

Hybridization* of an All MPI Application

* Creation of an application that exhibits three levels of parallelism, MPI between nodes, OpenMP** on the node and vectorized looping structures

** Why OpenMP? To provide performance portability.
OpenMP is the only threading construct that a compiler can analyze sufficiently to generate efficient threading on multi-core nodes and to generate efficient code for companion accelerators.

CAUTION!!

- Do not read “Automatic” into this presentation, the Hybridization of an application is difficult and efficient code only comes with a thorough interaction with the accelerator to generate the most efficient code and
 - High level OpenMP structures
 - Low level vectorization of major computational areas
- Performance is also dependent upon the location of the data. Best case is that the major computational arrays reside on the accelerator. Otherwise computational intensity of the accelerated kernel must be significant

**Cray's Hybrid Programming Environment
supplies tools for addressing these issues**

Three levels of Parallelism required

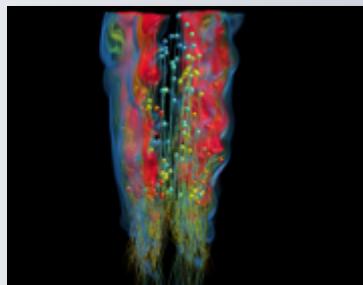
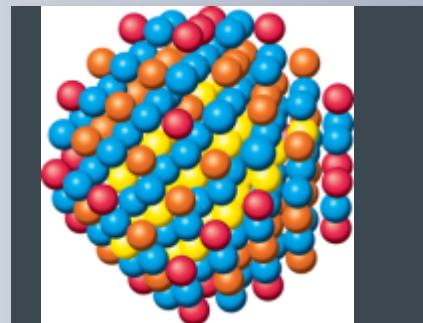
- Developers will continue to use MPI between nodes or sockets
- Developers must address using a shared memory programming paradigm on the node
- Developers must vectorize low level looping structures
- While there is a potential acceptance of new languages for addressing all levels directly. Most developers cannot afford this approach until they are assured that the new language will be accepted and the generated code is within a reasonable performance range

Titan: Early Science Applications

CRAY
THE SUPERCOMPUTER COMPANY

WL-LSMS

Role of material disorder, statistics, and fluctuations in nanoscale materials and systems.

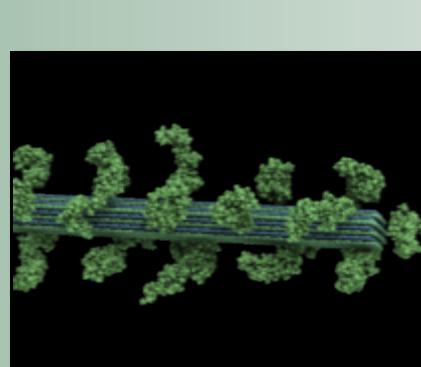
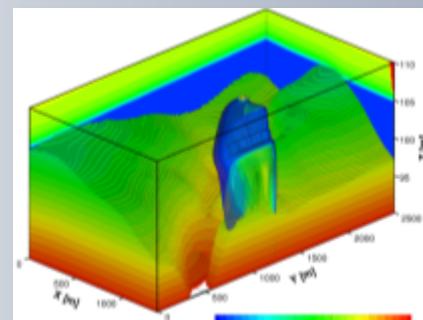


S3D

How are going to efficiently burn next generation diesel/bio fuels?

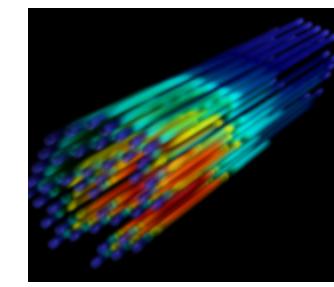
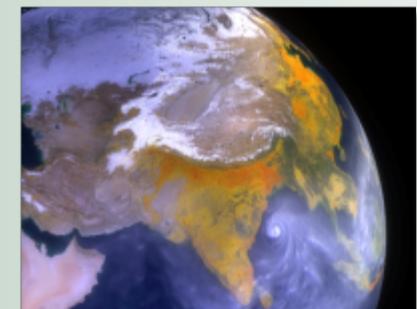
PFLOTRAN

Stability and viability of large scale CO₂ sequestration; predictive containment groundwater transport



CAM / HOMME

Answer questions about specific climate change adaptation and mitigation scenarios; realistically represent features like precipitation patterns/statistics and tropical storms



Denovo

Unprecedented high-fidelity radiation transport calculations that can be used in a variety of nuclear energy and technology applications.

Early Science Apps versus INCITE Awardees

- While the early Science Applications have had two years to prepare for Titan, the INCITE awardees are going to need to move to Titan in a tighter timeframe
 - Some will already be accelerator aware
 - We believe that OpenACC will facilitate porting to Titan



NVIDIA, Cray, PGI, CAPS Unveil ‘OpenACC’ Programming Standard for Parallel Computing

*Directives-based Programming Makes
Accelerating Applications Using
CPUs and GPUs Dramatically Easier*



OpenACC®

DIRECTIVES FOR ACCELERATORS

- A common directive programming model for **today's GPUs**
 - Announced at SC11 conference
 - Offers portability between compilers
 - Drawn up by: NVIDIA, Cray, PGI, CAPS
 - Multiple compilers offer portability, debugging, performance
 - Works for Fortran, C, C++
 - Standard available at www.OpenACC-standard.org
 - Initially implementations targeted at NVIDIA GPUs
- Current version: 1.0 (November 2011)
- Compiler support:
 - Cray CCE: partial now, complete in 2012
 - PGI Accelerator: released product in 2012
 - CAPS: released product in Q1 2012



The Portland Group



Support level

- Parallel -- complete
- Kernels -- complete*
- Loop -- complete
- Data -- complete*
- Update -- complete
- Wait -- complete
- Cache -- in progress (details to follow)
- Declare -- complete*
 - Fortran
 - C and C++
- Library calls -- complete
- Limitations
 - Implementation
 - Dope vector
 - C and C++
 - Specification
 - Function calls, uses IPA call tree flattening
 - C and C++



Enhancements

- **-hacc_model=**
- **Async**
 - auto_async_none
 - Default to all synchronous kernels and transfers
 - auto_async_kernels
 - kernels are executed asynchronously
 - auto_async_all
 - kernels and data motion execute asynchronously
- **Deep copy**
 - Fortran
 - C and C++
- **Addressing**
 - fast_addr
 - no_fast_addr
- **Pedantic***



Extended runtime interfaces

- Why?

