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

Back to the Futures – Combining different levels of parallelism

- 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.

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 cacciler 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

XK6 only has one GPU per node and one Interlagos Socket

- 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

Normal Profile – default Craypat report

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group 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)

Normal Profile – Using "setenv PAT_RT_HWPC 1"

```
================================================================================
  USER / parabola_
--------------------------------------------------------------------------------
  Time%                                              12.4%
  Time                                           9.438486 secs
  Imb. Time                                      0.851876 secs
  Imb. Time%                                         8.3%
  Calls                          0.265M/sec     2592000.0 calls
  PAPI_L1_DCM                   42.908M/sec     419719824 misses
  PAPI_TLB_DM                    0.048M/sec        474094 misses
  PAPI_L1_DCA                 1067.727M/sec   10444336795 refs
  PAPI_FP_OPS                 1808.848M/sec   17693862446 ops
  Average Time per Call                          0.000004 secs
  CrayPat Overhead : Time         75.3%
  User time (approx)              9.782 secs   21520125183 cycles  100.0% Time
  HW FP Ops / User time       1808.848M/sec   17693862446 ops   10.3%peak(DP)
  HW FP Ops / WCT             1808.848M/sec
  Computational intensity          0.82 ops/cycle     1.69 ops/ref
  MFLOPS (aggregate)         7409042.08M/sec
  TLB utilization              22030.09 refs/miss   43.028 avg uses
  D1 cache hit,miss ratios        96.0% hits          4.0% misses
  D1 cache utilization (misses)  24.88 refs/miss    3.111 avg hits
================================================================================
```

Re-compiling with –hprofile_generate

Loop Incl Time	Loop Hit	Loop Trips Min	Loop Trips Max	Function=/.LOOP[.] PE=HIDE
51.107386	500	0	16	|sweepx2_.LOOP.1.li.34
51.10682	8000	0	16	|sweepx2_.LOOP.2.li.35
50.373481	500	0	16	|sweepx1_.LOOP.1.li.34
50.372915	8000	0	16	|sweepx1_.LOOP.2.li.35
12.480442	1000	0	16	|sweepy_.LOOP.1.li.38
12.478967	16000	0	1	|sweepy_.LOOP.2.li.39
11.949236	500	0	16	|sweepz_.LOOP.05.li.54
11.948618	8000	0	1	|sweepz_.LOOP.06.li.55
5.479066	288000	0	1031	|riemann_.LOOP.2.li.63
3.082245	51168000	0	12	|riemann_.LOOP.3.li.64
1.796424	2592000	0	1032	|parabola_.LOOP.6.li.67
1.503023	2592000	0	1034	|parabola_.LOOP.2.li.30
1.377911	2592000	0	1032	|parabola_.LOOP.7.li.75
1.094964	2592000	0	1033	|parabola_.LOOP.4.li.44
0.815105	288000	0	1025	|remap_.LOOP.7.li.83
0.792899	2592000	0	1032	|parabola_.LOOP.5.li.53
0.76888	2592000	0	1032	|parabola_.LOOP.8.li.84
0.590497	128000	0	64	|sweepx2_.LOOP.3.li.38
0.505536	288000	0	1031	|riemann_.LOOP.1.li.44
0.478305	2592000	0	1034	|parabola_.LOOP.3.li.36
0.465781	2592000	0	1035	|parabola_.LOOP.1.li.24
0.463514	576000	0	1036	|paraset_.LOOP.1.li.117
0.362512	288000	0	1032	|states_.LOOP.2.li.64
0.338868	288000	0	1030	|evolve_.LOOP.4.li.70
0.335398	288000	0	1026	|remap_.LOOP.8.li.111

