Refactoring Applications for Future Hybrid Architectures

John M Levesque (Cray Inc)
Director Cray’s Supercomputing Center of Excellence
CTO Office - Applications
Are you ready for the future?

● You must move to a hybrid (MPI, threading & vector) architecture to prepare for the future
● You must start considering how to manage your arrays to have them close to the computational engine when you need them there
  ● We are moving to a more complex memory hierarchy that will require user intervention to achieve good performance.
● You must design your application to be performance portable across a wide range of systems
  ● Fortunately they will be similar enough that this will be possible
● Bottom Line – you must understand your application extremely well to achieve acceptable performance on today’s and tomorrow’s architectures
● OpenACC and OpenMP is part of a transition process to take an all-MPI code into the next generation of computers
A common directive programming model for today’s GPUs

- Announced at SC11 conference
- Offers portability between compilers
  - Drawn up by: NVIDIA, Cray, PGI, CAPS
  - Multiple compilers offer portability, debugging, permanence
- Works for Fortran, C, C++
  - 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
Strategic risk factors

Will there be machines to run my OpenACC code on?
- **Now?** Lots of Nvidia GPU accelerated systems
  - Cray XK7s: CSCS tödi, HLRS hermit, ORNL titan...
  - Lots of other GPU machines in Top100 (OpenACC is multi-vendor)
- **Future?** OpenACC can be targetted at other accelerators
  - PGI and CAPS already target Intel Xeon Phi, AMD GPUs
  - Plus you can always run on CPUs using same codebase

Will OpenACC continue?
- **Support?** Cray and PGI (at least) are committed to support OpenACC
  - Lots of big customer pressure to continue to run OpenACC
- **Develop?** OpenACC committee has finalized v2.0 of standard
  - Lots of new partners joined committee at end of last year

Will OpenACC be superseded by something else?
- **Auto-accelerating compilers?** If only!
  - Never really managed it for threading real HPC applications on the CPU
  - Data locality adds to the challenge
- **OpenMP accelerator directives?** OpenACC work not wasted
  - Very similar programming model; can transition when these release if wish
  - Cray (co-chair), PGI very active in OpenMP accelerator subcommittee
Programming for Hybrid Architectures
Nodes are becoming much more parallel
  • More processors/node
  • More threads/processor
  • Vector lengths are getting longer
  • Memory hierarchy is becoming more complex
  • Scalar performance is not increasing and will start decreasing
ALL accelerated nodes require vectorization at a good size to achieve reasonable performance
- Nvidia Kepler 32 length
- Intel MIC >8

Most of today’s compilers are designed for marginal vector performance on current Xeon systems, they need to understand future tradeoffs

User rewriting of loops is required for gaining good performance on future systems
Memory Hierarchy is becoming more complex

- As processors get faster, memory bandwidth cannot keep up
  - More complex caches
  - Non Uniform Memory Architecture (NUMA) for shared memory on node
  - Operand alignment is becoming more important
- Going forward this will become even more complex – two memories within same address space
  - Fast expensive memory
  - Slow less expensive memory
## Advantages & Disadvantages of the Node Architectures

<table>
<thead>
<tr>
<th></th>
<th>“hosted”</th>
<th>“self hosted”</th>
<th>Traditional Multi-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar Performance</td>
<td>Uses State of art Scalar Processor</td>
<td>Uses much slower scalar processor</td>
<td>Uses State of art Scalar Processor</td>
</tr>
<tr>
<td>Parallel Performance</td>
<td>Uses State of art Parallel Processor</td>
<td>Uses State of art Parallel Processor</td>
<td>Uses multi-core for performance</td>
</tr>
<tr>
<td>Data movement today</td>
<td>Significant Data Motion between Host/Accelerator</td>
<td>Minimal Data Motion</td>
<td>Minimal Data Motion</td>
</tr>
<tr>
<td>Data movement tomorrow</td>
<td>Significant Data Motion between Host/Accelerator</td>
<td>Significant Data Motion between Memory Levels</td>
<td>Minimal Data Motion</td>
</tr>
<tr>
<td>Amount of Memory</td>
<td>Limited on Accelerator</td>
<td>Limited fast Memory</td>
<td>Sufficient</td>
</tr>
</tbody>
</table>
Things we need to know about the application

- Where are the major arrays allocated and how are they accessed
  WHY – We need to understand how arrays can be allocated to assure most efficient access by major computational loops. (First touch, alignment, etc)

- Where are the major computational and communication regions
  WHY – We want to maintain a balance between computation and communication. How much time is spent in a computational region, what if any communication can be performed during that time

- Where is the major I/O performed
  WHY – Can we perform I/O asynchronously with computation/communication
**Goals**

- Develop a single source code that implements OpenMP and OpenACC in such a way that application can be efficiently run on:
  - Multi-core MPP systems
  - Multi-core MPP systems with companion accelerator
    - Nvidia
    - Intel
    - AMD
    - Whatever

- Clearly identify three levels of parallelism
  - MPI/PGAS between NUMA/UMA memory regions
  - Threading within the NUMA/UMA memory region
    - How this is implemented is important – OpenMP/OpenACC is most portable
  - SIMDization at a low level
    - How this is coded is important – compilers have different capability

- **We do want a performance/portable application at the end**
Working through a simple example

● Himeno
  ● This benchmark program takes measurements to proceed major loops in solving the Poisson's equation solution using the Jacobi iteration method.
  ● Single computational loop
  ● Easy to introduce OpenMP
  ● Good to see how to implement OpenACC
Step 1

Profile the Application
# Standard Profile

Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
</tr>
<tr>
<td>100.0%</td>
<td>66.812060</td>
<td>--</td>
<td>--</td>
<td>2372.1</td>
<td>Total</td>
</tr>
<tr>
<td>100.0%</td>
<td>66.811928</td>
<td>--</td>
<td>--</td>
<td>2167.0</td>
<td>USER</td>
</tr>
<tr>
<td>94.2%</td>
<td>62.965959</td>
<td>2.805426</td>
<td>4.3%</td>
<td>2.0</td>
<td>jacobi_.LOOPS</td>
</tr>
<tr>
<td>2.1%</td>
<td>1.385449</td>
<td>3.224803</td>
<td>71.1%</td>
<td>539.0</td>
<td>sendp1_</td>
</tr>
<tr>
<td>1.7%</td>
<td>1.167354</td>
<td>3.33783</td>
<td>75.2%</td>
<td>539.0</td>
<td>sendp2_</td>
</tr>
<tr>
<td>0.7%</td>
<td>0.461089</td>
<td>0.847765</td>
<td>65.8%</td>
<td>539.0</td>
<td>sendp3_</td>
</tr>
<tr>
<td>0.7%</td>
<td>0.455975</td>
<td>0.009942</td>
<td>2.2%</td>
<td>1.0</td>
<td>initmt_.LOOPS</td>
</tr>
<tr>
<td>0.4%</td>
<td>0.295538</td>
<td>0.302255</td>
<td>51.4%</td>
<td>1.0</td>
<td>initcomm_</td>
</tr>
<tr>
<td>0.1%</td>
<td>0.075184</td>
<td>0.015566</td>
<td>17.4%</td>
<td>1.0</td>
<td>himenobtxxp_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.005124</td>
<td>0.003026</td>
<td>37.7%</td>
<td>539.0</td>
<td>sendp_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000094</td>
<td>0.000009</td>
<td>8.7%</td>
<td>1.0</td>
<td>initmax_.LOOPS</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000090</td>
<td>0.000010</td>
<td>9.7%</td>
<td>1.0</td>
<td>initmax_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000042</td>
<td>0.000012</td>
<td>22.9%</td>
<td>1.0</td>
<td>exit</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000021</td>
<td>0.000005</td>
<td>20.2%</td>
<td>2.0</td>
<td>jacobi_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000009</td>
<td>0.000005</td>
<td>37.5%</td>
<td>1.0</td>
<td>initmt_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000088</td>
<td>--</td>
<td>--</td>
<td>2.0</td>
<td>MPI</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000087</td>
<td>0.000012</td>
<td>12.0%</td>
<td>1.0</td>
<td>mpi_finalize</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000001</td>
<td>0.000000</td>
<td>22.0%</td>
<td>1.0</td>
<td>mpi_init_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000044</td>
<td>--</td>
<td>--</td>
<td>203.1</td>
<td>ETC</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000043</td>
<td>0.000086</td>
<td>67.9%</td>
<td>203.1</td>
<td>__libc_csu_init</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000001</td>
<td>0.000068</td>
<td>100.0%</td>
<td>0.0</td>
<td>__STOP2</td>
</tr>
</tbody>
</table>
Table 2: Loop Stats by Function (from -hprofile_generate)