- Structured block
- Objects

- What?

```
void* cray_acc_create( void*, size_t );
void cray_acc_delete( void* );
void* cray_acc_copyin( void*, size_t );
void cray_acc_copyout( void*, size_t );
void cray_acc_update_device( void*, size_t );
void cray_acc_update_device_async( void *, size_t, int );
void cray_acc_update_host( void*, size_t );
void cray_acc_update_host_async( void *, size_t, int );
int cray_acc_is_present( void* );
int cray_acc_is_area_present( void*, size_t );
void* cray_acc_deviceptr( void* );
void* cray_acc_hostptr( void* );
```



Extended runtime interfaces

```
void *cray_acc_memcpy_to_host( void* destination,  
                               const void* source,  
                               size_t size );  
void *cray_acc_memcpy_to_host_async( void* destination,  
                                    const void* source,  
                                    size_t size,  
                                    int async_id );  
  
void *cray_acc_memcpy_to_device( void* destination,  
                                 const void * source,  
                                 size_t size );  
void *cray_acc_memcpy_to_device_async( void* destination,  
                                       const void * source,  
                                       size_t size,  
                                       int async_id );  
  
bool cray_acc_get_async_info( int async_id, void * async_info );
```

- **When?**
 - now
 - then next version of OpenACC

Using CRAY OpenACC Directives with C/C++

The real problem: POINTERS TO POINTERS

2d 'jagged' array which can not be treated as a 1d array

```
float **pa2d;
pa2d = (float**)calloc(50,sizeof(float*));
for(int j=0 ; j<50 ; j++){
// the jth row is (j+1) floats
    pa2d[j]=(float*)calloc(j+1,sizeof(float));
}

for(int j=0 ; j<50 ; j++){
    for(int i=0 ; i<j+1 ; i++){
        pa2d[j][i]=i+j;
    }
}
```

This can not be parallelized for the accelerator with data directives only.
It requires calls to runtime routines.

Using CRAY OpenACC Directives with C/C++

The real problem: POINTERS TO POINTERS

Some runtime routines to enable use of pointers to pointers:

```
void* acc_malloc( size_t );
void* cray_acc_create( void*, size_t );
void cray_acc_delete( void* );
```

```
// allocate and initialize memory
void* cray_acc_copyin( void*, size_t );
// copy memory from accelerator and free memory
void cray_acc_copyout( void*, size_t );
```

```
// copy host version of object to the accelerator
void cray_acc_update_device( void*, size_t );
// copy the accelerator version of the object to the host
void cray_acc_update_host( void*, size_t );
```

```
void *cray_acc_memcpy_to_host( void* destination,const void* source,
                               size_t size );
void *cray_acc_memcpy_to_device( void* destination,const void * source,
                                 size_t size );
```

```
void* cray_acc_deviceptr( void* );
void* cray_acc_hostptr( void* );
```

Using CRAY OpenACC Directives with C/C++

The real problem: POINTERS TO POINTERS

Setting up 2d jagged array on the accelerator:

Given: 2d host array

dimension 1: 50

dimension 2: jth row is $(j+1)$ floats

2d 'jagged' array which can not be treated as a 1d array:

```
float **pa2d; // original existing host 'jagged' 2d array to read on acc
float **trowptrs; // temp array to hold acc addresses on host
cray_acc_create(pa2d,50*sizeof(float*)); // create array of rows on acc
trowptrs = (float**)malloc(50*sizeof(float*));// alloc temp host memory for acc row pointers
for(int j=0 ; j<50 ; j++){
    trowptrs[j] = (float*)acc_malloc((j+1)*sizeof(float)); // allocate jth row on acc
                                                // save address on host
    cray_acc_memcpy_to_device(trowptrs[j],pa2d[j],(j+1)*sizeof(float)); // copy data to acc
}
void* pa2d_acc_addr = cray_acc_deviceptr(pa2d); // get the acc address for 2d array
cray_acc_memcpy_to_device(pa2d_acc_addr, trowptrs, 50*sizeof(float*));
// now the 2d array can be accessed from the accelerator
#pragma acc parallel loop present(pa2d[0:50])
for(int j=0 ; j<50 ; j++){
    for(int i=0 ; i<(j+1) ; i++){
        ... = pa2d[j][i];
    }
}
```

Using CRAY OpenACC Directives with C/C++

The real problem: POINTERS TO POINTERS

Create memory on accelerator.

Store address of created memory on host

Copy addresses to accelerator

Copy data to accelerator (before loop)

Copy data to host (after loop)

Many variations:

array already on accelerator

data not needed on host

Code similar to this is required whenever there is a pointer to a pointer.

Jagged 2d array is a pointer to a pointer

```
pa->pb->c;
```

```
pa->pb->pc->pd->e;
```

Using directives to give the compiler information

- Developing efficient OpenMP regions is not an easy task; however, the performance will definitely be worth the effort
- The next step will be to add OpenACC directives to allow for compilation of the same OpenMP regions to accelerator by the compiler.
 - With OpenACC data transfers between multi-core socket and the accelerator as well as utilization of registers and shared memory can be optimized.
 - With OpenACC user can control the utilization of the accelerator memory and functional units.

Task 1 – Identification of potential accelerator kernels

- Identify high level computational structures that account for a significant amount of time (95-99%)
 - To do this, one must obtain global runtime statistics of the application
 - High level call tree with subroutines and DO loops showing inclusive/exclusive time, min, max, average iteration counts.
- Identify major computational arrays
- Tools that will be needed
 - Advanced instrumentation to measure
 - DO loop statistics, iteration counts, inclusive time
 - Routine level sampling and profiling

Normal Profile – default Craypat report

Table 1: Profile by Function Group and Function

| Time% | Time | Imb. | Imb. | Calls | Group |
|--------|-----------|----------|-------|-----------|----------------------|
| | | Time | Time% | | Function |
| | | | | | PE=HIDE |
| 100.0% | 50.553984 | -- | -- | 6922023.0 | Total |
| 52.1% | 26.353695 | -- | -- | 6915004.0 | USER |
| 16.9% | 8.540852 | 0.366647 | 4.1% | 2592000.0 | parabola_ |
| 8.0% | 4.034867 | 0.222303 | 5.2% | 288000.0 | remap_ |
| 7.1% | 3.612980 | 0.862830 | 19.3% | 288000.0 | riemann_ |
| 3.7% | 1.859449 | 0.094075 | 4.8% | 288000.0 | ppmlr_ |
| 3.3% | 1.666590 | 0.064095 | 3.7% | 288000.0 | evolve_ |
| 2.6% | 1.315145 | 0.119832 | 8.4% | 576000.0 | paraset_ |
| 1.8% | 0.923711 | 0.048359 | 5.0% | 864000.0 | volume_ |
| 1.8% | 0.890751 | 0.064695 | 6.8% | 288000.0 | states_ |
| 1.4% | 0.719636 | 0.079651 | 10.0% | 288000.0 | flatten_ |
| 1.0% | 0.513454 | 0.019075 | 3.6% | 864000.0 | forces_ |
| 1.0% | 0.508696 | 0.023855 | 4.5% | 500.0 | sweepz_ |
| 1.0% | 0.504152 | 0.027139 | 5.1% | 1000.0 | sweepy_ |
| 37.9% | 19.149499 | -- | -- | 3512.0 | MPI |
| 28.7% | 14.487564 | 0.572138 | 3.8% | 3000.0 | mpi_alltoall |
| 8.7% | 4.391205 | 2.885755 | 39.7% | 2.0 | mpi_comm_split |
| 10.0% | 5.050780 | -- | -- | 3502.0 | MPI_SYNC |
| 6.9% | 3.483206 | 1.813952 | 52.1% | 3000.0 | mpi_alltoall_(sync) |
| 3.1% | 1.567285 | 0.606728 | 38.7% | 501.0 | mpi_allreduce_(sync) |

