

# Programming for Future

## Multi-Petaflop and Exaflop Computers

---

aka

Finding more parallelism in existing applications

# Cray Software Objectives for Accelerators

1. Provide baseline accelerator environment
  - Don't preclude use of tools developers/programmers are used to
2. Integrated Programming Environment
  - Extension of PE Cray has provided on XT/XE systems
  - Provide "bundled" 3<sup>rd</sup> party commonly used or expected software (compilers, libraries, tools)
3. Cray integrated programming environment include:
  - Greatly enhance the productivity of the programming writing new applications or porting existing applications to accelerators
  - Improve performance of existing applications by exploiting greater levels of parallelism
  - Maintain source compatibility between multi-core and accelerator versions of the code

- 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

- 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 OpenACC and the 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.

## Additional Note about Accelerators

- Many application developers, particularly ISVs have not put in the effort to convert their application to MPI. These applications potentially can utilize accelerators on the node to significantly enhance their performance
  - Most of these do use OpenMP parallelization – by adding OpenMP accelerator extensions, these application should be able to benefit from 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 compiler 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

- Cuda
- OpenCL
- Existing Fortran, C and C++ with extensions
  - Pthreads, Thread Building Blocks
  - 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

## comparison of Programming Models for the Node

	portability	Performance	Perf/portability	Ease of Use
Cuda	2	5	1	2
OpenCL	2	5	1	1
Existing Language with threading	3	4	2	3
Existing Language with directives	5	4	5	5

## Portability

	Nvidia GPGPU	AMD GPGPU	IBM Power	IBM BG	X86 multi-core
Cuda	x				
OpenCL	x	x			
Existing Language with threading			x	x	x
Existing Language with directives	x	x	x	x	x

## comparisons between Cuda and OpenMP accelerator extensions

- Cuda
  - Widely used programming model for effectively utilizing the accelerator
  - Flexibility to obtain good performance on the accelerator
- OpenMP accelerator extensions – things to prove
  - Are the directives powerful enough to allow the developer to pass information on to the compiler
  - Can the compiler generate code that get performance close to Cuda.

## Consider the following kernel

```

do k = 1,nz
  do j=1,ny
    SPECIES: do n=1,n_spec-1
      do i = 1,nx
        diffFlux(i,j,k,n,1) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,1) &
          + yspecies(i,j,k,n) * grad_mixMW(i,j,k,1) )
        diffFlux(i,j,k,n,2) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,2) &
          + yspecies(i,j,k,n) * grad_mixMW(i,j,k,2) )
        diffFlux(i,j,k,n,3) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,3) &
          + yspecies(i,j,k,n) * grad_mixMW(i,j,k,3) )
      enddo
      do i = 1,nx
        diffFlux(i,j,k,n_spec,1) = diffFlux(i,j,k,n_spec,1) - diffFlux(i,j,k,n,1)
        diffFlux(i,j,k,n_spec,2) = diffFlux(i,j,k,n_spec,2) - diffFlux(i,j,k,n,2)
        diffFlux(i,j,k,n_spec,3) = diffFlux(i,j,k,n_spec,3) - diffFlux(i,j,k,n,3)
      enddo
    enddo SPECIES
    do i = 1,nx
      grad_T(i,j,k,1) = -lambda(i,j,k) * grad_T(i,j,k,1)
      grad_T(i,j,k,2) = -lambda(i,j,k) * grad_T(i,j,k,2)
      grad_T(i,j,k,3) = -lambda(i,j,k) * grad_T(i,j,k,3)
    enddo
    do n=1,n_spec
      do i = 1,nx
        grad_T(i,j,k,1) = grad_T(i,j,k,1) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,1)
        grad_T(i,j,k,2) = grad_T(i,j,k,2) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,2)
        grad_T(i,j,k,3) = grad_T(i,j,k,3) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,3)
      enddo
    enddo
  enddo
enddo

```

## A rewrite for Cuda Fortran First extract to kernel and call



```
! transfer

grad_Ys_d = grad_Ys_h
diffFlux_d = diffFlux_h
yspecies_d = yspecies_h
h_spec_d = h_spec_h
ds_mixavg_d = ds_mixavg_h
grad_t_d = grad_t_h
grad_mixMW_d = grad_mixMW_h
lambda_d = lambda_h

grid = dim3((nxyz-1)/512+1,1,1)
threadBlock = dim3(512,1,1)

call calcDiffFlux<<<grid,threadBlock>>>(lambda_d, grad_T_d, grad_mixMW_d, diffFlux_d, &
    Ds_mixavg_d, yspecies_d, grad_Ys_d, h_spec_d)
ierr = cudaThreadSynchronize()
```

## A rewrite for Cuda Fortran And now the kernel (1)



```
i = (blockIdx%x-1)*blockDim%x + threadIdx%x
if (i <= nxxyz) then
    ! move first part of grad T here
    lambda_r = lambda(i)
    grad_T_1 = -lambda_r*grad_T(i,1)
    grad_T_2 = -lambda_r*grad_T(i,2)
    grad_T_3 = -lambda_r*grad_T(i,3)
    ! now diffFlux
    diffFlux_n_spec_1 = diffFlux(i,n_spec,1)
    diffFlux_n_spec_2 = diffFlux(i,n_spec,2)
    diffFlux_n_spec_3 = diffFlux(i,n_spec,3)
    grad_mixMW_1 = grad_mixMW(i,1)
    grad_mixMW_2 = grad_mixMW(i,2)
    grad_mixMW_3 = grad_mixMW(i,3)
    do n=1, n_spec-1
        Ds_mixavg_r = Ds_mixavg(i,n)
        yspecies_r = yspecies(i,n)
        diffFlux_1 = - Ds_mixavg_r * (grad_Ys(i,n,1) + yspecies_r*grad_mixMW_1)
        diffFlux_2 = - Ds_mixavg_r * (grad_Ys(i,n,2) + yspecies_r*grad_mixMW_2)
        diffFlux_3 = - Ds_mixavg_r * (grad_Ys(i,n,3) + yspecies_r*grad_mixMW_3)
        diffFlux_n_spec_1 = diffFlux_n_spec_1 - diffFlux_1
        diffFlux_n_spec_2 = diffFlux_n_spec_2 - diffFlux_2
        diffFlux_n_spec_3 = diffFlux_n_spec_3 - diffFlux_3
        h_spec_r = h_spec(i,n)
        grad_T_1 = grad_T_1 + h_spec_r*diffFlux_1
        grad_T_2 = grad_T_2 + h_spec_r*diffFlux_2
        grad_T_3 = grad_T_3 + h_spec_r*diffFlux_3
        diffFlux(i,n,1) = diffFlux_1
        diffFlux(i,n,2) = diffFlux_2
        diffFlux(i,n,3) = diffFlux_3
    enddo
```