<table>
<thead>
<tr>
<th>Loop</th>
<th>Loop</th>
<th>Loop</th>
<th>Loop</th>
<th>Loop</th>
<th>Function=/&gt;.LOOP[.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incl</td>
<td>Hit</td>
<td>Trips</td>
<td>Trips</td>
<td>Trips</td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>Time</td>
<td>Avg</td>
<td>Min</td>
<td>Max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65.985609</td>
<td>2</td>
<td>269.500</td>
<td>0</td>
<td>536</td>
<td>jacobi_.LOOP.1.li.206</td>
</tr>
<tr>
<td>52.004289</td>
<td>539</td>
<td>127.500</td>
<td>0</td>
<td>128</td>
<td>jacobi_.LOOP.2.li.209</td>
</tr>
<tr>
<td>51.999616</td>
<td>68722</td>
<td>127.500</td>
<td>0</td>
<td>128</td>
<td>jacobi_.LOOP.3.li.210</td>
</tr>
<tr>
<td>51.523908</td>
<td>8762119</td>
<td>255.500</td>
<td>0</td>
<td>256</td>
<td>jacobi_.LOOP.4.li.211</td>
</tr>
<tr>
<td>8.474166</td>
<td>539</td>
<td>127.500</td>
<td>0</td>
<td>128</td>
<td>jacobi_.LOOP.5.li.229</td>
</tr>
<tr>
<td>8.470748</td>
<td>68722</td>
<td>127.500</td>
<td>0</td>
<td>128</td>
<td>jacobi_.LOOP.6.li.230</td>
</tr>
<tr>
<td>7.510744</td>
<td>8762119</td>
<td>255.500</td>
<td>0</td>
<td>256</td>
<td>jacobi_.LOOP.7.li.231</td>
</tr>
<tr>
<td>0.302805</td>
<td>1</td>
<td>131</td>
<td>0</td>
<td>131</td>
<td>initmt_.LOOP.1.li.152</td>
</tr>
<tr>
<td>0.302796</td>
<td>131</td>
<td>131</td>
<td>0</td>
<td>131</td>
<td>initmt_.LOOP.2.li.153</td>
</tr>
<tr>
<td>0.302007</td>
<td>17161</td>
<td>259</td>
<td>0</td>
<td>259</td>
<td>initmt_.LOOP.3.li.154</td>
</tr>
<tr>
<td>0.153160</td>
<td>1</td>
<td>129.500</td>
<td>0</td>
<td>130</td>
<td>initmt_.LOOP.4.li.173</td>
</tr>
<tr>
<td>0.153145</td>
<td>129.500</td>
<td>129.500</td>
<td>0</td>
<td>130</td>
<td>initmt_.LOOP.5.li.174</td>
</tr>
<tr>
<td>0.152091</td>
<td>16770</td>
<td>257.500</td>
<td>0</td>
<td>258</td>
<td>initmt_.LOOP.6.li.175</td>
</tr>
<tr>
<td>0.000000</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>initmax_.LOOP.1.li.350</td>
</tr>
<tr>
<td>0.000000</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>initmax_.LOOP.3.li.365</td>
</tr>
<tr>
<td>0.000000</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>initmax_.LOOP.5.li.380</td>
</tr>
<tr>
<td>0.000000</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>initmax_.LOOP.6.li.387</td>
</tr>
<tr>
<td>0.000000</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>initmax_.LOOP.4.li.372</td>
</tr>
<tr>
<td>0.000000</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>initmax_.LOOP.2.li.357</td>
</tr>
</tbody>
</table>
Step 2

Use Reveal to scope the major computational loops
OpenMP directives on loop

! Directive inserted by Cray Reveal. May be incomplete.
!$OMP parallel do default(none)
!$OMP& private (i,j,k,s0,ss)
!$OMP& shared (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
!$OMP& reduction (+:wgosa)
  DO K=2,kmax-1
    DO J=2,jmax-1
      DO I=2,imax-1
        S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I+1,J,K)+a(I,J,K,3)*p(I,J,K+1)
        +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)) +p(I-1,J+1,K)-p(I-1,J-1,K))
        +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)) +p(I,J+1,K-1)+p(I,J-1,K-1))
        +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)) +p(I+1,J,K-1)+p(I-1,J,K-1))
        +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
        +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
        SS=(S0*a(I,J,K,4)-p(I,J,K)) *bnd(I,J,K)
        WGOSA=WGOSA+SS*SS
        wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
      enddo
    enddo
  enddo
OpenMP directives on loop – compiler listing

213. 1 ! Directive inserted by Cray Reveal. May be incomplete.
214. +1 !$OMP parallel do default(none)
215. 1 !$OMP& private (i,j,k,s0,ss)
216. 1 !$OMP& shared (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
217. 1 !$OMP& reduction (+:wgosa)
218. +1 DO K=2,kmax-1
219. +1 DO J=2,jmax-1
220. 1 DO I=2,imax-1
221. 1 m 3 Vr3 S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
222. 1 m 3 Vr3 1 +a(I,J,K,3)*p(I,J,K+1)
223. 1 m 3 Vr3 2 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K))
224. 1 m 3 Vr3 3 -p(I-1,J+1,K)+p(I-1,J-1,K))
225. 1 m 3 Vr3 4 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1))
226. 1 m 3 Vr3 5 -p(I,J+1,K-1)+p(I,J-1,K-1))
227. 1 m 3 Vr3 6 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1))
228. 1 m 3 Vr3 7 -p(I+1,J,K-1)+p(I-1,J,K-1))
229. 1 m 3 Vr3 8 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
230. 1 m 3 Vr3 9 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
231. 1 m 3 Vr3 SS=(S0*a(I,J,K,4)*p(I,J,K)) *bnd(I,J,K)
232. 1 m 3 Vr3 WGOSA=WGOSA+SS*SS
233. 1 m 3 Vr3 wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
234. 1 m 3 Vr3 enddo
235. 1 m 3 enddo
236. 1 m enddo
Step 3

Introduce OpenACC
This should only be done once a very good OpenMP/MPI application is in hand
Introducing OpenACC

```c
#ifdef _OPENACC
!$ACC parallel loop
!$ACC&  private (i,j,k,s0,ss)
!$ACC&  reduction (+:wgosa)
#else
!$OMP parallel do default(none)
!$OMP&  private (i,j,k,s0,ss)
!$OMP&  shared (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
!$OMP&  reduction (+:wgosa)
#endif
DO K=2,kmax-1
  DO J=2,jmax-1
    DO I=2,imax-1
      S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)+
        a(I,J,K,3)*p(I,J,K+1)+b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)-p(I-1,J+1,K)+p(I-1,J-1,K))+b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)-p(I,J+1,K-1)+p(I,J-1,K-1))+b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)-p(I+1,J,K-1)+p(I-1,J,K-1))+c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)+
        c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
      SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
      WGOSA=WGOSA+SS*SS
      wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
    enddo
  enddo
enddo
#ifdef _OPENACC
!$ACC end parallel loop
#endif
```
Introducing OpenACC – compiler listing

```c
1 #ifdef _OPENACC
2  + 1 G-----------------< !$ACC parallel loop
3  1 G                !$ACC& private (i,j,k,s0,ss)
4  1 G                !$ACC& reduction (+:wgosa)
5  1 G              #else
6  1 G                !$OMP parallel do default(none)
7  1 G                !$OMP& private (i,j,k,s0,ss)
8  1 G                !$OMP& shared (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
9  1 G                !$OMP& reduction (+:wgosa)
10  1 G            #endif
11  1 G g-------------< DO K=2,kmax-1
12  + 1 G g 4--------< DO J=2,jmax-1
13  1 G g 4 g--------< DO I=2,imax-1
14  1 G g 4 g g------< S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)+a(I,J,K,3)*p(I,J+1,K)
15  1 G g 4 g 1       +a(I,J,K,3)*p(I,J,K+1)
16  1 G g 4 g 2       +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K))
17  1 G g 4 g 3       -p(I-1,J+1,K)+p(I-1,J-1,K))
18  1 G g 4 g 4       +b(I,J,K,2)*(p(I,J+1,K-1)+p(I,J-1,K+1))
19  1 G g 4 g 5       -p(I,J+1,K-1)+p(I,J-1,K-1))
20  1 G g 4 g 6       +b(I,J,K,3)*(p(I+1,J+1,K)-p(I+1,J-1,K))
21  1 G g 4 g 7       -p(I+1,J,K-1)+p(I-1,J,K+1))
22  1 G g 4 g 8       +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
23  1 G g 4 g 9       +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
24  1 G g 4 g SS=(S0*a(I,J,K,4)*p(I,J,K)) bnd(I,J,K)
25  1 G g 4 g WGOSA=WGOSA+SS*SS
26  1 G g 4 g wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
27  1 G g 4 g-----> enddo
28  1 G g 4 g---------> enddo
29  1 G g 4 g---------> enddo
30  1 G g 4 g---------> enddo
31  1 G              #ifdef _OPENACC
32  + 1 G-----------------< !$ACC end parallel loop
33  1 G              #endif
```
Introducing OpenACC – compiler listing

