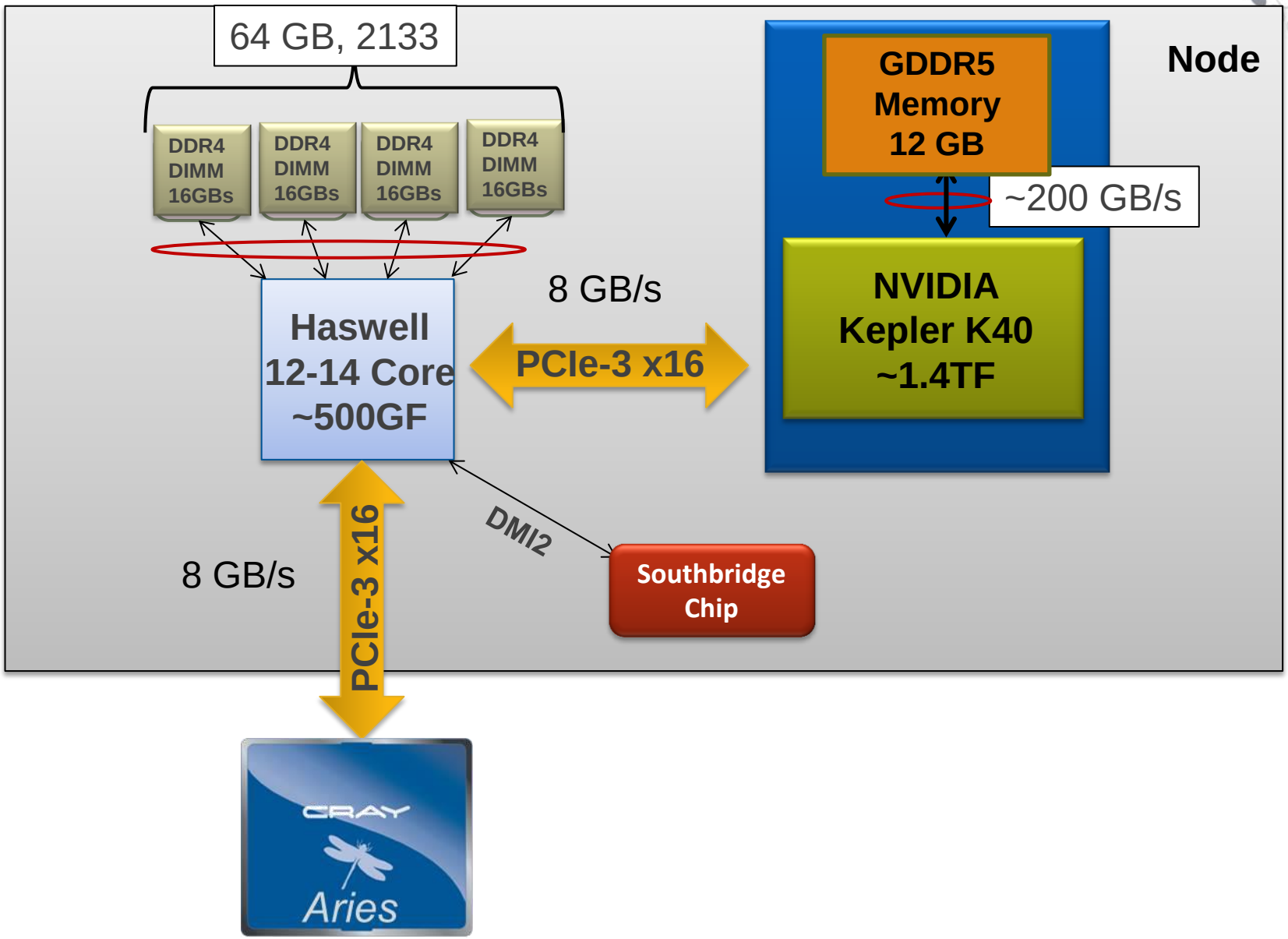
Performance challenges when using OpenACC to port applications

John M Levesque
Director
Cray's Supercomputing Center of Excellence
CTO Office - Applications

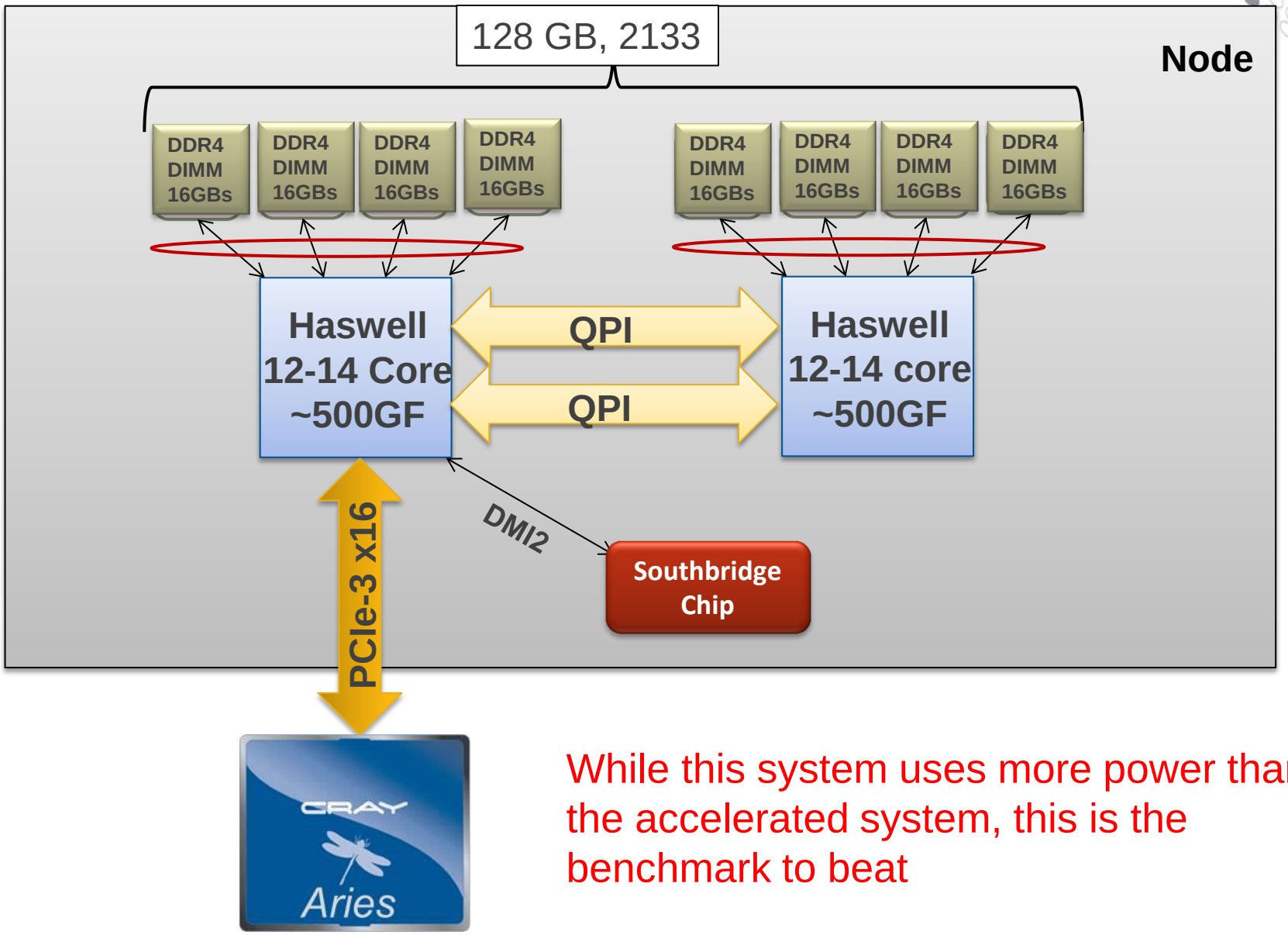
Challenges

- Obtain Performance enhancements on a high speed accelerator attached to a state-of-the-art micro-processor through a relatively low speed PCI express channel