- 1) Unrolled all dir loops
- 2) Combined two n\_spec loops
- 3) Moved around computation
- 4) Assigned most arrays to temps

## A rewrite for Cuda Fortran And now the kernel (2)



```
! do n = n_spec iteration and write out final data

h_spec_r = h_spec(i,n_spec)
grad_T_1 = grad_T_1 + h_spec_r*diffFlux_n_spec_1
grad_T_2 = grad_T_2 + h_spec_r*diffFlux_n_spec_2
grad_T_3 = grad_T_3 + h_spec_r*diffFlux_n_spec_3

grad_T(i,1) = grad_T_1
grad_T(i,2) = grad_T_2
grad_T(i,3) = grad_T_3

diffFlux(i,n_spec,1) = diffFlux_n_spec_1
diffFlux(i,n_spec,2) = diffFlux_n_spec_2
diffFlux(i,n_spec,3) = diffFlux_n_spec_3
endif
```

## Or Add directives

```

!$acc parallel private(i,j,k,n)
!$acc loop
o k = 1,nz
do j=1,ny
    SPECIES: do n=1,n_spec-1
        do i = 1,nx
            diffFlux(i,j,k,n,1) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,1) &
                + yspecies(i,j,k,n) * grad_mixMW(i,j,k,1) )
            diffFlux(i,j,k,n,2) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,2) &
                + yspecies(i,j,k,n) * grad_mixMW(i,j,k,2) )
            diffFlux(i,j,k,n,3) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,3) &
                + yspecies(i,j,k,n) * grad_mixMW(i,j,k,3) )
        enddo
        do i = 1,nx
            diffFlux(i,j,k,n_spec,1) = diffFlux(i,j,k,n_spec,1) - diffFlux(i,j,k,n,1)
            diffFlux(i,j,k,n_spec,2) = diffFlux(i,j,k,n_spec,2) - diffFlux(i,j,k,n,2)
            diffFlux(i,j,k,n_spec,3) = diffFlux(i,j,k,n_spec,3) - diffFlux(i,j,k,n,3)
        enddo
    enddo SPECIES
    do i = 1,nx
        grad_T(i,j,k,1) = -lambda(i,j,k) * grad_T(i,j,k,1)
        grad_T(i,j,k,2) = -lambda(i,j,k) * grad_T(i,j,k,2)
        grad_T(i,j,k,3) = -lambda(i,j,k) * grad_T(i,j,k,3)
    enddo
    do n=1,n_spec
        do i = 1,nx
            grad_T(i,j,k,1) = grad_T(i,j,k,1) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,1)
            grad_T(i,j,k,2) = grad_T(i,j,k,2) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,2)
            grad_T(i,j,k,3) = grad_T(i,j,k,3) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,3)
        enddo
    enddo
enddo
!$acc end loop
!$acc end parallel

```

## What the compiler Shows

```

218. 1 G-----< !$acc parallel private(i,j,k,n)
219. 1 G           !$acc loop
220. 1 G C-----<      do k = 1,nz
221. 1 G C g-----<      do j=1,ny
222. 1 G C g 5----<          SPECIES: do n=1,n_spec-1
223. 1 G C g 5 gf--<          do i = 1,nx
224. 1 G C g 5 gf          ! driving force is just the gradient in mole fraction:
225. 1 G C g 5 gf          diffFlux(i,j,k,n,1) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,1) &
226. 1 G C g 5 gf          + yspecies(i,j,k,n) * grad_mixMW(i,j,k,1) )
227. 1 G C g 5 gf          diffFlux(i,j,k,n,2) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,2) &
228. 1 G C g 5 gf          + yspecies(i,j,k,n) * grad_mixMW(i,j,k,2) )
229. 1 G C g 5 gf          diffFlux(i,j,k,n,3) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,3) &
230. 1 G C g 5 gf          + yspecies(i,j,k,n) * grad_mixMW(i,j,k,3) )
231. 1 G C g 5 gf-->
232. 1 G C g 5 f---<
233. 1 G C g 5 f
234. 1 G C g 5 f
235. 1 G C g 5 f
236. 1 G C g 5 f--->
237. 1 G C g 5---->
238. 1 G C g g----<
239. 1 G C g g
240. 1 G C g g
241. 1 G C g g
242. 1 G C g g---->
243. 1 G C g 5----<
244. 1 G C g 5 g---<
245. 1 G C g 5 g
246. 1 G C g 5 g
247. 1 G C g 5 g
248. 1 G C g 5 g--->
249. 1 G C g 5---->
250. 1 G C g----->
251. 1 G C----->
252. 1 G           !$acc end loop
253. 1 G-----> !$acc end parallel

```

## Legend to compiler Notes

Primary Loop Type	Modifiers
A - Pattern matched	a - atomic memory operation
C - Collapsed	b - blocked
D - Deleted	c - conditional and/or computed
E - Cloned	f - fused
G - Accelerated	g - partitioned
I - Inlined	i - interchanged
M - Multithreaded	m - partitioned
V - Vectorized	n - non-blocking remote transfer
	p - partial
	r - unrolled
	s - shortloop
	w - unwound