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||||        |            |            |    vhone_
6|||||   3.1% |   3.663423 |   768000.0 |sweepx1_.LOOP.2.li.35
7|||||        |            |            | sweepx1_.LOOP.1.li.34
8|||||        |            |            |  sweepx1_.LOOPS
9|||||        |            |            |   sweepx1_
10||||        |            |            |    vhone_
||||||=================================================
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|||        |            |            |    vhone_
4|||   1.6% |   1.852820 |   384000.0 |sweepx1_.LOOP.2.li.35
5|||        |            |            | sweepx1_.LOOP.1.li.34
6|||        |            |            |  sweepx1_.LOOPS
7|||        |            |            |   sweepx1_
8|||        |            |            |    vhone_
|||=================================================
```

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

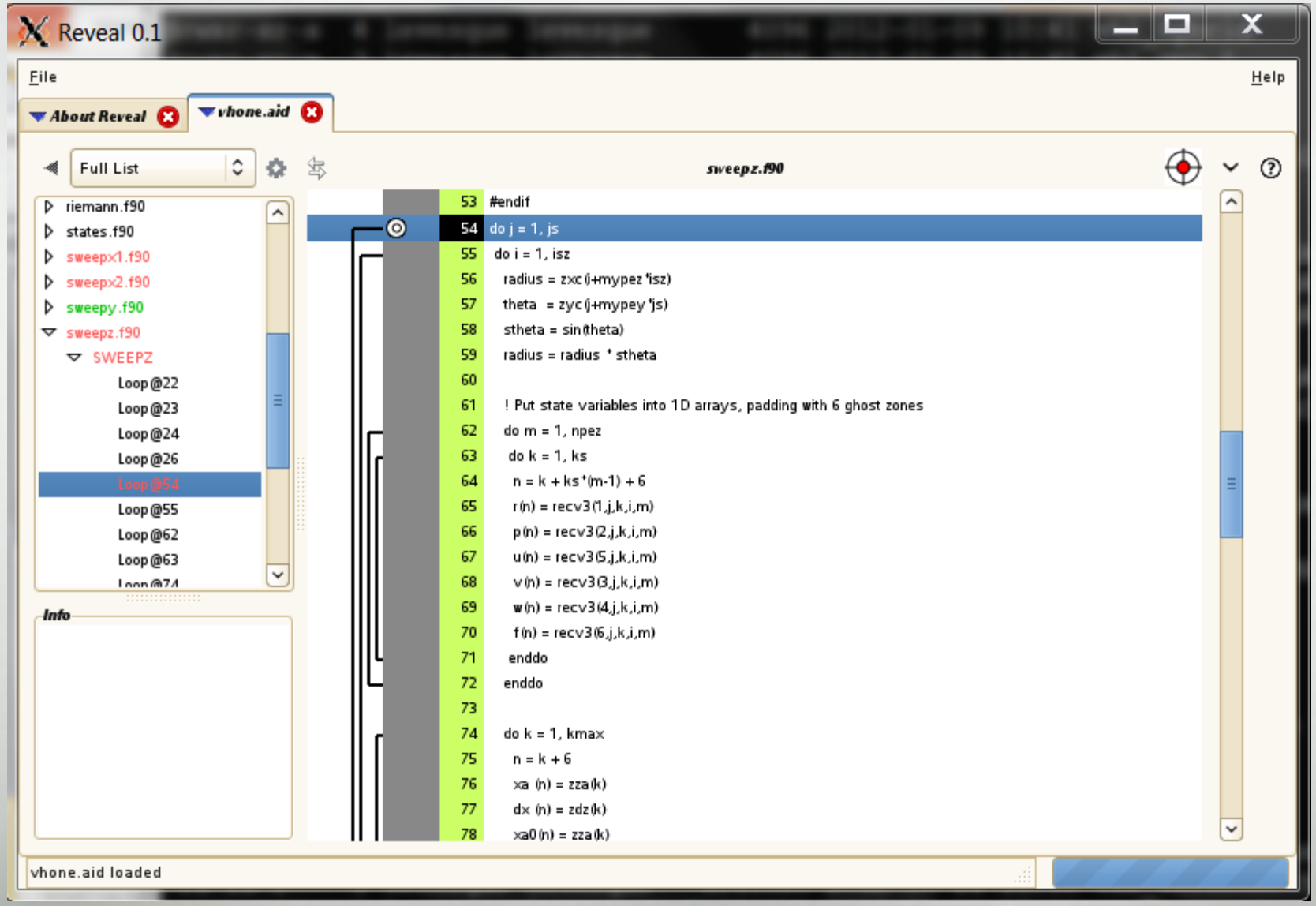
Task 2 Parallel Analysis, Scoping and Vectorization (Cont)

- 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>

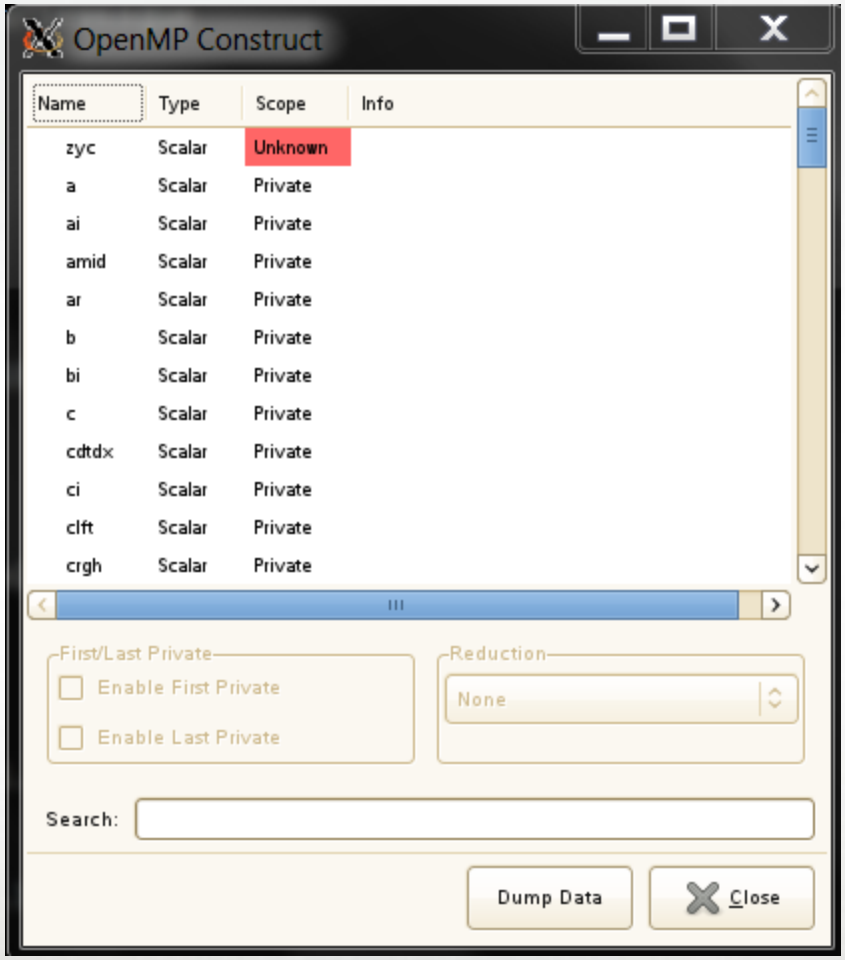
Current output of –homp_analyze

```
<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>
```

Main window of reveal



Scoping window



What we end up finding out

Private Variables in module, need to use Threadprivate

```
!$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

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

 INTEGER, PARAMETER :: imax = 64, jmax =1024, kmax = 1024    ! Memory dimens
 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 var

 INTEGER :: isy, isz, js, ks, Ya2abuff_size, Za2abuff_size
 INTEGER :: npe, npey, npez, mype, mypey, mypez  ! # of pes and local pe num
 INTEGER :: MPI_COMM_ROW, MPI_COMM_COL
 INTEGER :: VH1_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(nvar,imax/pey,imax/pez,kmax) :: send4
```

1: 1 Default text

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

 INTEGER, PARAMETER :: imax = 64, jmax =1024, kmax = 1024    ! Memory dimens
 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 var

 INTEGER :: isy, isz, js, ks, Ya2abuff_size, Za2abuff_size
 INTEGER :: npe, npey, npez, mype, mypey, mypez  ! # of pes and local pe num
 INTEGER :: MPI_COMM_ROW, MPI_COMM_COL
 INTEGER :: VH1_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(nvar,imax/pey,imax/pez,kmax) :: send4
```

1: 1 Default text

Differences in OpenMP version

```
module sweepsize
!========================================================================
! 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(imax,jmax,kmax) + 12
!------------------------------------------------------------------------

integer, parameter :: maxsweep=1050

end module sweepsize

module sweeps
!========================================================================
! Data structures used in 1D sweeps, dimensioned maxsweep  (set in sweepsize
!------------------------------------------------------------------------

use sweepsize

character(len=1) :: sweep                                ! direction of
integer :: nmin, nmax, ngeom, nleft, nright              ! number of fir

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 sweeps
```

```
module sweepsize
!========================================================================
! 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(imax,jmax,kmax) + 12
!------------------------------------------------------------------------

integer, parameter :: maxsweep=1050

end module sweepsize

module sweeps
!========================================================================
! Data structures used in 1D sweeps, dimensioned maxsweep  (set in sweepsize
!------------------------------------------------------------------------

use sweepsize

character(len=1) :: sweep                                ! direction of
integer :: nmin, nmax, ngeom, nleft, nright              ! number of fir
!$omp threadprivate (r, p, e, q, u, v, w,xa, xa0, dx, dx0, dvol,f, flat,para
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 sweeps
```

Differences in OpenMP version

```
#endif

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) = zfl(i,j,k)

     xa0(n) = zxa(i)
     dx0(n) = zdx(i)
     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
```

```
#endif
!$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) = zfl(i,j,k)

     xa0(n) = zxa(i)
     dx0(n) = zdx(i)
     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
```

