
79.    1 2-->   enddo
80.    1----> enddo
```

ftn-6254 ftn: VECTOR RIEMANN, File = riemann.f90, Line = 64

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         wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67.    1 2         wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68.    1 2         wlft (l) = clft(l) * sqrt(wlft(l))
69.    1 2         wrgh (l) = crgh(l) * sqrt(wrgh(l))
70.    1 2         zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71.    1 2         zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72.    1 2         zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
73.    1 2         zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) –
                                                                    prgh(l)))
74.    1 2         umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75.    1 2         umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76.    1 2         pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) /
                                                              (zrgh(l) - zlft(l))
77.    1 2         pmid (l) = max(smallp,pmid(l))
78.    1 2         if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
79.    1 2-->   enddo
80.    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          wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
68.    1 Vr2          wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
69.    1 Vr2          wlft (l) = clft(l) * sqrt(wlft(l))
70.    1 Vr2          wrgh (l) = crgh(l) * sqrt(wrgh(l))
71.    1 Vr2          zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
72.    1 Vr2          zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
73.    1 Vr2          zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
74.    1 Vr2          zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
75.    1 Vr2          umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
76.    1 Vr2          umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
77.    1 Vr2          pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / &
78.    1 Vr2                    (zrgh(l)-zlft(l))
79.    1 Vr2          pmid (l) = max(smallp,pmid(l))
80.    1 Vr2          if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) then
81.    1 Vr2          converged(l) = .T.
82.    1 Vr2          endif
83.    1 Vr2         endif
84.    1 Vr2-->    enddo
85.  + 1          if(all(converged(lmin:lmax)))exit
86.    1------>  enddo
```

SWEEPZ

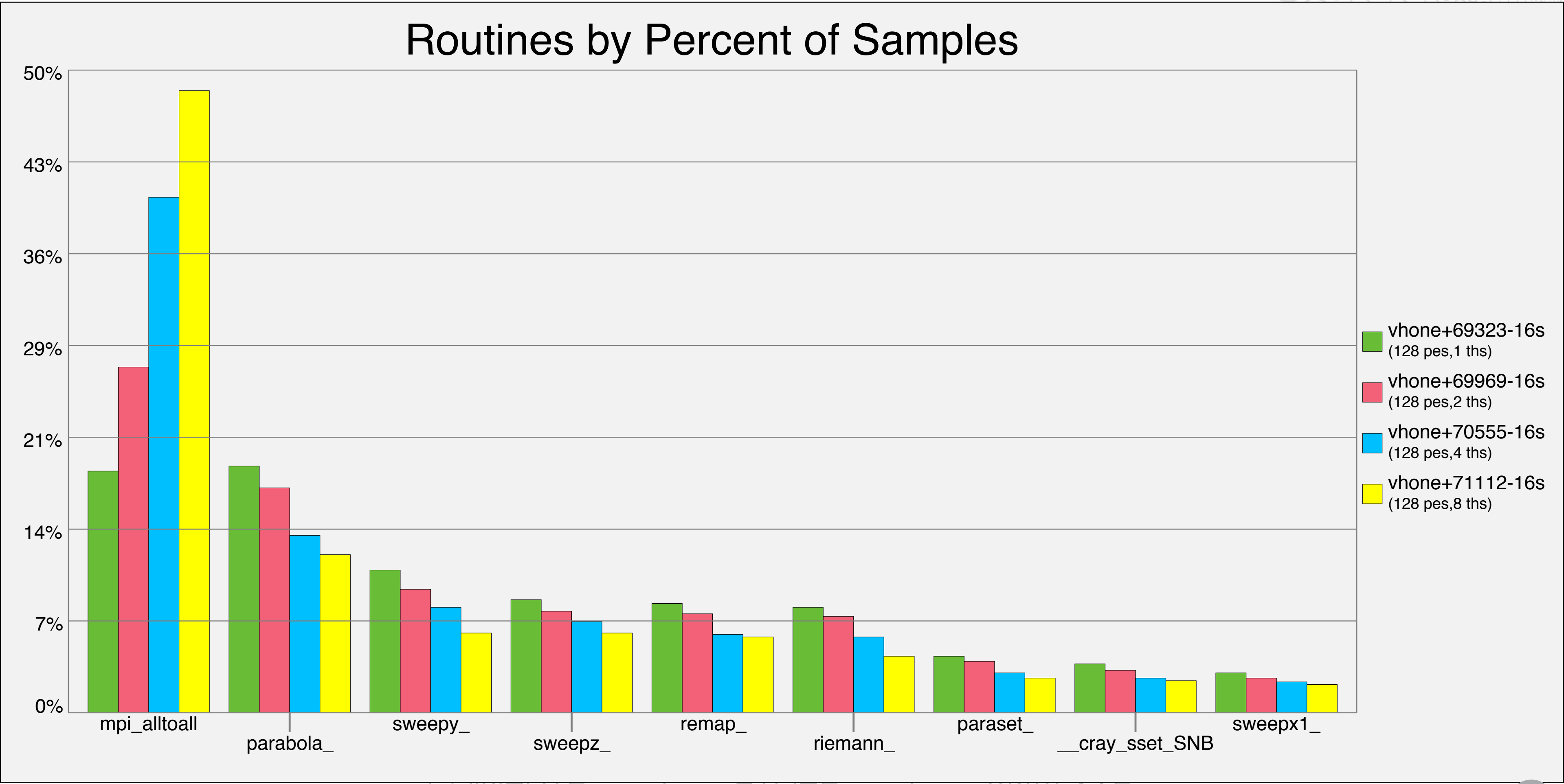
```
||  10.3% |  1,679.0 |     -- |     -- | sweepz_
3|        |          |        |        |  levesque/CUG_2018/VH1_version1/sweepz.f90
||||-------------------------------------------------------------------------
4|||    3.0% |     489.4 |  21.6 |   4.3% | line.27
4|||    1.3% |     211.9 |  38.1 |  15.4% | line.64
4|||    1.0% |     165.0 |  43.0 |  20.8% | line.104
4|||    2.7% |     447.2 |  36.8 |   7.7% | line.121
```