## And a little restructuring

```

 !$acc parallel private(t_1,t_2,t_3,I,j,k,n)
 !$acc loop collapse(3)
   do k = 1,nz
     do j=1,ny
       do i = 1,nx
         grad_T(i,j,k,1) = -lambda(i,j,k) * grad_T(i,j,k,1)
         grad_T(i,j,k,2) = -lambda(i,j,k) * grad_T(i,j,k,2)
         grad_T(i,j,k,3) = -lambda(i,j,k) * grad_T(i,j,k,3)
         t_1=diffFlux(i,j,k,n_spec,1)
         t_2=diffFlux(i,j,k,n_spec,2)
         t_3=diffFlux(i,j,k,n_spec,3)
         SPECIES: do n=1,n_spec
           if(n < n_spec) then
             diffFlux(i,j,k,n,1) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,1) &
               + yspecies(i,j,k,n) * grad_mixMW(i,j,k,1) )
             diffFlux(i,j,k,n,2) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,2) &
               + yspecies(i,j,k,n) * grad_mixMW(i,j,k,2) )
             diffFlux(i,j,k,n,3) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,3) &
               + yspecies(i,j,k,n) * grad_mixMW(i,j,k,3) )
             t_1 = t_1 - diffFlux(i,j,k,n,1)
             t_2 = t_2 - diffFlux(i,j,k,n,2)
             t_3 = t_3 - diffFlux(i,j,k,n,3)
           end if
           grad_T(i,j,k,1) = grad_T(i,j,k,1) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,1)
           grad_T(i,j,k,2) = grad_T(i,j,k,2) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,2)
           grad_T(i,j,k,3) = grad_T(i,j,k,3) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,3)
         enddo SPECIES
         diffFlux(i,j,k,n_spec,1) = t_1
         diffFlux(i,j,k,n_spec,2) = t_2
         diffFlux(i,j,k,n_spec,3) = t_3
       enddo
     enddo
   enddo
 !$acc end loop
 !$acc end parallel

```

## What the compiler Shows



```
219. 1 G-----< !$acc parallel private(t_1,t_2,t_3,I,j,k,n)
220. 1 G           !$acc loop collapse(3)
221. 1 G C-----<      do k = 1,nz
222. 1 G C C-----<          do j=1,ny
223. 1 G C C g----<            do i = 1,nx
224. 1 G C C g           grad_T(i,j,k,1) = -lambda(i,j,k) * grad_T(i,j,k,1)
225. 1 G C C g           grad_T(i,j,k,2) = -lambda(i,j,k) * grad_T(i,j,k,2)
226. 1 G C C g           grad_T(i,j,k,3) = -lambda(i,j,k) * grad_T(i,j,k,3)
228. 1 G C C g           t_1=diffFlux(i,j,k,n_spec,1)
229. 1 G C C g           t_2=diffFlux(i,j,k,n_spec,2)
230. 1 G C C g           t_3=diffFlux(i,j,k,n_spec,3)
232. 1 G C C g 6--<      SPECIES: do n=1,n_spec
234. 1 G C C g 6        if(n < n_spec) then
235. 1 G C C g 6          diffFlux(i,j,k,n,1) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,1) &
236. 1 G C C g 6                  + yspecies(i,j,k,n) * grad_mixMW(i,j,k,1) )
237. 1 G C C g 6          diffFlux(i,j,k,n,2) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,2) &
238. 1 G C C g 6                  + yspecies(i,j,k,n) * grad_mixMW(i,j,k,2) )
239. 1 G C C g 6          diffFlux(i,j,k,n,3) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,n,3) &
240. 1 G C C g 6                  + yspecies(i,j,k,n) * grad_mixMW(i,j,k,3) )
242. 1 G C C g 6          t_1 = t_1 - diffFlux(i,j,k,n,1)
243. 1 G C C g 6          t_2 = t_2 - diffFlux(i,j,k,n,2)
244. 1 G C C g 6          t_3 = t_3 - diffFlux(i,j,k,n,3)
246. 1 G C C g 6        end if
248. 1 G C C g 6        grad_T(i,j,k,1) = grad_T(i,j,k,1) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,1)
249. 1 G C C g 6        grad_T(i,j,k,2) = grad_T(i,j,k,2) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,2)
250. 1 G C C g 6        grad_T(i,j,k,3) = grad_T(i,j,k,3) + h_spec(i,j,k,n)*diffFlux(i,j,k,n,3)
251. 1 G C C g 6
252. 1 G C C g 6-->      enddo SPECIES
254. 1 G C C g          diffFlux(i,j,k,n_spec,1) = t_1
255. 1 G C C g          diffFlux(i,j,k,n_spec,2) = t_2
256. 1 G C C g          diffFlux(i,j,k,n_spec,3) = t_3
258. 1 G C C g---->      enddo
259. 1 G C C---->      enddo
260. 1 G C---->      enddo
261. 1 G           !$acc end loop
262. 1 G-----> !$acc end parallel
```

And the timings  
Ignoring Data Transfer\*



	Original OpenMP Across entire node	Cuda Fortran	Directive Approach	Directive Approach Restructured
Kernel Only	.0417 Seconds	.0061 Seconds	.0113 Seconds	.0067 Seconds

\* In S3D all of the arrays used in this computation will reside on the Accelerator prior to the invocation of the kernel.

## Ease of Use

Kernel	Lines of Code	Cuda lines of Code Added or changed	Lines of Code added or changed for directives	Directive lines of Code added
comp_heat	65	45	0	4
comp_heat (opt)	65	45	9	4
Getrates	5869	6362	0	4
Divergent_sphere	89	44	0	4
Gradient_sphere	45	42	0	4



## 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

## Gathering High Level looping statistics

Table 1: Profile by Function Group and Function

Time%	Time	Imb.	Imb.	Calls	Group
		Time	Time%		Function
					PE=HIDE
					Thread=HIDE
100.0%	96.774934	--	--	6083.9	Total
<hr/>					
100.0%	96.766637	--	--	6056.9	USER
<hr/>					
18.7%	18.125552	15.748098	49.6%	200.0	streaming_.LOOPS
16.4%	15.904828	5.941168	29.0%	200.0	recolor_.LOOP@li.723
11.3%	10.909372	0.428680	4.0%	1.0	initmpi_.LOOP@li.313
9.9%	9.614664	0.047006	0.5%	1.0	read_parallel_.LOOPS
8.5%	8.220680	3.385983	31.1%	200.0	streaming_.LOOP@li.774
7.9%	7.622818	1.101631	13.5%	200.0	streaming_exchange_.LOOPS
4.9%	4.699645	1.298570	23.1%	200.0	collisiona_.LOOP@li.456
4.4%	4.257534	1.162477	22.9%	200.0	collisionb_.LOOP@li.607
3.8%	3.637698	0.803541	19.3%	201.0	cal_velocity_.LOOP@li.874
3.5%	3.421273	0.450034	12.4%	200.0	wall_boundary_.LOOP@li.802
2.2%	2.119447	5.224182	75.9%	201.0	injection_.LOOPS
2.1%	2.031823	2.732934	61.2%	200.0	collisionb_.LOOPS
1.3%	1.252738	0.188434	13.9%	201.0	injection_.LOOP@li.967