Differences in OpenMP version

```
!-------------------------------------------------------------------------
! Calculate the domain of dependence along space coordinate,
! C*dt, by multiplying the Eulerian sound speed in
! each zone by dt.  Divide by two to save flops later.
! If angular coordinate, then divide out by radius to get physical dimension

hdt    = 0.5*dt
do n = nmin-4, nmax+4
  Cdtdx (n) = sqrt(gam*p(n)/r(n))/(dx(n)*radius)
```

```
  svel      = max(svel,Cdtdx(n))
```

```
  Cdtdx (n) = Cdtdx(n)*hdt
  fCdtdx(n) = 1. - fourthd*Cdtdx(n)
enddo

! Include gravitational and ficticious forces
call forces(xa,grav,fict)

! Obtain averages of rho, u, and P over the domain (+/-)Cdt
!                    lft is the + wave on the left  side of the boundary
!                    rgh is the - wave on the right side of the boundary

do n = nmin-4, nmax+4
```

```
!-------------------------------------------------------------------------
! Calculate the domain of dependence along space coordinate,
! C*dt, by multiplying the Eulerian sound speed in
! each zone by dt.  Divide by two to save flops later.
! If angular coordinate, then divide out by radius to get physical dimension

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

! Include gravitational and ficticious forces
call forces(xa,grav,fict)

! Obtain averages of rho, u, and P over the domain (+/-)Cdt
!                    lft is the + wave on the left  side of the boundary
!                    rgh is the - wave on the right side of the boundary

do n = nmin-4, nmax+4
```

Differences in runtime

All MPI on 4096 cores	**43.01 seconds**
Hybrid 256 nodesx16 threads	**45.05 seconds**

Task 3 Moving from OpenMP to OpenACC

- 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

```
#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

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

do k = 1, ks
 do j = 1, js
     theta=0.0
     stheta=0.0
     radius=0.0

   ! 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) = zfl(i,j,k)

     xa0(n) = zxa(i)
     dx0(n) = zdx(i)
     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
```

1: 1 Default text

```
!$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) = zfl(i,j,k)

     xa0(n) = zxa(i)
     dx0(n) = zdx(i)
     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
```

1: 1 Default text

Changes to remove THREADPRIVATE and CRITICAL REGION

```
      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
```

1: 1 Default text

```
      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
```

1: 1 Default text

Changes to remove THREADPRIVATE

```
   subroutine ppmlr (svel, sweep, nmin, nmax, ngeom, nleft, nright,r, p, e,
      xa, xa0, dx, dx0, dvol, f, flat, para,radius, theta, stheta)

! Using the 1D arrays of rho, u, and P, perform a 1D lagrangian hydrodynamic
!   - set boundary conditions by filling ghost zones
!   - obtain parabolic interpolations of rho, u, P
!   - compute input states from these interpolations for Riemann problem
!   - call the Riemann solver to find the time averages umid, pmid
!   - evolve the finite difference equations to get updated values of rho, u
! Then perform a conservative remap back to the Eulerian grid
!---------------------------------------------------------------------

! GLOBALS
use global
use sweepsize

IMPLICIT NONE
integer :: sweep                                ! direction of sweep: x,
integer :: nmin, nmax, ngeom, nleft, nright             ! number of fir
real, dimension(maxsweep) :: r, p, e, q, u, v, w,svel       ! fluid va
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


! LOCALS
```

1: 1 Default text

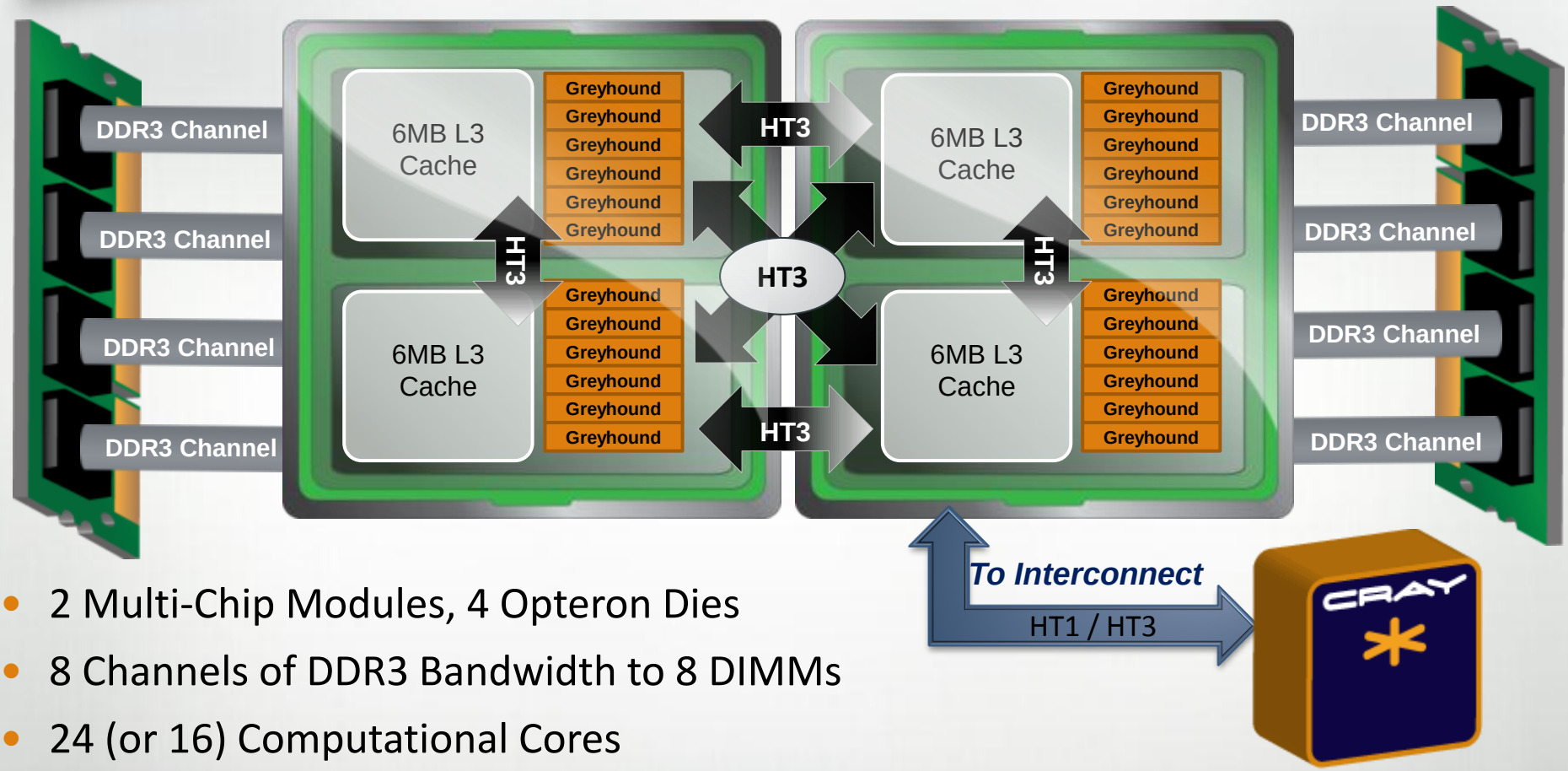
```
subroutine ppmlr

! Using the 1D arrays of rho, u, and P, perform a 1D lagrangian hydrodynamic
!   - set boundary conditions by filling ghost zones
!   - obtain parabolic interpolations of rho, u, P
!   - compute input states from these interpolations for Riemann problem
!   - call the Riemann solver to find the time averages umid, pmid
!   - evolve the finite difference equations to get updated values of rho, u
! Then perform a conservative remap back to the Eulerian grid
!---------------------------------------------------------------------

! GLOBALS
use global
use sweeps

IMPLICIT NONE


! LOCALS
```