Re-compiling with `-hprofile_generate "pat_report -O callers"`

```

100.0% | 117.646170 | 13549032.0 |Total
|-----
| 75.4% | 88.723495 | 13542013.0 |USER
||-----
|| 10.7% | 12.589734 | 2592000.0 |parabola_
|||-----
3|| 7.1% | 8.360290 | 1728000.0 |remap_.LOOPS
4|| | | | remap_
5|| | | | ppmlr_
|||||-----
6|||| 3.2% | 3.708452 | 768000.0 |sweepx2_.LOOP.2.li.35
7|||| | | | sweepx2_.LOOP.1.li.34
8|||| | | | sweepx2_.LOOPS
9|||| | | | sweepx2_
10||| | | | vphone_
6|||| 3.1% | 3.663423 | 768000.0 |sweepx1_.LOOP.2.li.35
7|||| | | | sweepx1_.LOOP.1.li.34
8|||| | | | sweepx1_.LOOPS
9|||| | | | sweepx1_
10||| | | | vphone_
|||||=====
3|| 3.6% | 4.229443 | 864000.0 |ppmlr_
|||-----
4||| 1.6% | 1.880874 | 384000.0 |sweepx2_.LOOP.2.li.35
5||| | | | sweepx2_.LOOP.1.li.34
6||| | | | sweepx2_.LOOPS
7||| | | | sweepx2_
8||| | | | vphone_
4||| 1.6% | 1.852820 | 384000.0 |sweepx1_.LOOP.2.li.35
5||| | | | sweepx1_.LOOP.1.li.34
6||| | | | sweepx1_.LOOPS
7||| | | | sweepx1_
8||| | | | vphone_

```

Converting the MPI application to a Hybrid OpenMP/MPI application

Task 2 Parallel Analysis, Scoping and Vectorization

- Investigate parallelizability of high level looping structures
 - Often times one level of loop is not enough, must have several parallel loops
 - User must understand what high level DO loops are in fact independent.
 - Without tools, variable scoping of high level loops is very difficult
 - Loops must be more than independent, their variable usage must adhere to private data local to a thread or global shared across all the threads
- Investigate vectorizability of lower level Do loops
 - Cray compiler has been vectorizing complex codes for over 30 years



Example Report – Inclusive Loop Time

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

| Loop Incl Time Total | Loop Hit | Loop Trips Avg | Loop Trips | Loop Trips | Function=/.LOOP[.] PE=HIDE |
|-------------------------------|-------------|----------------------|---------------|---------------|-------------------------------|
| | | | Min | Max | |
| 8.995914 | 100 | 25 | 0 | 25 | sweeypy_.LOOP.1.li.33 |
| 8.995604 | 2500 | 25 | 0 | 25 | sweeypy_.LOOP.2.li.34 |
| 8.894750 | 50 | 25 | 0 | 25 | sweepz_.LOOP.05.li.49 |
| 8.894637 | 1250 | 25 | 0 | 25 | sweepz_.LOOP.06.li.50 |
| 4.420629 | 50 | 25 | 0 | 25 | sweepx2_.LOOP.1.li.29 |
| 4.420536 | 1250 | 25 | 0 | 25 | sweepx2_.LOOP.2.li.30 |
| 4.387534 | 50 | 25 | 0 | 25 | sweepx1_.LOOP.1.li.29 |
| 4.387457 | 1250 | 25 | 0 | 25 | sweepx1_.LOOP.2.li.30 |
| 2.523214 | 187500 | 107 | 0 | 107 | riemann_.LOOP.2.li.63 |
| 1.541299 | 20062500 | 12 | 0 | 12 | riemann_.LOOP.3.li.64 |
| 0.863656 | 1687500 | 104 | 0 | 108 | parabola_.LOOP.6.li.67 |

Reveal



New analysis and code restructuring assistant...

Uses both the performance toolset and CCE's program library functionality to provide static and runtime analysis information

Assists user with the code optimization phase by **correlating source code with analysis** to help identify which areas are key candidates for optimization

Key Features

Annotated source code with compiler optimization information

- Provides feedback on critical dependencies that prevent optimizations

Scoping analysis

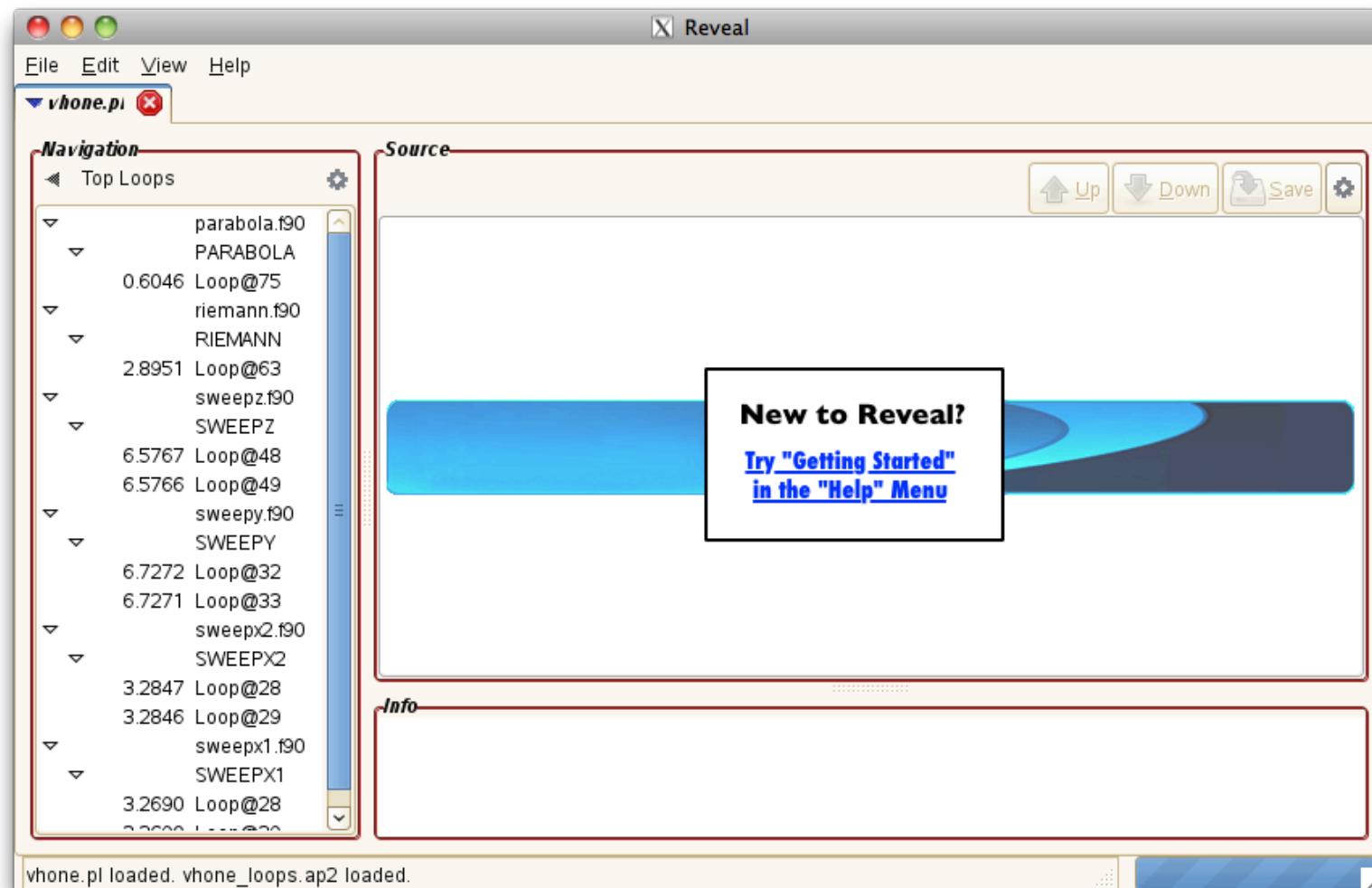
- Identifies shared, private and ambiguous arrays
- Allows user to privatize ambiguous arrays
- Allows user to override dependency analysis