ftn-6405 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  A region starting at line 215 and ending at line 244 was placed on the accelerator.

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "p" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "a" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "b" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "c" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "wrk1" to accelerator, free at line 244 (acc_copyin).

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "bnd" to accelerator, free at line 244 (acc_copyin).

ftn-6416 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  If not already present: allocate memory and copy whole array "wrk2" to accelerator, copy back at line 244 (acc_copy).

ftn-6415 ftn: ACCEL File = himenoBMTxpr.f, Line = 215
  Allocate memory and copy variable "wgosa" to accelerator, copy back at line 244 (acc_copy).

ftn-6430 ftn: ACCEL File = himenoBMTxpr.f, Line = 224
  A loop starting at line 224 was partitioned across the thread blocks.
Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>62.073674</td>
<td>--</td>
<td>--</td>
<td>7627.1</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>95.6%</td>
<td>59.368154</td>
<td>--</td>
<td>--</td>
<td>2565.0</td>
<td>USER</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
<td></td>
</tr>
<tr>
<td>81.0%</td>
<td>50.284595</td>
<td>1.058725</td>
<td>2.1%</td>
<td>284.0</td>
<td>jacobi_.ACC_COPY@li.215</td>
<td></td>
</tr>
<tr>
<td>3.8%</td>
<td>2.359195</td>
<td>0.094854</td>
<td>3.9%</td>
<td>284.0</td>
<td>jacobi_.ACC_COPY@li.244</td>
<td></td>
</tr>
<tr>
<td>3.3%</td>
<td>2.055168</td>
<td>0.131544</td>
<td>6.1%</td>
<td>2.0</td>
<td>jacobi_</td>
<td></td>
</tr>
<tr>
<td>2.6%</td>
<td>1.616893</td>
<td>0.457704</td>
<td>22.4%</td>
<td>284.0</td>
<td>jacobi_.ACC_SYNC_WAIT@li.244</td>
<td></td>
</tr>
<tr>
<td>2.5%</td>
<td>1.535406</td>
<td>0.280701</td>
<td>15.7%</td>
<td>1.0</td>
<td>exit</td>
<td></td>
</tr>
<tr>
<td>2.3%</td>
<td>1.410941</td>
<td>0.284518</td>
<td>17.0%</td>
<td>1.0</td>
<td>initmt_.LOOP@li.156</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.054959</td>
<td>0.001335</td>
<td>2.4%</td>
<td>1.0</td>
<td>initmt_</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.039723</td>
<td>0.005092</td>
<td>11.5%</td>
<td>284.0</td>
<td>jacobi_.ACC_ASYNC_KERNEL@li.215</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.003376</td>
<td>0.003567</td>
<td>52.2%</td>
<td>284.0</td>
<td>jacobi_.ACC_REGION@li.215</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.002758</td>
<td>0.000392</td>
<td>12.7%</td>
<td>284.0</td>
<td>sendp3_</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.002172</td>
<td>0.000380</td>
<td>15.1%</td>
<td>284.0</td>
<td>sendp_</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.001459</td>
<td>0.000189</td>
<td>11.7%</td>
<td>284.0</td>
<td>sendp1_</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.001155</td>
<td>0.000109</td>
<td>8.8%</td>
<td>284.0</td>
<td>sendp2_</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000276</td>
<td>0.000590</td>
<td>69.2%</td>
<td>1.0</td>
<td>himenobmtxp_</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000047</td>
<td>0.000007</td>
<td>13.3%</td>
<td>1.0</td>
<td>initcomm_</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000025</td>
<td>0.000005</td>
<td>15.9%</td>
<td>1.0</td>
<td>initmax_</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000006</td>
<td>0.000018</td>
<td>74.8%</td>
<td>1.0</td>
<td>initmt_.REGION@li.156</td>
<td></td>
</tr>
</tbody>
</table>
Steps 4, 5, 6, 7, 8

Once one loop is analyzed, now look at next highest compute loop, perform steps 2 and 3
Moving to Version 2

● **Introduce !$acc data region in main routine**
  ● Move data to the device and initialize data on the device
  ● Now must move data back to the host for the message passing
OpenACC Version 2 – Using Data regions – a)

```c
#ifdef __OPENACC
!$acc data present(a,p,b,c,bnd,wrk1)present_or_create(wgosa)
#endif

DO loop=1,nn
  gosa=0.0
  wgosa=0.0
!
! Directive inserted by Cray Reveal. May be incomplete.
#ifdef __OPENACC
!$ACC parallel loop
!$ACC&   private (i,j,k,s0,ss)
!$ACC&   reduction (+:wgosa)
#else
!$OMP parallel do default(none)
!$OMP&   private (i,j,k,s0,ss)
!$OMP&   shared (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
!$OMP&   reduction (+:wgosa)
#endif
DO K=2,kmax-1
  DO J=2,jmax-1
    DO I=2,imax-1
      S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
      +a(I,J,K,3)*p(I,J,K+1)
      +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
      -p(I-1,J+1,K)+p(I-1,J-1,K))
      +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
      -p(I,J+1,K-1)+p(I,J-1,K-1))
      +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
      -p(I+1,J,K-1)+p(I-1,J,K-1))
      +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
      +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
      SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
      WGOSA=WGOSA+SS*SS
      wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
    enddo
  enddo
enddo
#ifdef __OPENACC
!$ACC end parallel loop
#endif
```
OpenACC Version 2 – Using Data regions –b)

C
#ifdef __OPENACC
!$ACC parallel loop
!$ACC private (i,j,k)
#else
!$OMP parallel do default(none)
!$OMP private (i,j,k)
!$OMP shared (p,wrk2)
#endif
DO K=2,kmax-1
  DO J=2,jmax-1
    DO I=2,imax-1
      p(I,J,K)=wrk2(I,J,K)
    enddo
  enddo
enddo
#ifdef __OPENACC
!$ACC end parallel loop
!$acc update host(p,wgosa)
#endif
C
call sendp(ndx,ndy,ndz)
C
#ifdef __OPENACC
!$acc update device(p)
#endif
call mpi_allreduce(wgosa,
  >       gosa,
  >       1,
  >       mpi_real4,
  >       mpi_sum,
  >       mpi_comm_world,
  >       ierr)
C
  enddo
#endif
#ifdef __OPENACC
!$acc end data
#endif
OpenACC Version 2 – Using Data regions –a)