XC-30 Haswell Compute Node with Accelerator

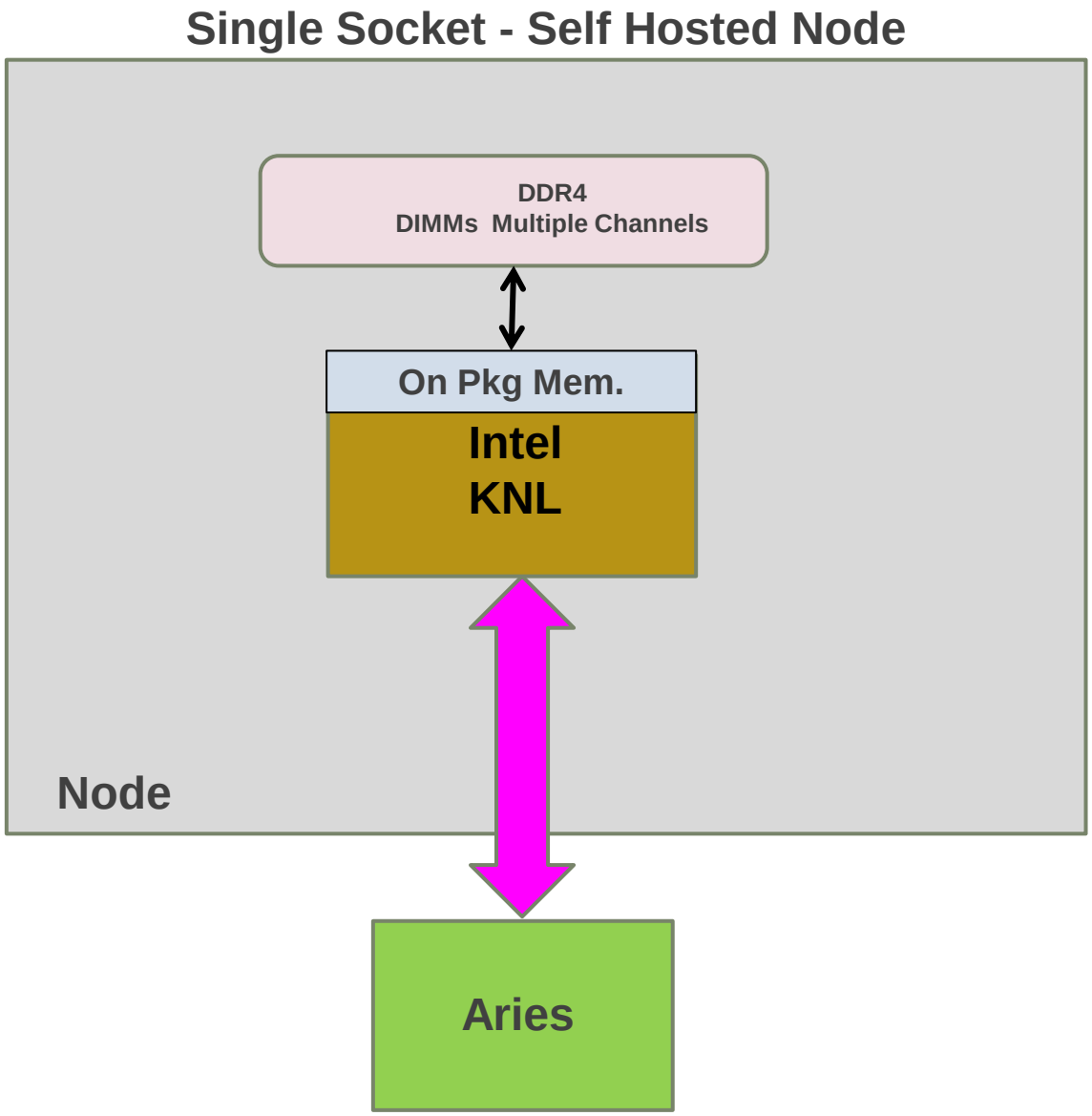


Haswell Compute Node



While this system uses more power than the accelerated system, this is the benchmark to beat

Future XC KNL Node



Common Characteristics of next generation nodes

- **Multiple memories**
  - High bandwidth relatively small memory
  - Low bandwidth larger memory
- **Lots and lots of threading**
  - Typical multi-threading to utilize cores
  - Hyper-threading
- **Vectorization is more important**

You Snooze and you will lose

Challenges

- Obtain Performance enhancements on a high speed accelerator attached to a state-of-the-art micro-processor through a relatively low speed PCI express channel
- **Computational performance on the accelerator for gather/scatter operations will be better than on the host; however, care must be taken to optimize them as much as possible**
- **The biggest challenge is to orchestrate all the data movement, on and off the accelerator and between the nodes in the most efficient manner. It should be possible to hide all computation under data movement**
- **Taking advantage of the four types of parallelism**
  - MPI between nodes
  - Threading on the node (Including host and accelerator)
  - SIMDization on the accelerator
  - Streaming on the host and the accelerator

Himeno

- **The following simple benchmark has many issues that are similar to those in O&G**
- **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

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE
100.0%	66.812060	--	--	2372.1	Total
100.0%	66.811928	--	--	2167.0	USER
94.2%	62.965959	2.805426	4.3%	2.0	jacobi_.LOOPS
2.1%	1.385449	3.224803	71.1%	539.0	sendp1_
1.7%	1.167354	3.333783	75.2%	539.0	sendp2_
0.7%	0.461089	0.847765	65.8%	539.0	sendp3_
0.7%	0.455975	0.009942	2.2%	1.0	initmt_.LOOPS
0.4%	0.295538	0.302255	51.4%	1.0	initcomm_
0.1%	0.075184	0.015566	17.4%	1.0	himenobmtxp_
0.0%	0.005124	0.003026	37.7%	539.0	sendp_
0.0%	0.000094	0.000009	8.7%	1.0	initmax_.LOOPS
0.0%	0.000090	0.000010	9.7%	1.0	initmax_
0.0%	0.000042	0.000012	22.9%	1.0	exit
0.0%	0.000021	0.000005	20.2%	2.0	jacobi_
0.0%	0.000009	0.000005	37.5%	1.0	initmt_
0.0%	0.000088	--	--	2.0	MPI
0.0%	0.000087	0.000012	12.0%	1.0	mpi_finalize_
0.0%	0.000001	0.000000	22.0%	1.0	mpi_init_
0.0%	0.000044	--	--	203.1	ETC
0.0%	0.000043	0.000086	67.9%	203.1	__libc_csu_init
0.0%	0.000001	0.000068	100.0%	0.0	_STOP2

Loop Profile

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

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

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,J+1,K)
     1                 +a(I,J,K,3)*p(I,J,K+1)
     2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
     4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
     5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
     6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
     7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
     8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9                 +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 m----------<          DO K=2,kmax-1
219.   + 1 m 3--------<             DO J=2,jmax-1
220.     1 m 3 Vr3----<                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

```
#ifdef _OPENACC
!$ACC  parallel loop
!$ACC&   private (s0,ss)
!$ACC&   reduction (+:wgosa)
#else
!$OMP  parallel do default(none)
!$OMP&   private (s0,ss)
!$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)
     1                 +a(I,J,K,3)*p(I,J,K+1)
     2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
     4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
     5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
     6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
     7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
     8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9                 +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
```

Introducing OpenACC –compiler listing

```
214.     1                  #ifdef GPU
215.   + 1 G------------<   !$ACC  parallel loop
216.     1 G                !$ACC&   private (i,j,k,s0,ss)
217.     1 G                !$ACC&   reduction (+:wgosa)
218.     1 G                #else
219.     1 G                !$OMP  parallel do default(none)
220.     1 G                !$OMP&   private (i,j,k,s0,ss)
221.     1 G                !$OMP&   shared  (a,b,bnd,c,imax,jmax,kmax,omega,p,wrk1,wrk2)
222.     1 G                !$OMP&   reduction (+:wgosa)
223.     1 G                #endif
224.     1 G g----------<             DO K=2,kmax-1
225.   + 1 G g 4--------<                DO J=2,jmax-1
226.     1 G g 4 g------<                   DO I=2,imax-1
227.     1 G g 4 g                             S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
228.     1 G g 4 g                1                 +a(I,J,K,3)*p(I,J,K+1)
229.     1 G g 4 g                2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
230.     1 G g 4 g                3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
231.     1 G g 4 g                4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
232.     1 G g 4 g                5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
233.     1 G g 4 g                6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
234.     1 G g 4 g                7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
235.     1 G g 4 g                8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
236.     1 G g 4 g                9                 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
237.     1 G g 4 g                             SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
238.     1 G g 4 g                             WGOSA=WGOSA+SS*SS
239.     1 G g 4 g                             wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
240.     1 G g 4 g------>                   enddo
241.     1 G g 4-------->                enddo
242.     1 G g---------->             enddo
243.     1 G                #ifdef GPU
244.     1 G------------> !$ACC  end parallel loop
245.     1                  #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.
```