Source code navigation

- Uses performance data collected through CrayPat



Reveal with Loop Work Estimates





Visualize Loopmark with Performance Information

The screenshot shows the Reveal application interface for visualizing performance information. The window title is "Reveal". The menu bar includes File, Edit, View, Help, and a tab labeled "vhone.pi". The left panel, titled "Navigation", lists various source files and their performance metrics:

- parabola.f90 (PARABOLA): 0.6046 Loop@75
- riemann.f90
- RIEMANN: 2.8951 Loop@63
- sweepz.f90
- SWEEPZ: 6.5767 Loop@48
- 6.5766 Loop@49
- sweezy.f90
- SWEEZY: 6.7272 Loop@32
- 6.7271 Loop@33
- sweepx2.f90
- SWEEPX2: 3.2847 Loop@28
- 3.2846 Loop@29
- sweepx1.f90
- SWEEPX1: 3.2690 Loop@28
- 3.2690 Loop@29

The right panel displays the source code for "parabola.f90". The code is color-coded: lines 74 and 84 are green, while others are black. Annotations are shown above lines 74 and 84. The status bar at the bottom indicates: "vhone.pi loaded. vhone_loops.ap2 loaded."

Annotations in the image:

- A yellow speech bubble points to the "Navigation" panel with the text "Performance feedback".
- A yellow speech bubble points to the source code area with the text "Loopmark and optimization annotations".
- A yellow speech bubble points to the "Info - Line 75" panel with the text "Compiler feedback".

Source code (lines 74-87):

```
74
75 do n = nmin, nmax
76   if(scrch1(n) <= 0.0) then
77     ar(n) = a(n)
78     al(n) = a(n)
79   endif
80   if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
81   if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
82 enddo
83
84 do n = nmin, nmax
85   deltaa(n)= ar(n) - al(n)
86   a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
87 enddo
```

Info - Line 75

- A loop starting at line 75 was unrolled 2 times.
- A loop starting at line 75 was vectorized.

Visualize CCE's Loopmark with Performance Profile (2)



Reveal

File Edit View Help

vphone.pl

Navigation

- Program View
- riemann.f90
- remap.f90
- evolve.f90
- volume.f90
- forces.f90
- ppmlr.f90
- states.f90
- flatten.f90
- sweepz.f90
- sweeppy.f90
- boundary.f90
- prin.f90
- sweepx2.f90
- 0.53% SWEEPX2
 - Loop@28
 - Loop@29
 - Loop@32
 - Loop@33**
 - Loop@44
 - Loop@58
- sweepx1.f90

Source - /lus/sonexion/heidi/reveal/sweepx2.f90

```

L 32 do m = 1, npey
Lr8 33 do i = 1, isy
34 n = i + isy*(m-1) + 6
35 r(n) = recv2(1,k,i,j,m)
36 p(n) = recv2(2,k,i,j,m)
37 u(n) = recv2(3,k,i,j,m)
38 v(n) = recv2(4,k,i,j,m)
39 w(n) = recv2(5,k,i,j,m)
40 f(n) = recv2(6,k,i,j,m)
41 enddo
42 enddo
43 do i = 1,imax
44 n = i + 6
45

```

Info - Line 33

- A loop starting at line 33 was not vectorized because it does not have enough iterations.
- A loop starting at line 33 was unrolled 8 times.

Explain

OPT_INFO: A loop starting at line %s was unrolled.

The compiler unrolled the loop. Unrolling creates a number of copies of the loop body. When unrolling an outer loop, the compiler attempts to fuse replicated inner loops - a transformation known as unroll-and-jam. The compiler will always employ the unroll-and-jam mode when unrolling an outer loop; literal outer loop unrolling may occur when unrolling to satisfy a user directive (pragma).

This message indicates that unroll-and-jam was performed with respect to the identified loop. A different message is issued when literal outer loop unrolling is performed, as this transformation is far less likely to be beneficial.

For sake of illustration, the following contrasts unroll-and-jam with literal outer loop unrolling.

```

# 426 "ptmp/ulib/buildslaves/pdgcs-81-edition-build/tbs/build/release/pdgcs/pdgcstfn.msg.c"
DO J=1,10
DO I=1,100
A(I,J)=B(I,J) + 42.0
ENDDO
DO J=1,10,2
DO I=1,100
A(I,J)=B(I,J) + 42.0 !unroll-and-jam
A(I,J+1)=B(I,J+1) + 42.0
ENDDO
ENDDO
DO J=1,10,2
DO I=1,100
A(I,J)=B(I,J) + 42.0 !literal outer unroll
ENDDO
DO I=1,100
A(I,J+1)=B(I,J+1) + 42.0
ENDDO
ENDDO

```

The literal outer unroll code performs the same sequence of memory operations as the original nest, while the unroll-and-jam transformation interleaves operations from outer loop iterations. The compiler employs literal outerloop unrolling only when the data dependencies in the loop, or a control flow impediment, prevent fusion of the replicated inner loops. Literal outer loop unrolling is generally not desirable. It is provided to ensure expected behavior and for those rare instances where the user has determined that it is beneficial.