Table 1: Profile by Function and Callers

Time%	Time	Calls	Group
			Function
			Caller
			PE=HIDE
100.0%	333.310995	120405.1	Total
-----			
96.0%	320.005729	99841.2	USER
-----			
90.0%	299.917011	2000.0	compute_forces_elastic_dev_.LOOPS
3          compute_forces_elastic_dev_			
4          compute_forces_elastic_.LOOPS			
5          compute_forces_elastic_			
6          iterate_time_.LOOPS			
7          iterate_time_			
8          specfem3d_			
9          xspecfem3d_			
2.2%	7.348773	1000.0	it_update_displacement_scheme_.LOOPS
3          it_update_displacement_scheme_			
4          iterate_time_.LOOPS			
5          iterate_time_			
6          specfem3d_			
7          xspecfem3d_			
1.8%	6.030917	1000.0	compute_forces_elastic_.LOOPS
3          compute_forces_elastic_			
4          iterate_time_.LOOPS			
5          iterate_time_			
6          specfem3d			
7          xspecfem3d_			
0.9%	2.843254	1.0	locate_receivers_.LOOPS
3          locate_receivers_			
4          setup_receivers_.LOOPS			
5          setup_receivers_			
6          setup_sources_receivers_			
7          specfem3d_			
8          xspecfem3d_			
0.4%	1.477712	1.0	write_vtk_data_elem_1_.LOOPS
3          write_vtk_data_elem_1_			
4          rmd_setup_inner_outer_elems_.LOOPS			
5          rmd_setup_inner_outer_elems_			
6          read_mesh_databases_			
7          specfem3d_			
8          xspecfem3d_			

## Gathering High Level looping statistics

Table 2: Loop Stats from -hprofile\_generate

Loop	Loop Incl	Loop Incl	Loop Hit	Loop Trips	Loop	Function=/.LOOP\.
86.0%	464.222802	464.222802	1	200.0	novec	lbgm3d2p_d_.LOOP.4.li.118
9.0%	48.424200	0.242121	200	34.0	novec	recolor_.LOOP.0.li.722
6.8%	36.439476	0.000455	80000	101.0	novec	recolor_.LOOP.1.li.723
6.7%	36.394050	0.000005	8080000	101.0	novec	recolor_.LOOP.2.li.724
5.6%	30.165525	0.000001	21111225	15.0	novec	recolor_.LOOP.3.li.744
4.6%	24.770886	0.123854	200	14.0	novec	recolor_.LOOP.0.li.773
4.5%	24.500305	0.122502	200	34.0	novec	collisionb_.LOOP.5.li.606
4.4%	23.493726	0.117469	200	34.0	novec	wall_boundary_.LOOP.0.li.801
4.3%	23.470821	0.116770	201	34.0	novec	injection_.LOOP.0.li.966
4.3%	23.415045	0.116493	201	34.0	novec	cal_velocity_.LOOP.0.li.873
4.3%	23.307203	0.116536	200	34.0	novec	collisiona_.LOOP.0.li.455
4.3%	23.256005	0.116280	200	34.0	novec	collisionb_.LOOP.0.li.570
3.5%	18.858521	18.858521	1	400.0	novec	read_parallel_.LOOP.0.li.854
2.8%	15.080327	0.037701	400	1014776.0	novec	recolor_.LOOP.1.li.774
1.5%	8.200184	0.000103	80000	101.0	novec	collisiona_.LOOP.1.li.456
1.5%	8.148148	0.000001	8080000	101.0	novec	collisiona_.LOOP.2.li.457
1.4%	7.553060	0.000094	80000	101.0	novec	collisionb_.LOOP.6.li.607
1.4%	7.489768	0.000001	8080000	101.0	sunwind	collisionb_.LOOP.7.li.608
1.2%	6.399606	0.000080	80400	101.0	novec	cal_velocity_.LOOP.1.li.874
1.2%	6.344608	0.000001	8120400	101.0	sunwind	cal_velocity_.LOOP.2.li.875
1.2%	6.229104	0.000078	80000	101.0	novec	wall_boundary_.LOOP.1.li.802
1.1%	6.145206	0.000001	8080000	101.0	sunwind	wall_boundary_.LOOP.2.li.803

## Current benchmark we are working

```
do ii=1,ntimes
    if(ip == 0) then
        print *, 'step = ',ii*npasses
    endif

    do jj=1,npasses
        if(ip == 0) then
            write(*,*) '***** step = *****', ((ii-1)*npasses)+jj
        endif
        call set_boundary_macro_press2
        call set_boundary_micro_press
        call collisiona
        call collisionb
        call recolor
        call streaming
        call wall_boundary
        call cal_velocity
        call injection
    end do

    etime = mpi_wtime()
    call saturation

end do
```

## Converting the MPI application to a Hybrid OpenMP/MPI application

### Task 2 Parallel Analysis, Scoping and Vectorization

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

## Task 2 Parallel Analysis and Scoping

- Tools that will be needed
  - Whole program analysis scoping tool with User interaction
    - The compiler performs an initial parallelization analysis to identify obvious inhibitors to parallelization
    - The User instructs the compiler to ignore various inhibitors if possible
    - The compiler performs an initial scoping analysis and presents the User with concerns with array usage
    - The User works with the environment to trace variables through the high level looping structure, works with the compiler to scope the variables in question.
  - Vectorization Feedback from the compiler
    - Tremendous experience from years of vector architectures