SWEEPZ

```
+ i------------< do j = 1, js
+ i i----------<  do m = 1, npey
+ i i i--------<   do i = 1, isy
  i i i             n = i + isy*(m-1)
+ i i i 4------<    do k = 1, ks
  i i i 4 VR--<>      send3(1,j,k,n) = recv2(1,k,i,j,m)
  i i i 4             send3(2,j,k,n) = recv2(2,k,i,j,m)
  i i i 4             send3(3,j,k,n) = recv2(3,k,i,j,m)
  i i i 4             send3(4,j,k,n) = recv2(4,k,i,j,m)
  i i i 4             send3(5,j,k,n) = recv2(5,k,i,j,m)
  i i i 4             send3(6,j,k,n) = recv2(6,k,i,j,m)
  i i i 4------>    enddo
  i i i-------->   enddo
  i i---------->  enddo
  i------------> enddo
```

SWEEPZ (continued)

```
                                !---------------------------------------------
       M---------------<  !$OMP PARALLEL DO
     + M im------------<  do j = 1, js
     + M im i----------<   do m = 1, npey
     + M im i i--------<    do i = 1, isy
       M im i i              n = i + isy*(m-1)
     + M im i i 5------<     do k = 1, ks
       M im i i 5 VR--<>       send3(1,j,k,n) = recv2(1,k,i,j,m)
       M im i i 5              send3(2,j,k,n) = recv2(2,k,i,j,m)
       M im i i 5              send3(3,j,k,n) = recv2(3,k,i,j,m)
       M im i i 5              send3(4,j,k,n) = recv2(4,k,i,j,m)
       M im i i 5              send3(5,j,k,n) = recv2(5,k,i,j,m)
       M im i i 5              send3(6,j,k,n) = recv2(6,k,i,j,m)
       M im i i 5------>     enddo
       M im i i-------->    enddo
       M im i---------->   enddo
       M im----------->>  enddo
```


Routines by Percent of Samples
50%
43%
36%
29%
21%
14%
7%
0%
mpi_alltoall
parabola_
sweepy_
sweepz_
remap_
riemann_
paraset_
__cray_sset_SNB
sweepx1_
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)

Profile for 1 Thread and 4 Threads

```
Table 1:  Profile by Function     1 Thread

  Samp% |     Samp |   Imb. |  Imb. | Group
        |          |   Samp | Samp% |  Function=[MAX10]
        |          |        |       |   PE=HIDE

 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

  Samp% |     Samp |   Imb. |  Imb. | Group
        |          |   Samp | Samp% |  Function=[MAX10]
        |          |        |       |   PE=HIDE
        |          |        |       |    Thread=HIDE

 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
||======================================================
```