OpenACC Version 1 – Profile output

```
Table 1:  Profile by Function Group and Function

 Time%  |     Time  |    Imb.  |  Imb.  | Calls  |Group
        |           |    Time  | Time%  |        | Function
        |           |          |        |        |  PE=HIDE
        |           |          |        |        |   Thread=HIDE

 100.0% | 62.073674 |       -- |     -- | 7627.1 |Total
|-------------------------------------------------------------------------------
|  95.6% | 59.368154 |       -- |     -- | 2565.0 |USER
||------------------------------------------------------------------------------
||  81.0% | 50.284595 | 1.058725 |   2.1% |  284.0 |jacobi_.ACC_COPY@li.215
||   3.8% |  2.359195 | 0.094854 |   3.9% |  284.0 |jacobi_.ACC_COPY@li.244
||   3.3% |  2.055168 | 0.131544 |   6.1% |    2.0 |jacobi_
||   2.6% |  1.616893 | 0.457704 |  22.4% |  284.0 |jacobi_.ACC_SYNC_WAIT@li.244
||   2.5% |  1.535406 | 0.280701 |  15.7% |    1.0 |exit
||   2.3% |  1.410941 | 0.284518 |  17.0% |    1.0 |initmt_.LOOP@li.156
||   0.1% |  0.054959 | 0.001335 |   2.4% |    1.0 |initmt_
||   0.1% |  0.039723 | 0.005092 |  11.5% |  284.0 |jacobi_.ACC_ASYNC_KERNEL@li.215
||   0.0% |  0.003376 | 0.003567 |  52.2% |  284.0 |jacobi_.ACC_REGION@li.215
||   0.0% |  0.002758 | 0.000394 |  12.7% |  284.0 |sendp3_
||   0.0% |  0.002172 | 0.000380 |  15.1% |  284.0 |sendp_
||   0.0% |  0.001459 | 0.000189 |  11.7% |  284.0 |sendp1_
||   0.0% |  0.001155 | 0.000109 |   8.8% |  284.0 |sendp2_
||   0.0% |  0.000276 | 0.000590 |  69.2% |    1.0 |himenobmtxp_
||   0.0% |  0.000047 | 0.000007 |  13.3% |    1.0 |initcomm_
||   0.0% |  0.000025 | 0.000005 |  15.9% |    1.0 |initmax_
||   0.0% |  0.000006 | 0.000018 |  74.8% |    1.0 |initmt_.REGION@li.156
.
```

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
- **Introduces the DIFFICULT task of managing host and device copies of the data**
  - Now must move data back to the host for the message passing

OpenACC Version 2 – Using Data regions –a)

```
#ifdef _OPENACC
!$acc data present(a,p,b,c,bnd,wrk1)present_or_create(wgosa)
#endif
C
      DO loop=1,nn
         gosa=0.0
         wgosa=0.0
! Directive inserted by Cray Reveal.  May be incomplete.
#ifdef _OPENACC
!$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)
     1                 +a(I,J,K,3)*p(I,J,K+1)
     2                 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
     3                 -p(I-1,J+1,K)+p(I-1,J-1,K))
     4                 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
     5                 -p(I,J+1,K-1)+p(I,J-1,K-1))
     6                 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
     7                 -p(I+1,J,K-1)+p(I-1,J,K-1))
     8                 +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
     9                 +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
#ifdef _OPENACC
!$acc end data
#endif
.
```

OpenACC Version 2 – Using Data regions –a)

```
240.  + G------------< !$acc data present(a,p,b,c,bnd,wrk1)present_or_create(wgosa)
 241.    G             #endif
 242.    G             C
 243.  + G 2----------<       DO loop=1,nn
 244.    G 2                    gosa=0.0
 245.    G 2                    wgosa=0.0
 246.    G 2           ! Directive inserted by Cray Reveal.  May be incomplete.
 247.    G 2           #ifdef _OPENACC
 248.  + G 2 G--------< !$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 g 5----<             DO J=2,jmax-1
 259.    G 2 G g 5 g--<               DO I=2,imax-1
 260.    G 2 G g 5 g                    S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
 261.    G 2 G g 5 g         1               +a(I,J,K,3)*p(I,J,K+1)
 262.    G 2 G g 5 g         2               +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
 263.    G 2 G g 5 g         3               -p(I-1,J+1,K)+p(I-1,J-1,K))
 264.    G 2 G g 5 g         4               +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
 265.    G 2 G g 5 g         5               -p(I,J+1,K-1)+p(I,J-1,K-1))
 266.    G 2 G g 5 g         6               +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
 267.    G 2 G g 5 g         7               -p(I+1,J,K-1)+p(I-1,J,K-1))
 268.    G 2 G g 5 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 g 5 g         9               +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
 270.    G 2 G g 5 g                    SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
 271.    G 2 G g 5 g                    WGOSA=WGOSA+SS*SS
 272.    G 2 G g 5 g                    wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
 273.    G 2 G g 5 g-->               enddo
 274.    G 2 G g 5---->             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)

```
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          !$OMP&   shared  (p,wrk2)
 287.    G 2 G          #endif
 288.    G 2 G g------<           DO K=2,kmax-1
 289.  + G 2 G g 5----<              DO J=2,jmax-1
 290.    G 2 G g 5 g--<                 DO I=2,imax-1
 291.    G 2 G g 5 g                      p(I,J,K)=wrk2(I,J,K)
 292.    G 2 G g 5 g-->                 enddo
 293.    G 2 G g 5---->              enddo
 294.    G 2 G g------>           enddo
 295.    G 2 G          #ifdef _OPENACC
 296.    G 2 G--------> !$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.
```

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.
.
```

OpenACC Version 2 – Profile

```
Table 1:  Profile by Function Group and Function

 Time%  |     Time  |    Imb.   |  Imb.   |  Calls  |Group
        |           |    Time   | Time%   |         | Function
        |           |           |         |         |  PE=HIDE

 100.0% | 35.679931 |       -- |     -- | 36436.1 |Total
|-------------------------------------------------------------------------------
|  92.2% | 32.914300 |       -- |     -- | 19121.0 |USER
||------------------------------------------------------------------------------
||  34.9% | 12.445154 | 0.259110 |   2.1% |  1005.0 |jacobi_.ACC_COPY@li.303
||  31.5% | 11.256881 | 0.247093 |   2.2% |  1005.0 |jacobi_.ACC_COPY@li.297
||  17.1% |  6.089251 | 0.845903 |  12.4% |  1005.0 |jacobi_.ACC_SYNC_WAIT@li.277
||   8.2% |  2.916580 | 1.242691 |  30.4% |  1005.0 |jacobi_.ACC_SYNC_WAIT@li.296
||   0.2% |  0.076624 | 0.010203 |  11.9% |  1005.0 |jacobi_.ACC_ASYNC_KERNEL@li.248
||   0.1% |  0.049299 | 0.006400 |  11.7% |  1005.0 |jacobi_.ACC_ASYNC_KERNEL@li.281
||   0.1% |  0.019482 | 0.008660 |  31.3% |  1005.0 |jacobi_.ACC_COPY@li.248
||   0.0% |  0.009641 | 0.001109 |  10.5% |     2.0 |jacobi_.ACC_DATA_REGION@li.240
||   0.0% |  0.007366 | 0.000622 |   7.9% |  1005.0 |sendp3_
||   0.0% |  0.005379 | 0.000703 |  11.7% |  1005.0 |jacobi_.ACC_REGION@li.248
||   0.0% |  0.004826 | 0.000634 |  11.8% |  1005.0 |sendp1_
||   0.0% |  0.004560 | 0.000721 |  13.9% |  1005.0 |jacobi_.ACC_UPDATE@li.303
||   0.0% |  0.004302 | 0.000850 |  16.8% |  1005.0 |jacobi_.ACC_SYNC_WAIT@li.297
||   0.0% |  0.004042 | 0.000214 |   5.1% |  1005.0 |sendp_
||   0.0% |  0.003855 | 0.000379 |   9.1% |  1005.0 |sendp2_
.
```