240. + G----------< !$acc data present(a,p,b,c,bnd,wrk1) present_or_create(wgosa)
241.     G       #endif
242.     G       C
243. + G 2------< DO loop=1,nn
gosa=0.0
wgosa=0.0
247. G 2      ifdef _OPENACC
248. + G 2 G----< !$ACC parallel loop
249. G 2 G     !$ACC& private (i,j,k,s0,ss)
250. G 2 G     !$ACC& reduction (+:wgosa)
251. G 2 G     #else
252. G 2 G     !$OMP parallel do default(none)
253. G 2 G     !$OMP& private (i,j,k,s0,ss)
254. G 2 G     !$OMP& shared (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
255. G 2 G     !$OMP& reduction (+:wgosa)
256. G 2 G     #endif
257. G 2 G g----< DO K=2,kmax-1
258. + G 2 G g5----< DO J=2,jmax-1
259. G 2 G g5 g--< DO I=2,imax-1
260. G 2 G g5 g  80=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
261. G 2 G g5 g  1 +a(I,J,K,3)*p(I,J,K+1)
262. G 2 G g5 g  2 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K))
263. G 2 G g5 g  3 -p(I-1,J+1,K)+p(I-1,J-1,K))
264. G 2 G g5 g  4 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1))
265. G 2 G g5 g  5 -p(I+1,J,K-1)+p(I+1,J,K-1))
266. G 2 G g5 g  6 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1))
267. G 2 G g5 g  7 -p(I+1,J,K-1)+p(I-1,J,K-1))
268. G 2 G g5 g  8 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
269. G 2 G g5 g  9 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
270. G 2 G g5 g  SS=S0*a(I,J,K,4)-p(I,J,K)*bnd(I,J,K)
271. G 2 G g5 g  WGOSA=WGOSA+SS*SS
272. G 2 G g5 g  wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
273. G 2 G g5 g-- enddo
274. G 2 G g5---- enddo
275. G 2 G g---- enddo
276. G 2 G     ifdef _OPENACC
277. G 2 G------- !$ACC end parallel loop.
OpenACC Version 2 – Using Data regions –b)

```c
279.    G 2              C
280.    G 2              ifdef _OPENACC
281.    + G 2 G--------< !$ACC parallel loop
282.    G 2 G            !$ACC&   private (i,j,k)
283.    G 2 G            #else
284.    G 2 G            !$OMP parallel do default(none)
285.    G 2 G            !$OMP&   private (i,j,k)
286.    G 2 G            !$OMP&   shared (p,wrk2)
287.    G 2 G            #endif
288.    G 2 G g------<      DO K=2,kmax-1
289.    G 2 G             DO J=2,jmax-1
290.    G 2 G             DO I=2,imax-1
291.    G 2 G             p(I,J,K)=wrk2(I,J,K)
292.    G 2 G             enddo
293.    G 2 G             enddo
294.    G 2 G g-------->   enddo
295.    G 2 G              ifdef _OPENACC
296.    G 2 G-------->   !$ACC end parallel loop
297.    G 2              !$acc update host(p,wgosa)
298.    G 2              #endif
299.    G 2              C
300.    + G 2              call sendp(ndx,ndy,ndz)
301.    G 2              C
302.    G 2              ifdef _OPENACC
303.    G 2              !$acc update device(p)
304.    G 2              #endif
305.    + G 2              call mpi_allreduce(wgosa,
306.    G 2                      gosa,
307.    G 2                      1,
308.    G 2                      mpi_real4,
309.    G 2                      mpi_sum,
310.    G 2                      mpi_comm_world,
311.    G 2                      ierr)
312.    G 2              C
313.    G 2-------------->   enddo
314.    G              ifdef _OPENACC
315.    G-------------->   !$acc end data.
```

8/7/2013
ATPESC - OpenACC Workshop
OpenACC Version 2 – Using Data regions –c)

ftn-6413 ftn: ACCEL File = himenoBMTxpr.f, Line = 240
   A data region was created at line 240 and ending at line 315.

ftn-6422 ftn: ACCEL File = himenoBMTxpr.f, Line = 240
   If not already present: allocate memory for variable "wgosa" on accelerator, free at line 315 (acc_share).

ftn-6288 ftn: VECTOR File = himenoBMTxpr.f, Line = 243
   A loop starting at line 243 was not vectorized because it contains a call to subroutine "sendp" on line 300.

ftn-6405 ftn: ACCEL File = himenoBMTxpr.f, Line = 248
   A region starting at line 248 and ending at line 277 was placed on the accelerator.

ftn-6416 ftn: ACCEL File = himenoBMTxpr.f, Line = 248
   If not already present: allocate memory and copy whole array "wrk2" to accelerator, copy back at line 277 (acc_copy).

ftn-6430 ftn: ACCEL File = himenoBMTxpr.f, Line = 257
   A loop starting at line 257 was partitioned across the thread blocks.

ftn-6509 ftn: ACCEL File = himenoBMTxpr.f, Line = 258
   A loop starting at line 258 was not partitioned because a better candidate was found at line 259.

ftn-6412 ftn: ACCEL File = himenoBMTxpr.f, Line = 258
   A loop starting at line 258 will be redundantly executed.

ftn-6430 ftn: ACCEL File = himenoBMTxpr.f, Line = 259
   A loop starting at line 259 was partitioned across the 128 threads within a threadblock.

ftn-6405 ftn: ACCEL File = himenoBMTxpr.f, Line = 281
   A region starting at line 281 and ending at line 296 was placed on the accelerator.

ftn-6418 ftn: ACCEL File = himenoBMTxpr.f, Line = 281
   If not already present: allocate memory and copy whole array "wrk2" to accelerator, free at line 296 (acc_copyin).

ftn-6430 ftn: ACCEL File = himenoBMTxpr.f, Line = 288
   A loop starting at line 288 was partitioned across the thread blocks.
### Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>35.679931</td>
<td>--</td>
<td>--</td>
<td>36436.1</td>
<td>Total</td>
</tr>
<tr>
<td>92.2%</td>
<td>32.914300</td>
<td>--</td>
<td>--</td>
<td>19121.0</td>
<td>USER</td>
</tr>
<tr>
<td>34.9%</td>
<td>12.445154</td>
<td>0.259110</td>
<td>2.1%</td>
<td>1005.0</td>
<td>jacobi_.ACC_COPY@li.303</td>
</tr>
<tr>
<td>31.5%</td>
<td>11.256881</td>
<td>0.247093</td>
<td>2.2%</td>
<td>1005.0</td>
<td>jacobi_.ACC_COPY@li.297</td>
</tr>
<tr>
<td>17.1%</td>
<td>6.089251</td>
<td>0.845903</td>
<td>12.4%</td>
<td>1005.0</td>
<td>jacobi_.ACC_SYNC_WAIT@li.277</td>
</tr>
<tr>
<td>8.2%</td>
<td>2.916580</td>
<td>1.242691</td>
<td>30.4%</td>
<td>1005.0</td>
<td>jacobi_.ACC_SYNC_WAIT@li.296</td>
</tr>
<tr>
<td>0.2%</td>
<td>0.076624</td>
<td>0.010203</td>
<td>11.9%</td>
<td>1005.0</td>
<td>jacobi_.ACC_ASYNC_KERNEL@li.248</td>
</tr>
<tr>
<td>0.1%</td>
<td>0.049299</td>
<td>0.006400</td>
<td>11.7%</td>
<td>1005.0</td>
<td>jacobi_.ACC_ASYNC_KERNEL@li.281</td>
</tr>
<tr>
<td>0.1%</td>
<td>0.019482</td>
<td>0.000622</td>
<td>31.3%</td>
<td>1005.0</td>
<td>jacobi_.ACC_COPY@li.248</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.009641</td>
<td>0.001109</td>
<td>10.5%</td>
<td>2.0</td>
<td>jacobi_.ACC_DATA_REGION@li.240</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.007366</td>
<td>0.000622</td>
<td>7.9%</td>
<td>1005.0</td>
<td>sendp3_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.005379</td>
<td>0.000703</td>
<td>11.7%</td>
<td>1005.0</td>
<td>jacobi_.ACC_REGION@li.248</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.004826</td>
<td>0.000634</td>
<td>11.8%</td>
<td>1005.0</td>
<td>sendp1_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.004560</td>
<td>0.000721</td>
<td>13.9%</td>
<td>1005.0</td>
<td>jacobi_.ACC_UPDATE@li.303</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.004302</td>
<td>0.000850</td>
<td>16.8%</td>
<td>1005.0</td>
<td>jacobi_.ACC_SYNC_WAIT@li.297</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.004042</td>
<td>0.000214</td>
<td>5.1%</td>
<td>1005.0</td>
<td>sendp_</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.003855</td>
<td>0.00379</td>
<td>9.1%</td>
<td>1005.0</td>
<td>sendp2_</td>
</tr>
</tbody>
</table>
Steps 9, 10, 11, 12

