Application Development for the Cray XK6

Jeff Larkin & John Levesque
Cray’s Supercomputing Center of Excellence
Outline

• Why should you care about this tutorial
  • The next generation of programming challenges

• Converting to Hybrid OpenMP/MPI
  • Identifying high level OpenMP loops
  • Using the Cray Scoping tool
  • Using the program library (-hwp)
  • NUMA effects on the XK6 node
  • Comparing Hybrid OpenMP/MPI to all MPI
  • Using the progress engine for overlapping MPI and computation

• Looking at methods of acceleration
  • Using Cuda with OpenACC and Cuda Fortran, Visual profiler, command line profiler, libsci being used with OpenACC

• A systematic approach for converting a Hybrid OpenMP/MPI application to OpenACC
  • Using OpenACC
  • First, let the compiler do most of the work
  • Using Craypat to identify the most time consuming portions of the accelerated code
  • Optimizing the OpenACC code
    • Most optimizations will improve OpenMP code
  • Employing Cuda and/or Cuda Fortran in an OpenACC application
Programming for Future

Multi-Petaflop and Exaflop Computers

aka
Finding more parallelism in existing applications
• Fact
  • For the next decade all HPC system will basically have the same architecture
    • Message passing between nodes
    • Multi-threading within the node – MPI will not do
    • Vectorization at the lower level

• Fact
  • Current petascale applications are not structured to take advantage of these architectures
    • Current – 80-90% of application use a single level of parallelism, message passing between the cores of the MPP system
    • Looking forward, application developers are faced with a significant task in preparing their applications for the future
Cray’s Programming Environment

- Tools for identifying additional parallel structures within the application
  - Investigation of looping structures within a complex application
  - Scoping tools for investigating the parallelizability of high level looping structures
- Tools for maintaining performance portable applications
  - Supply compiler that accepts directives from OpenMP sub-committee formulating extensions to target companion accelerators
  - Application developer able to develop a single code that can run efficiently on multi-core nodes with or without accelerator
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.
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 cacceler 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
Possible Programming Models for the Node

- One MPI task/node *
  - Cuda
  - OpenCL
- Existing Fortran, C and C++ with extensions
  - Comment line directives
    - OpenMP extensions for Accelerators

All of these programming models require the application developer to replace MPI within the node – to develop Hybrid versions of the application

*Later this year – this will be relaxed

Actually some applications are using more
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.

- Tools that will be needed
  - Advanced instrumentation to measure
    - DO loop statistics, iteration counts, inclusive time
    - Routine level sampling and profiling
### Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>50.553984</td>
<td>--</td>
<td>--</td>
<td>6922023.0</td>
<td>Total</td>
</tr>
<tr>
<td>52.1%</td>
<td>26.353695</td>
<td>--</td>
<td>--</td>
<td>6915004.0</td>
<td>USER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.9%</td>
<td>8.540852</td>
<td>0.366647</td>
<td>4.1%</td>
<td>2592000.0</td>
<td>parabola_</td>
</tr>
<tr>
<td>8.0%</td>
<td>4.034867</td>
<td>0.222303</td>
<td>5.2%</td>
<td>288000.0</td>
<td>remap_</td>
</tr>
<tr>
<td>7.1%</td>
<td>3.612980</td>
<td>0.862830</td>
<td>19.3%</td>
<td>288000.0</td>
<td>riemann_</td>
</tr>
<tr>
<td>3.7%</td>
<td>1.859449</td>
<td>0.094075</td>
<td>4.8%</td>
<td>288000.0</td>
<td>ppmlr_</td>
</tr>
<tr>
<td>3.3%</td>
<td>1.666590</td>
<td>0.064095</td>
<td>3.7%</td>
<td>288000.0</td>
<td>evolve_</td>
</tr>
<tr>
<td>2.6%</td>
<td>1.315145</td>
<td>0.119832</td>
<td>8.4%</td>
<td>576000.0</td>
<td>paraset_</td>
</tr>
<tr>
<td>1.8%</td>
<td>0.923711</td>
<td>0.048359</td>
<td>5.0%</td>
<td>864000.0</td>
<td>volume_</td>
</tr>
<tr>
<td>1.8%</td>
<td>0.890751</td>
<td>0.06495</td>
<td>6.8%</td>
<td>288000.0</td>
<td>states_</td>
</tr>
<tr>
<td>1.4%</td>
<td>0.719636</td>
<td>0.079651</td>
<td>10.0%</td>
<td>288000.0</td>
<td>flatten_</td>
</tr>
<tr>
<td>1.0%</td>
<td>0.513454</td>
<td>0.019075</td>
<td>3.6%</td>
<td>864000.0</td>
<td>forces_</td>
</tr>
<tr>
<td>1.0%</td>
<td>0.504152</td>
<td>0.027139</td>
<td>5.1%</td>
<td>1000.0</td>
<td>sweepz_</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37.9%</td>
<td>19.149499</td>
<td>--</td>
<td>--</td>
<td>3512.0</td>
<td>MPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28.7%</td>
<td>14.487564</td>
<td>0.572138</td>
<td>3.8%</td>
<td>3000.0</td>
<td>mpi_alltoall</td>
</tr>
<tr>
<td>8.7%</td>
<td>4.391205</td>
<td>2.885755</td>
<td>39.7%</td>
<td>2.0</td>
<td>mpi_comm_split</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0%</td>
<td>5.050780</td>
<td>--</td>
<td>--</td>
<td>3502.0</td>
<td>MPI_SYNC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.9%</td>
<td>3.483206</td>
<td>1.813952</td>
<td>52.1%</td>
<td>3000.0</td>
<td>mpi_alltoall_(sync)</td>
</tr>
<tr>
<td>3.1%</td>
<td>1.567285</td>
<td>0.606728</td>
<td>38.7%</td>
<td>501.0</td>
<td>mpi_allreduce_(sync)</td>
</tr>
</tbody>
</table>
Normal Profile – Using “setenv PAT_RT_HWPC 1”

<table>
<thead>
<tr>
<th>USER / parabola_</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
</tr>
<tr>
<td>Time</td>
</tr>
<tr>
<td>Imb. Time</td>
</tr>
<tr>
<td>Imb. Time%</td>
</tr>
<tr>
<td>Calls</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
</tr>
<tr>
<td>Average Time per Call</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
</tr>
<tr>
<td>User time (approx)</td>
</tr>
<tr>
<td>HW FP Ops / User time</td>
</tr>
<tr>
<td>HW FP Ops / WCT</td>
</tr>
<tr>
<td>Computational intensity</td>
</tr>
<tr>
<td>MFLOPS (aggregate)</td>
</tr>
<tr>
<td>TLB utilization</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
</tr>
</tbody>
</table>