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

OpenACC Version 3 – a)

```
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
#endif
#ifdef _OPENACC
!$ACC  parallel loop
!$ACC&   private (ii,j,k)
#else
!$OMP  parallel do
!$OMP&   private (ii,j,k)
!$OMP&   shared  (p,wrk2)
#endif
         DO K=1,kmax
            DO J=1,jmax
                ii = j + (k-1)*jmax
                  pack_p11(ii)=p(2,j,k)
                  pack_p12(ii)=p(imax-1,j,k)
            enddo
         enddo
#ifdef _OPENACC
!$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
         DO K=1,kmax
            DO I=1,imax
                jj = i +(k-1)*imax
                  pack_p21(jj)=p(i,2,k)
                  pack_p22(jj)=p(i,jmax-1,k)
            enddo
         enddo
#ifdef _OPENACC
!$ACC  end parallel loop
#endif
#ifdef _OPENACC
!$ACC  parallel loop
!$ACC&   private (i,j,kk)
#else
!$OMP  parallel do
!$OMP&   private (i,j,kk)
#endif
         DO J=1,jmax
            DO I=1,imax
                kk = i + (j-1)*imax
                  pack_p31(kk)=p(i,j,2)
                  pack_p32(kk)=p(i,j,kmax-1)
            enddo
         enddo
#ifdef _OPENACC
!$ACC  end parallel loop
#endif
#ifdef _OPENACC
!$acc update host(pack_p11,pack_p12,pack_p21,pack_p22,
!$acc&              pack_p31,pack_p32,wgosa)
#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)
C
C
```

OpenACC Version 3 – b)

```
#ifdef _OPENACC
!$acc update device(unpack_p11,unpack_p12,unpack_p21,unpack_p22,
!$acc&              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)
!$OMP&   shared  (p,wrk2)
#endif
         DO K=1,kmax
            DO J=1,jmax
                ii = j + (k-1)*jmax
                  p(2,j,k)=unpack_p11(ii)
                  p(imax-1,j,k)=unpack_p12(ii)
            enddo
         enddo
#ifdef _OPENACC
!$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
         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_p32(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
         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
!$ACC&   private (i,j,kk)
#else
!$OMP  parallel do
!$OMP&   private (i,j,kk)
#endif
         DO J=1,jmax
            DO I=1,imax
                kk = i +(j-1)*imax
                  p(i,j,2)=unpack_p31(kk)
                  p(i,j,kmax-1)=unpack_p32(kk)
            enddo
         enddo
#ifdef _OPENACC
!$ACC  end parallel loop
#endif
)
c
C
—
.
```

Comparisons of Himeno Versions

Version of Himeno	GFLOPS
Original running on 16 nodes 4 MPI/Node	78.75
OMP version1 running on 16 nodes 4 MPI/Node 4 threads/MPI task	169.5
OMP version2 running on 16 nodes 4 MPI/Node 4 threads/MPI task	180.9
OpenACC version 1 running on 16 nodes 4 MPI/Node 1 GPU	45.991
OpenACC version 2 running on 16 nodes 4 MPI/Node 1 GPU	256.9
OpenACC version 3 running on 16 nodes 4 MPI/Node 1 GPU	593.3
Last Version running in OpenMP mode	180.99

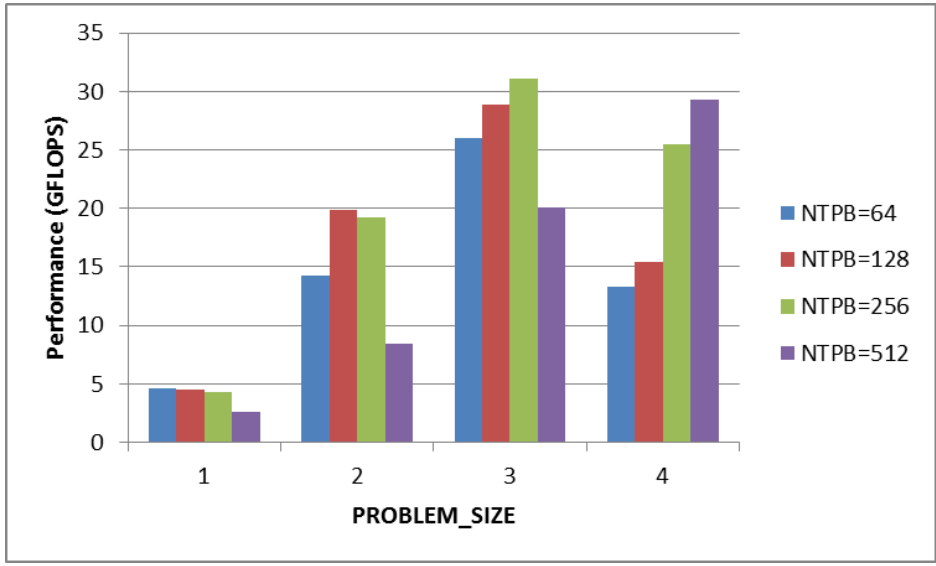
Not finished – need to consider

- Optimization of the kernel
- Optimization of the data movement

Scalar Himeno performance

- **PROBLEM_SIZE=1,2,3,4**
  - For a parallel problem, this simulates strong scaling choices
  - Here we look at all 4 options (all double precision)
- **vector_length choices: NTPB=64,128,256,512**
  - Easy tuning choice, but needs a recompile with CCE
    - but you need to put in vector_length(NTPB) clauses
    - and compile with -DNTPB=<value>
  - CCE defaults to 128

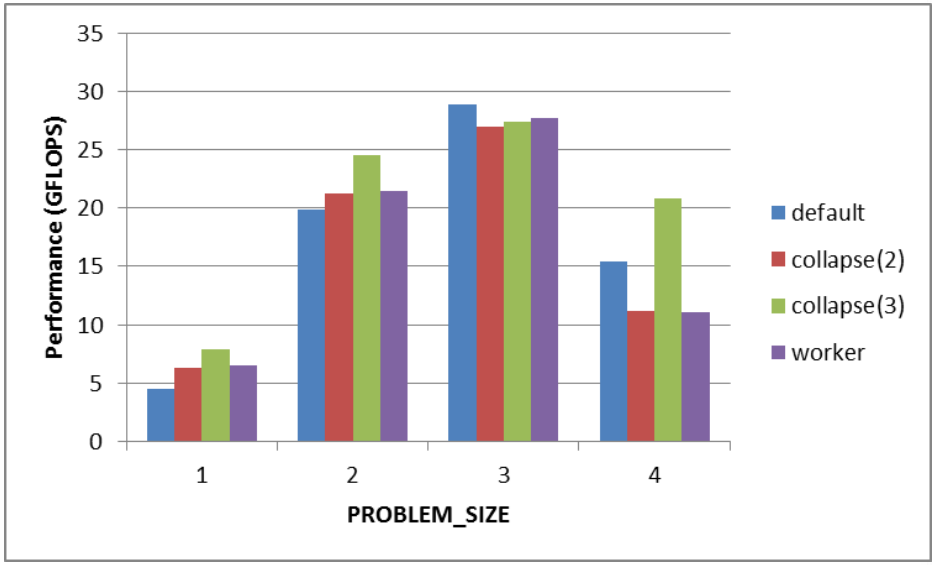
- **This has a BIG effect**
  - best NTPB relative to default:
  - PROBLEM_SIZE=1: + 2%
  - PROBLEM_SIZE=2: default
  - PROBLEM_SIZE=3: + 8%
  - PROBLEM_SIZE=4: +90%



Scalar Himeno performance

- **PROBLEM_SIZE=1,2,3,4**
- **4 algorithm choices for jacobi stencil kernel**
  - default scheduling
  - collapse(2) inner i,j-loops
  - collapse(3) all loops
  - worker schedule for j-loop
- **vector_length: default**

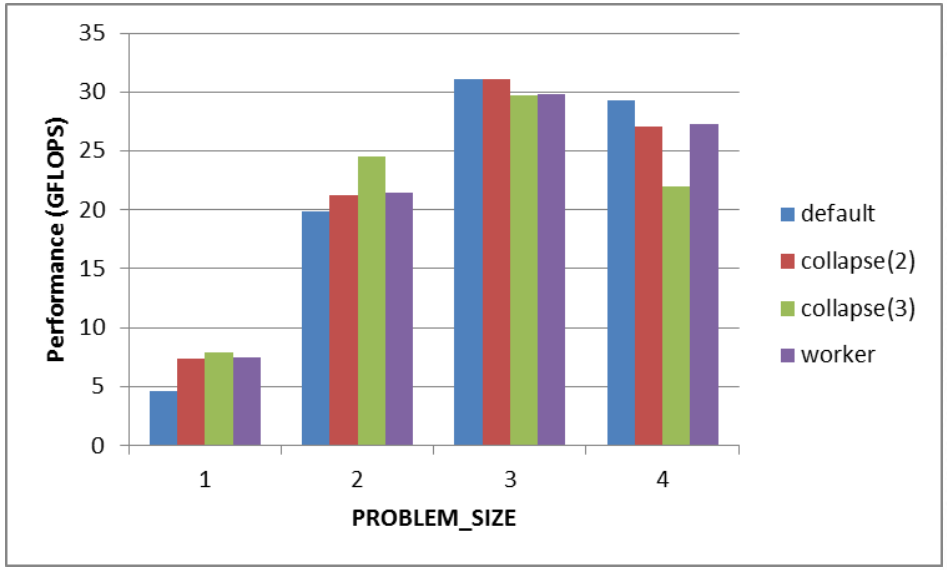
- **Effect also quite big**
  - best NTPB relative to default:
  - PROBLEM_SIZE=1: +76%
  - PROBLEM_SIZE=2: +23%
  - PROBLEM_SIZE=3: default
  - PROBLEM_SIZE=4: +35%