## Lets scope out the potential OpenMP loops

```
-- Loop starting at line 221
auto shared(cell,local_lx,local_ly,lz,rho,uxyz)
-- Loop starting at line 262
auto shared(b,cell,local_lx,local_ly,lz,r,rho,uxyz)
-- Loop starting at line 438
auto shared(b,cell,local_lx,local_ly,lz,r,rho,uxyz)
-- Loop starting at line 558
auto shared(cell,grad,local_lx,local_ly,lz,rho,wet)
-- Loop starting at line 572
auto shared(cell,grad,local_lx,local_ly,lz,wet)
-- Loop starting at line 591
auto shared
(b,cell,ci1,ci10,ci11,ci12,ci13,ci14,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,grad,
 local_lx,local_ly,lz)
-- Loop starting at line 712
auto firstprivate(crit,icrit)
auto shared(b,cell,cix,ciy,ciz,local_lx,local_ly,lz,r,rho,uxyz)
```

**msg-obj: fi FAIL -- Value/Shared Scope Conflict.**

```
-- Loop starting at line 784
auto shared(b,index,index_max,r)
-- Loop starting at line 812
auto shared(b,cell,local_lx,local_ly,lz,r)
-- Loop starting at line 965
auto shared(b,cell,local_lx,local_ly,lz,r,rho,uxyz)
-- Loop starting at line 1125
auto shared(b,bbar,blue,cell,local_lx,local_ly,lz,r,rbar,red,rho,surf)
auto reduction(:bbar,:rbar)
```

## Vectorization Problem in recolor – call to maxloc

```

723. 1-----<    do k=0,lz-1
724. 1 2-----<    do j=0,local_ly-1
725. 1 2 3-----<    do i=0,local_lx-1
726. 1 2 3          if (cell(i,j,k)==0) then
727. 1 2 3            fi(0) = r0(i,j,k) + b0(i,j,k)
728. 1 2 3            fi(1) = r1(i,j,k) + b1(i,j,k)
729. 1 2 3            fi(2) = r2(i,j,k) + b2(i,j,k)
742. 1 2 3
743. 1 2 3 Vw---<>          crit(1:15) = (cix(0:14)*fx(i,j,k)+ciy(0:14)*fy(i,j,k)+&
744. 1 2 3                                     ciz(0:14)*fz(i,j,k))
745. 1 2 3 4----<          do l=0,14
746. 1 2 3 4 w--<>          max_loc      = maxloc(crit)
747. 1 2 3 4          mm           = max_loc(1)
748. 1 2 3 4          crit(mm)     = worst_value
749. 1 2 3 4          mm           = mm-1
750. 1 2 3 4          frac_r       = max(0.0d0,min(rho_r(i,j,k),fi(mm)))
751. 1 2 3 4          rho_r(i,j,k) = rho_r(i,j,k)-frac_r
752. 1 2 3 4
753. 1 2 3 4          R(i,j,k,mm) = frac_r
754. 1 2 3 4          B(i,j,k,mm) = fi(mm) - R(i,j,k,mm)
755. 1 2 3 4---->          end do
756. 1 2 3          end if
757. 1 2 3----->          end do
758. 1 2----->          end do
759. 1----->          end do

```

## Recolor Routine – now vectorized

```

do k=0,lz-1
    do j=0,local_ly-1
        do i=0,local_lx-1
            crit(i,:)=0.0
            icrit(i,:)= 0
            if (cell(i,j,k)==0) then
                o o o
                do iii = 1,14
                    do ii = 1,14
                        if(crit(i,ii).gt.crit(i,ii+1))then
                            crit_temp = crit(i,ii+1)
                            icrit_temp = icrit(i,ii+1)
                            crit(i,ii+1)=crit(i,ii)
                            icrit(i,ii+1) = icrit(i,ii)
                            crit(i,ii)=crit_temp
                            icrit(i,ii) = icrit_temp
                        endif
                    enddo
                enddo

                do ii = 15,1,-1
                    mm           = icrit(i,ii)-1
                    frac_r       = max(0.0d0,min(rho(2,i,j,k),fi(i,mm)))
                    rho(2,i,j,k) = rho(2,i,j,k)-frac_r
                enddo
            endif
        enddo
    enddo
enddo

```

## 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
- compilation of OpenMP regions to accelerator by the compiler is approaching the performance of hand-coded CUDA or OpenCL with the advantage that it results in portable code. And it will only get better.
  - With OpenMP extensions targeting accelerators, data transfers between multi-core socket and the accelerator can be optimized. Utilization of registers and shared memory can also be optimized
  - With OpenMP extensions targeting accelerators, user can control the utilization of the accelerator memory and functional units.

## Now we start adding directives -recolor

```
!$acc parallel_loop private(mm,i,j,k,l,fi,crit,frac_r,ii,crit_max,crit_temp,
!$acc&                                icrit_temp,icrit,iii)
do k=0,lz-1
do j=0,local_ly-1
do i=0,local_lx-1
crit(i,:)=0.0
icrit(i,:)= 0
if (cell(i,j,k)==0) then
    fi(i,0) = r(i,j,k,0) + b(i,j,k,0)
    fi(i,1) = r(i,j,k,1) + b(i,j,k,1)
    fi(i,2) = r(i,j,k,2) + b(i,j,k,2)
    fi(i,3) = r(i,j,k,3) + b(i,j,k,3)
    fi(i,4) = r(i,j,k,4) + b(i,j,k,4)
    fi(i,5) = r(i,j,k,5) + b(i,j,k,5)
    fi(i,6) = r(i,j,k,6) + b(i,j,k,6)
    fi(i,7) = r(i,j,k,7) + b(i,j,k,7)
    fi(i,8) = r(i,j,k,8) + b(i,j,k,8)
    fi(i,9) = r(i,j,k,9) + b(i,j,k,9)
    fi(i,10) = r(i,j,k,10) + b(i,j,k,10)
```