CUG Stuttgart April 30, 2012
Re-compiling with --hprofile_generate

<table>
<thead>
<tr>
<th>Loop Incl Time</th>
<th>Loop Hit</th>
<th>Loop Trips</th>
<th>Loop Trips</th>
<th>Function=/&gt;.LOOP[.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>51.107386</td>
<td>500</td>
<td>0</td>
<td>16</td>
<td>sweepx2_.LOOP.1.li.34</td>
</tr>
<tr>
<td>51.10682</td>
<td>8000</td>
<td>0</td>
<td>16</td>
<td>sweepx2_.LOOP.2.li.35</td>
</tr>
<tr>
<td>50.373481</td>
<td>500</td>
<td>0</td>
<td>16</td>
<td>sweepx1_.LOOP.1.li.34</td>
</tr>
<tr>
<td>50.372915</td>
<td>8000</td>
<td>0</td>
<td>16</td>
<td>sweepx1_.LOOP.2.li.35</td>
</tr>
<tr>
<td>12.480442</td>
<td>1000</td>
<td>0</td>
<td>16</td>
<td>sweepy_.LOOP.1.li.38</td>
</tr>
<tr>
<td>12.478967</td>
<td>16000</td>
<td>0</td>
<td>1</td>
<td>sweepy_.LOOP.2.li.39</td>
</tr>
<tr>
<td>11.949236</td>
<td>500</td>
<td>0</td>
<td>16</td>
<td>sweepz_.LOOP.05.li.54</td>
</tr>
<tr>
<td>11.948618</td>
<td>8000</td>
<td>0</td>
<td>1</td>
<td>sweepz_.LOOP.06.li.55</td>
</tr>
<tr>
<td>5.479066</td>
<td>288000</td>
<td>0</td>
<td>1031</td>
<td>riemann_.LOOP.2.li.63</td>
</tr>
<tr>
<td>3.082245</td>
<td>51168000</td>
<td>0</td>
<td>12</td>
<td>riemann_.LOOP.3.li.64</td>
</tr>
<tr>
<td>1.796424</td>
<td>2592000</td>
<td>0</td>
<td>1032</td>
<td>parabola_.LOOP.6.li.67</td>
</tr>
<tr>
<td>1.503023</td>
<td>2592000</td>
<td>0</td>
<td>1034</td>
<td>parabola_.LOOP.2.li.30</td>
</tr>
<tr>
<td>1.377911</td>
<td>2592000</td>
<td>0</td>
<td>1032</td>
<td>parabola_.LOOP.7.li.75</td>
</tr>
<tr>
<td>1.094964</td>
<td>2592000</td>
<td>0</td>
<td>1033</td>
<td>parabola_.LOOP.4.li.44</td>
</tr>
<tr>
<td>0.815105</td>
<td>288000</td>
<td>0</td>
<td>1025</td>
<td>remap_.LOOP.7.li.83</td>
</tr>
<tr>
<td>0.792899</td>
<td>2592000</td>
<td>0</td>
<td>1032</td>
<td>parabola_.LOOP.5.li.53</td>
</tr>
<tr>
<td>0.76888</td>
<td>2592000</td>
<td>0</td>
<td>1032</td>
<td>parabola_.LOOP.8.li.84</td>
</tr>
<tr>
<td>0.590497</td>
<td>128000</td>
<td>0</td>
<td>64</td>
<td>sweepx2_.LOOP.3.li.38</td>
</tr>
<tr>
<td>0.505536</td>
<td>288000</td>
<td>0</td>
<td>1031</td>
<td>riemann_.LOOP.1.li.44</td>
</tr>
<tr>
<td>0.478305</td>
<td>2592000</td>
<td>0</td>
<td>1034</td>
<td>parabola_.LOOP.3.li.36</td>
</tr>
<tr>
<td>0.465781</td>
<td>2592000</td>
<td>0</td>
<td>1035</td>
<td>parabola_.LOOP.1.li.24</td>
</tr>
<tr>
<td>0.463514</td>
<td>576000</td>
<td>0</td>
<td>1036</td>
<td>paraset_.LOOP.1.li.117</td>
</tr>
<tr>
<td>0.362512</td>
<td>288000</td>
<td>0</td>
<td>1032</td>
<td>states_.LOOP.2.li.64</td>
</tr>
<tr>
<td>0.338868</td>
<td>288000</td>
<td>0</td>
<td>1030</td>
<td>evolve_.LOOP.4.li.70</td>
</tr>
<tr>
<td>0.335398</td>
<td>288000</td>
<td>0</td>
<td>1026</td>
<td>remap_.LOOP.8.li.111</td>
</tr>
</tbody>
</table>
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_ |
| 7.1% | 8.360290 | 1728000.0 | remap_.LOOPS |
| | | | remap_ |
| | | | ppmlr_ |
| 3.2% | 3.708452 | 768000.0 | sweepx2_.LOOP.2.li.35 |
| 3.1% | 3.663423 | 768000.0 | sweepx1_.LOOP.2.li.35 |
| 3.6% | 4.229443 | 864000.0 | ppmlr_ |
| 1.6% | 1.880874 | 384000.0 | sweepx2_.LOOP.2.li.35 |
| 1.6% | 1.852820 | 384000.0 | sweepx1_.LOOP.2.li.35 |

CUG Stuttgart April 30, 2012
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
Task 2 Parallel Analysis, Scoping and Vectorization (Cont)

- Current scoping tool, -homp_analyze, is meant to interface to a code restructuring GUI called “reveal”. At this time, we need to use cryptic output and massage it with editor/script.
  - !dir$ omp_analyze_loop

- In order to utilize scoping tool for loops that contain procedures the program library need to be employed
  - -hwp –hpl=vhone.aid
    - This will do an initial pass of the code, checking for error and then at the load it will build the program library and perform the analysis

- Compiler will be very conservative
  - <object_message kind="warn">LastPrivate of array may be very expensive.</object_message>
Current output of –homp_analyze

<construct kind="loop" begin_line="54" end_line="119">
  <construct_message kind="warn">Call or I/O at line 100 of sweepz.f90</construct_message>
  <construct_message kind="warn">Call or I/O at line 84 of sweepz.f90</construct_message>
  <object state="known">
    <symbol name="dotflo"/>
    <scope source="compiler"> <shared/> </scope>
  </object>
  <object state="known">
    <symbol name="dt"/>
    <scope source="compiler"> <shared/> </scope>
  </object>
  <object state="known">
    <symbol name="dvol"/>
    <scope source="compiler"> <private first="true" last="true"/> </scope>
    <object_message kind="warn">LastPrivate of array may be very expensive.</object_message>
  </object>
  <object state="known">
    <symbol name="dx"/>
    <scope source="compiler"> <private first="true" last="true"/> </scope>
    <object_message kind="warn">LastPrivate of array may be very expensive.</object_message>
  </object>
  <object state="known">
    <symbol name="dx0"/>
    <scope source="compiler"> <private first="true" last="true"/> </scope>
    <object_message kind="warn">LastPrivate of array may be very expensive.</object_message>
  </object>
  <object state="known">
    <symbol name="e"/>
    <scope source="compiler"> <private first="true" last="true"/> </scope>
    <object_message kind="warn">LastPrivate of array may be very expensive.</object_message>
  </object>
</construct>
Main window of reveal
### Scoping window

#### OpenMP Construct

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scope</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>zyc</td>
<td>Scalar</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>ai</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>amid</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>ar</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>bi</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>cdt dx</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>ci</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>cfft</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>crgh</td>
<td>Scalar</td>
<td>Private</td>
<td></td>
</tr>
</tbody>
</table>

- **First/Last Private**
  - [ ] Enable First Private
  - [ ] Enable Last Private

- **Reduction**
  - None

- **Search**

- **Dump Data**

- **Close**
What we end up finding out

**Private Variables in module, need to use Threadprivate**

```fortran
!$omp threadprivate (r, p, e, q, u, v, w, xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta)
real, dimension(maxsweep) :: r, p, e, q, u, v, w             ! fluid variables
real, dimension(maxsweep) :: xa, xa0, dx, dx0, dvol         ! coordinate values
real, dimension(maxsweep) :: f, flat                       ! flattening parameter
real, dimension(maxsweep,5) :: para                        ! parabolic interpolation coefficients
real :: radius, theta, stheta

Reduction variable down callchain, need to use

!$OMP CRITICAL;!$OMP END CRITICAL

hdt = 0.5*dt
do n = nmin-4, nmax+4
  Cdtdx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
enddo
!$omp critical
do n = nmin-4, nmax+4
  svel = max(svel, Cdtdx(n))
enddo
!$omp end critical
do n = nmin-4, nmax+4
  Cdtdx (n) = Cdtdx(n)*hdt
  fCdtdx(n) = 1. - fourthd*Cdtdx(n)
enddo
```
Differences in OpenMP version

```fortran
module zone

! (formerly zone.h) global (3D) data arrays

INTEGER, PARAMETER :: imax = 64, jmax = 1024, kmax = 1024 ! Memory dimensions
INTEGER, PARAMETER :: pey = 64, pez = 64 ! number of MPI tasks

## for 2D: ^^^ IF kmax=1, MUST HAVE pez=1 ##
INTEGER, PARAMETER :: nvar = 6 ! number of primitive fluid variables

INTEGER :: isy, isz, js, ks, Ya2abuff_size, Za2abuff_size
INTEGER :: npe, npey, npez, mype, mypez ! # of pes and local pe numbers
INTEGER :: MPI_COMM_ROW, MPI_COMM_COL
INTEGER :: VHI_DATATYPE

INTEGER :: ngeomx, ngeomy, ngeomz ! XYZ Geometry flag
INTEGER :: nleftx, nlefty, nleftz ! XYZ Lower Boundary Condition
INTEGER :: nrightx, nrighty, nrightz ! XYZ Upper Boundary Condition

REAL, DIMENSION(imax, jmax, pey, kmax/pez) :: zro, zpr, zux, zuy, zuz, zfl
REAL, DIMENSION(imax) :: zxa, zdx, zxc
REAL, DIMENSION(jmax) :: zya, zdy, zyc
REAL, DIMENSION(kmax) :: zza, zdz, zzc

REAL, DIMENSION(nvar, kmax/pez, jmax/pey, imax) :: send1
REAL, DIMENSION(nvar, kmax/pez, imax/pey, jmax) :: send2
REAL, DIMENSION(nvar, jmax/pey, kmax/pez, imax) :: send3
REAL, DIMENSION(ivar, jmax/pey, imax/pey, kmax) :: send4