Scalar Himeno performance

- **PROBLEM_SIZE=1,2,3,4**
- **4 algorithm choices for jacobi stencil kernel**
  - default scheduling
  - collapse(2) inner i,j-loops
  - collapse(3) all loops
  - worker schedule for j-loop
- **vector_length: best for each**

- **Final improvements**
  - compared to default
    - scheduling and vector_length:
  - PROBLEM_SIZE=1: +76%
  - PROBLEM_SIZE=2: +23%
  - PROBLEM_SIZE=3: + 8%
  - PROBLEM_SIZE=4: +90%



The initial naïve approach

```
209.  + 1 G----------< !$ACC  parallel loop
210.    1 G           !$ACC&   private (s0,ss)
211.    1 G           !$ACC&   reduction (+:wgosa)
212.    1 G g--------<         DO K=2,kmax-1
213.  + 1 G g 4------<             DO J=2,jmax-1
214.    1 G g 4 g----<                DO I=2,imax-1
215.    1 G g 4 g                        S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
216.    1 G g 4 g          1                  +a(I,J,K,3)*p(I,J,K+1)
217.    1 G g 4 g          2                  +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
218.    1 G g 4 g          3                  -p(I-1,J+1,K)+p(I-1,J-1,K))
219.    1 G g 4 g          4                  +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
220.    1 G g 4 g          5                  -p(I,J+1,K-1)+p(I,J-1,K-1))
221.    1 G g 4 g          6                  +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
222.    1 G g 4 g          7                  -p(I+1,J,K-1)+p(I-1,J,K-1))
223.    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)
224.    1 G g 4 g          9                  +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
225.    1 G g 4 g                        SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
226.    1 G g 4 g                        WGOSA=WGOSA+SS*SS
227.    1 G g 4 g                        wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
228.    1 G g 4 g---->                enddo
229.    1 G g 4------>             enddo
230.    1 G g------->>           enddo
```

Parallelizing all three loops

```
209.  + 1 G----------< !$ACC  parallel loop gang
210.    1 G           !$ACC&   private (s0,ss)
211.    1 G           !$ACC&   reduction (+:wgosa)
212.    1 G g--------<          DO K=2,kmax-1
213.    1 G g         !$ACC  loop worker
214.    1 G g         !$ACC&   private (s0,ss)
215.    1 G g         !$ACC&   reduction (+:wgosa)
216.    1 G g g------<             DO J=2,jmax-1
217.    1 G g g       !$ACC  loop vector
218.    1 G g g       !$ACC&   private (s0,ss)
219.    1 G g g       !$ACC&   reduction (+:wgosa)
220.    1 G g g g----<                 DO I=2,imax-1
221.    1 G g g g                         S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
222.    1 G g g g          1                   +a(I,J,K,3)*p(I,J,K+1)
223.    1 G g g g          2                   +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
224.    1 G g g g          3                   -p(I-1,J+1,K)+p(I-1,J-1,K))
225.    1 G g g g          4                   +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
226.    1 G g g g          5                   -p(I,J+1,K-1)+p(I,J-1,K-1))
227.    1 G g g g          6                   +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
228.    1 G g g g          7                   -p(I+1,J,K-1)+p(I-1,J,K-1))
229.    1 G g g g          8                   +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
230.    1 G g g g          9                   +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
231.    1 G g g g                         SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
232.    1 G g g g                         WGOSA=WGOSA+SS*SS
233.    1 G g g g                         wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
234.    1 G g g g----->                 enddo
235.    1 G g g------>                enddo
236.    1 G g------->>              enddo
```

Using the collapse clause

```
209.  + 1 G----------< !$ACC  parallel loop collapse(3)
210.    1 G           !$ACC&   private (s0,ss)
211.    1 G           !$ACC&   reduction (+:wgosa)
212.    1 G C--------<         DO K=2,kmax-1
213.    1 G C C------<             DO J=2,jmax-1
214.  + 1 G C C g----<                 DO I=2,imax-1
215.    1 G C C g                       S0=a(I,J,K,1)*p(I+1,J,K)+a(I,J,K,2)*p(I,J+1,K)
216.    1 G C C g           1                +a(I,J,K,3)*p(I,J,K+1)
217.    1 G C C g           2                +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)
218.    1 G C C g           3                -p(I-1,J+1,K)+p(I-1,J-1,K))
219.    1 G C C g           4                +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)
220.    1 G C C g           5                -p(I,J+1,K-1)+p(I,J-1,K-1))
221.    1 G C C g           6                +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)
222.    1 G C C g           7                -p(I+1,J,K-1)+p(I-1,J,K-1))
223.    1 G C C g           8                +c(I,J,K,1)*p(I-1,J,K)+c(I,J,K,2)*p(I,J-1,K)
224.    1 G C C g           9                +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
225.    1 G C C g                       SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
226.    1 G C C g                       WGOSA=WGOSA+SS*SS
227.    1 G C C g                       wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
228.    1 G C C g---->                enddo
229.    1 G C C------>              enddo
230.    1 G C------->>            enddo
```

Comparisons of Himeno Versions

Version of Himeno	Major Loop Time
Naïve Approach	0.925593
Parallelizing all three loops	0.936273
Using Collapse	0.616912

Each of the three loop lengths are 128

Scalar Himeno tuning conclusions

- **Tuning can have a big effect, relative to default**
  - It is worth doing, but only after you optimise the data locality
    - data region gave 44x speedup; kernel tuning gave less than 2x
  - We gained something at every problem size (don't reject a mere 2x!)
  - Only explored vector_length and basic scheduling (collapse, worker)
  - The only change was OpenACC directives; CPU version same

- **General conclusions for scalar Himeno:**
  - Larger vector_length suited larger problem sizes
    - 64 best for PS1; 128 best for PS2; 256 best for PS3; 512 best for PS4
  - Once we had optimised vector_length
    - collapse(3) was best for small problems (PS1, PS2)
    - default scheduling was best for large problems (PS3, PS4)

- **What else could we try?**
  - cache clause: could have benefits for the expensive stencil kernel
  - async clause: scalar Himeno is too simple for task-based overlap
    - async is important in the parallel case

OpenACC async clause

- **async[(handle)] clause for parallel, update directives**
  - Launch accelerator region/data transfer asynchronously
  - Operations with same handle guaranteed to execute sequentially
    - as for CUDA streams
  - Operations with different handles can overlap
    - if the hardware permits it and runtime chooses to schedule it:
    - can potentially overlap:
      - PCIe transfers in both directions
      - Plus multiple kernels
    - can overlap up to 16 parallel streams with Fermi
  - streams identified by handle (integer-valued)
    - tasks with same handle execute sequentially
    - can wait on one, more or all tasks

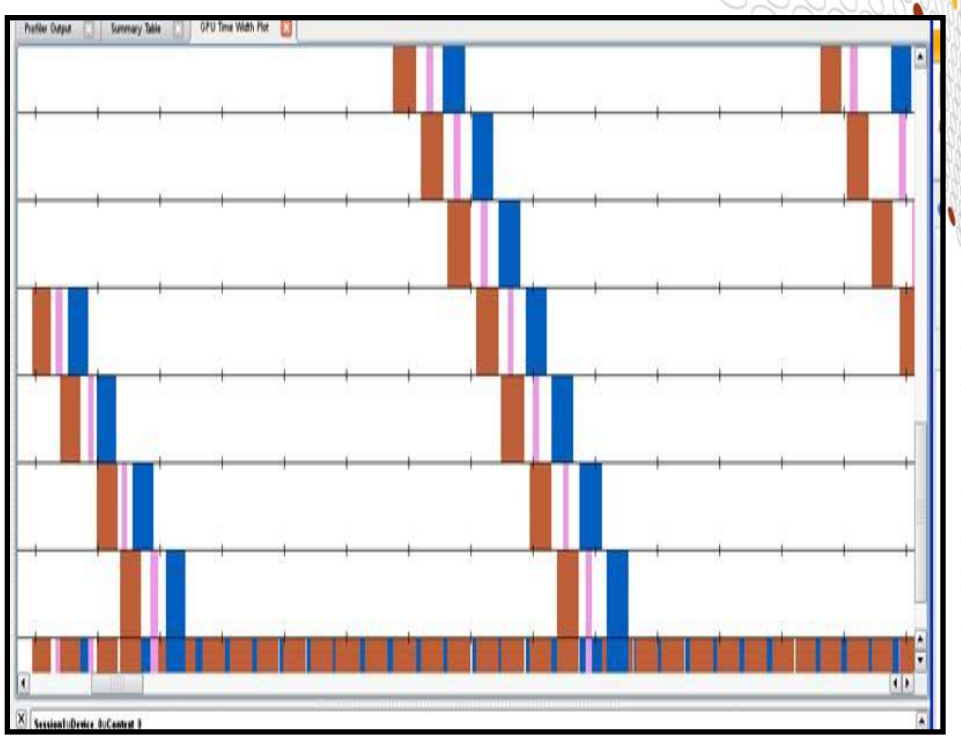
- **!$acc wait: waits for completion of all streams of tasks**
  - !$acc wait(handle) waits for a specified stream to complete
- **Runtime API library functions**
  - can also be used to wait or test for completion