Optimizing data transfers
Moving to Version 3

- Pack MPI buffers on the accelerator
  - Difficult here because MPI data types are used
  - Significantly reduce the data moved back and forth
C
#define _OPENACC
!$ACC parallel loop
!$ACC& private (i,j,k)
#else
!$OMP parallel do default(none)
!$OMP& private (i,j,k)
!$OMP& shared (p,wrk2)
#endif
DO K=2,kmax-1
  DO J=2,jmax-1
    DO I=2,imax-1
      p(I,J,K)=wrk2(I,J,K)
    enddo
  enddo
enddo
#ifdef _OPENACC
!$ACC parallel loop
!$ACC& private (i,j,k)
#else
!$OMP parallel do
!$OMP& private (i,j,k)
!$OMP& shared (p,wrk2)
#endif
DO K=1,kmax
  DO J=1,jmax
    ii = j + (k-1)*imax
    pack_p11(ii)=p(i,2,k)
    pack_p12(ii)=p(i,jmax-1,k)
  enddo
enddo
#endif
C
#endif _OPENACC
!$ACC update host(pack_p11,pack_p12,pack_p21,pack_p22,
!$ACC&               pack_p31,pack_p32,wgosa)
#endif
C
#endif _OPENACC
!$ACC parallel loop
#endif _OPENACC
!$ACC parallel loop
!$ACC& private (i,j,k)
#else
!$OMP parallel do
!$OMP& private (i,j,k)
!$OMP& shared (p,wrk2)
#endif
DO K=1,kmax
  DO J=1,jmax
    kk = i + (j-1)*imax
    pack_p31(kk)=p(i,j,2)
    pack_p32(kk)=p(i,j,kmax-1)
  enddo
enddo
#endif
C
call sendp(ndx,ndy,ndz,
& pack_p11,pack_p12,pack_p21,pack_p22,
& pack_p31,pack_p32,unpack_p11,unpack_p12,
& unpack_p21,unpack_p22,unpack_p31,unpack_p32)
OpenACC Version 3 – b)

```c
#ifdef _OPENACC
$acc update device(unpack_p11, unpack_p12, unpack_p21, unpack_p22, unpack_p31, unpack_p32)
#endif
C
#ifdef _OPENACC
$ACC parallel loop
$ACC& private (ii, j, k)
#else
$OMP parallel do
$OMP& private (ii, j, k)
#endif
shared (p, wrk2)
#endif
DO K=1, kmax
  DO J=1, jmax
    ii = j + (k-1)*jmax
    p(2, j, k) = unpack_p11(ii)
    p(imax-1, j, k) = unpack_p12(ii)
  enddo
enddo
#ifdef _OPENACC
$ACC end parallel loop
#endif

#ifdef _OPENACC
$ACC parallel loop
$ACC& private (i, jj, k)
#else
$OMP parallel do
$OMP& private (i, jj, k)
#endif
shared (p, wrk2)
#endif
DO K=1, kmax
  DO I=1, imax
    jj = i + (k-1)*imax
    p(i, 2, k) = unpack_p21(jj)
    p(i, jmax-1, k) = unpack_p22(jj)
  enddo
enddo
#ifdef _OPENACC
$ACC end parallel loop
#endif

#ifdef _OPENACC
$ACC parallel loop
$ACC& private (i, jj, k)
#else
$OMP parallel do
$OMP& private (i, jj, k)
#endif
shared (p, wrk2)
#endif
DO K=1, kmax
  DO J=1, jmax
    ii = j + (k-1)*jmax
    p(2, j, k) = unpack_p11(ii)
    p(imax-1, j, k) = unpack_p12(ii)
  enddo
enddo
#ifdef _OPENACC
$ACC end parallel loop
#endif

#ifdef _OPENACC
$ACC parallel loop
$ACC& private (i, jj, k)
#else
$OMP parallel do
$OMP& private (i, jj, k)
#endif
shared (p, wrk2)
#endif
DO K=1, kmax
  DO I=1, imax
    kk = i + (j-1)*imax
    p(i, jj, 2) = unpack_p31(kk)
    p(i, jj, kmax-1) = unpack_p32(kk)
  enddo
enddo
#ifdef _OPENACC
$ACC end parallel loop
#endif
```
## Comparisons of Himeno Versions

<table>
<thead>
<tr>
<th>Version of Himeno</th>
<th>GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original running on 16 nodes 4 MPI/Node</td>
<td>78.75</td>
</tr>
<tr>
<td>OMP version1 running on 16 nodes 4 MPI/Node 4 threads/MPI task</td>
<td>169.5</td>
</tr>
<tr>
<td>OMP version2 running on 16 nodes 4 MPI/Node 4 threads/MPI task</td>
<td>180.9</td>
</tr>
<tr>
<td>OpenACC version 1 running on 16 nodes 4 MPI/Node 1 GPU</td>
<td>45.991</td>
</tr>
<tr>
<td>OpenACC version 2 running on 16 nodes 4 MPI/Node 1 GPU</td>
<td>256.9</td>
</tr>
<tr>
<td>OpenACC version 3 running on 16 nodes 4 MPI/Node 1 GPU</td>
<td>593.3</td>
</tr>
<tr>
<td>Last Version running in OpenMP mode</td>
<td>180.99</td>
</tr>
</tbody>
</table>
#![host_data use_device]

#ifdef _OPENACC

!$acc data present(f)
!$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
#endif

!importantacc end host_data
!$acc end data
!importantacc end data
11. Gather *perftools* statistics on code and identify bottlenecks