```

```fortran
module zone

! (formerly zone.h) global (3D) data arrays

INTEGER, PARAMETER :: imax = 64, jmax = 1024, kmax = 1024 ! Memory dimensions
INTEGER, PARAMETER :: pey = 16, pez = 16 ! number of MPI tasks

## for 2D: ^^^ IF kmax=1, MUST HAVE pez=1 ##
INTEGER, PARAMETER :: nvar = 6 ! number of primitive fluid variables

INTEGER :: isy, isz, js, ks, Ya2abuff_size, Za2abuff_size
INTEGER :: npe, npey, npez, mype, mypez ! # of pes and local pe numbers
INTEGER :: MPI_COMM_ROW, MPI_COMM_COL
INTEGER :: VHI_DATATYPE

INTEGER :: ngeomx, ngeomy, ngeomz ! XYZ Geometry flag
INTEGER :: nleftx, nlefty, nleftz ! XYZ Lower Boundary Condition
INTEGER :: nrightx, nrighty, nrightz ! XYZ Upper Boundary Condition

REAL, DIMENSION(imax, jmax, pey, kmax/pez) :: zro, zpr, zux, zuy, zuz, zfl
REAL, DIMENSION(imax) :: zxa, zdx, zxc
REAL, DIMENSION(jmax) :: zya, zdy, zyc
REAL, DIMENSION(kmax) :: zza, zdz, zzc

REAL, DIMENSION(nvar, kmax/pez, jmax/pey, imax) :: send1
REAL, DIMENSION(nvar, kmax/pez, imax/pey, jmax) :: send2
REAL, DIMENSION(nvar, jmax/pey, kmax/pez, imax) :: send3
REAL, DIMENSION(ivar, jmax/pey, imax/pey, kmax) :: send4
```

CUG Stuttgart April 30, 2012
Differences in OpenMP version

module sweeplsize
! Dimension of 1D sweeps. maxsweep must be as long as the longest of the
! 3D arrays PLUS the ghost zones:
! ie, must have maxsweep >= max(1m, jmax, kmax) + 12
integer, parameter :: maxsweep=1050
end module sweeplsize

module sweep
! Data structures used in 1D sweeps, dimensioned maxsweep (set in sweeplsize)
use sweeplsize

character(len=1) :: sweep ! direction of
integer :: nmin, nmax, ngeom, nleft, nright ! number of fre
real, dimension(maxsweep) :: r, p, e, q, u, v, w ! fluid variabl
real, dimension(maxsweep) :: xa, xa0, dx, dx0, dvol ! coordinate va
real, dimension(maxsweep) :: f, flat ! flattening pa
real, dimension(maxsweep,5) :: para ! parabolic int
real :: radius, theta, stheta !
end module sweep
Differences in OpenMP version

```plaintext
!$omp parallel do private(i,j,k,n)
do k = 1, ks
do j = 1, js

! Put state variables into 1D arrays, padding with 6 ghost zones
do i = 1,imax
n = i + 6
r(n) = zro(i,j,k)
p(n) = zpr(i,j,k)
u(n) = zux(i,j,k)
v(n) = zuy(i,j,k)
w(n) = zuz(i,j,k)
f(n) = zfli(i,j,k)
enddo

! Do 1D hydro update using PPMLR
call ppmlr
#else DEBUGX
print *, 'In sweepx', svel
endif
```
Differences in runtime

All MPI on 4096 cores 43.01 seconds
Hybrid 256 nodes x 16 threads 45.05 seconds
Things that are different between OpenMP and OpenACC

- Cannot have CRITICAL REGION down callchain
- Cannot have THREADPRIVATE
- Vectorization is much more important
- Cache/Memory Optimization much more important
- No EQUIVALENCE

Currently both OpenMP and OpenACC must be included in the source

```c
#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
```
Changes to remove THREADPRIVATE and CRITICAL REGION

```plaintext
xa (n) = zxa(i)
dx (n) = zdx(i)
p (n) = max(smallp,p(n))
e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
enddo
!
! Do 1D hydro update using PPMLR

! call ppmlr (svel0, sweep, nmin, nmax, ngeom, nleft, nright, r, p, e, q, u,
! xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, stheta)

do n = nmin-4, nmax+4
  svel = max(svel,svel0(n))
enddo

#ifdef DEBUGX
  print *, 'In sweepx1', svel
#endif

! Put updated values into send buffer for sweepy, dropping ghost zones
do i = 1, imax
  n = i + 6
  send1(1,k,j,i) = r(n)
  send1(2,k,j,i) = p(n)
  send1(3,k,j,i) = u(n)
  send1(4,k,j,i) = v(n)
  send1(5,k,j,i) = w(n)
  send1(6,k,j,i) = f(n)
enddo
```

```plaintext
xa (n) = zxa(i)
dx (n) = zdx(i)
p (n) = max(smallp,p(n))
e (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
enddo
!
! Do 1D hydro update using PPMLR

! call ppmlr

#ifdef DEBUGX
  print *, 'In sweepx1', svel
#endif

! Put updated values into send buffer for sweepy, dropping ghost zones
do i = 1, imax
  n = i + 6
  send1(1,k,j,i) = r(n)
  send1(2,k,j,i) = p(n)
  send1(3,k,j,i) = u(n)
  send1(4,k,j,i) = v(n)
  send1(5,k,j,i) = w(n)
  send1(6,k,j,i) = f(n)
enddo
```
Differences in runtime

- All MPI on 4096 cores: 43.01 seconds
- Hybrid 256 nodesx16 threads: 45.05 seconds
- Rest Hybrid 256x16 threads: 48.03 seconds
• 2 Multi-Chip Modules, 4 Opteron Dies
• 8 Channels of DDR3 Bandwidth to 8 DIMMs
• 24 (or 16) Computational Cores
  • 64 KB L1 and 512 KB L2 caches for each core
  • 6 MB of shared L3 cache on each die
• Dies are fully connected with HT3
• Snoop Filter Feature Allows 4 Die SMP to scale well
By default MPI messaging is not synchronous; however, the MPI library has been modified to utilize Cray core specialization to execute a progress engine for overlapping message passing with computation. Especially important for large messages that come from using OpenMP on the node.

```bash
#PBS -N S3D
#PBS -j oe

setenv MPICH_NEMESIS_ASYNC_PROGRESS 1
setenv MPICH_MAX_THREAD_SAFETY multiple
cd $PBS_O_WORKDIR
cp -f ../input/s3d.in.110592_64 ../input/s3d.in
setenv OMP_NUM_THREADS 14
aprun -n 64 -N 1 -d 14 -r 2 ../oldbuild/s3d.x
```

- set two environment vars
- give MPI two cores
Resultant Hybrid S3D Performance

Weak Scaling, Lower is Better

- **Wall Clock Time (Seconds)**
  - **Hybrid S3D**
  - **All MPI S3D**

**Number of cores (12 cores/node)**

CUG Stuttgart April 30, 2012
Evaluating Methods of Acceleration

Accelerated/Heterogeneous computing is still the wild west, programmers have many options with varying

- Maturity
- Ease of use
- Portability
- Performance

The most commonly considered paradigms are

- CUDA for C (Nvidia)
- OpenCL (Nvidia and others)
- CUDA Fortran (Portland Group)
- Directives (Cray, PGI, CAPS, and others)
- Libraries (Cray, CULA, MAGMA, and others)
CUDA for C

Parallel programming extensions to C/C++ designed and supported by Nvidia

Pros:
• Mature
• High performance
• Widely taught
• Strong tool support

Cons:
• Nvidia Only
• Learning Curve
• Duplicate code
OpenCL

Standard for parallel programming in C/C++ with strong focus on portability

Pros:
• Portable code (multiple vendors, variety of accelerators and CPUs)
• Improving tool support

Cons:
• High learning curve with significant boilerplate code
• Duplicate code
• Performance portability still to be proven
CUDA Fortran

CUDA extensions for FORTRAN developed by PGI and Nvidia.

Pros:
- Performance comparable with CUDA C while remaining in FORTRAN
- FORTRAN syntactic sugar

Cons:
- Nvidia and PGI Only
- Learning Curve
- Duplicate code
Directives

Hints embedded within the source code to guide compiler to generate accelerator code.

Pros:
- Single source code
- Multiple vendor support (OpenACC)
- Lower learning curve
- Can integrate with existing CUDA code

Cons:
- Immature
- Performance will typically lag CUDA slightly

Notice nothing here says “automatic.” Directives still require work by the programmer.
Libraries

Standard interfaces, such as BLAS, LAPACK, and FFTs that have been accelerated and packaged in a library.