## Now we start adding directives -collisionb

```

!$acc parallel loop private(k,j,i,fx_tmp,fy_tmp,fz_tmp,cif2)
do k=0,lz-1
  do j=0,local_ly-1
    do i=0,local_lx-1
      fx_tmp = 0.0d0
      fy_tmp = 0.0d0
      fz_tmp = 0.0d0
      if (cell(i,j,k)==0) then
        fx_tmp =grad(i+1,j ,k )-grad(i-1,j ,k )&
          +grad(i+1,j+1,k+1)-grad(i-1,j+1,k+1)&
          +grad(i+1,j-1,k+1)+grad(i+1,j+1,k-1)&
          -grad(i-1,j-1,k-1)+grad(i+1,j-1,k-1)&
          -grad(i-1,j+1,k-1)-grad(i-1,j-1,k+1)
        fy_tmp =grad(i ,j+1,k )-grad(i ,j-1,k )&
          +grad(i+1,j+1,k+1)+grad(i-1,j+1,k+1)&
          -grad(i+1,j-1,k+1)+grad(i+1,j+1,k-1)&
          -grad(i-1,j-1,k-1)-grad(i+1,j-1,k-1)&
          +grad(i-1,j+1,k-1)-grad(i-1,j-1,k+1)
        fz_tmp =grad(i ,j ,k+1)-grad(i ,j ,k-1)&
          +grad(i+1,j+1,k+1)+grad(i-1,j+1,k+1)&
          +grad(i+1,j-1,k+1)-grad(i+1,j+1,k-1)&
          -grad(i-1,j-1,k-1)-grad(i+1,j-1,k-1)&

```

## 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 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?

## Task 4 Fine tuning of accelerated program

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

Several other vendors are supplying similar performance gathering tools

## Useful tools contd.

- Craypat profiling
  - Tracing: "pat\_build -u <executable>" (can do APA sampling first)
  - "pat\_report -O accelerator <.xf file>"; -T also useful
    - Other pat\_report tables (as of perftools/5.2.1.7534)
      - acc\_fu flat table of accelerator events
      - acc\_time call tree sorted by accelerator time
      - acc\_time\_fu flat table of accelerator events sorted by accelerator time
      - acc\_show\_by\_ct regions and events by calltree sorted alphabetically

## Run and gather runtime statistics

Table 1: Profile by Function Group and Function

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

## Keep data on the accelerator with acc\_data region

```

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

```

Now when we do communication we have to update the host

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

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

## Packing the buffers on the accelerator



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

## Final Profile - bulk of time in kernel execution

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

# 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

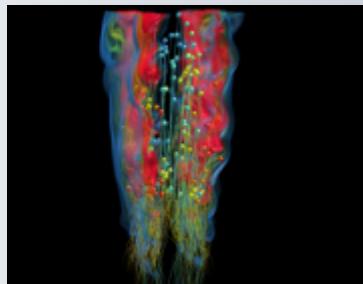
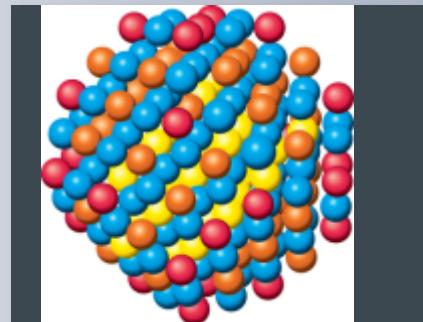
- Objective: Enhance productivity related to porting applications to hybrid multi-core systems
- Four core components
  - Cray Statistics Gathering Facility on host and GPU
  - Cray Optimization Explorer – Scoping Tools (COE)
  - Cray Compilation Environment (CCE)
  - Cray GPU Libraries

# Titan: Early Science Applications

**CRAY**  
THE SUPERCOMPUTER COMPANY

## WL-LSMS

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

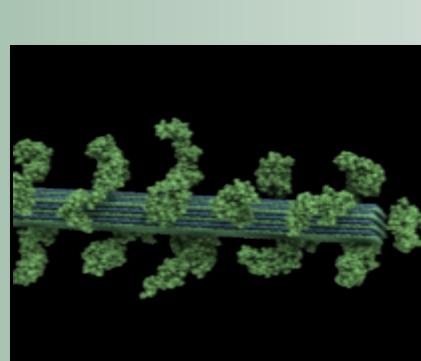
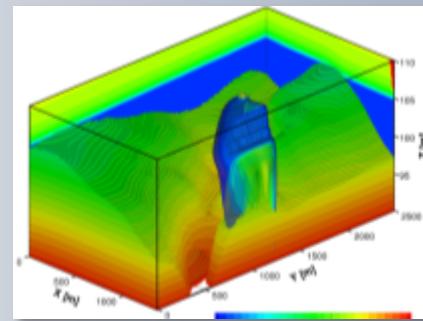


## S3D

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

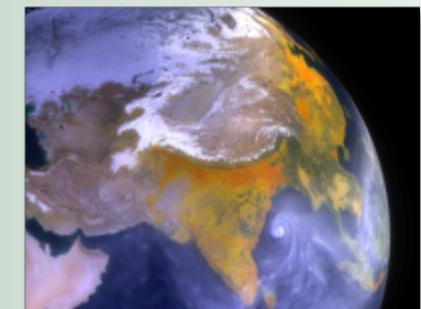
## PFLOTRAN

Stability and viability of large scale CO<sub>2</sub> sequestration; predictive containment groundwater transport