Table 1: Time and Bytes Transferred for Accelerator Regions

<table>
<thead>
<tr>
<th>Acc %</th>
<th>Acc Time</th>
<th>Host Time</th>
<th>Acc Copy</th>
<th>Acc Copy</th>
<th>Events</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>130.491</td>
<td>140.390</td>
<td>50831</td>
<td>96209</td>
<td>897204</td>
<td>Total</td>
</tr>
<tr>
<td>17.1%</td>
<td>22.301</td>
<td>0.118</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>reaction_rate_vec_.ACC_ASYNC_KERNEL@li.167</td>
</tr>
<tr>
<td>8.9%</td>
<td>11.634</td>
<td>0.069</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_KERNEL@li.916</td>
</tr>
<tr>
<td>5.1%</td>
<td>6.594</td>
<td>0.810</td>
<td>6961</td>
<td>--</td>
<td>8400</td>
<td>derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.</td>
</tr>
<tr>
<td>4.5%</td>
<td>5.815</td>
<td>0.004</td>
<td>--</td>
<td>100</td>
<td>computecoefficients_r_.ACC_ASYNC_KERNEL@li.155</td>
<td></td>
</tr>
<tr>
<td>3.7%</td>
<td>4.829</td>
<td>0.820</td>
<td>6961</td>
<td>--</td>
<td>8400</td>
<td>derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.</td>
</tr>
<tr>
<td>3.5%</td>
<td>4.503</td>
<td>0.872</td>
<td>6961</td>
<td>--</td>
<td>8400</td>
<td>derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.</td>
</tr>
<tr>
<td>3.1%</td>
<td>4.022</td>
<td>0.176</td>
<td>--</td>
<td>6497</td>
<td>1800</td>
<td>derivative_z_pack_np_.ACC_ASYNC_COPY@li.577</td>
</tr>
<tr>
<td>2.9%</td>
<td>3.842</td>
<td>0.241</td>
<td>--</td>
<td>6497</td>
<td>1800</td>
<td>derivative_z_pack_np_.ACC_ASYNC_COPY@li.619</td>
</tr>
<tr>
<td>2.9%</td>
<td>3.809</td>
<td>0.018</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_KERNEL@li.1737</td>
</tr>
<tr>
<td>2.8%</td>
<td>3.598</td>
<td>0.071</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_KERNEL@li.1680</td>
</tr>
<tr>
<td>2.7%</td>
<td>3.517</td>
<td>2.074</td>
<td>6961</td>
<td>--</td>
<td>8400</td>
<td>derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.</td>
</tr>
<tr>
<td>2.3%</td>
<td>3.060</td>
<td>0.009</td>
<td>--</td>
<td>19491</td>
<td>100</td>
<td>integrate_.ACC_ASYNC_COPY@li.75</td>
</tr>
<tr>
<td>2.2%</td>
<td>2.856</td>
<td>0.174</td>
<td>--</td>
<td>6497</td>
<td>1800</td>
<td>derivative_y_pack_np_.ACC_ASYNC_COPY@li.650</td>
</tr>
<tr>
<td>2.1%</td>
<td>2.801</td>
<td>0.175</td>
<td>--</td>
<td>6497</td>
<td>1800</td>
<td>derivative_y_pack_np_.ACC_ASYNC_COPY@li.608</td>
</tr>
<tr>
<td>1.9%</td>
<td>2.529</td>
<td>0.068</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_KERNEL@li.624</td>
</tr>
<tr>
<td>1.9%</td>
<td>2.526</td>
<td>0.080</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_KERNEL@li.1636</td>
</tr>
<tr>
<td>1.8%</td>
<td>2.402</td>
<td>0.084</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_KERNEL@li.400</td>
</tr>
<tr>
<td>1.8%</td>
<td>2.399</td>
<td>0.066</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_KERNEL@li.450</td>
</tr>
<tr>
<td>1.8%</td>
<td>2.375</td>
<td>2.799</td>
<td>--</td>
<td>7341</td>
<td>600</td>
<td>save_bc_deriv1$rhsf_.ACC_ASYNC_COPY@li.248</td>
</tr>
<tr>
<td>1.7%</td>
<td>2.251</td>
<td>0.777</td>
<td>6961</td>
<td>--</td>
<td>8400</td>
<td>derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.</td>
</tr>
<tr>
<td>1.6%</td>
<td>2.145</td>
<td>0.770</td>
<td>6961</td>
<td>--</td>
<td>8400</td>
<td>derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.</td>
</tr>
<tr>
<td>1.6%</td>
<td>2.043</td>
<td>0.043</td>
<td>--</td>
<td>--</td>
<td>100</td>
<td>computecoefficients_r_.ACC_ASYNC_KERNEL@li.225</td>
</tr>
<tr>
<td>1.5%</td>
<td>1.938</td>
<td>0.066</td>
<td>--</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_KERNEL@li.493</td>
</tr>
<tr>
<td>1.4%</td>
<td>1.877</td>
<td>0.172</td>
<td>--</td>
<td>6497</td>
<td>1800</td>
<td>derivative_x_pack_np_.ACC_ASYNC_COPY@li.640</td>
</tr>
<tr>
<td>1.3%</td>
<td>1.734</td>
<td>1.674</td>
<td>3544</td>
<td>--</td>
<td>600</td>
<td>rhsf_.ACC_ASYNC_COPY@li.2436</td>
</tr>
<tr>
<td>1.1%</td>
<td>1.444</td>
<td>1.270</td>
<td>--</td>
<td>464.062</td>
<td>6600</td>
<td>derivative_x_pack_.ACC_ASYNC_COPY@li.598</td>
</tr>
<tr>
<td>1.0%</td>
<td>1.254</td>
<td>0.027</td>
<td>--</td>
<td>--</td>
<td>700</td>
<td>calc_primary_vars_.ACC_ASYNC_KERNEL@li.47</td>
</tr>
<tr>
<td>1.0%</td>
<td>1.247</td>
<td>0.160</td>
<td>--</td>
<td>6497</td>
<td>1800</td>
<td>derivative_x_pack_np_.ACC_ASYNC_COPY@li.598</td>
</tr>
</tbody>
</table>
Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
</tr>
<tr>
<td>100.0%</td>
<td>174.160022</td>
<td>--</td>
<td>--</td>
<td>4867603.0</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>92.4%</td>
<td>160.926071</td>
<td>--</td>
<td>--</td>
<td>2676780.0</td>
<td>USER</td>
<td></td>
</tr>
<tr>
<td>12.8%</td>
<td>22.319336</td>
<td>0.000000</td>
<td>0.0%</td>
<td>600.0</td>
<td>reaction_rate_vec_.ACC_SYNC_WAIT@li.5008</td>
<td></td>
</tr>
<tr>
<td>10.3%</td>
<td>17.997279</td>
<td>0.000000</td>
<td>0.0%</td>
<td>600.0</td>
<td>rhsf_.ACC_DATA_REGION@li.256</td>
<td></td>
</tr>
<tr>
<td>7.6%</td>
<td>13.238744</td>
<td>0.000000</td>
<td>0.0%</td>
<td>6600.0</td>
<td>derivative_z_pack_.ACC_ASYNC_KERNEL@li.479</td>
<td></td>
</tr>
<tr>
<td>3.4%</td>
<td>5.842934</td>
<td>0.000000</td>
<td>0.0%</td>
<td>3000.0</td>
<td>derivative_xyz_wait_np$derivative_m_.ACC_SYNC_WAIT@li.567</td>
<td></td>
</tr>
<tr>
<td>3.3%</td>
<td>5.817360</td>
<td>0.000000</td>
<td>0.0%</td>
<td>100.0</td>
<td>computecoefficients_r_.ACC_SYNC_WAIT@li.222</td>
<td></td>
</tr>
<tr>
<td>2.7%</td>
<td>4.743826</td>
<td>0.000321</td>
<td>0.0%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.2268</td>
<td></td>
</tr>
<tr>
<td>2.3%</td>
<td>3.991119</td>
<td>0.000000</td>
<td>0.0%</td>
<td>6600.0</td>
<td>derivative_x_send_.ACC_SYNC_WAIT@li.717</td>
<td></td>
</tr>
<tr>
<td>1.8%</td>
<td>3.072952</td>
<td>0.000000</td>
<td>0.0%</td>
<td>100.0</td>
<td>integrate_.ACC_SYNC_WAIT@li.75</td>
<td></td>
</tr>
<tr>
<td>1.7%</td>
<td>3.040157</td>
<td>0.000000</td>
<td>0.0%</td>
<td>201600.0</td>
<td>deriv_inplane_1</td>
<td></td>
</tr>
<tr>
<td>1.7%</td>
<td>3.024576</td>
<td>0.000000</td>
<td>0.0%</td>
<td>560.0</td>
<td>filter$filter_m</td>
<td></td>
</tr>
<tr>
<td>1.7%</td>
<td>3.019308</td>
<td>0.000000</td>
<td>0.0%</td>
<td>700.0</td>
<td>calc_primary_vars_.ACC_SYNC_WAIT@li.157</td>
<td></td>
</tr>
<tr>
<td>1.6%</td>
<td>2.798427</td>
<td>0.000000</td>
<td>0.0%</td>
<td>600.0</td>
<td>save_bc_deriv1$rhsf_.ACC_ASYNC_COPY@li.248</td>
<td></td>
</tr>
<tr>
<td>1.2%</td>
<td>2.111812</td>
<td>0.000000</td>
<td>0.0%</td>
<td>201600.0</td>
<td>deriv_inplane_2</td>
<td></td>
</tr>
<tr>
<td>1.2%</td>
<td>2.071792</td>
<td>0.000000</td>
<td>0.0%</td>
<td>8400.0</td>
<td>derivative_xyz_wait_np$derivative_m_.ACC_ASYNC_COPY@li.544</td>
<td></td>
</tr>
<tr>
<td>1.2%</td>
<td>2.006773</td>
<td>0.000000</td>
<td>0.0%</td>
<td>100.0</td>
<td>computecoefficients_r_.ACC_SYNC_WAIT@li.1748</td>
<td></td>
</tr>
<tr>
<td>1.1%</td>
<td>1.975207</td>
<td>0.000000</td>
<td>0.0%</td>
<td>6600.0</td>
<td>derivative_z_pack_.ACC_ASYNC_COPY@li.454</td>
<td></td>
</tr>
<tr>
<td>1.1%</td>
<td>1.914216</td>
<td>0.000000</td>
<td>0.0%</td>
<td>100.0</td>
<td>controller$rk_m</td>
<td></td>
</tr>
<tr>
<td>1.0%</td>
<td>1.673879</td>
<td>0.000000</td>
<td>0.0%</td>
<td>600.0</td>
<td>rhsf_.ACC_ASYNC_COPY@li.2436</td>
<td></td>
</tr>
<tr>
<td>0.9%</td>
<td>1.615192</td>
<td>0.000000</td>
<td>0.0%</td>
<td>600.0</td>
<td>rhsf_.ACC_ASYNC_COPY@li.2187</td>
<td></td>
</tr>
<tr>
<td>0.9%</td>
<td>1.598921</td>
<td>0.000000</td>
<td>0.0%</td>
<td>600.0</td>
<td>rhsf_.ACC_ASYNC_COPY@li.2189</td>
<td></td>
</tr>
<tr>
<td>0.9%</td>
<td>1.586929</td>
<td>0.000000</td>
<td>0.0%</td>
<td>600.0</td>
<td>rhsf_.ACC_ASYNC_COPY@li.2191</td>
<td></td>
</tr>
<tr>
<td>0.7%</td>
<td>1.268257</td>
<td>0.000000</td>
<td>0.0%</td>
<td>6600.0</td>
<td>derivative_x_pack_.ACC_ASYNC_COPY@li.463</td>
<td></td>
</tr>
<tr>
<td>0.6%</td>
<td>1.080301</td>
<td>0.001090</td>
<td>0.1%</td>
<td>600.0</td>
<td>rhsf_.LOOP@li.2411</td>
<td></td>
</tr>
<tr>
<td>0.5%</td>
<td>0.949635</td>
<td>0.000000</td>
<td>0.0%</td>
<td>100.0</td>
<td>integrate_.ACC_SYNC_WAIT@li.145</td>
<td></td>
</tr>
<tr>
<td>0.5%</td>
<td>0.892484</td>
<td>0.000000</td>
<td>0.0%</td>
<td>67200.0</td>
<td>point_der1_y</td>
<td></td>
</tr>
<tr>
<td>0.5%</td>
<td>0.888298</td>
<td>0.000000</td>
<td>0.0%</td>
<td>67200.0</td>
<td>point_der1_x</td>
<td></td>
</tr>
<tr>
<td>0.5%</td>
<td>0.870532</td>
<td>0.000000</td>
<td>0.0%</td>
<td>100.0</td>
<td>integrate_.ACC_SYNC_WAIT@li.99</td>
<td></td>
</tr>
</tbody>
</table>
12. If bottleneck is data copies and you did a good job on 9. - look at packing buffers on the accelerator