Pros:
• Requires little/no code modification
• The *hard work* has been done by someone else
• Integrates with both CUDA and Directives

Cons:
• Immature
• Completely opaque to user
## Comparison of Programming Models

<table>
<thead>
<tr>
<th>Name</th>
<th>Ease of Use</th>
<th>Portability</th>
<th>Performance</th>
<th>Maturity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA for C</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>OpenCL</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>CUDA Fortran</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Directives</td>
<td>4</td>
<td>4 (improving)</td>
<td>3 (improving)</td>
<td>2</td>
</tr>
<tr>
<td>Libraries</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Scale: 1 - Low, 5 - High
CUDA Command-line Profiler

The CUDA Command-line Profiler can be enabled for any accelerated executable, regardless of programming model.

Enabling the Profiler:

```
export CUDA_PROFILE=1 or setenv CUDA_PROFILE 1
```

Customizing the Profile

1. Create a text file listing the events to record
2. Set the environment variable `CUDA_PROFILE_CONFIG` to point to this file.

Only a limited number of counters can be active in a given run, so it may be necessary to make multiple runs and combine the data.
The CUDA Command Line Profiler can output data for the CUDA Visual Profiler.

Set `CUDA_PROFILE_CSV` to 1 to output data in CSV format.

Choose import from CUDA Visual Profiler and select this file.

**NOTE:** CUDA Visual Profiler 4.0 is unable to handle very large files, so it should be used from short runs or kernels. Version 4.1 is able to accept larger files, but requires a custom `CUDA_PROFILE_CONFIG` that includes the `gpustarttimestamp` event.

**HINT:** CUDA Profiler 4.0 is not MPI-aware, so all processes write to the same file. This gets better in 4.1. If you get an error about CSV formatting, this is probably why, check the end of the file for junk.
Provide basic scientific libraries optimized for hybrid CPU and accelerator systems (XK6)
Independent to, but fully compatible with openACC
Dual goal:
1. Base performance of GPU with minimal (or no) code change
   libsci_acc simple interface
2. Advanced performance of the GPU with controls for data movement
   libsci_acc device interface
does not imply that always need expert interfaces to get great performance
Simple interface

Supports the standard API in original form
Will perform all GPU dirty-work for you

- Initialize data structures on GPU
- Split your problem into a CPU portion and GPU portion
- Copy data to the GPU memory from CPU memory
- Perform GPU and CPU operations
- Copy data back to CPU memory

Library-heavy codes can use GPUs with no code change
Is not only a tool for simple usage

- If you don’t need the data on GPU afterwards, use the simple interface

Simple API has automatic adaptation
You can pass either host pointers or device pointers to simple interface

A is host memory

\[ \text{dgetrf}(M, N, A, \text{lda}, \text{ipiv}, \ & \text{info}) \]

- Performs hybrid operation on GPU
- if problem is too small, performs host operation

Pass Device memory

\[ \text{dgetrf}(M, N, d\ A, \text{lda}, \text{ipiv}, \ & \text{info}) \]

- Performs hybrid operation on GPU

BLAS 1 and 2 performs computation local to the data location

- CPU-GPU data transfer is too expensive to exploit hybrid execution
Device interface

Device interface gives higher degrees of control
Requires that you have already copied your data to
the device memory

API

- Every routine in libsci has a version with _acc suffix
- E.g. dgetrf_acc
- This resembles standard API except for the suffix and the device pointers
Sometimes apps may want to force ops on the CPU
- Need to preserve GPU memory
- Want to perform something in parallel
- Don’t want to incur transfer cost for a small op

can force any operation to occur on CPU with _cpu version
Every routine has a _cpu entry-point
API is exactly standard otherwise
## BLAS
- [s,d,c,z]GEMM
- [s,d,c,z]TRSM
- [z,c]HEMM
- [s,d,c,z]SYMM
- [s,d,c,z]SYRK
- [z,d]HERK
- [s,d,c,z]SYR2K
- [s,d,c,z]TRMM
- All level 2 BLAS
- All level 1 BLAS

## LAPACK
- [d,z]GETRF
- [d,z]GETRS
- [d,z]POTRF
- [d,z]POTRS
- [d,z]GESDD
- [d,z]GEBRD
- [d,z]GEQRF
- [d,z]GELQF
Targets

- **BLAS**
  - [s,d,c,z]GEMM
  - [s,d,c,z]TRSM
  - [z,c]HEMM
  - [s,d]SYMM
  - [s,d,c,z]SYRK
  - [z,d]HERK
  - [s,d,c,z]SYR2K
  - [s,d,c,z]TRMM
  - All level 2 BLAS
  - All level 1 BLAS

- **LAPACK**
  - [d,z]GETRF
  - [d,z]GETRS
  - [d,z]POTRF
  - [d,z]POTRS
  - [d,z]GESDD
  - [d,z]GESDD
  - [d,z]GEBRD
  - [d,z]GEQRF
  - [d,z]GELQF

**Full-HYBRID**  **HYBRID is planned**  **No HYBRID**
Libsci_acc Example

Starting with a code that relies on dgemm.

The library will check the parameters at runtime.

If the size of the matrix multiply is large enough, the library will run it on the GPU, handling all data movement behind the scenes.

NOTE: Input and Output data are in CPU memory.

call dgemm('n','n',m,n,k,alpha,&a,lda,b,ldb,beta,c,ldc)
Libsci_acc Example

If the rest of the code uses OpenACC, it’s possible to use the library with directives.

All data management performed by OpenACC.

Calls the device version of dgemm.

All data is in CPU memory before and after data region.

```
!$acc data copy(a,b,c)

!$acc parallel
!Do Something
!$acc end parallel

!$acc host_data use_device(a,b,c)
call dgemm_acc('n','n',m,n,k,&
    alpha,a,lda,&
    b,ldb,beta,c,ldc)

!$acc end host_data
!$acc end data
```
Libsci_acc Example

Libsci_acc is a bit smarter than this.

Since ‘a,’ ‘b’, and ‘c’ are device arrays, the library knows it should run on the device.

So just dgemm is sufficient.

```fortran
$acc data copy(a,b,c)
$acc parallel
!Do Something
$acc end parallel

$acc host_data use_device(a,b,c)
call dgemm ('n','n',m,n,k,&
   alpha,a,lda,&
   b,ldb,beta,c,ldc)

$acc end host_data
$acc end data
```
NVIDIA, Cray, PGI, CAPS Unveil ‘OpenACC’ Programming Standard for Parallel Computing