OpenACC async clause

- **First attempt**
  - a simple pipeline:
  - processes array, slice by slice
    - copy data to GPU,
    - process on GPU,
    - bring back to CPU
  - can overlap 3 streams at once
    - use slice number as stream handle
      - don't worry if number gets too large
      - OpenACC runtime maps it back into allowable range (using MOD function)

```
REAL(kind=dp) ::
a(Nvec,Nchunks),b(Nvec,Nchunks)

!$acc data create(a,b)
DO j = 1,Nchunks
!$acc update device(a(:,j)) async(j)

!$acc parallel loop async(j)
  DO i = 1,Nvec
    b(i,j) = <function of a(i,j)>
  ENDDO

!$acc update host(b(:,j)) async(j)

ENDDO
!$acc wait
!$acc end data
```

OpenACC async results



- **Execution times (on Cray XK6):**
  - CPU: 3.76s
  - OpenACC, blocking: 1.10s
  - OpenACC, async: 0.34s
- **NVIDIA Visual profiler:**
  - time flows left to right
  - streams stacked vertically
    - only 7 of 16 streams fit in window
    - **red**: data transfer to GPU
    - **pink**: computational on GPU
    - **blue**: data transfer from GPU
  - vertical slice shows what is overlapping
    - collapsed view at bottom
  - async handle modded by number of streams
    - so see multiple coloured bars per stream (looking horizontally)
- **Alternative to pipelining is task-based overlap**
  - Harder to arrange; needs knowledge of data flow in specific application
  - May (probably will) require application restructuring (maybe helps CPU)
  - Some results later in Himeno Case Study

!$acc host_data use_device

```
#ifdef GPU
!$acc data present(f)
!$acc host_data use_device(f)
#endif
  if( deriv_z_list(idx)%packed ) then
    deriv_z_list(idx)%packed = .false.
    if(deriv_z_list(idx)%neg_nbr>=0) then
       call MPI_ISend(f(1,1,1),(mx*my*iorder/2),&
                      MPI_REAL8,deriv_z_list(idx)%neg_nbr,deriv_list_size + idx, &
                      gcomm,deriv_z_list(idx)%req(2),ierr)
    endif
    if(deriv_z_list(idx)%pos_nbr>=0) then
     ! send ghost cells to neighbor on (+z) side
       nm = mz + 1 - iorder/2
       call MPI_ISend(f(1,1,nm),(mx*my*iorder/2), &
                      MPI_REAL8,deriv_z_list(idx)%pos_nbr,idx, &
                      gcomm,deriv_z_list(idx)%req(4),ierr)
    endif
  else
    if(deriv_z_list(idx)%neg_nbr>=0) then
       call MPI_ISend(f(1,1,1),(mx*my*iorder/2),&
                      MPI_REAL8,deriv_z_list(idx)%neg_nbr,deriv_list_size + idx, &
                      gcomm,deriv_z_list(idx)%req(2),ierr)
    endif

    if(deriv_z_list(idx)%pos_nbr>=0) then
     ! send ghost cells to neighbor on (+z) side
       nm = mz + 1 - iorder/2
       call MPI_ISend(f(1,1,nm),(mx*my*iorder/2), &
                      MPI_REAL8,deriv_z_list(idx)%pos_nbr,idx, &
                      gcomm,deriv_z_list(idx)%req(4),ierr)
    endif
  endif
#ifdef GPU
!$acc end host_data
!$acc end data
#endif
```

User in Denmark, mentioned in Jack's talk is using MPI-3 with use_device and getting excellent communication performance.

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
```

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 elminating transfers to and from the host.**

Strategy for refactoring the application

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

There is Vectorization and then there is good Vectorization

```
DIRECTION: do m=1,3
    SPECIES: do n=1,n_spec-1

        ! driving force is just the gradient in mole fraction:
        diffFlux(:,:,:,n,m) = - Ds_mixavg(:,:,:,n) * ( grad_Ys(:,:,:,n,m)  &
                              + Ys(:,:,:,n) * grad_mixMW(:,:,:,m) )

        diffFlux(:,:,:,n_spec,m) = diffFlux(:,:,:,n_spec,m) - diffFlux(:,:,:,n,m)