if (vary_in_x==1) then
    call derivative_x_pack_np( nx, ny, nz, yspecies(1,1,1,1), 5, 'yspecies-x', n_spec, 25)
endif
if (vary_in_y==1) then
    call derivative_y_pack_np( nx, ny, nz, yspecies(1,1,1,1), 5, 'yspecies-y', n_spec, 27)
endif
if (vary_in_z==1) then
    call derivative_z_pack_np( nx, ny, nz, yspecies(1,1,1,1), 5, 'yspecies-z', n_spec, 29)
endif
if (vary_in_x==1) then
    call derivative_x_pack( nx, ny, nz, temp(1,1,1), 4, 'temp-x', 19)
    call derivative_x_pack( nx, ny, nz, u(1,1,1,1), 1, 'u-x1', 1)
    call derivative_x_pack( nx, ny, nz, u(1,1,1,2), 2, 'u-x2', 7)
    call derivative_x_pack( nx, ny, nz, u(1,1,1,3), 3, 'u-x3', 13)
endif
if (vary_in_y==1) then
    call derivative_y_pack( nx, ny, nz, temp(1,1,1), 4, 'temp-y', 21)
    call derivative_y_pack( nx, ny, nz, u(1,1,1,1), 1, 'u-y1', 3)
    call derivative_y_pack( nx, ny, nz, u(1,1,1,2), 2, 'u-y2', 9)
    call derivative_y_pack( nx, ny, nz, u(1,1,1,3), 3, 'u-y3', 15)
endif
if (vary_in_z==1) then
    call derivative_z_pack( nx, ny, nz, temp(1,1,1), 4, 'temp-z', 23)
    call derivative_z_pack( nx, ny, nz, u(1,1,1,1), 1, 'u-z1', 5)
    call derivative_z_pack( nx, ny, nz, u(1,1,1,2), 2, 'u-z2', 11)
    call derivative_z_pack( nx, ny, nz, u(1,1,1,3), 3, 'u-z3', 17)
endif
13. If bottleneck is kernel performance

- You absolutely have to vectorize on a good vector length; that is, greater than or equal to 32 (32 is a warp, 128 is 4 warps)
- You need to have thousands of the warps waiting to kick off to amortize latency to memory
- Watch out for register spills
- Watch out for overflowing shared memory
A Simdized OpenACC loop

\[
\text{do } i = 1, \text{ nx*ny*nz, ms}
\]
\[
\text{ ml } = i
\]
\[
\text{ mu } = \min(i+ms-1, \text{ nx*ny*nz})
\]
DIRECTION: do m=1,3
\[
\text{diffFlux(ml:mu,1,1,n_spec,m) } = 0.0
\]
\[
\text{grad_mixMW(ml:mu,1,1,m)=grad_mixMW(ml:mu,1,1,m)}\quad \text{*avmolwt(ml:mu,1,1)}
\]
SPECIES: do n=1,n_spec-1
\[
\text{diffFlux(ml:mu,1,1,n,m)=}-D_{s_{mixavg}}(ml:mu,1,1,n)\quad \text{&}
\]
\[
(\text{grad}_{Ys}(ml:mu,1,1,n,m)+Y_{s}(ml:mu,1,1,n)\quad \text{&}
\]
\[
\text{grad_mixMW(ml:mu,1,1,m) } \quad)
\]
\[
\text{diffFlux(ml:mu,1,1,n_spec,m)=} \quad \text{&}
\]
\[
\text{diffFlux(ml:mu,1,1,n_spec,m)-} \quad \text{&}
\]
\[
\text{diffFlux(ml:mu,1,1,n,m)}
\]
\[
\text{enddo SPECIES}
\]
\[
\text{enddo DIRECTION}
\]
\[
\text{enddo}
\]
A Better Simdized OpenACC loop

do i = 1, nx*ny*nz, ms
  ml = i
  mu = min(i+ms-1, nx*ny*nz)
  difftemp1 = 0.0
  difftemp2 = 0.0
  difftemp3 = 0.0
  do m = ml.mu
    grad_mixMW(m,1,1,1) = grad_mixMW(m,1,1,1)* avmolwt(m,1,1)
    grad_mixMW(m,1,1,2) = grad_mixMW(m,1,1,2)* avmolwt(m,1,1)
    grad_mixMW(m,1,1,3) = grad_mixMW(m,1,1,3)* avmolwt(m,1,1)
  do n=1,n_spec-1
    diffFlux(m,1,1,n,1) = - ds_mxvg(m,1,1,n)* ( grad_Ys(m,1,1,n,1) + yspecies(m,1,1,n)*grad_mixMW(m,1,1,1) )
    diffFlux(m,1,1,n,2) = - ds_mxvg(m,1,1,n)* ( grad_Ys(m,1,1,n,2) + yspecies(m,1,1,n)*grad_mixMW(m,1,1,2) )
    diffFlux(m,1,1,n,3) = - ds_mxvg(m,1,1,n)* ( grad_Ys(m,1,1,n,3) + yspecies(m,1,1,n)*grad_mixMW(m,1,1,3) )
    difftemp1 = difftemp1-diffFlux(m,1,1,n,1)
    difftemp2 = difftemp2-diffFlux(m,1,1,n,2)
    difftemp3 = difftemp3-diffFlux(m,1,1,n,3)
  enddo! n
  diffFlux(m,1,1,n_spec,1) = difftemp1
  diffFlux(m,1,1,n_spec,2) = difftemp2
  diffFlux(m,1,1,n_spec,3) = difftemp3
  grad_T(m,1,1,1) = -lambda(m,1,1)* grad_T(m,1,1,1)
  grad_T(m,1,1,2) = -lambda(m,1,1)* grad_T(m,1,1,2)
  grad_T(m,1,1,3) = -lambda(m,1,1)* grad_T(m,1,1,3)
  do n=1,n_spec
    grad_T(m,1,1,1) = grad_T(m,1,1,1)+ h_spec(m,1,1,n)*diffFlux(m,1,1,n,1)
    grad_T(m,1,1,2) = grad_T(m,1,1,2)+ h_spec(m,1,1,n)*diffFlux(m,1,1,n,2)
    grad_T(m,1,1,3) = grad_T(m,1,1,3)+ h_spec(m,1,1,n)*diffFlux(m,1,1,n,3)
  enddo! n
  enddo! m
enddo! i
Temperature Interpolation loop

tmp1(ml:mu) = e0(ml:mu) - 0.5*tmp1(ml:mu)
LOOPM: DO m = ml, mu
   icount = 1
   r_gas = Ru*avmolwt(m)
   yspec(:) = ys(m, :)
ITERATION: DO
   cpmix(m) = mixCp( yspec, temp(m) )
   enthmix = mixEnth( yspec, temp(m) )
   deltat = &
   ( tmp1(m) - (enthmix- &
      r_gas*temp(m)) ) &
   / ( cpmix(m) - r_gas )
   temp(m) = temp(m) + deltat
   IF( ABS(deltat) < atol ) THEN
      cpmix(m) = mixCp( yspec, &
      temp(m) )
      EXIT ITERATION
   ELSEIF( icount > icountmax ) THEN
      STOP
   ELSE
      icount = icount + 1
   ENDIF
ENDDO ITERATION
ENDDO LOOPM
Temperature Interpolation loop