Directives-based Programming Makes Accelerating Applications Using CPUs and GPUs Dramatically Easier
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, permanence
- 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
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 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
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.

```c
#ifndef GPU
$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
                           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,&
                           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 x, xa0, dx, dx0, dvol, f, flat, para, radius, theta, sttheta)&
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 !$omp& xa, xa0, dx, dx0, dvol, f, flat, para, radius, theta, sttheta)&
52. G !$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     sttheta=0.0
59. G g 3     radius=0.0
60. G g 3 g-----< do i = 1,imax
61. G g 3 g     n = i + 6
62. G g 3 g     r (n) = zro(i,j,k)
63. G g 3 g     p (n) = zpr(i,j,k)
64. G g 3 g     u (n) = zux(i,j,k)
65. G g 3 g     v (n) = zuy(i,j,k)
66. G g 3 g     w (n) = zuz(i,j,k)
67. G g 3 g     f (n) = zfl(i,j,k)
68. G g 3 g     xa0(n) = zxa(i)
69. G g 3 g     dx0(n) = zdx(i)
70. G g 3 g     xa (n) = zxa(i)
71. G g 3 g     dx (n) = zdx(i)
72. G g 3 g     p  (n) = max(smallp,p(n))
73. G g 3 g     e  (n) = p(n)/(r(n)*gamm)+0.5*(u(n)**2+v(n)**2+w(n)**2)
74. G g 3 g------> enddo
77. G g 3 g------> ! 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, sttheta)
A region starting at line 46 and ending at line 104 was placed on the accelerator.

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

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

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

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

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

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

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?
### Table 1: Time and Bytes Transferred for Accelerator Regions

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

---

**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 SIMT 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
Improving Vectorization

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
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
Lets look at some other examples
### Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time %</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb. Time %</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>83.277477</td>
<td>--</td>
<td>--</td>
<td>851.0</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td><strong>---</strong></td>
<td><strong>---</strong></td>
<td><strong>---</strong></td>
<td><strong>---</strong></td>
<td><strong>---</strong></td>
</tr>
<tr>
<td>51.3%</td>
<td>42.762837</td>
<td>--</td>
<td>--</td>
<td>703.0</td>
<td>ACCELERATOR</td>
</tr>
<tr>
<td></td>
<td><strong>---</strong></td>
<td><strong>---</strong></td>
<td><strong>---</strong></td>
<td><strong>---</strong></td>
<td><strong>---</strong></td>
</tr>
<tr>
<td><strong>18.8%</strong></td>
<td><strong>15.672371</strong></td>
<td><strong>1.146276</strong></td>
<td><strong>7.3%</strong></td>
<td>20.0</td>
<td>recolor_.SYNC_COPY@li.790 not good</td>
</tr>
<tr>
<td><strong>16.3%</strong></td>
<td><strong>13.585707</strong></td>
<td><strong>0.404190</strong></td>
<td><strong>3.1%</strong></td>
<td>20.0</td>
<td>recolor_.SYNC_COPY@li.793 not good</td>
</tr>
<tr>
<td><strong>7.5%</strong></td>
<td><strong>6.216010</strong></td>
<td><strong>0.873830</strong></td>
<td><strong>13.1%</strong></td>
<td>20.0</td>
<td>lbm3d2p_d_.ASYNC_KERNEL@li.116</td>
</tr>
<tr>
<td><strong>1.6%</strong></td>
<td><strong>1.337119</strong></td>
<td><strong>0.193826</strong></td>
<td><strong>13.5%</strong></td>
<td>20.0</td>
<td>lbm3d2p_d_.ASYNC_KERNEL@li.119</td>
</tr>
<tr>
<td><strong>1.6%</strong></td>
<td><strong>1.322690</strong></td>
<td><strong>0.059387</strong></td>
<td><strong>4.6%</strong></td>
<td>1.0</td>
<td>lbm3d2p_d_.ASYNC_COPY@li.100</td>
</tr>
<tr>
<td><strong>1.0%</strong></td>
<td><strong>0.857149</strong></td>
<td><strong>0.245369</strong></td>
<td><strong>23.7%</strong></td>
<td>20.0</td>
<td>collisionb_.ASYNC_KERNEL@li.586</td>
</tr>
<tr>
<td><strong>1.0%</strong></td>
<td><strong>0.822911</strong></td>
<td><strong>0.172468</strong></td>
<td><strong>18.5%</strong></td>
<td>20.0</td>
<td>lbm3d2p_d_.ASYNC_KERNEL@li.114</td>
</tr>
<tr>
<td><strong>0.9%</strong></td>
<td><strong>0.786618</strong></td>
<td><strong>0.386807</strong></td>
<td><strong>35.2%</strong></td>
<td>20.0</td>
<td>injection_.ASYNC_KERNEL@li.1119</td>
</tr>
<tr>
<td><strong>0.9%</strong></td>
<td><strong>0.727451</strong></td>
<td><strong>0.221332</strong></td>
<td><strong>24.9%</strong></td>
<td>20.0</td>
<td>lbm3d2p_d_.ASYNC_KERNEL@li.118</td>
</tr>
</tbody>
</table>
Keep data on the accelerator with acc_data region