Integrated message
'explain support'

Explain other message... Close



View Pseudo Code for Inlined Functions

Reveal

File Edit View Help

vhone.pl

Navigation

Program View

- parabola.f90
- riemann.f90
- remap.f90
- evolve.f90
- volume.f90
- force.f90
- plot.f90
- start.f90
- flatten.f90
- sweepz.f90
- sweezy.f90
- boundary.f90
- prin.f90
- sweepx2.f90
- sweepx1.f90
- dtcon.f90
- vhone.f90
- init.f90
- 0.00% GRID Loop@199
- 0.01% INIT

Inlined call sites marked

Source - /lus/sonexion/heidi/reveal/init.f90

```

IVr2 88 call grid(imax,xmin,xmax,zxa,zxc,zdx)
      t$26 = 100
      t$27 = 100
      $I_L88_100 = 0
      !dir$ ivdep
      do
          zxa(1 + $I_L88_100) = 9.9999998e-3 * $I_L88_100
          zdx(1 + $I_L88_100) = 9.9999998e-3
          zxc(1 + $I_L88_100) = 4.9999999e-3 + ( 9.9999998e-3 *
          $I_L88_100 = 1 + $I_L88_100
          if ( $I_L88_100 >= 100 ) exit
      enddo
      If 89 call grid(jmax,ymin,ymax,zya,zyc,zdy)
      Tf 90 call grid(kmax,zmin,zmax,zzc,zzd,zdz)

```

Info - Line 89:

- A divide was turned into a multiply by a reciprocal
- A loop starting at line 89 was fused with the loop starting at line 88.
- The call to leaf routine "grid" was textually inlined.

vhone.pl loaded. vhone_loops.ap2 loaded.

Expand to see pseudo code



Scoping Assistance – Review Scoping Results

Navigation

File Edit View Help

vhone.pl

Source - /lus/sonexion/heidi/reveal/swee

| Name | Type | Scope | Info |
|---------|--------|------------|---|
| f | Array | Unresolved | FAIL:Last defining iteration not known for variable that is live on exit. WARN:LastPrivate of array may be very expensive. |
| flat | Array | Unresolved | FAIL:Last defining iteration not known for variable that is live on exit. WARN:LastPrivate of array may be very expensive. |
| q | Array | Unresolved | FAIL:Last defining iteration not known for variable that is live on exit. WARN:LastPrivate of array may be very expensive. |
| zyc | Array | Conflict | |
| i | Scalar | Private | |
| j | Scalar | Private | |
| k | Scalar | Private | |
| m | Scalar | Private | |
| isz | Scalar | Shared | |
| js | Scalar | Shared | |
| ks | Scalar | Shared | |
| mpey | Scalar | Shared | |
| mpeyz | Scalar | Shared | |
| ngeomz | Scalar | Shared | |
| nleftz | Scalar | Shared | |
| npey | Scalar | Shared | |
| nrightz | Scalar | Shared | |

Info - Line 48

- A loop starting at line 48 was not vectored. Loop has been flattened.

Parallelization inhibitor messages are provided to assist user with analysis

Loops with scoping information are highlighted – red needs user assistance

User addresses parallelization issues for unresolved variables

Scoping Assistance – User Resolves Issues



Use Reveal's OpenMP parallelization tips

Click on variable to view all occurrences in loop

Reveal – OpenMP Tips

OpenMP Tips

- Reduction in an inlined function
- Scoping conflict with inlined variable
- Scoping conflict with locally visible array
 - An array requires conflicting scopes at different locations.
It may be possible to declare and use a different array for the private array uses.

sweepx2.f90

```
! Loop over each row...
  = 1, ks
  = 1, js
  ! Put state variables i
  do m = 1, npey
    do i = 1, isy
      n = i + isy*(m-1) +
      r(n) = recv2(1,k,i,j)
      p(n) = recv2(2,k,i,j)
      u(n) = recv2(3,k,i,j)
      v(n) = recv2(4,k,i,j)
      w(n) = recv2(5,k,i,j)
      f(n) = recv2(6,k,i,j)
    enddo
  enddo
```

Info - Line 28

A loop starting at line 28 was not vectorized because it contains a call to subroutine "ppmlr" on line 55.

Loop has been flattened.

Loop has been flattened.

OpenMP Scope Selector

| Name | Type | Scope | Info |
|--------|--------|------------|--|
| f | Array | Unresolved | FAIL! Last defining iteration not known for variable that is live on exit.:W |
| flat | Array | Unresolved | FAIL! Last defining iteration not known for variable that is live on exit.:W |
| q | Array | Unresolved | FAIL! Last defining iteration not known for variable that is live on exit.:W |
| isy | Scalar | Shared | |
| js | Scalar | Shared | |
| ks | Scalar | Shared | |
| ngeom | Scalar | Shared | |
| nleft | Scalar | Shared | |
| npey | Scalar | Shared | |
| nright | Scalar | Shared | |
| recv2 | Array | Shared | |
| zdx | Array | Shared | |
| zfl | Array | Shared | |
| zpr | Array | Shared | |
| zro | Array | Shared | |
| zux | Array | Shared | |
| zuy | Array | Shared | |
| zuz | Array | Shared | |

First/Last Private
 Enable First Private
 Enable Last Private

None

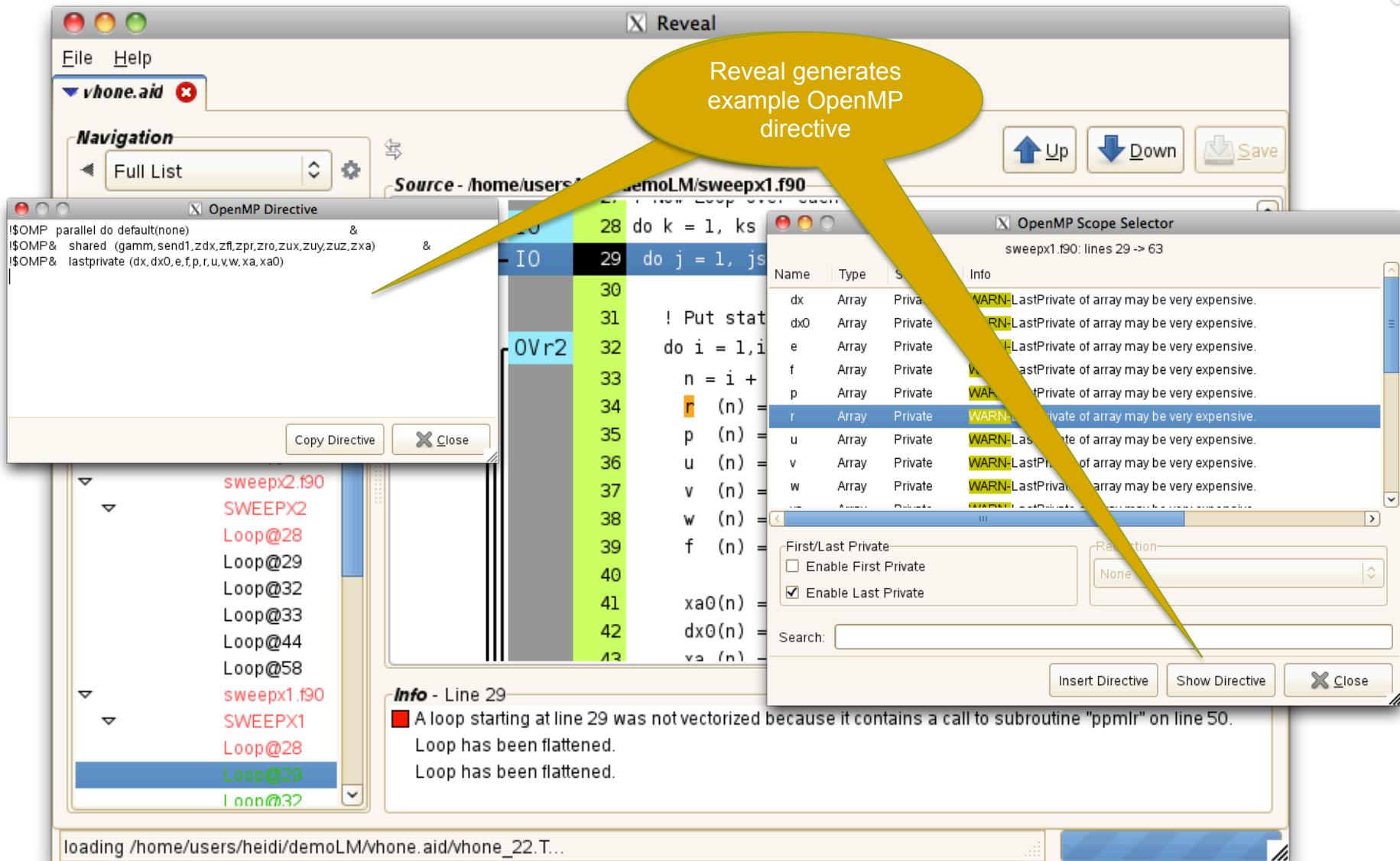
Search:

Insert Directive Show Directive Close

loading /home/users/heidi/demoLM/vphone.aid/vphone_22.T...



Scoping Assistance – Generate Directive



At this point we should have some idea of the major arrays

- 1) Which arrays are used in the major computational routines?
- 2) Where else are these arrays used?
- 3) Are other arrays used with identified arrays
- 4) Go to 1

This is extremely difficult in Fortran and more so in C and C++. We could really use a tool that identified where in the code certain range of memory was used.

Task 3 Correctness Debugging

- Run transformed application on the accelerator and investigate the correctness and performance
 - Run as OpenMP application on multi-core socket
 - Use multi-core socket Debugger - DDT
 - Run as Hybrid multi-core application across multi-core socket and accelerator
- Tools That will be needed
 - Information that was supplied by the directives/user's interaction with the compiler

Task 4 Letting the Compiler do all the work

- The only requirement for using the !\$acc parallel loop is that the user specify the private variables and the compiler will do the rest.
 - If subroutine calls are contained in the loop, -hwp must be used.

```
#ifdef GPU
 !$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
 !$acc&    xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta)&
 !$acc&    reduction(max:svel)
#else
 !$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
 !$omp&    xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta)&
 !$omp&    reduction(max:svel)
#endif
```

- The Compiler will then show:
 - All data motion required to run the loop on the accelerator.
 - Show how it handled the looping structures in the parallel region

Compiler list for SWEEPX1

```

45.          #ifdef GPU
46. G-----< !$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
47. G           !$acc&      xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta) &
48. G           !$acc&      reduction(max:svel)
49. G           #else
50. G           !$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
51. G           !$omp&      xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta) &
52. G           !$omp&      reduction(max:svel)
53. G           #endif
55. G g-----< do k = 1, ks
56. G g 3-----< do j = 1, js
57. G g 3           theta=0.0
58. G g 3           stheta=0.0
59. G g 3           radius=0.0
62. G g 3 g-----< do i = 1,imax
63. G g 3 g           n = i + 6
64. G g 3 g           r (n) = zro(i,j,k)
65. G g 3 g           p (n) = zpr(i,j,k)
66. G g 3 g           u (n) = zux(i,j,k)
67. G g 3 g           v (n) = zuy(i,j,k)
68. G g 3 g           w (n) = zuz(i,j,k)
69. G g 3 g           f (n) = zfl(i,j,k)
71. G g 3 g           xa0(n) = zxa(i)
72. G g 3 g           dx0(n) = zdx(i)
73. G g 3 g           xa (n) = zxa(i)
74. G g 3 g           dx (n) = zdx(i)
75. G g 3 g           p (n) = max(smallp,p(n))
76. G g 3 g           e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
77. G g 3 g-----> enddo
79. G g 3           ! Do 1D hydro update using PPMLR
80. G g 3 gr2 I--> call ppmlr (svel0, sweep, nmin, nmax, ngeom, nleft, nright,r, p, e, q, u, v, w, &
81. G g 3           xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta)
82. G g 3

```

Compiler list for SWEEPX1

ftn-6405 ftn: ACCEL File = sweepx1.f90, Line = 46

A region starting at line 46 and ending at line 104 was placed on the accelerator.

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zro" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zpr" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zux" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zuy" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zuz" to accelerator, free at line 104 (acc_copyin).

ftn-6418 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "zfl" to accelerator, free at line 104 (acc_copyin).

ftn-6416 ftn: ACCEL File = sweepx1.f90, Line = 46

If not already present: allocate memory and copy whole array "send1" to accelerator, copy back at line 104 (acc_copy).

Task 5 Fine tuning of accelerated program

- Understand current performance bottlenecks
 - Is data transfer between multi-core socket and accelerator a bottleneck?
 - Is shared memory and registers on the accelerator being used effectively?
 - Is the accelerator code utilizing the MIMD parallel units?
 - Is the shared memory parallelization load balanced?
 - Is the low level accelerator code vectorized?
 - Are the memory accesses effectively utilizing the memory bandwidth?

Profile of Accelerated Version 1

Table 1: Time and Bytes Transferred for Accelerator Regions

| Acc Time% | Acc Time | Host Time | Acc Copy In (MBytes) | Acc Copy Out (MBytes) | Calls | Function |
|-----------|----------|-----------|-------------------------|--------------------------|-------|---------------------------|
| | | | | | | PE=HIDE Thread=HIDE |
| 100.0% | 58.363 | 67.688 | 24006.022 | 16514.196 | 14007 | Total |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| 30.3% | 17.697 | 0.022 | -- | -- | 1000 | sweepy_.ACC_KERNEL@li.47 |
| 22.0% | 12.827 | 0.010 | -- | -- | 500 | sweepx2_.ACC_KERNEL@li.46 |
| 21.2% | 12.374 | 0.013 | -- | -- | 500 | sweepz_.ACC_KERNEL@li.67 |
| 14.0% | 8.170 | 0.013 | -- | -- | 500 | sweepx1_.ACC_KERNEL@li.46 |
| 3.9% | 2.281 | 1.161 | 12000.004 | -- | 1000 | sweepy_.ACC_COPY@li.47 |
| 2.0% | 1.162 | 0.601 | 6000.002 | -- | 500 | sweepz_.ACC_COPY@li.67 |
| 1.6% | 0.953 | 0.014 | -- | 6000.004 | 1000 | sweepy_.ACC_COPY@li.129 |
| 1.0% | 0.593 | 0.546 | 3000.002 | -- | 500 | sweepx1_.ACC_COPY@li.46 |
| 1.0% | 0.591 | 0.533 | 3000.002 | -- | 500 | sweepx2_.ACC_COPY@li.46 |
| 0.8% | 0.494 | 0.015 | -- | 3000.002 | 500 | sweepx2_.ACC_COPY@li.107 |
| 0.8% | 0.485 | 0.007 | -- | 3000.002 | 500 | sweepx1_.ACC_COPY@li.104 |
| 0.8% | 0.477 | 0.007 | -- | 3000.002 | 500 | sweepz_.ACC_COPY@li.150 |
| 0.4% | 0.250 | 0.016 | -- | 1503.174 | 500 | vhone_.ACC_COPY@li.251 |
| 0.0% | 0.005 | 0.005 | 6.012 | -- | 1 | vhone_.ACC_COPY@li.205 |
| 0.0% | 0.001 | 0.000 | -- | 6.012 | 1 | vhone_.ACC_COPY@li.283 |
| 0.0% | 0.001 | 0.000 | -- | 5.000 | 1 | vhone_.ACC_COPY@li.266 |
| ===== | ===== | ===== | ===== | ===== | ===== | ===== |