Memory Analysis for Optimized VH1

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

  Samp% |  Samp | Imb. |  Imb. |  MEM_LOAD_UOPS_RETIRED | RESOURCE_STALLS | Group
        |       | Samp | Samp% |     :HIT_LFB:precise=2 |            :ANY |  Function=[MAX10]
        |       |      |       |                        |                 |   Source
        |       |      |       |                        |                 |    Line
        |       |      |       |                        |                 |     PE=HIDE

 100.0% | 506.4 |   -- |    -- | 52,444,505,648,539,880 | 810,810,111,305 | Total
|--------------------------------------------------------------------------------
|  67.0% | 339.1 |   -- |    -- | 35,351,497,873,955,312 | 551,686,946,473 | USER
||-------------------------------------------------------------------------------
||  25.6% | 129.5 |   -- |    -- | 13,561,376,423,707,960 | 207,864,527,154 | parabola_
3|        |       |      |       |                        |                 |  parabola.f90
||||----------------------------------------------------------------------------
4|||  14.0% |  71.1 | 77.9 | 52.7% |  7,540,450,746,779,697 | 113,994,086,387 | line.26
4|||   1.0% |   5.2 |  8.8 | 63.0% |    510,173,395,620,708 |   8,400,427,348 | line.32
4|||   8.1% |  41.0 | 41.0 | 50.4% |  4,241,915,862,200,189 |  65,974,673,046 | line.46
4|||   1.3% |   6.8 |  9.2 | 58.0% |    714,682,558,419,997 |  10,882,073,895 | line.55
||||============================================================================
||  12.1% |  61.2 |   -- |    -- |  6,449,735,211,652,689 | 103,694,034,743 | sweepy_
3|        |       |      |       |                        |                 |  sweepy.f90
||||----------------------------------------------------------------------------
4|||   3.2% |  16.2 | 19.8 | 55.6% |  1,653,665,489,088,910 |  27,467,001,609 | line.46
4|||   6.4% |  32.2 | 34.8 | 52.4% |  3,465,660,652,265,039 |  54,562,573,915 | line.47
||||============================================================================
||  10.2% |  51.9 |   -- |    -- |  5,479,965,955,430,281 |  81,379,551,948 | sweepz_
3|        |       |      |       |                        |                 |  sweepz.f90
||||----------------------------------------------------------------------------
4|||   2.4% |  12.3 | 19.7 | 62.0% |  1,356,797,349,204,156 |  21,098,054,630 | line.28
4|||   1.8% |   8.9 | 13.1 | 60.0% |    945,580,000,330,081 |  13,804,851,073 | line.65
4|||   3.7% |  18.8 | 28.2 | 60.4% |  1,961,528,744,947,444 |  29,502,970,062 | line.66
```

Code Responsible for Memory Bandwidth

Difficult to merge these loops – compiler did merge two

```
                !-------------------------------------------------------------
     Vr4---<    do n = nmin-2, nmax+1
     Vr4          diffa(n) = a(n+1) - a(n)
     Vr4--->    enddo

                !                                            Equation 1.7 of C&W
                !     da(j) = D1 * (a(j+1) - a(j)) + D2 * (a(j) - a(j-1))
     fVr4--<    do n = nmin-1, nmax+1
     fVr4        da(n) = para(n,4) * diffa(n) + para(n,5) * diffa(n-1)
     fVr4        da(n) = sign( min(abs(da(n)), 2.0*abs(diffa(n-1)), 2.0*abs(diffa(n))), da(n) )
     fVr4-->    enddo

                !     zero out da(n) if a(n) is a local max/min
     f-----<    do n = nmin-1, nmax+1
     f            if(diffa(n-1)*diffa(n) < 0.0) da(n) = 0.0
     f----->    enddo

                !                                            Equation 1.6 of C&W
                !     a(j+.5) = a(j) + C1 * (a(j+1)-a(j)) + C2 * dma(j+1) + C3 * dma(j)
                ! MONOT: Limit ar(n) to the range defined by a(n) and a(n+1)

     Vr2---<    do n = nmin-1, nmax
     Vr2          ar(n) = a(n) + para(n,1)*diffa(n) + para(n,2)*da(n+1) + para(n,3)*da(n)
     Vr2        !  ar(n) = max(ar(n),min(a(n),a(n+1)))
     Vr2        !  ar(n) = min(ar(n),max(a(n),a(n+1)))
     Vr2          al(n+1) = ar(n)
     Vr2--->    enddo
```

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.

Q&A

John Levesque
levesque@cray.com

Heidi Poxon
heidi@cray.com