```fortran
!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,& !$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

```fortran
!$acc parallel_loop private(k,j,i)
  do j=0,local_ly-1
    do i=0,local_lx-1
      if (cell(i,j,0)==1) then
        grad (i,j,-1) = (1.0d0-wet)*db*press
      else
        grad (i,j,-1) = db*press
      end if
    grad (i,j,lz) = grad(i,j,lz-1)
    end do
  end do
!$acc end parallel_loop
!$acc update host(grad)
!$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
!$acc update host(send_e,send_w)
call mpi_irecv(recv_w, bufsize(2),mpi_double_precision,w_id, &
  tag(25),mpi_comm_world,irequest_in(25),ierr)
call mpi_isend(send_w, bufsize(2),mpi_double_precision,w_id, &
  tag(26),& mpi_comm_world,irequest_out(25),ierr)
call mpi_waitall(2,irequest_in(25),istatus_req,ierr)
call mpi_waitall(2,irequest_out(25),istatus_req,ierr)
!$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)
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>37.9%</td>
<td>236.592782</td>
<td>--</td>
<td>--</td>
<td>11403.0</td>
<td>ACCELERATOR</td>
<td></td>
</tr>
<tr>
<td>15.7%</td>
<td>98.021619</td>
<td>43.078137</td>
<td>31.0%</td>
<td>200.0</td>
<td>lbm3d2p_d_.ASYNC_KERNEL@li.129</td>
<td></td>
</tr>
<tr>
<td>3.7%</td>
<td>23.359080</td>
<td>2.072147</td>
<td>8.3%</td>
<td>200.0</td>
<td>lbm3d2p_d_.ASYNC_KERNEL@li.127</td>
<td></td>
</tr>
<tr>
<td>3.6%</td>
<td>22.326085</td>
<td>1.469419</td>
<td>6.3%</td>
<td>200.0</td>
<td>lbm3d2p_d_.ASYNC_KERNEL@li.132</td>
<td></td>
</tr>
<tr>
<td>3.0%</td>
<td>19.035232</td>
<td>1.464608</td>
<td>7.3%</td>
<td>200.0</td>
<td>collisionb_.ASYNC_KERNEL@li.599</td>
<td></td>
</tr>
<tr>
<td>2.6%</td>
<td>16.216648</td>
<td>3.505232</td>
<td>18.1%</td>
<td>200.0</td>
<td>lbm3d2p_d_.ASYNC_KERNEL@li.131</td>
<td></td>
</tr>
<tr>
<td>2.5%</td>
<td>15.401916</td>
<td>8.093716</td>
<td>35.0%</td>
<td>200.0</td>
<td>injection_.ASYNC_KERNEL@li.1116</td>
<td></td>
</tr>
<tr>
<td>1.9%</td>
<td>11.734026</td>
<td>4.488785</td>
<td>28.1%</td>
<td>200.0</td>
<td>recolor_.ASYNC_KERNEL@li.786</td>
<td></td>
</tr>
<tr>
<td>0.9%</td>
<td>5.530201</td>
<td>2.132243</td>
<td>28.3%</td>
<td>200.0</td>
<td>collisionb_.SYNC_COPY@li.593</td>
<td></td>
</tr>
<tr>
<td>0.8%</td>
<td>4.714995</td>
<td>0.518495</td>
<td>10.1%</td>
<td>200.0</td>
<td>collisionb_.SYNC_COPY@li.596</td>
<td></td>
</tr>
<tr>
<td>0.6%</td>
<td>3.738615</td>
<td>2.986891</td>
<td>45.1%</td>
<td>200.0</td>
<td>collisionb_.ASYNC_KERNEL@li.568</td>
<td></td>
</tr>
<tr>
<td>0.4%</td>
<td>2.656962</td>
<td>0.454093</td>
<td>14.8%</td>
<td>1.0</td>
<td>lbm3d2p_d_.ASYNC_COPY@li.100</td>
<td></td>
</tr>
<tr>
<td>0.4%</td>
<td>2.489231</td>
<td>2.409892</td>
<td>50.0%</td>
<td>200.0</td>
<td>streaming_exchange_.ASYNC_COPY@li.810</td>
<td></td>
</tr>
<tr>
<td>0.4%</td>
<td>2.487132</td>
<td>2.311190</td>
<td>48.9%</td>
<td>200.0</td>
<td>streaming_exchange_.ASYNC_COPY@li.625</td>
<td></td>
</tr>
<tr>
<td>0.2%</td>
<td>1.322791</td>
<td>0.510645</td>
<td>28.3%</td>
<td>200.0</td>
<td>streaming_exchange_.SYNC_COPY@li.622</td>
<td></td>
</tr>
<tr>
<td>0.2%</td>
<td>1.273771</td>
<td>0.288743</td>
<td>18.8%</td>
<td>200.0</td>
<td>streaming_exchange_.SYNC_COPY@li.574</td>
<td></td>
</tr>
<tr>
<td>0.2%</td>
<td>1.212260</td>
<td>0.298053</td>
<td>20.0%</td>
<td>200.0</td>
<td>streaming_exchange_.SYNC_COPY@li.759</td>
<td></td>
</tr>
<tr>
<td>0.2%</td>
<td>1.208250</td>
<td>0.422182</td>
<td>26.3%</td>
<td>200.0</td>
<td>streaming_exchange_.SYNC_COPY@li.806</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.696120</td>
<td>0.442372</td>
<td>39.5%</td>
<td>200.0</td>
<td>streaming_exchange_.ASYNC_KERNEL@li.625</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.624982</td>
<td>0.379697</td>
<td>38.4%</td>
<td>200.0</td>
<td>streaming_exchange_.ASYNC_KERNEL@li.525</td>
<td></td>
</tr>
</tbody>
</table>
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
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
WL-LSMS
Role of material disorder, statistics, and fluctuations in nanoscale materials and systems.

S3D
How are going to efficiently burn next generation diesel/biofuels?

PFLOTRAN
Stability and viability of large scale CO$_2$ 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

LAMMPS
Biofuels: An atomistic model of cellulose (blue) surrounded by lignin molecules comprising a total of 3.3 million atoms. Water not shown.

Denovo
Unprecedented high-fidelity radiation transport calculations that can be used in a variety of nuclear energy and technology applications.
S3D – A DNS solver

- Structured Cartesian mesh flow solver
- Solves compressible reacting Navier-Stokes, energy and species conservation equations.
  - 8th order explicit finite difference method
    - 4th order Runge-Kutta integrator with error estimator
- Detailed gas-phase thermodynamic, chemistry and molecular transport property evaluations
- Lagrangian particle tracking
- MPI-1 based spatial decomposition and parallelism
- Fortran code. Does not need linear algebra, FFT or solver libraries.

Developed and maintained at CRF, Sandia (Livermore) with BES and ASCR sponsorship. PI – Jacqueline H. Chen (jhchen@sandia.gov)
Benchmark Problem and Profile

- A benchmark problem was defined to closely resemble the target simulation
  - 52 species n-heptane chemistry and $48^3$ grid points per node

- $48^3 \times 18,500$ nodes = 2 billion grid points
- Target problem would take two months on today’s Jaguar

- Code was benchmarked and profiled on dual-hex core XT5
- Several kernels identified and extracted into stand-alone driver programs
Team:

Ramanan Sankaran ORNL
Ray Grout NREL
John Levesque Cray

Goals:

▪ Convert S3D to a hybrid multi-core application suited for a multi-core node with or without an accelerator.
▪ Be able to perform the computation entirely on the accelerator.
  - Arrays and data able to reside entirely on the accelerator.
  - Data sent from accelerator to host CPU for halo communication, I/O and monitoring only.

Strategy:

▪ To program using both hand-written and generated code.
  - Hand-written and tuned CUDA*.
  - Automated Fortran and CUDA generation for chemistry kernels
  - Automated code generation through compiler directives

▪ S3D is now a part of Cray’s compiler development test cases
### Original S3D

<table>
<thead>
<tr>
<th>Time Step</th>
<th>Method</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solve_Drive</td>
<td></td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>Integrate</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>RHS</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>get mass fraction</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>get_velocity</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>calc_inv_avg</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>calc_temp</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>Compute Grads</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>Diffusive Flux</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>Derivatives</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>reaction rates</td>
</tr>
</tbody>
</table>
### Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Function</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td>PE=HIDE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>284.732812</td>
<td>--</td>
<td>--</td>
<td>156348682.1</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>92.1%</td>
<td>262.380782</td>
<td>--</td>
<td>--</td>
<td>155578796.1</td>
<td>USER</td>
<td></td>
</tr>
</tbody>
</table>

| 12.4% | 35.256420 | 0.237873 | 0.7% | 391200.0 | ratt_i_.LOOPS |
| 9.6%  | 27.354247 | 0.186752 | 0.7% | 391200.0 | ratx_i_.LOOPS |
| 7.7%  | 21.911069 | 1.037701 | 4.5% | 1562500.0 | mcedif_.LOOPS |
| 5.4%  | 15.247551 | 2.389440 | 13.6% | 35937500.0 | mceval4_ |
| 5.2%  | 14.908749 | 4.123319 | 21.7% | 600.0 | rhsf_.LOOPS |
| 4.7%  | 13.495568 | 1.229034 | 8.4% | 35937500.0 | mceval4_.LOOPS |
| 4.6%  | 12.985353 | 0.620839 | 4.6% | 701.0 | calc_temp$thermchem_m_.LOOPS |
| 4.3%  | 12.274200 | 0.167054 | 1.3% | 1562500.0 | mca_vis_new$transport_m_.LOOPS |
| 4.0%  | 11.363281 | 0.606625 | 5.1% | 600.0 | computespeciesdiffflux$transport_m_.LOOPS |
| 2.9%  | 8.257434 | 0.743004 | 8.3% | 21921875.0 | mixcp$thermchem_m_ |
| 2.9%  | 8.150646 | 0.205423 | 100.0 | integrate_.LOOPS |
| 2.4%  | 6.942384 | 0.078555 | 1.1% | 391200.0 | qssa_i_.LOOPS |
| 2.3%  | 6.430820 | 0.481475 | 7.0% | 21921875.0 | mixcp$thermchem_m_.LOOPS |
| 2.0%  | 5.588500 | 0.343099 | 5.8% | 600.0 | computeheatflux$transport_m_.LOOPS |
| 1.8%  | 5.252285 | 0.062576 | 1.2% | 391200.0 | rdwdot_i_.LOOPS |
| 1.7%  | 4.801062 | 0.723213 | 13.1% | 31800.0 | derivative_x_calc_.LOOPS |
| 1.6%  | 4.461274 | 1.310813 | 22.7% | 31800.0 | derivative_y_calc_.LOOPS |
| 1.5%  | 4.327627 | 1.290121 | 23.0% | 31800.0 | derivative_z_calc_.LOOPS |
| 1.4%  | 3.963951 | 0.138844 | 3.4% | 701.0 | get_mass_frac$variables_m_.LOOPS |
## Restructured S3D for multi-core systems

<table>
<thead>
<tr>
<th>Time Step</th>
<th>Runge K</th>
<th>grid loop-omp</th>
<th>get mass fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>grid loop-omp</td>
<td>get_velocity</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>grid loop-omp</td>
<td>calc_inv_avg</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>grid loop-omp</td>
<td>calc_temp</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>grid loop-omp</td>
<td>Compute Grads</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>grid loop-omp</td>
<td>Diffusive Flux</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>grid loop-omp</td>
<td>Derivatives</td>
</tr>
<tr>
<td>Time Step</td>
<td>Runge K</td>
<td>grid loop-omp</td>
<td>reaction rates</td>
</tr>
</tbody>
</table>
## Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb. Time%</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
</tr>
<tr>
<td>85.3%</td>
<td>539.077983</td>
<td>--</td>
<td>--</td>
<td>144908.0</td>
<td>USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21.7%</td>
<td>136.950871</td>
<td>0.583731</td>
<td>0.5%</td>
<td>600.0</td>
<td>rhsf_</td>
</tr>
<tr>
<td>14.7%</td>
<td>93.237279</td>
<td>0.132829</td>
<td>0.2%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.1084</td>
</tr>
<tr>
<td>8.7%</td>
<td>55.047054</td>
<td>0.309278</td>
<td>0.6%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.1098</td>
</tr>
<tr>
<td>6.3%</td>
<td>40.129463</td>
<td>0.265153</td>
<td>0.8%</td>
<td>100.0</td>
<td>integrate_</td>
</tr>
<tr>
<td>5.8%</td>
<td>36.647080</td>
<td>0.237180</td>
<td>0.7%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.1211</td>
</tr>
<tr>
<td>5.6%</td>
<td>35.264114</td>
<td>0.091537</td>
<td>0.3%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.194</td>
</tr>
<tr>
<td>3.7%</td>
<td>23.624271</td>
<td>0.054666</td>
<td>0.3%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.320</td>
</tr>
<tr>
<td>2.7%</td>
<td>17.211435</td>
<td>0.095793</td>
<td>0.6%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.540</td>
</tr>
<tr>
<td>2.4%</td>
<td>15.471160</td>
<td>0.358690</td>
<td>2.6%</td>
<td>14400.0</td>
<td>derivative_y_calc_buff_r_.LOOP@li.1784</td>
</tr>
<tr>
<td>2.4%</td>
<td>15.113374</td>
<td>1.020242</td>
<td>7.2%</td>
<td>14400.0</td>
<td>derivative_z_calc_buff_r_.LOOP@li.1822</td>
</tr>
<tr>
<td>2.3%</td>
<td>14.335142</td>
<td>0.144579</td>
<td>1.1%</td>
<td>14400.0</td>
<td>derivative_x_calc_buff_r_.LOOP@li.1794</td>
</tr>
<tr>
<td>1.9%</td>
<td>11.794965</td>
<td>0.073742</td>
<td>0.7%</td>
<td>600.0</td>
<td>integrate_.LOOP@li.96</td>
</tr>
<tr>
<td>1.7%</td>
<td>10.747430</td>
<td>0.063508</td>
<td>0.7%</td>
<td>600.0</td>
<td>computespeciesdiffflux2$transport_m_.LOOP</td>
</tr>
<tr>
<td>1.5%</td>
<td>9.733830</td>
<td>0.096476</td>
<td>1.1%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.247</td>
</tr>
<tr>
<td>1.2%</td>
<td>7.649953</td>
<td>0.043920</td>
<td>0.7%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.274</td>
</tr>
<tr>
<td>0.8%</td>
<td>5.116578</td>
<td>0.008031</td>
<td>0.2%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.398</td>
</tr>
<tr>
<td>0.6%</td>
<td>3.966540</td>
<td>0.089513</td>
<td>2.5%</td>
<td>1.0</td>
<td>s3d_</td>
</tr>
<tr>
<td>0.3%</td>
<td>2.027255</td>
<td>0.017375</td>
<td>1.0%</td>
<td>100.0</td>
<td>integrate_.LOOP@li.73</td>
</tr>
<tr>
<td>0.2%</td>
<td>1.318550</td>
<td>0.001374</td>
<td>0.1%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.376</td>
</tr>
<tr>
<td>0.2%</td>
<td>0.986124</td>
<td>0.017854</td>
<td>2.0%</td>
<td>600.0</td>
<td>rhsf_.REGION@li.1096</td>
</tr>
<tr>
<td>0.1%</td>
<td>0.700156</td>
<td>0.027669</td>
<td>4.3%</td>
<td>1.0</td>
<td>exit</td>
</tr>
</tbody>
</table>
Advantage of raising loops

- Create good granularity OpenMP Loop
- Improves cache re-use
- Reduces Memory usage significantly
- Creates a good potential kernel for an accelerator
## Restructured S3D for multi-core systems

<table>
<thead>
<tr>
<th>Time Step – acc_data</th>
<th>S3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve_Drive</td>
<td></td>
</tr>
<tr>
<td>Integrate</td>
<td></td>
</tr>
<tr>
<td>RHS</td>
<td></td>
</tr>
<tr>
<td>grid loop -ACC</td>
<td>get mass fraction</td>
</tr>
<tr>
<td>grid loop-ACC</td>
<td>get_velocity</td>
</tr>
<tr>
<td>grid loop-ACC</td>
<td>calc_inv_avg</td>
</tr>
<tr>
<td>grid loop-ACC</td>
<td>calc_temp</td>
</tr>
<tr>
<td>grid loop-ACC</td>
<td>Compute Grads</td>
</tr>
<tr>
<td>grid loop-ACC</td>
<td>Diffusive Flux</td>
</tr>
<tr>
<td>grid loop-ACC</td>
<td>Derivatives</td>
</tr>
<tr>
<td>grid loop-ACC</td>
<td>reaction rates</td>
</tr>
</tbody>
</table>
What does OpenACC look like