    enddo SPECIES
enddo DIRECTION
```

This Vectorizes and it runs very slow

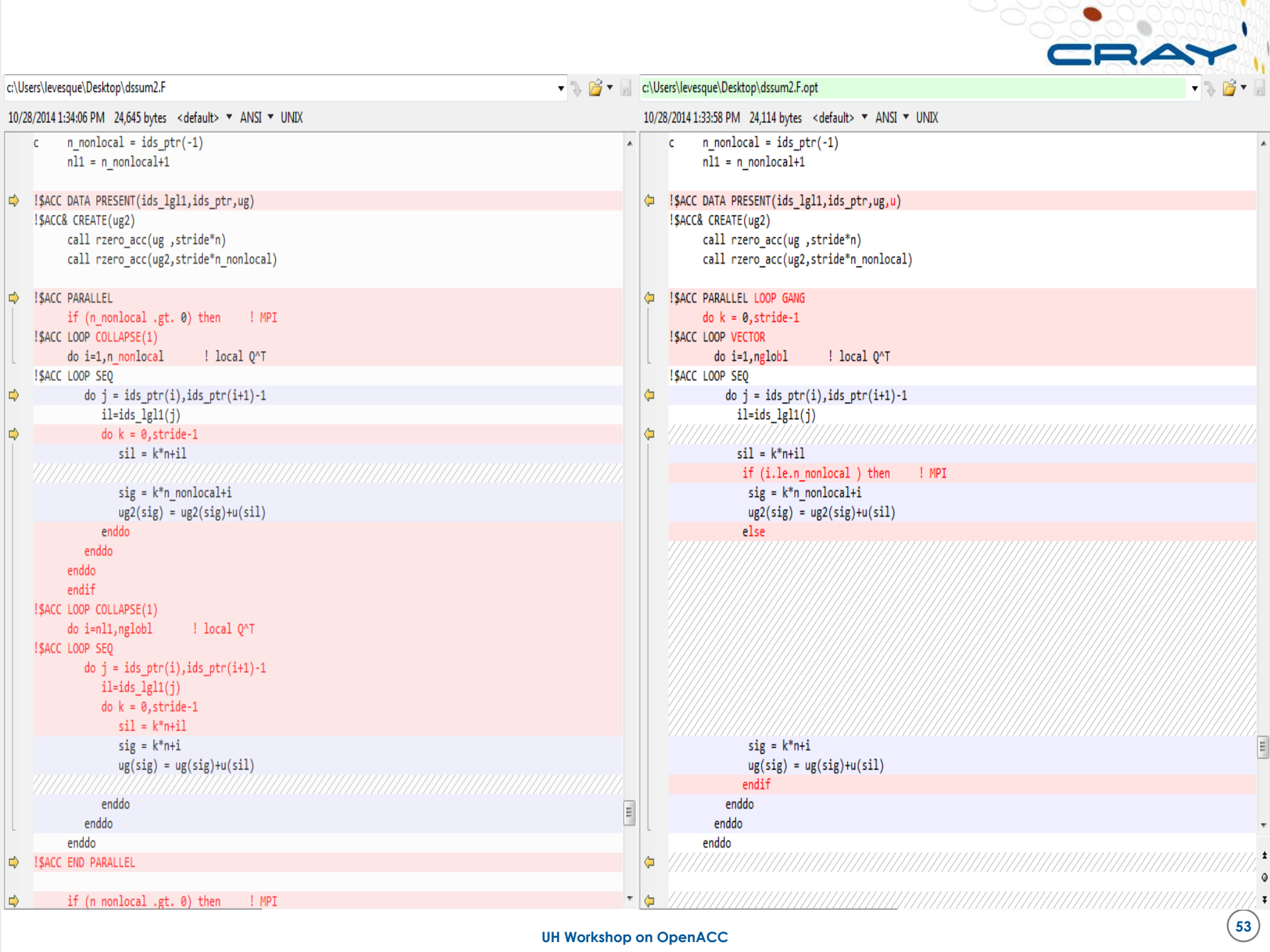
There is Vectorization and then there is good Vectorization

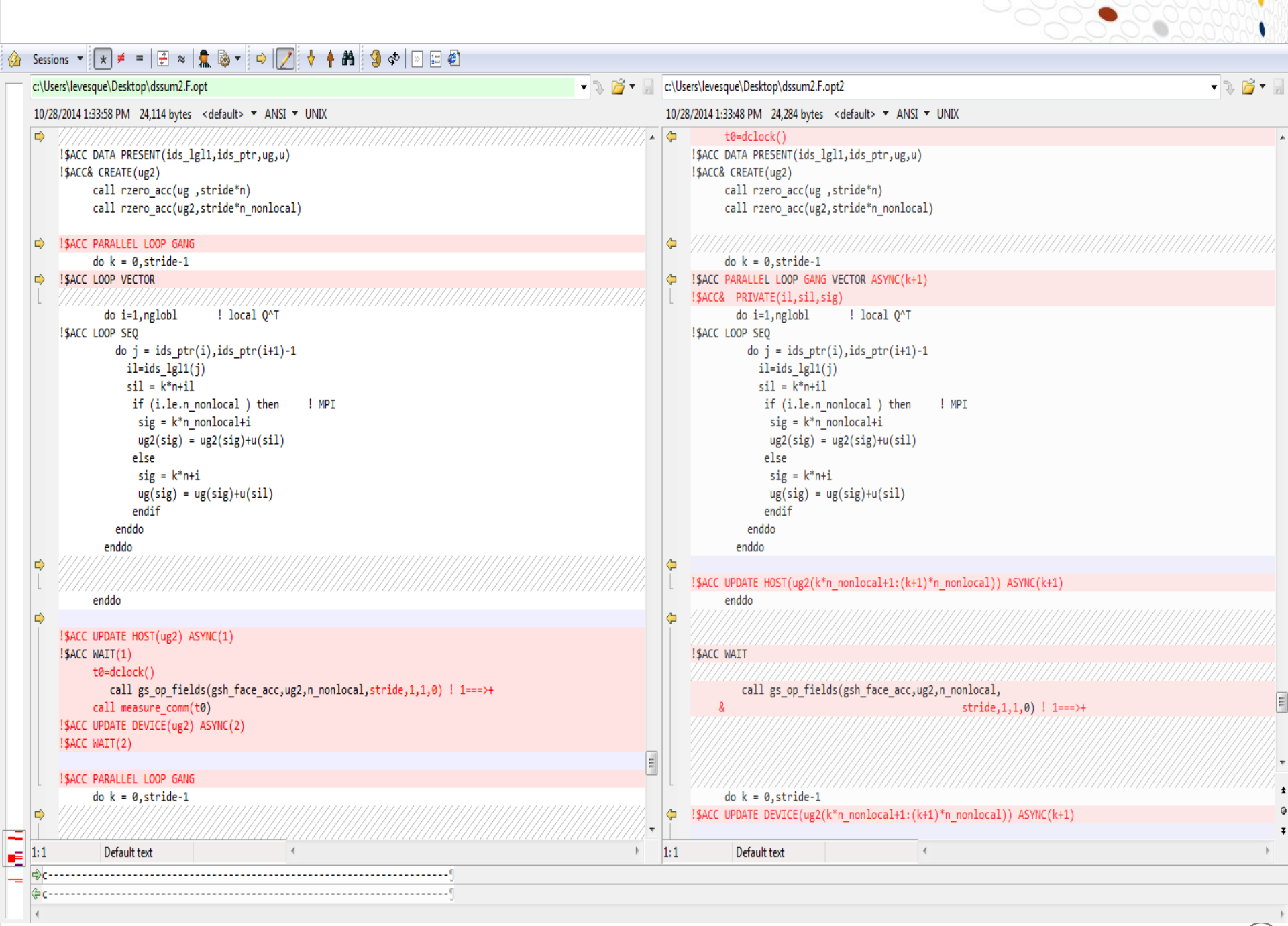
This Vectorizes and it runs very fast