1: 1 Default text

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**

NUMA Regions on 24-core Magny Cours



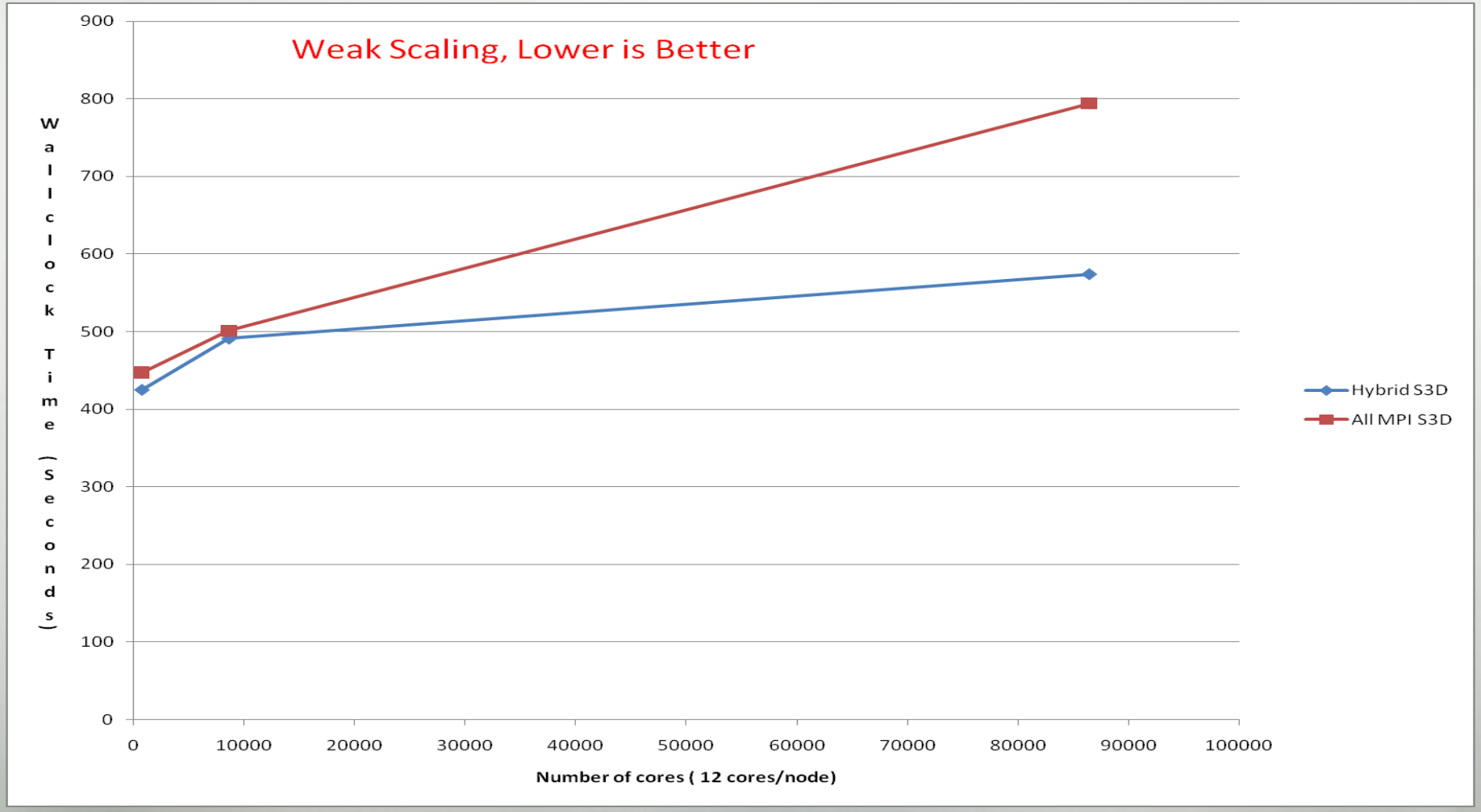
- 2 Multi-Chip Modules, 4 Opteron Dies
- 8 Channels of DDR3 Bandwidth to 8 DIMMs
- 24 (or 16) Computational Cores
  - 64 KB L1 and 512 KB L2 caches for each core
  - 6 MB of shared L3 cache on each die
- Dies are fully connected with HT3
- Snoop Filter Feature Allows 4 Die SMP to scale well

Cray's Core Specialization and Asynchronous MPI

- 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

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

setenv MPICH_NEMESIS_ASYNC_PROGRESS 1          ← set two environment vars
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  ← give MPI two cores
```

Resultant Hybrid S3D Performance



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

Name	Ease of Use	Portability	Performance	Maturity
CUDA for C	2	3	5	5
OpenCL	2	4	4	4
CUDA Fortran	2	2	4	4
Directives	4	4 (improving)	3 (improving)	2
Libraries	5	4	4	3

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.

CUDA Command Line & Visual Profiler

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.

Libsci_acc

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

Adaptation in Simple interface

You can pass either host pointers or device pointers to simple interface

A is host memory

```
dgetrf(M, N, A, lda, ipiv, &info)
```

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

Pass Device memory

```
dgetrf(M, N, d_A, lda, ipiv, &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

CPU interface

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

Contents – As of April 2012

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 that this.

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

So just dgemm is sufficient.

```
!$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
```

CRAY
THE SUPERCOMPUTER COMPANY

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

CRAY THE SUPERCOMPUTER COMPANY

- 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, permaner
  - 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

CAPS CRAY THE SUPERCOMPUTER COMPANY NVIDIA The Portland Group

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

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 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

```
!     -use previous guess for pmid to get wavespeeds: wlft, wrgh
!     -find the slope in the u-P plane for each state: zlft, zrgh
!     -use the wavespeeds and pmid to guess umid on each side: umidl, umidr
!     -project tangents from (pmid,umidl) and (pmid,umidr) to get new pmid
!     -make sure pmid does not fall below floor value for pressure

  do n = 1, 12
do l = lmin, lmax
    pmold(l) = pmid(l)
    wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
    wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
    wlft (l) = clft(l) * sqrt(wlft(l))
    wrgh (l) = crgh(l) * sqrt(wrgh(l))
    zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
    zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
    zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
    zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
    umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
    umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
    pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (zrgh(l
    pmid (l) = max(smallp,pmid(l))
    tol_check = abs(pmid(l)-pmold(l))/pmid(l)
enddo
    if (all(tol_check(lmin:lmax).lt. tol )) exit
  enddo

! Calculate umid by averaging umidl, umidr based on new pmid
```

```
!     -use previous guess for pmid to get wavespeeds: wlft, wrgh
!     -find the slope in the u-P plane for each state: zlft, zrgh
!     -use the wavespeeds and pmid to guess umid on each side: umidl, umidr
!     -project tangents from (pmid,umidl) and (pmid,umidr) to get new pmid
!     -make sure pmid does not fall below floor value for pressure


do l = lmin, lmax
  do n = 1, 12
    pmold(l) = pmid(l)
    wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
    wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
    wlft (l) = clft(l) * sqrt(wlft(l))
    wrgh (l) = crgh(l) * sqrt(wrgh(l))
    zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
    zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
    zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
    zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
    umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
    umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
    pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (zrgh(l
    pmid (l) = max(smallp,pmid(l))
    if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
  enddo


enddo

! Calculate umid by averaging umidl, umidr based on new pmid
```