```c
#ifdef GPU
 !$acc data present_or_create( avmolwt, cpcoef_aa, cpcoef_bb, cpmix, &
 !$acc&   enthcoef_aa, enthcoef_bb, &
 !$acc&   gamma, invEnthInc, lrmcwrk, molwt_c, n_spec, pressure, q, &
 !$acc&   neighbor, nsc, &
 !$acc&   rhs, rmcwrk, Ru, temp, temp_hibound, temp_lobound, u, vary_in_x, &
 !$acc&   vary_in_y, &
 !$acc&   vary_in_z, volum, yspecies, ds_mxvg, diffflux,&
 !$acc&   diffusion, grad_mixmw, grad_t, grad_u, grad_ys, h_spec, lambda, &
 !$acc&   rr_r, rs_therm_diff, tmmp, tmmp2, tmmpdx,voltmp, vscsty,&
 !$acc&   neg_fs_x_buf, neg_fs_y_buf,neg_fs_z_buf, pos_fs_x_buf, &
 !$acc&   pos_fs_y_buf, pos_fs_z_buf, &
 !$acc&   neg_f_x_buf, neg_f_y_buf,neg_f_z_buf, pos_f_x_buf, pos_f_y_buf, &
 !$acc&   pos_f_z_buf,mixmw)&
 !$acc&   copyin( jstage,scale1x,scale1y,scale1z,ds,ae,be,ce,de,molwt)
 #endif
```
What does OpenACC look like