Differences in runtime

| | |
|-----------------------------|----------------|
| All MPI on 4096 cores | 43.01 seconds |
| Hybrid 256 nodesx16 threads | 45.05 seconds |
| Rest Hybrid 256x16 threads | 47.58 seconds |
| OpenACC 256xgpu | 105.92 seconds |

Task 4 Fine tuning of accelerated program

- Tools that will be needed:
 - Compiler feedback on parallelization and vectorization of input application
 - Hardware counter information from the accelerator to identify bottlenecks in the execution of the application.
 - Information on memory utilization
 - Information on performance of SIMD units

Several other vendors are supplying similar performance gathering tools

Useful tools contd.

- Craypat profiling
 - Tracing: "pat_build -u <executable>" (can do APA sampling first)
 - "pat_report -O accelerator <.xf file>"; -T also useful
 - Other pat_report tables (as of perftools/5.2.1.7534)
 - acc_fu flat table of accelerator events
 - acc_time call tree sorted by accelerator time
 - acc_time_fu flat table of accelerator events sorted by accelerator time
 - acc_show_by_ct regions and events by calltree sorted alphabetically

Run and gather runtime statistics

Table 1: Profile by Function Group and Function

| Time % | Time | Imb. Time | Imb. | Calls | Group |
|--------|-----------|-----------|--------|-------|--------------------------------------|
| | | | Time % | | Function |
| | | | | | PE='HIDE' |
| | | | | | Thread='HIDE' |
| 100.0% | 83.277477 | -- | -- | 851.0 | Total |
| <hr/> | | | | | |
| 51.3% | 42.762837 | -- | -- | 703.0 | ACCELERATOR |
| <hr/> | | | | | |
| 18.8% | 15.672371 | 1.146276 | 7.3% | 20.0 | recolor_.SYNC_COPY@li.790 ↪ not good |
| 16.3% | 13.585707 | 0.404190 | 3.1% | 20.0 | recolor_.SYNC_COPY@li.793 ↪ not good |
| 7.5% | 6.216010 | 0.873830 | 13.1% | 20.0 | lbtm3d2p_d_.ASYNC_KERNEL@li.116 |
| 1.6% | 1.337119 | 0.193826 | 13.5% | 20.0 | lbtm3d2p_d_.ASYNC_KERNEL@li.119 |
| 1.6% | 1.322690 | 0.059387 | 4.6% | 1.0 | lbtm3d2p_d_.ASYNC_COPY@li.100 |
| 1.0% | 0.857149 | 0.245369 | 23.7% | 20.0 | collisionb_.ASYNC_KERNEL@li.586 |
| 1.0% | 0.822911 | 0.172468 | 18.5% | 20.0 | lbtm3d2p_d_.ASYNC_KERNEL@li.114 |
| 0.9% | 0.786618 | 0.386807 | 35.2% | 20.0 | injection_.ASYNC_KERNEL@li.1119 |
| 0.9% | 0.727451 | 0.221332 | 24.9% | 20.0 | lbtm3d2p_d_.ASYNC_KERNEL@li.118 |

Keep data on the accelerator with acc_data region

```
!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$acc& ci12,ci13,ci14,r,b,uxyz,cell,rho,grad,index_max,index,&
!$acc& ciy,ciz,wet,np,streaming_sbuf1, &
!$acc& streaming_sbuf1,streaming_sbuf2,streaming_sbuf4,streaming_sbuf5,&
!$acc& streaming_sbuf7s,streaming_sbuf8s,streaming_sbuf9n,streaming_sbuf10s,&
!$acc& streaming_sbuf11n,streaming_sbuf12n,streaming_sbuf13s,streaming_sbuf14n,&
!$acc& streaming_sbuf7e,streaming_sbuf8w,streaming_sbuf9e,streaming_sbuf10e,&
!$acc& streaming_sbuf11w,streaming_sbuf12e,streaming_sbuf13w,streaming_sbuf14w, &
!$acc& streaming_rbuf1,streaming_rbuf2,streaming_rbuf4,streaming_rbuf5,&
!$acc& streaming_rbuf7n,streaming_rbuf8n,streaming_rbuf9s,streaming_rbuf10n,&
!$acc& streaming_rbuf11s,streaming_rbuf12s,streaming_rbuf13n,streaming_rbuf14s,&
!$acc& streaming_rbuf7w,streaming_rbuf8e,streaming_rbuf9w,streaming_rbuf10w,&
!$acc& streaming_rbuf11e,streaming_rbuf12w,streaming_rbuf13e,streaming_rbuf14e, &
!$acc& send_e,send_w,send_n,send_s,recv_e,recv_w,recv_n,recv_s)
do ii=1,ntimes
    o o o
    call set_boundary_macro_press2
    call set_boundary_micro_press
    call collisiona
    call collisionb
    call recolor
```

Now when we do communication we have to update the host

```
!$acc parallel_loop private(k,j,i)
do j=0,local_ly-1
  do i=0,local_lx-1
    if (cell(i,j,0)==1) then
      grad (i,j,-1) = (1.0d0-wet)*db*press
    else
      grad (i,j,-1) = db*press
    end if
    grad (i,j,lz) = grad(i,j,lz-1)
  end do
end do
 !$acc end parallel_loop
 !$acc update host(grad)
 call mpi_barrier(MPI_COMM_WORLD,ierr)
 call grad_exchange
 !$acc update device(grad)
```