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**

Improving Cache/Shared Memory Usage

```
!-----------------------------------------------------------------------
  do n = nmin, nmax
    diffanm2 = a(n-1) - a(n-2)
    diffanm1 = a(n) - a(n-1)
    diffan = a(n+1) - a(n)




    diffanp1 = a(n+2) - a(n+1)
   danm1 = para(n-1,4) * diffanm1 + para(n-1,5) * diffanm2
   danm1 = sign( min(abs(danm1), 2.0*abs(diffanm2), 2.0*abs(diffanm1)), danm
    if(diffanm2*diffanm1 < 0.0) danm1 = 0.0
   dan = para(n,4) * diffan + para(n,5) * diffanm1
   dan = sign( min(abs(dan), 2.0*abs(diffanm1), 2.0*abs(diffan)), dan )




    if(diffanm1*diffan < 0.0) dan = 0.0
   danp1 = para(n+1,4) * diffanp1 + para(n+1,5) * diffan
   danp1 = sign( min(abs(danp1), 2.0*abs(diffan), 2.0*abs(diffanp1)), danp1
    if(diffan*diffanp1 < 0.0) danp1 = 0.0




    arn = a(n) + para(n,1)*diffan + para(n,2)*danp1 + para(n,3)*dan
```

```
!-----------------------------------------------------------------------
  do n = nmin-2, nmax+1




    diffa(n) = a(n+1) - a(n)
  enddo

!                                                       Equation 1.7 of C&W
!     da(j) = D1 * (a(j+1) - a(j)) + D2 * (a(j) - a(j-1))
  do n = nmin-1, nmax+1




   da(n) = para(n,4) * diffa(n) + para(n,5) * diffa(n-1)
   da(n) = sign( min(abs(da(n)), 2.0*abs(diffa(n-1)), 2.0*abs(diffa(n))), da
  enddo

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

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

  do n = nmin-1, nmax
    ar(n) = a(n) + para(n,1)*diffa(n) + para(n,2)*da(n+1) + para(n,3)*da(n)
```