```c
#ifndef GPU
!$acc parallel loop gang private(i,ml,mu)
#else
!$omp parallel private(i, ml, mu)
!$omp do
#endif
  do i = 1, nx*ny*nz, ms
    ml = i
    mu = min(i+ms-1, nx*ny*nz)
    call calc_gamma_r( gamma, cpmix, avmolwt, ml, mu)
    call calc_press_r( pressure, q(1,1,1,4), temp, avmolwt, ml, mu )
    call calc_specEnth_allpts_r(temp, h_spec, ml, mu)
  end do
#endif
~
```
What does OpenACC look like

! X - direction communication - (+) side
reqcount = 0
do i = 1, derivcount
    if( deriv_x_list(i)%pos_nbr >= 0 .and. &
        deriv_x_list(i)%inuse ) then

        reqcount = reqcount + 1
        req(reqcount) = deriv_x_list(i)%req(3)

    endif
enddo
if( reqcount > 0 ) then
    !write(*,'(1i4,1a,1i4)') myid, 'x pos waiting on ', reqcount
    call MPI_WAITALL( reqcount, req, stat, ierr )
#endif GPU_ASYNC
#$acc update device(pos_f_x_buf(:,:,1:reqcount)) async(1)
#endif
For the next year, until we can call subroutines and functions on the accelerator, the compiler must inline all subroutines and functions within a acc_region.

- This is performed automatically by the compiler
  - Can be incrementally controlled by using compile line options
    - hwp –hpl=<path to program library>
There are several things that inhibit the inlining of the call chain beneath the acc_region:

- Call to subroutines and functions that the compiler does not see
- I/O, STOP, etc (Not anymore)
- Array shape changing through argument passing
- Dummy arguments
  - Real*8 dummy(*), dummy_2d(nx,*)
Successful Inlining

248. !$acc parallel_loop private(i,ml,mu)
249.  1------< do i = 1, nx*ny*nz, ms
250.  1               ml = i
251.  1               mu = min(i+ms-1, nx*ny*nz)
252.  1               call get_mass_frac_r( q, volum, yspecies, ml, mu) ! get Ys from rho*Ys, volum from rho
253.  1               call get_velocity_vec_r( u, q, volum, ml, mu) ! fill the velocity vector
254.  1               call calc_inv_avg_mol_wt_r( yspecies, avmolwt, mixMW, ml, mu) ! set inverse of mixture MW
255.  1--------> end do
Inliner diagnostics

333. 1-------< do n=1,n_spec
334. 1     itmp = n + 4
335. 1     !call computeScalarGradient_calc( yspecies(:,;:,n), grad_Ys(:,;:,n,:), itmp )
336. 1     call computeScalarGradient5d_calc( yspecies(1,1,1,n), &

^ ftn-3007 ftn: ERROR RHSF, File = rhsf.f90, Line = 336, Column = 12
Routine "write_date_and_time", referenced in "terminate_run", was not inlined because a scalar actual argument at position 1 is being mapped to an array dummy argument.

^ ftn-3007 ftn: ERROR RHSF, File = rhsf.f90, Line = 336, Column = 12
Routine "write_date_and_time", referenced in "terminate_run", was not inlined because a scalar actual argument at position 1 is being mapped to an array dummy argument.

^ ftn-3007 ftn: ERROR RHSF, File = rhsf.f90, Line = 336, Column = 12
Routine "write_date_and_time", referenced in "terminate_run", was not inlined because a scalar actual argument at position 2 is being mapped to an array dummy argument.

^ ftn-3007 ftn: ERROR RHSF, File = rhsf.f90, Line = 336, Column = 12
Routine "write_date_and_time", referenced in "terminate_run", was not inlined because a scalar actual argument at position 1 is being mapped to an array dummy argument.
Currently many compiler internal errors are given when forms are encountered that inhibit acceleration

- Calls within the acc_region
  - These can be identified by using the inliner

- Derived Types
  - These are being worked

- Dummy arguments

- Etc.
Early Software/Hardware Issues

• Finding lots of bugs in tools and compiler
  • Cannot fix them until they are identified
• Identified bottleneck in MPI messaging between GPUs
  • This is being addressed by Cray/Nvidia
    • Want zero transfer messages – GPU directly to other GPU
• Directives are emerging – changing
  • Usage is identifying new capabilities – pipelining
• Future GPUs will have a higher performance advantage over x86 sockets
The latest version of S3D with OpenMP and OpenACC performs
- 1.8X faster on XK6 with GPU than a dual-CPU XE6
- 2.2X faster on XK6 with GPU than a XK6 without GPU
## Comparisons between OpenMP and OpenACC

<table>
<thead>
<tr>
<th>Speedup</th>
<th>OpenMP</th>
<th>Acc</th>
<th>Acc</th>
<th>Host</th>
<th>Acc Copy</th>
<th>Acc Copy</th>
<th>Calls</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time%</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>In</td>
<td>Out</td>
<td>PE=HIDE</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>----------</td>
<td>----------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(MBytes)</td>
<td>(MBytes)</td>
<td>Thread=HIDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>----------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.00%</td>
<td>187.783</td>
<td>234.011</td>
<td>148248</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>81843.75</td>
<td>1960504</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Speed</th>
<th>OpenMP</th>
<th>Acc</th>
<th>Acc</th>
<th>Host</th>
<th>Acc Copy</th>
<th>Acc Copy</th>
<th>Calls</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>In</td>
<td>Out</td>
<td>PE=HIDE</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>----</td>
<td>----</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(MBytes)</td>
<td>(MBytes)</td>
<td>Thread=HIDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>----------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.094895</td>
<td>94.22</td>
<td>24.00%</td>
<td>44.976</td>
<td>0.042</td>
<td>--</td>
<td>--</td>
<td>600reaction_rate_vec_.ACC_KERNEL@li.165</td>
<td></td>
</tr>
<tr>
<td>4.362416</td>
<td>65</td>
<td>7.90%</td>
<td>14.9</td>
<td>0.002</td>
<td>--</td>
<td>--</td>
<td>100computecoefficients_r_.ACC_KERNEL@li.141</td>
<td></td>
</tr>
<tr>
<td>1.781216</td>
<td>19.8</td>
<td>5.90%</td>
<td>11.116</td>
<td>0.001</td>
<td>--</td>
<td>--</td>
<td>600rhf_.ACC_COPY@li.1806</td>
<td></td>
</tr>
<tr>
<td>2.889151</td>
<td>29.4</td>
<td>5.40%</td>
<td>10.176</td>
<td>0.037</td>
<td>--</td>
<td>--</td>
<td>600rhf_.ACC_KERNEL@li.379</td>
<td></td>
</tr>
<tr>
<td>2.527231</td>
<td>25.29</td>
<td>5.30%</td>
<td>10.007</td>
<td>0.039</td>
<td>--</td>
<td>--</td>
<td>600rhf_.ACC_KERNEL@li.1712</td>
<td></td>
</tr>
<tr>
<td>0.314012</td>
<td>3.05</td>
<td>5.20%</td>
<td>9.713</td>
<td>0.039</td>
<td>--</td>
<td>--</td>
<td>600rhf_.ACC_KERNEL@li.1810</td>
<td></td>
</tr>
<tr>
<td>1.124026</td>
<td>7.93</td>
<td>3.80%</td>
<td>7.055</td>
<td>0.037</td>
<td>--</td>
<td>--</td>
<td>600rhf_.ACC_KERNEL@li.1758</td>
<td></td>
</tr>
<tr>
<td>4.46391</td>
<td>25.48</td>
<td>3.00%</td>
<td>5.708</td>
<td>0.037</td>
<td>--</td>
<td>--</td>
<td>600rhf_.ACC_KERNEL@li.419</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.00%</td>
<td>5.627</td>
<td>5.63</td>
<td>--</td>
<td>--</td>
<td>600reaction_rate_vec_.ACC_COPY@li.4958</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.90%</td>
<td>5.373</td>
<td>5.376</td>
<td>--</td>
<td>--</td>
<td>600rhf_.ACC_COPY@li.2212</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.70%</td>
<td>3.131</td>
<td>0.008</td>
<td>--</td>
<td>19490.63</td>
<td>100integrate_.ACC_COPY@li.74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.521674</td>
<td>25.48</td>
<td>1.40%</td>
<td>2.676</td>
<td>0.014</td>
<td>--</td>
<td>--</td>
<td>700calc_primary_vars_.ACC_KERNEL@li.42</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.40%</td>
<td>2.584</td>
<td>0.086</td>
<td>14175</td>
<td>--</td>
<td>600rhf_.ACC_COPY@li.366</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.20%</td>
<td>2.266</td>
<td>1.526</td>
<td>--</td>
<td>6496.875</td>
<td>92400derivative_z_pack_np_.ACC_COPY@li.351</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.925433</td>
<td>13.35</td>
<td>1.20%</td>
<td>2.253</td>
<td>0.008</td>
<td>--</td>
<td>--</td>
<td>600rhf_.ACC_KERNEL@li.989</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.10%</td>
<td>2.104</td>
<td>0.042</td>
<td>--</td>
<td>12993.75</td>
<td>1800derivative_y_pack_np_.ACC_COPY@li.429</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.10%</td>
<td>2.104</td>
<td>0.073</td>
<td>--</td>
<td>12993.75</td>
<td>1800derivative_x_pack_np_.ACC_COPY@li.433</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.10%</td>
<td>2.053</td>
<td>1.247</td>
<td>--</td>
<td>6496.875</td>
<td>92400derivative_z_pack_np_.ACC_COPY@li.340</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.48176</td>
<td>7.74</td>
<td>0.90%</td>
<td>1.727</td>
<td>0.009</td>
<td>--</td>
<td>--</td>
<td>600integrate_.ACC_KERNEL@li.113</td>
<td></td>
</tr>
</tbody>
</table>
GPU Annotated Timeline

Integration loop – 6 times per call to compute transport coefficients

Transport Coefficients: 7%

Reaction rates: 18%

Derivatives: Kernels only 2%, and should get near-perfect overlap w/ memcpys. But w/ current overheads and synchronous memcpys, currently totals 21%. (We will fix this.)

RHS: 24% ... but low GPU utilization in these kernels, so could be less. (We will improve this.)

GPU is idle appx. 23% of total time, largely due to missing MPI overlap. (We will improve this.)

Example of missed overlap opportunity, this one costing 6.5% of total time. (We will fix this.)
Future Developments

• Timeline shows where improvements can be obtained
  – Asynchronous updates – improved performance from 2.5 sec/ts to 2.05 sec/ts
  – Overlapping MPI with GPU computation - needs
    • GPU direct
      – Available later in the year
    • Use !$acc host_data use_device directive to simply communication between device and host
      – Significantly cleans up code
  – Cuda proxy for running multiple MPI ranks on node and sharing the GPU
    • This would be used if the overlap and GPU direct succeeds on fully utilizing the GPU
#ifdef GPU
!$acc data present(f)
!$acc host_data use_device(f)
#endif
if( deriv_z_list(idx)%packed ) then
  deriv_z_list(idx)%packed = .false.
if(deriv_z_list(idx)%neg_nbr>=0) then
  call MPI_Isend(f(1,1,1),(mx*my*iorder/2),
    MPI_REAL8,deriv_z_list(idx)%neg_nbr,deriv_list_size + idx, &
    gcomm,deriv_z_list(idx)%req(2),ierr)
endif
if(deriv_z_list(idx)%pos_nbr>=0) then
  ! send ghost cells to neighbor on (+z) side
  nm = mz + 1 - iorder/2
  call MPI_Isend(f(1,1,nm),(mx*my*iorder/2), &
    MPI_REAL8,deriv_z_list(idx)%pos_nbr, idx, &
    gcomm,deriv_z_list(idx)%req(4),ierr)
endif
else
  if(deriv_z_list(idx)%neg_nbr>=0) then
    call MPI_Isend(f(1,1,1),(mx*my*iorder/2), &
      MPI_REAL8,deriv_z_list(idx)%neg_nbr,deriv_list_size + idx, &
      gcomm,deriv_z_list(idx)%req(2),ierr)
  endif
endif
if(deriv_z_list(idx)%pos_nbr>=0) then
  ! send ghost cells to neighbor on (+z) side
  nm = mz + 1 - iorder/2
  call MPI_Isend(f(1,1,nm),(mx*my*iorder/2), &
    MPI_REAL8,deriv_z_list(idx)%pos_nbr, idx, &
    gcomm,deriv_z_list(idx)%req(4),ierr)
endif
#endif
!$acc end host_data
!$acc end data
#endif
Thank you. Questions?