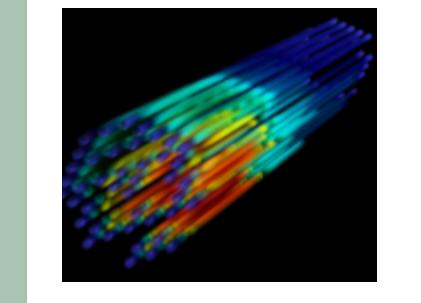
## CAM / HOMME

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



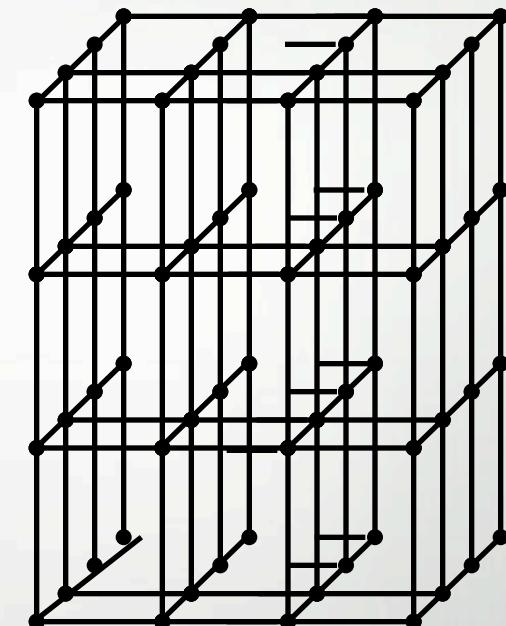
## Denovo

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



## S3D – A DNS solver

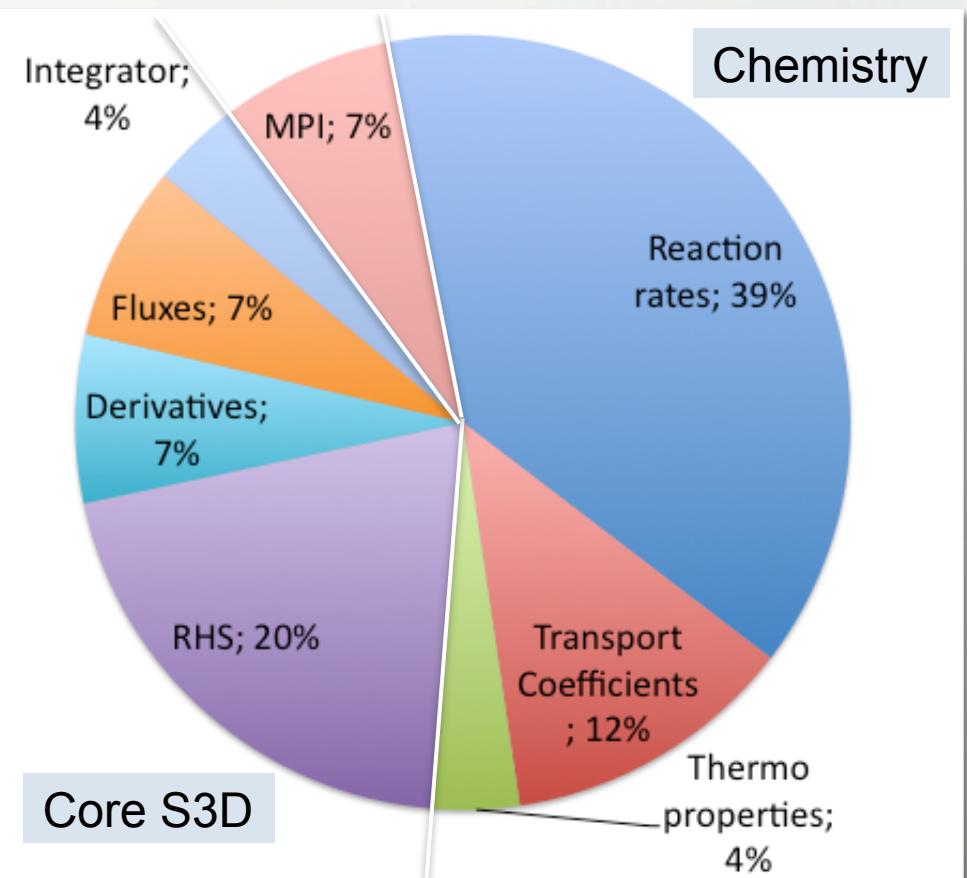
- Structured Cartesian mesh flow solver
- Solves compressible reacting Navier-Stokes, energy and species conservation equations.
  - 8<sup>th</sup> order explicit finite difference method
  - 4<sup>th</sup> order Runge-Kutta integrator with error estimator
- Detailed gas-phase thermodynamic, chemistry 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](mailto: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.	Imb.	Calls	Group
		Time	Time%		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

		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

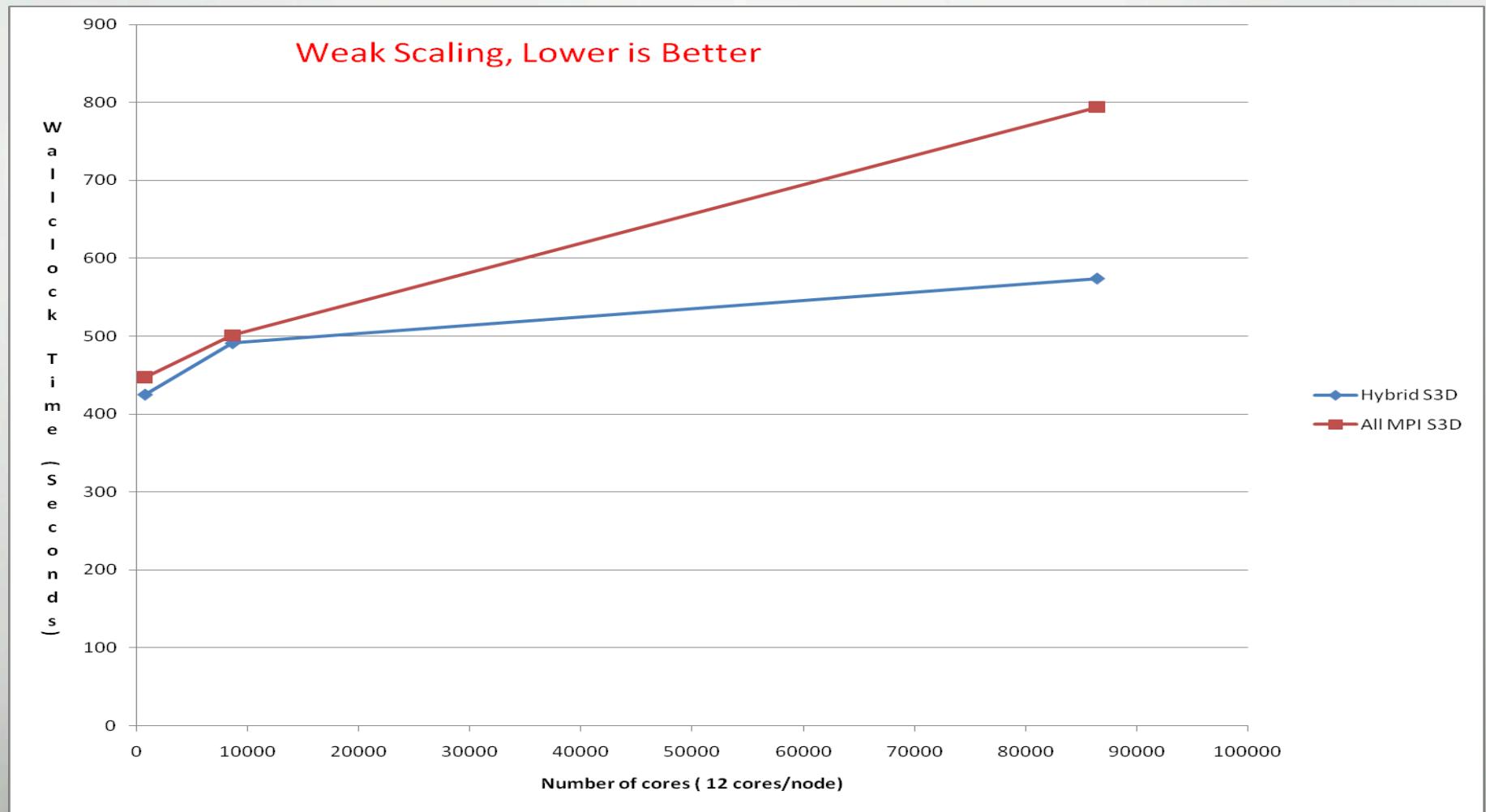
Table 1: Profile by Function Group and Function