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

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
|-------------------------------------------------------------------
|  51.3% | 42.762837 |       -- |     -- | 703.0 |ACCELERATOR
||------------------------------------------------------------------
||  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 |lbm3d2p_d_.ASYNC_KERNEL@li.116
||   1.6% |  1.337119 | 0.193826 |  13.5% |  20.0 |lbm3d2p_d_.ASYNC_KERNEL@li.119
||   1.6% |  1.322690 | 0.059387 |   4.6% |   1.0 |lbm3d2p_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 |lbm3d2p_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 |lbm3d2p_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
||-----------------------------------------------------------------------
||  15.7% |  98.021619 | 43.078137 |  31.0% |   200.0 |lbm3d2p_d_.ASYNC_KERNEL@li.129
||   3.7% |  23.359080 |  2.072147 |   8.3% |   200.0 |lbm3d2p_d_.ASYNC_KERNEL@li.127
||   3.6% |  22.326085 |  1.469419 |   6.3% |   200.0 |lbm3d2p_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 |lbm3d2p_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 |lbm3d2p_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
```

Useful tools

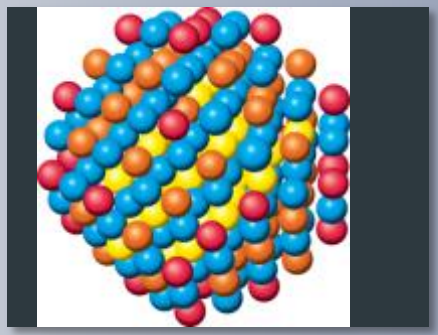
- Compiler feedback:
  - -ra to generate *.lst loopmark files (equivalent for C)
  - -rd to generate *.cg and *.opt files
    - *.cg useful to understand synchronisation points (CAF and ACC)
  - "ptxas -v *.ptx" gives information on register and shared memory usage (no way yet for user to adjust this)
- 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

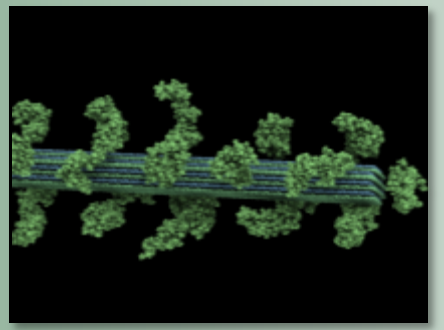
Cray GPU Programming Environment

- 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

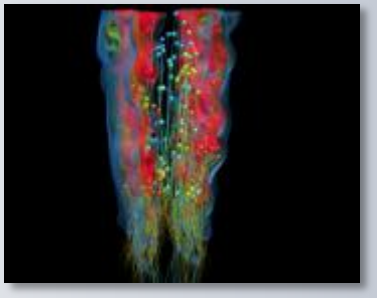
Titan: Early Science Applications

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



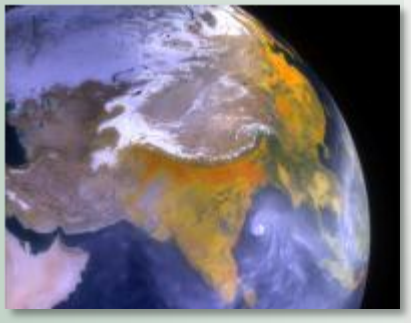


LAMMPS
Biofuels: An atomistic model of cellulose (blue) surrounded by lignin molecules comprising a total of 3.3 million atoms. Water not shown.

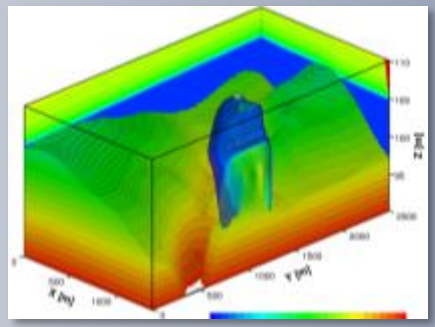


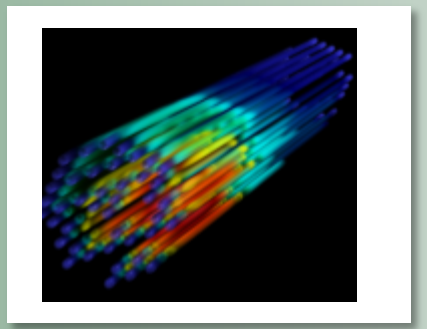
S3D
How are going to efficiently burn next generation diesel/bio fuels?
.

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



PFLOTRAN
Stability and viability of large scale CO_2 sequestration; predictive containment groundwater transport

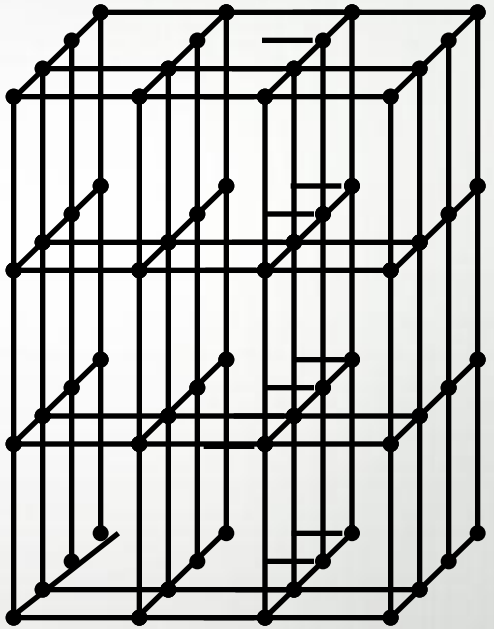




Denovo
Unprecedented high-fidelity radiation transport calculations that can be used in a variety of nuclear energy and technology applications.

S3D – A DNS solver

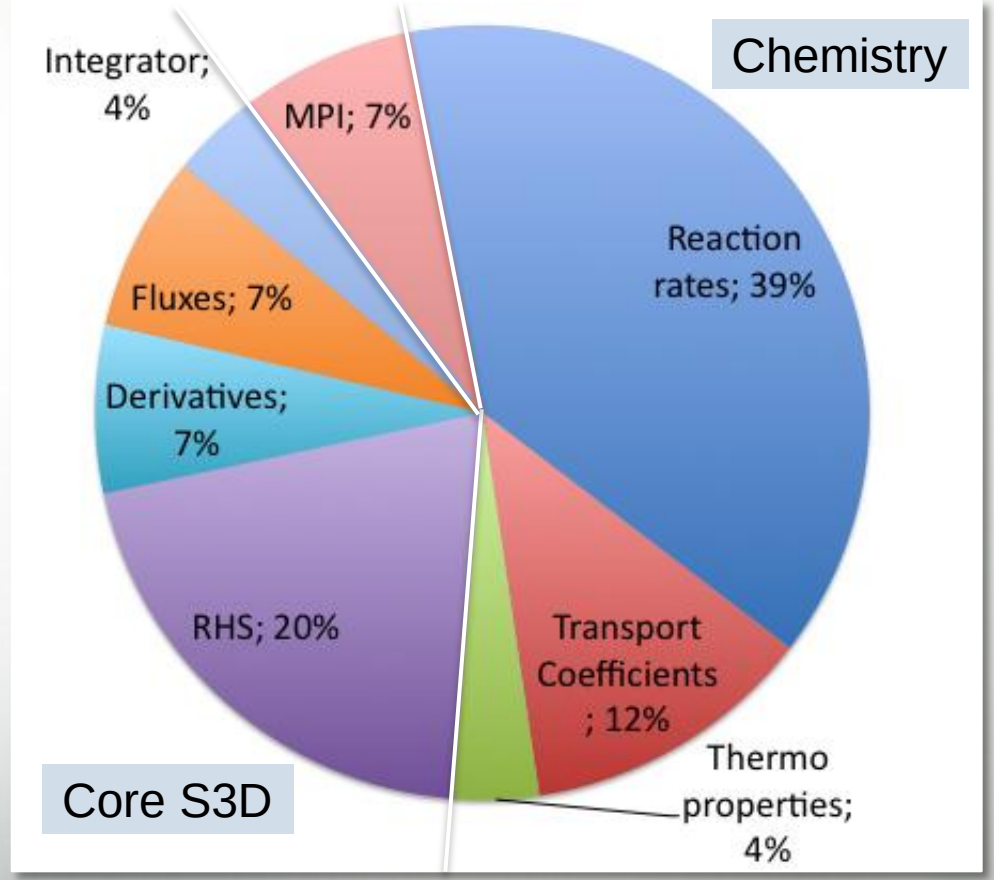
- Structured Cartesian mesh flow solver
- Solves compressible reacting Navier-Stokes, energy and species conservation equations.
  - 8th order explicit finite difference method
  - 4th order Runge-Kutta integrator with error estimator
- Detailed gas-phase thermodynamic, chemistry and molecular transport 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 * 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



Acceleration Strategy

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

		S3D		
Time Step		Solve_Drive		
Time Step	Runge K	Integrate		
Time Step	Runge K	RHS		
Time Step	Runge K		get mass fraction	I,j,k,n_spec loops
Time Step	Runge K		get_velocity	I,j,k,n_spec loops
Time Step	Runge K		calc_inv_avg	I,j,k,n_spec loops
Time Step	Runge K		calc_temp	I,j,k,n_spec loops
Time Step	Runge K		Compute Grads	I,j,k,n_spec loops
Time Step	Runge K		Diffusive Flux	I,j,k,n_spec loops
Time Step	Runge K		Derivatives	I,j,k,n_spec loops
Time Step	Runge K		reaction rates	I,j,k,n_spec loops

Profile from Original S3D

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group / Function / PE=HIDE / Thread=HIDE
100.0%	284.732812	--	--	156348682.1	Total
92.1%	262.380782	--	--	155578796.1	USER
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$thermchem_m_.LOOPS
4.3%	12.274200	0.167054	1.3%	1562500.0	mcavis_new$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	2.5%	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

		S3D	
Time Step		Solve_Drive	
Time Step	Runge K	Integrate	
Time Step	Runge K	RHS	
Time Step	Runge K	grid loop -omp	get mass fraction
Time Step	Runge K	grid loop-omp	get_velocity
Time Step	Runge K	grid loop-omp	calc_inv_avg
Time Step	Runge K	grid loop-omp	calc_temp
Time Step	Runge K	grid loop-omp	Compute Grads
Time Step	Runge K	grid loop-omp	Diffusive Flux
Time Step	Runge K	grid loop-omp	Derivatives
Time Step	Runge K	grid loop-omp	reaction rates

Statistics from running S3D

Table 1: Profile by Function Group and Function

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

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

		S3D	
Time Step – acc_data		Solve_Drive	
Time Step– acc_data	Runge K	Integrate	
Time Step– acc_data	Runge K	RHS	
Time Step– acc_data	Runge K	grid loop -ACC	get mass fraction
Time Step– acc_data	Runge K	grid loop-ACC	get_velocity
Time Step– acc_data	Runge K	grid loop-ACC	calc_inv_avg
Time Step– acc_data	Runge K	grid loop-ACC	calc_temp
Time Step– acc_data	Runge K	grid loop-ACC	Compute Grads
Time Step– acc_data	Runge K	grid loop-ACC	Diffusive Flux
Time Step– acc_data	Runge K	grid loop-ACC	Derivatives
Time Step– acc_data	Runge K	grid loop-ACC	reaction rates

What does OpenACC look like