But we would rather not send the entire grad array back – how about

Packing the buffers on the accelerator



```
!$acc data present(grad,recv_w,recv_e,send_e,send_w,recv_n,&
!$acc&           recv_s,send_n,send_s)
!$acc parallel_loop
do k=-1,lz
  do j=-1,local_ly
    send_e(j,k) = grad(local_lx-1,j ,k)
    send_w(j,k) = grad(0      ,j ,k)
  end do
end do
!$acc end parallel_loop
!$acc update host(send_e,send_w)
call mpi_irecv(recv_w, bufsize(2),mpi_double_precision,w_id, &
               tag(25),mpi_comm_world,irequest_in(25),ierr)
  o  o  o
call mpi_isend(send_w, bufsize(2),mpi_double_precision,w_id, &
               tag(26),& mpi_comm_world,irequest_out(26),ierr)
call mpi_waitall(2,irequest_in(25),istatus_req,ierr)
call mpi_waitall(2,irequest_out(25),istatus_req,ierr)
!$acc update device(recv_e,recv_w)
!$acc parallel
!$acc loop
do k=-1,lz
  do j=-1,local_ly
    grad(local_lx ,j ,k) = recv_e(j,k)
    grad(-1       ,j ,k) = recv_w(j,k)
```

Final Profile - bulk of time in kernel execution

| | | | | | |
|-------|------------|-----------|-------|---------|---|
| 37.9% | 236.592782 | -- | -- | 11403.0 | ACCELERATOR |
| <hr/> | | | | | |
| 15.7% | 98.021619 | 43.078137 | 31.0% | 200.0 | lbtm3d2p_d_.ASYNC_KERNEL@li.129 |
| 3.7% | 23.359080 | 2.072147 | 8.3% | 200.0 | lbtm3d2p_d_.ASYNC_KERNEL@li.127 |
| 3.6% | 22.326085 | 1.469419 | 6.3% | 200.0 | lbtm3d2p_d_.ASYNC_KERNEL@li.132 |
| 3.0% | 19.035232 | 1.464608 | 7.3% | 200.0 | collisionb_.ASYNC_KERNEL@li.599 |
| 2.6% | 16.216648 | 3.505232 | 18.1% | 200.0 | lbtm3d2p_d_.ASYNC_KERNEL@li.131 |
| 2.5% | 15.401916 | 8.093716 | 35.0% | 200.0 | injection_.ASYNC_KERNEL@li.1116 |
| 1.9% | 11.734026 | 4.488785 | 28.1% | 200.0 | recolor_.ASYNC_KERNEL@li.786 |
| 0.9% | 5.530201 | 2.132243 | 28.3% | 200.0 | collisionb_.SYNC_COPY@li.593 |
| 0.8% | 4.714995 | 0.518495 | 10.1% | 200.0 | collisionb_.SYNC_COPY@li.596 |
| 0.6% | 3.738615 | 2.986891 | 45.1% | 200.0 | collisionb_.ASYNC_KERNEL@li.568 |
| 0.4% | 2.656962 | 0.454093 | 14.8% | 1.0 | lbtm3d2p_d_.ASYNC_COPY@li.100 |
| 0.4% | 2.489231 | 2.409892 | 50.0% | 200.0 | streaming_exchange_.ASYNC_COPY@li.810 |
| 0.4% | 2.487132 | 2.311190 | 48.9% | 200.0 | streaming_exchange_.ASYNC_COPY@li.625 |
| 0.2% | 1.322791 | 0.510645 | 28.3% | 200.0 | streaming_exchange_.SYNC_COPY@li.622 |
| 0.2% | 1.273771 | 0.288743 | 18.8% | 200.0 | streaming_exchange_.SYNC_COPY@li.574 |
| 0.2% | 1.212260 | 0.298053 | 20.0% | 200.0 | streaming_exchange_.SYNC_COPY@li.759 |
| 0.2% | 1.208250 | 0.422182 | 26.3% | 200.0 | streaming_exchange_.SYNC_COPY@li.806 |
| 0.1% | 0.696120 | 0.442372 | 39.5% | 200.0 | streaming_exchange_.ASYNC_KERNEL@li.625 |
| 0.1% | 0.624982 | 0.379697 | 38.4% | 200.0 | streaming_exchange_.ASYNC_KERNEL@li.525 |

GPU Direct

```
#ifdef GPU
!$acc host_data use_device(neg_f_x_buf)
#endif
    call MPI_IRecv(neg_f_x_buf(1,1,1,idx),(my*mz*iorder/2),&
                    MPI_REAL8,deriv_x_list(idx)%neg_nbr,idx,&
                    gcomm,deriv_x_list(idx)%req(1),ierr)

#ifndef GPU
!$acc end host_data
#endif
#endif
if(lnbr(2)>=0) then
#ifndef GPU
!$acc host_data use_device(pos_f_x_buf)
#endif
    call MPI_IRecv(pos_f_x_buf(1,1,1,idx),(my*mz*iorder/2),&
                    MPI_REAL8,deriv_x_list(idx)%pos_nbr,idx+deriv_list_size,&
                    gcomm,deriv_x_list(idx)%req(3),ierr)

#ifndef GPU
!$acc end host_data
#endif
#endif
```

Two ways of using GPU Direct on sends

```
#ifdef GPU_STREAMS
 !$acc update host(neg_fs_x_buf(:, :, :, idx:idx+np-1)) async(istr+1)
#else
 !$acc host_data use_device(neg_fs_x_buf)
#endif

#ifndef GPU_STREAMS
    call cray_mpif_isend_openacc(c_loc(neg_fs_x_buf(1,1,1,idx)),(np*my*mz*iorder/2), &
                                MPI_REAL8,deriv_x_list(idx)%pos_nbr,idx, &
                                gcomm,istr+1,deriv_x_list(idx)%req(4),ierr)
#else
    call MPI_ISend(neg_fs_x_buf(1,1,1,idx),(np*my*mz*iorder/2), &
                  MPI_REAL8,deriv_x_list(idx)%pos_nbr,idx, &
                  gcomm,deriv_x_list(idx)%req(4),ierr)
#endif

#ifdef GPU
#ifndef GPU_STREAMS
 !$acc end host_data
#endif
#endif
```

Sharing the GPU between MPI tasks

- Non exclusive mode
 - Currently being used. May have problems
- Cuda Proxy
 - Allows n MPI tasks to share GPU
 - Currently being tested in house

Useful tools

- Compiler feedback:
 - -ra to generate *.lst loopmark files (equivalent for C)
 - -rd to generate *.cg and *.opt files
 - *.cg useful to understand synchronisation points (CAF and ACC)
 - "ptxas -v *.ptx" gives information on register and shared memory usage (no way yet for user to adjust this)
- Runtime feedback (no recompilation needed)
 - "export CRAY_ACC_DEBUG=[1,2,3]" commentary to STDERR
 - NVIDIA compute profiler works with CUDA and directives
 - "export COMPUTE_PROFILE=1"
 - gives information on timings and occupancy in separate file
 - "more /opt/nvidia/cuda/<version>/doc/Compute_Profiler.txt" for documentation
 - Vince Graziano has a great script for summarising the output

Cray GPU Programming Environment

- Objective: Enhance productivity related to porting applications to hybrid multi-core systems
- Four core components
 - Cray Statistics Gathering Facility on host and GPU
 - Cray Optimization Explorer – Scoping Tools (COE)
 - Cray Compilation Environment (CCE)
 - Cray GPU Libraries