ITERATION: do
   do m = ml, mu
      !-- compute mixture heat capacity and enthalpy for this temperature
      n = max(1,min(2001,int((temp(m)-temp_lobound)*invEnthInc)+1))
      cpmix(m) = 0.0
      do mm=1,n_spec
         cpmix(m) = cpmix(m) + &
         yspecies(m,mm)*(cpCoef_aa(mm,n) * temp(m) + cpCoef_bb(mm,n) )
      enddo
      enthmix(m) = 0.0
      do mm=1,n_spec
         enthmix(m) = enthmix(m) + yspecies(m,mm)*(enthCoef_aa(mm,n)*temp(m) + enthCoef_bb(mm,n))
      enddo
      !-- calculate deltat, new temp
      !   remember tmp1 holds the internal energy
      deltat(m) = ( tmp1(m) - (enthmix(m)-Ru*avmolwt(m)*temp(m)) ) &
                   /( cpmix(m) - Ru*avmolwt(m) )
      if(iconsverge(m).eq.0)temp(m) = temp(m) + deltat(m)
   enddo
   do m = ml, mu
      !-- check for convergence
      ! if( abs(deltat(m)) < atol.and.iconsverge(m).eq.0 ) then  ! converged
      !     BUG- FIX AUG-16-04 - cpmix was not updated after successful convergence
      !     iconsverge(m) = 1
      !     n = max(1,min(2001,int((temp(m)-temp_lobound)*invEnthInc)+1))
      !     cpmix(m) = 0.0
      !     do mm=1,n_spec
      !        cpmix(m) = cpmix(m) + &
      !        yspecies(m,mm)*(cpCoef_aa(mm,n) * temp(m) + cpCoef_bb(mm,n) )
      !     enddo
      !   endif
   enddo
Temperature Interpolation loop

if(all(iconverge(ml:mu).eq.1))EXIT ITERATION
    EXIT ITERATION
    do m = ml,mu
        if(iconverge(m).eq.0)then
            if( icount(m) > icountmax ) then ! maximum count violation
                write(6,'(a)')'calc_temp cannot converge after 100 iterations'
                write(6,'(a)') 'for processor with rank =',myid
                write(6,'(a)') 'm=',m
                stop !ugly termination but that's the way it is without doing a broadcast
            else
                icount(m) = icount(m) + 1
            endif
        endif
    enddo
enddo ITERATION
end do
14. Consider introducing CUDA streams

if (vary_in_x==1) then
  call derivative_x_pack_np( nx, ny, nz, yspecies(1,1,1,1), 5, 'yspecies-x', n_spec, 25)
endif
if (vary_in_y==1) then
  call derivative_y_pack_np( nx, ny, nz, yspecies(1,1,1,1), 5, 'yspecies-y', n_spec, 27)
endif
if (vary_in_z==1) then
  call derivative_z_pack_np( nx, ny, nz, yspecies(1,1,1,1), 5, 'yspecies-z', n_spec, 29)
endif
if (vary_in_x==1) then
  call derivative_x_pack( nx, ny, nz, temp(1,1,1), 4, 'temp-x', 19)
call derivative_x_pack( nx, ny, nz, u(1,1,1,1), 1, 'u-x1', 1)
call derivative_x_pack( nx, ny, nz, u(1,1,1,2), 2, 'u-x2', 7)
call derivative_x_pack( nx, ny, nz, u(1,1,1,3), 3, 'u-x3', 13)
endif
if (vary_in_y==1) then
  call derivative_y_pack( nx, ny, nz, temp(1,1,1), 4, 'temp-y', 21)
call derivative_y_pack( nx, ny, nz, u(1,1,1,1), 1, 'u-y1', 3)
call derivative_y_pack( nx, ny, nz, u(1,1,1,2), 2, 'u-y2', 9)
call derivative_y_pack( nx, ny, nz, u(1,1,1,3), 3, 'u-y3', 15)
endif
if (vary_in_z==1) then
  call derivative_z_pack( nx, ny, nz, temp(1,1,1), 4, 'temp-z', 23)
call derivative_z_pack( nx, ny, nz, u(1,1,1,1), 1, 'u-z1', 5)
call derivative_z_pack( nx, ny, nz, u(1,1,1,2), 2, 'u-z2', 11)
call derivative_z_pack( nx, ny, nz, u(1,1,1,3), 3, 'u-z3', 17)
endif
! Start communication - the _prep routines do posts and sends using buffer
! identified by itmp
call computeScalarGradient_prep_np(yspecies(1,1,1,1), 5,25,n_spec)
  itmp = 4
  istr = 19
  call computeScalarGradient_prep( temp, itmp, istr )
  itmp = 1
  istr = 1
  call computeVectorGradient_prep( u, itmp,istr )
endif
14. Continued

do i = 1, reqcount
    call MPI_WAITANY(reqcount,req,index, stat, ierr )
if(direction(index).eq.1)then
!$acc update device(pos_f_x_buf(:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.2)then
!$acc update device(neg_f_x_buf(:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.3)then
!$acc update device(pos_f_y_buf(:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.4)then
!$acc update device(neg_f_y_buf(:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.5)then
!$acc update device(pos_f_z_buf(:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
if(direction(index).eq.6)then
!$acc update device(neg_f_z_buf(:,:,idx(index):idx(index)+nb(index)-1)) async(isync)
endif
isync=isync+1
  enddo
15. Start looking at timelines showing communication, host execution and accelerator
Strategy for refactoring the application

1. First and foremost – Profile the application
   Must identify looping structure within the time step loop
   Use –h profile_generate on compile and –Ocalltree or –Ocallers

2. Use Reveal to identify scoping of variables in the major loop – may call subroutines and functions
   The idea is to first generate OpenMP version of the loop and then add some OpenACC

3. Use OpenACC to identify data motion require to run with companion accelerator
   Once scoping is obtained, the OpenACC compiler will indicate what data would need to be moved to run on the accelerator – user must have the variable scoping correct

4. Once one loop is analyze, now look at next highest compute loop, perform steps 2 and 3.

5. Soon multiple loops can be combined within a OpenACC data region for eliminating transfers to and from the host.
6. Work outward until a data region encompasses a communication, I/O or looping structure more suited for the host
   a. Must use updates to move data to and from the host to supply host with up-to-date data

7. Move data region outside time step loop
   a. Now must account for all updates to keep host and accelerator with consistent data

8. Test versions after each step – don’t worry about performance yet – just accuracy

9. The compiler may introduce data transfer so look at –rm listing for each individual OpenACC loop.

10. Optimize/Minimize data transfers first by using present on data clause.
Strategy for refactoring the application

11. Gather perftools statistics on code and identify bottlenecks

12. If bottleneck is data copies look at step 9

13. If bottleneck is kernel performance
   A. Look at –rm and see what the compiler did to optimize the loop
   B. Ideally we want three levels of parallelism, gang, worker, vector
   C. Inner loop needs to be g on listing
   D. If inner loop is indicated by a loop level, that means that it is running in scalar – BAD

14. Consider introducing CUDA streams
   A. Either by taking an outer loop that cannot be parallelized due to communication and running that in a streaming mode
   B. Taking several independent operations and running that in a stream mode

15. Start looking at timelines showing communication, host execution and accelerator
   A. What can be overlapped