```
#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

```
#ifdef GPU
!$acc parallel loop  gang  private(i,ml,mu)
#else
!$omp parallel private(i, ml, mu)
!$omp do
#endif
 do i = 1, nx*ny*nz, ms
  ml = i
  mu =  min(i+ms-1, nx*ny*nz)
  call calc_gamma_r( gamma, cpmix, avmolwt, ml, mu)
  call calc_press_r( pressure, q(1,1,1,4), temp, avmolwt, ml, mu )
  call calc_specEnth_allpts_r(temp, h_spec, ml, mu)
end do
#ifdef GPU
!$acc end parallel loop
#else
!$omp end parallel
#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 )
#ifdef GPU_ASYNC
!$acc update device(pos_f_x_buf(:,:,:,1:reqcount)) async(1)
#endif
   endif
```

Interprocedural Analysis with Inlining

- 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>

Whole Program Analysis -hwp

- 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  I         call get_mass_frac_r( q, volum, yspecies, ml, mu)            ! get Ys from rho*Ys, volum from rho
253.  1  I         call get_velocity_vec_r( u, q, volum, ml, mu)                ! fill the velocity vector
254.  1  I         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.
```

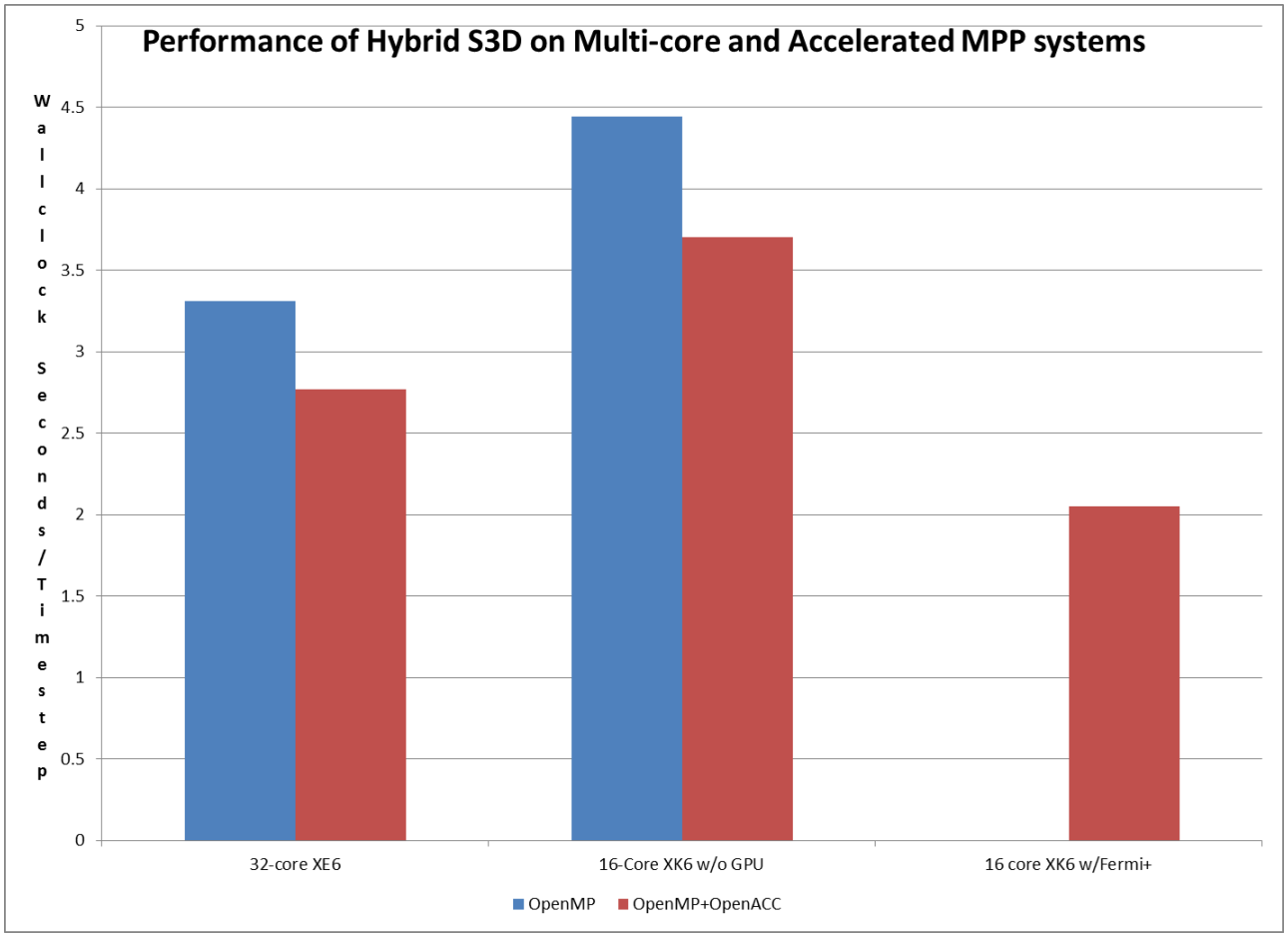
All Compiler Internal Errors are not errors

- 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

Performance of S3D Hybrid Code (4/30/12)



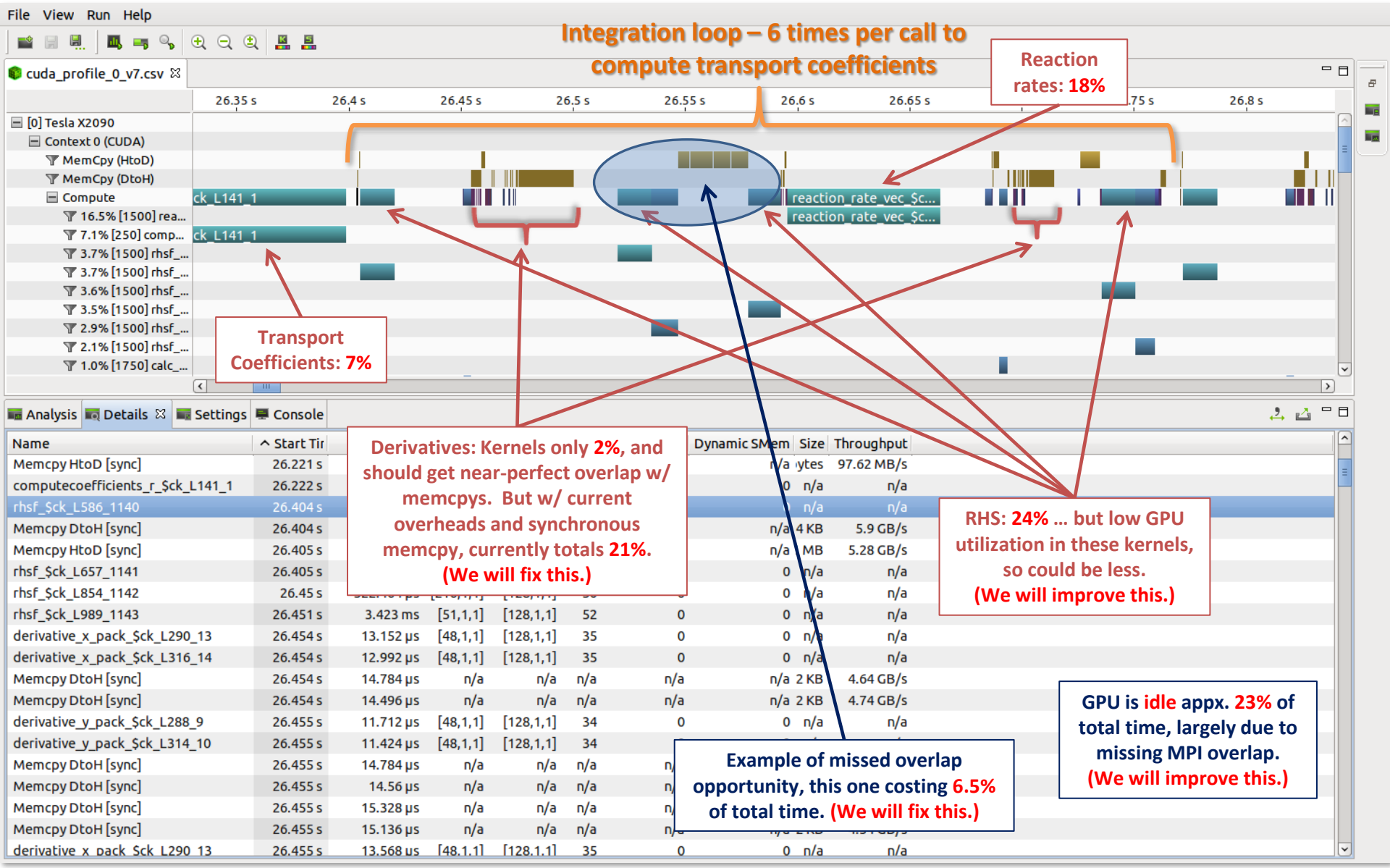
The latest version of S3D with OpenMP and OpenACC performs

- 1.8X faster on XK6 with GPU than a dual-CPU XE6
- 2.2X faster on XK6 with GPU than a XK6 without GPU

Comparions between OpenMP and OpenACC

Speedup	OpenMP Time	Acc Time%	Acc Time	Host Time	Acc Copy In (MBytes)	Acc Copy Out (MBytes)	Calls	Function PE=HIDE Thread=HIDE
		100.00%	187.783	234.011	148248	81843.75	1960504	Total
		---	---	---	---	---	---	---
2.094895	94.22	24.00%	44.976	0.042	--	--	600	reaction_rate_vec_.ACC_KERNEL@li.165
4.362416	65	7.90%	14.9	0.002	--	--	100	computecoefficients_r_.ACC_KERNEL@li.141
		7.50%	14.074	0.064	77456.25	--	600	rhsf_.ACC_COPY@li.1806
1.781216	19.8	5.90%	11.116	0.01	--	--	600	rhsf_.ACC_KERNEL@li.657
2.889151	29.4	5.40%	10.176	0.037	--	--	600	rhsf_.ACC_KERNEL@li.379
2.527231	25.29	5.30%	10.007	0.039	--	--	600	rhsf_.ACC_KERNEL@li.1712
0.314012	3.05	5.20%	9.713	0.039	--	--	600	rhsf_.ACC_KERNEL@li.1810
1.124026	7.93	3.80%	7.055	0.037	--	--	600	rhsf_.ACC_KERNEL@li.1758