Time%	Time	Imb.	Imb.	Calls	Group
		Time	Time%		Function
85.3%	539.077983	--	--	144908.0	USER
<hr/>					
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

## Resultant Hybrid S3D Performance



		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

## Sample from S3D – Hybrid-Multi-core

### Part 1

```

 !$acc data copyin(q,volum) shared(yspecies,u,avmolwt,mixMW,temp)
 !$acc parallel_loop private(i,ml,mu)
 do i = 1, nx*ny*nz, ms
   ml = i
   mu = min(i+ms-1, nx*ny*nz)
   call get_mass_frac_r( q, volum, yspecies, ml, mu)
   call get_velocity_vec_r( u, q, volum, ml, mu)
   call calc_inv_avg_mol_wt_r( yspecies, avmolwt, mixMW, ml, mu)
   volttmp(ml:mu,1,1)=q(ml:mu,1,1,5)*volum(ml:mu,1,1)
   call calc_temp_r(temp, volttmp, u, yspecies, cpmix, avmolwt, ml, mu)
 end do
 !$acc end parallel_loop
 ! Start communication - the _prep routines do posts and sends
 ! using buffer identified by itmp
 itmp = 1
 !$acc acc_update host(u,temp,yspecies)
 call computeVectorGradient_prep( u, itmp )
 call computeScalarGradient_prep( temp, itmp )
 do n=1,n_spec
   call computeScalarGradient_prep( yspecies(:,:,:,:n), itmp )
 enddo
 ! Compute remaining properties whilst communication is underway
 !$acc parallel_loop private(i,ml,mu)
 do i = 1, nx*ny*nz, ms
   ml = i
   mu = min(i+ms-1, nx*ny*nz)
   call calc_gamma_r( gamma, cpmix, avmolwt, ml, mu)
   call calc_press_r( pressure, q(:,:,:,:4), temp, avmolwt, ml, mu )
   call calc_specEnth_allpts_r(temp, h_spec, ml, mu)
 end do
 !$acc end parallel_loop

```

```
! Now wait for communication
call derivative_xyz_wait( itmp )
calc_buff_internal_wait = .false.
itmp = 1
!$acc update device(u,temp,yspecies)
call computeVectorGradient_calc( u, grad_u, itmp )
call computeScalarGradient_calc( temp, grad_T, itmp )
!$acc parallel_loop private(n,itmp)
do n=1,n_spec
    itmp = n + 4
    call computeScalarGradient5d_calc( yspecies(1,1,1,n), &
        grad_Ys(1,1,1,1,1), n_spec, n, itmp,sscale_1x,sscale_1y,sscale_1z )
enddo
!$acc end parallel_loop
!$acc end data
```

## Current Status of putting S3D on XK6

- Internal XK6 with 160 nodes of
  - Interlagos
  - Fermi +
- Hybrid S3D running across entire system without accelerators
- Hybrid S3D with OpenACC using the accelerator running across the entire system
  - Overlapping computation on device with computation on host with communication (Using core specialization)
- Current work
  - Preparing for async option on update to overlap data movement
  - Cleaning up code
    - Removing work-around for compiler bugs

# 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 |      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.

## Inliner diagnostics ( -rmp )

^

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.

## Inliner diagnostics ( -rmp )

^

ftn-3021 ftn: ERROR RHSF, File = rhsf.f90, Line = 336, Column = 12

Routine "mpi\_finalize", referenced in "terminate\_run", was not inlined because the compiler was unable to locate the routine to expand it inline.

^

ftn-3021 ftn: ERROR RHSF, File = rhsf.f90, Line = 336, Column = 12

Routine "mpi\_barrier", referenced in "terminate\_run", was not inlined because the compiler was unable to locate the routine to expand it inline.

^

ftn-3021 ftn: ERROR RHSF, File = rhsf.f90, Line = 336, Column = 12

Routine "mpi\_wait", referenced in "derivative\_y\_calc\_buff\_r", was not inlined because the compiler was unable to locate the routine to expand it inline.

^

ftn-3021 ftn: ERROR RHSF, File = rhsf.f90, Line = 336, Column = 12

Routine "mpi\_wait", referenced in "derivative\_y\_calc\_buff\_r", was not inlined because the compiler was unable to locate the routine to expand it inline.

## 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

# Thank you. Questions?