4.46391	25.48	3.00%	5.708	0.037	--	--	600	rhsf_.ACC_KERNEL@li.419
		3.00%	5.627	5.63	--	--	600	reaction_rate_vec_.ACC_COPY@li.4958
		2.90%	5.373	5.376	--	--	600	rhsf_.ACC_COPY@li.2212
		1.70%	3.131	0.008	--	19490.63	100	integrate_.ACC_COPY@li.74
9.521674	25.48	1.40%	2.676	0.014	--	--	700	calc_primary_vars_.ACC_KERNEL@li.42
		1.40%	2.584	0.086	14175	--	600	rhsf_.ACC_COPY@li.366
		1.20%	2.266	1.526	--	6496.875	92400	derivative_z_pack_np_.ACC_COPY@li.351
5.925433	13.35	1.20%	2.253	0.008	--	--	600	rhsf_.ACC_KERNEL@li.989
		1.10%	2.104	0.042	--	12993.75	1800	derivative_y_pack_np_.ACC_COPY@li.429
		1.10%	2.104	0.073	--	12993.75	1800	derivative_x_pack_np_.ACC_COPY@li.433
		1.10%	2.053	1.247	--	6496.875	92400	derivative_z_pack_np_.ACC_COPY@li.340
4.48176	7.74	0.90%	1.727	0.009	--	--	600	integrate_.ACC_KERNEL@li.113

GPU Annotated Timeline



Name	Start Tir					Dynamic SMem	Size	Throughput
Memcpy HtoD [sync]	26.221 s						ytes	97.62 MB/s
computecoefficients_r_$ck_L141_1	26.222 s					0	n/a	n/a
rhsf_$ck_L586_1140	26.404 s						n/a	n/a
Memcpy DtoH [sync]	26.404 s					n/a	4 KB	5.9 GB/s
Memcpy HtoD [sync]	26.405 s					n/a	MB	5.28 GB/s
rhsf_$ck_L657_1141	26.405 s					0	n/a	n/a
rhsf_$ck_L854_1142	26.45 s					0	n/a	n/a
rhsf_$ck_L989_1143	26.451 s	3.423 ms	[51,1,1]	[128,1,1]	52	0	0	n/a	n/a
derivative_x_pack_$ck_L290_13	26.454 s	13.152 µs	[48,1,1]	[128,1,1]	35	0	0	n/a	n/a
derivative_x_pack_$ck_L316_14	26.454 s	12.992 µs	[48,1,1]	[128,1,1]	35	0	0	n/a	n/a
Memcpy DtoH [sync]	26.454 s	14.784 µs	n/a	n/a	n/a	n/a	n/a	2 KB	4.64 GB/s
Memcpy DtoH [sync]	26.454 s	14.496 µs	n/a	n/a	n/a	n/a	n/a	2 KB	4.74 GB/s
derivative_y_pack_$ck_L288_9	26.455 s	11.712 µs	[48,1,1]	[128,1,1]	34	0	0	n/a	n/a
derivative_y_pack_$ck_L314_10	26.455 s	11.424 µs	[48,1,1]	[128,1,1]	34				
Memcpy DtoH [sync]	26.455 s	14.784 µs	n/a	n/a	n/a	n/			
Memcpy DtoH [sync]	26.455 s	14.56 µs	n/a	n/a	n/a	n/			
Memcpy DtoH [sync]	26.455 s	15.328 µs	n/a	n/a	n/a	n/			
Memcpy DtoH [sync]	26.455 s	15.136 µs	n/a	n/a	n/a	n/			
derivative_x_pack_$ck_L290_13	26.455 s	13.568 µs	[48,1,1]	[128,1,1]	35	0	0	n/a	n/a

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

!$acc host_data use_device

```
#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

    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
#ifdef GPU
!$acc end host_data
!$acc end data
#endif
```

Thank you. Questions?