```
do n=1,n_spec-1
      do k=1,nz
      do j=1,ny
      do i=1,nx
         diffFlux(i,j,k,1,n) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,1,n)  &
                             + Ys(i,j,k,n) * ( grad_mixMW(i,j,k,1)    &
                             + (1 - molwt(n)*avmolwt(i,j,k)) * grad_P(i,j,k,1)/Press(i,j,k)))
         diffFlux(i,j,k,2,n) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,2,n)  &
                             + Ys(i,j,k,n) * ( grad_mixMW(i,j,k,2)    &
                             + (1 - molwt(n)*avmolwt(i,j,k)) * grad_P(i,j,k,2)/Press(i,j,k)))
         diffFlux(i,j,k,3,n) = - Ds_mixavg(i,j,k,n) * ( grad_Ys(i,j,k,3,n)  &
                             + Ys(i,j,k,n) * ( grad_mixMW(i,j,k,3)    &
                             + (1 - molwt(n)*avmolwt(i,j,k)) * grad_P(i,j,k,3)/Press(i,j,k)))
         diffFlux(i,j,k,1,n) = diffFlux(i,j,k,1,n)    &
            - Ds_mixavg(i,j,k,n) * Rs_therm_diff(i,j,k,n) * molwt(n) &
            * avmolwt(i,j,k) * grad_T(i,j,k,1) / Temp(i,j,k)
         diffFlux(i,j,k,2,n) = diffFlux(i,j,k,2,n)    &
            - Ds_mixavg(i,j,k,n) * Rs_therm_diff(i,j,k,n) * molwt(n) &
            * avmolwt(i,j,k) * grad_T(i,j,k,2) / Temp(i,j,k)
         diffFlux(i,j,k,3,n) = diffFlux(i,j,k,3,n)    &
            - Ds_mixavg(i,j,k,n) * Rs_therm_diff(i,j,k,n) * molwt(n) &
            * avmolwt(i,j,k) * grad_T(i,j,k,3) / Temp(i,j,k)
       diffFlux(i,j,k,1,n_spec) = diffFlux(i,j,k,1,n_spec) - diffFlux(i,j,k,1,n)
       diffFlux(i,j,k,2,n_spec) = diffFlux(i,j,k,2,n_spec) - diffFlux(i,j,k,2,n)
       diffFlux(i,j,k,3,n_spec) = diffFlux(i,j,k,3,n_spec) - diffFlux(i,j,k,3,n)
    enddo
    enddo
    enddo
    enddo
```

Table 5: Time and Bytes Transferred for Accelerator Regions

Host Time%	Host Time	Acc Time	Acc Copy In (MBytes)	Acc Copy Out (MBytes)	Events	Calltree Thread=HIDE
100.0%	114.753	155.234	158873	154495	802662	Total
100.0%	114.753	155.234	158873	154495	802662	nekton_
						cem_solve_
						cem_solver_
						time_advancing_pde_
						time_advancing_pde_.ACC_DATA_REGION@li.931
97.9%	112.309	152.970	154495	154495	780000	cem_
						cem_maxwell_op_rk_acc_
97.0%	111.304	149.856	154495	154495	600000	cem_maxwell_op_acc_
94.5%	108.394	128.471	154495	154495	225000	cem_maxwell_flux_acc_
						cem_maxwell_flux3d_acc_
						cem_maxwell_flux3d_acc_.ACC_DATA_REGION@li.421
93.8%	107.681	126.929	154495	154495	175000	gs_op_fields_acc_
						gs_op_fields_acc_.ACC_DATA_REGION@li.886
29.1%	33.390	12.933	--	--	30000	gs_op_fields_acc_.ACC_REGION@li.891
28.7%	32.890	--	--	--	5000	gs_op_fields_acc_.ACC_SYNC_WAIT@li.918
0.4%	0.459	12.918	--	--	10000	gs_op_fields_acc_.ACC_ASYNC_KERNEL@li.891
0.0%	0.025	--	--	--	5000	gs_op_fields_acc_.ACC_REGION@li.891(exclusive)
0.0%	0.008	0.008	--	--	5000	gs_op_fields_acc_.ACC_COPY@li.918
0.0%	0.008	0.008	--	--	5000	gs_op_fields_acc_.ACC_COPY@li.891
27.8%	31.850	63.034	154495	--	15000	gs_op_fields_acc_.ACC_UPDATE@li.925
27.7%	31.832	63.034	154495	--	10000	gs_op_fields_acc_.ACC_COPY@li.925
0.0%	0.018	--	--	--	5000	gs_op_fields_acc_.ACC_UPDATE@li.925(exclusive)
17.2%	19.773	39.150	--	154495	20000	gs_op_fields_acc_.ACC_UPDATE@li.921
17.2%	19.715	39.150	--	154495	10000	gs_op_fields_acc_.ACC_COPY@li.921
0.0%	0.041	--	--	--	5000	gs_op_fields_acc_.ACC_SYNC_WAIT@li.921
0.0%	0.016	--	--	--	5000	gs_op_fields_acc_.ACC_UPDATE@li.921(exclusive)


c:\Users\levesque\Desktop\dssum2.F
10/28/2014 1:34:06 PM 24,645 bytes <default> ANSI UNIX
c n_nonlocal = ids_ptr(-1)
nl1 = n_nonlocal+1
!$ACC DATA PRESENT(ids_lgl1,ids_ptr,ug)
!$ACC& CREATE(ug2)
call rzero_acc(ug ,stride*n)
call rzero_acc(ug2,stride*n_nonlocal)
!$ACC PARALLEL
if (n_nonlocal .gt. 0) then ! MPI
!$ACC LOOP COLLAPSE(1)
do i=1,n_nonlocal ! local Q^T
!$ACC LOOP SEQ
do j = ids_ptr(i),ids_ptr(i+1)-1
il=ids_lgl1(j)
do k = 0,stride-1
sil = k*n+il
sig = k*n_nonlocal+i
ug2(sig) = ug2(sig)+u(sil)
enddo
enddo
enddo
endif
!$ACC LOOP COLLAPSE(1)
do i=nl1,nglobl ! local Q^T
!$ACC LOOP SEQ
do j = ids_ptr(i),ids_ptr(i+1)-1
il=ids_lgl1(j)
do k = 0,stride-1
sil = k*n+il
sig = k*n+i
ug(sig) = ug(sig)+u(sil)
enddo
enddo
enddo
!$ACC END PARALLEL
if (n nonlocal .gt. 0) then ! MPI
c:\Users\levesque\Desktop\dssum2.F.opt
10/28/2014 1:33:58 PM 24,114 bytes <default> ANSI UNIX
c n_nonlocal = ids_ptr(-1)
nl1 = n_nonlocal+1
!$ACC DATA PRESENT(ids_lgl1,ids_ptr,ug,u)
!$ACC& CREATE(ug2)
call rzero_acc(ug ,stride*n)
call rzero_acc(ug2,stride*n_nonlocal)
!$ACC PARALLEL LOOP GANG
do k = 0,stride-1
!$ACC LOOP VECTOR
do i=1,nglobl ! local Q^T
!$ACC LOOP SEQ
do j = ids_ptr(i),ids_ptr(i+1)-1
il=ids_lgl1(j)
sil = k*n+il
if (i.le.n_nonlocal) then ! MPI
sig = k*n_nonlocal+i
ug2(sig) = ug2(sig)+u(sil)
else
sig = k*n+i
ug(sig) = ug(sig)+u(sil)
endif
enddo
enddo
enddo


Sessions
c:\Users\levesque\Desktop\dssum2.F.opt
10/28/2014 1:33:58 PM 24,114 bytes <default> ANSI UNIX
!$ACC DATA PRESENT(ids_lgl1,ids_ptr,ug,u)
!$ACC& CREATE(ug2)
call rzero_acc(ug ,stride*n)
call rzero_acc(ug2,stride*n_nonlocal)
!$ACC PARALLEL LOOP GANG
do k = 0,stride-1
!$ACC LOOP VECTOR
do i=1,nglobl ! local Q^T
!$ACC LOOP SEQ
do j = ids_ptr(i),ids_ptr(i+1)-1
il=ids_lgl1(j)
sil = k*n+il
if (i.le.n_nonlocal) then ! MPI
sig = k*n_nonlocal+i
ug2(sig) = ug2(sig)+u(sil)
else
sig = k*n+i
ug(sig) = ug(sig)+u(sil)
endif
enddo
enddo
enddo
!$ACC UPDATE HOST(ug2) ASYNC(1)
!$ACC WAIT(1)
t0=dclock()
call gs_op_fields(gsh_face_acc,ug2,n_nonlocal,stride,1,1,0) ! 1===>+
call measure_comm(t0)
!$ACC UPDATE DEVICE(ug2) ASYNC(2)
!$ACC WAIT(2)
!$ACC PARALLEL LOOP GANG
do k = 0,stride-1
1:1 Default text
c:\Users\levesque\Desktop\dssum2.F.opt2
10/28/2014 1:33:48 PM 24,284 bytes <default> ANSI UNIX
t0=dclock()
!$ACC DATA PRESENT(ids_lgl1,ids_ptr,ug,u)
!$ACC& CREATE(ug2)
call rzero_acc(ug ,stride*n)
call rzero_acc(ug2,stride*n_nonlocal)
do k = 0,stride-1
!$ACC PARALLEL LOOP GANG VECTOR ASYNC(k+1)
!$ACC& PRIVATE(il,sil,sig)
do i=1,nglobl ! local Q^T
!$ACC LOOP SEQ
do j = ids_ptr(i),ids_ptr(i+1)-1
il=ids_lgl1(j)
sil = k*n+il
if (i.le.n_nonlocal) then ! MPI
sig = k*n_nonlocal+i
ug2(sig) = ug2(sig)+u(sil)
else
sig = k*n+i
ug(sig) = ug(sig)+u(sil)
endif
enddo
enddo
!$ACC UPDATE HOST(ug2(k*n_nonlocal+1:(k+1)*n_nonlocal)) ASYNC(k+1)
enddo
!$ACC WAIT
call gs_op_fields(gsh_face_acc,ug2,n_nonlocal,
& stride,1,1,0) ! 1===>+
do k = 0,stride-1
!$ACC UPDATE DEVICE(ug2(k*n_nonlocal+1:(k+1)*n_nonlocal)) ASYNC(k+1)
1:1 Default text

```
!$ACC PARALLEL LOOP COLLAPSE(4) GANG WORKER VECTOR
!$ACC&     private(tmpr1,tmpr2,tmpr3)
!dir$ NOBLOCKING
      do e = 1,nelt
         do k = 1,nz1
         do j = 1,ny1
         do i = 1,nx1
            tmpr1 = 0.0
            tmpr2 = 0.0
            tmpr3 = 0.0
!$ACC LOOP SEQ
            do l=1,nx1
               tmpr1=tmpr1+d(i,l)*u1(l,j,k,e)
               tmpr2=tmpr2+d(i,l)*u2(l,j,k,e)
               tmpr3=tmpr3+d(i,l)*u3(l,j,k,e)
            enddo
            u1r(i,j,k,e) = tmpr1
            u2r(i,j,k,e) = tmpr2
            u3r(i,j,k,e) = tmpr3
         enddo
         enddo
         enddo
      enddo
!$ACC END PARALLEL LOOP
!$ACC PARALLEL LOOP COLLAPSE(4) GANG WORKER VECTOR
!$ACC&     private(tmps1,tmps2,tmps3)
!dir$ NOBLOCKING
      do e = 1,nelt
         do k = 1,nz1
         do j = 1,ny1
         do i = 1,nx1
            tmps1 = 0.0
            tmps2 = 0.0
            tmps3 = 0.0
!$ACC LOOP SEQ
            do l=1,nx1
               tmps1=tmps1+d(j,l)*u1(i,l,k,e)
               tmps2=tmps2+d(j,l)*u2(i,l,k,e)
               tmps3=tmps3+d(j,l)*u3(i,l,k,e)
            enddo
            u1s(i,j,k,e) = tmps1
            u2s(i,j,k,e) = tmps2
            u3s(i,j,k,e) = tmps3
         enddo
         enddo
         enddo
```

Another triple nested loop

```
!$ACC PARALLEL LOOP COLLAPSE(4) GANG WORKER VECTOR
!$ACC&     private(tmpr1,tmpr2,tmpr3,tmps1,tmps2,tmps3,
!$ACC&             tmpt1,tmpt2,tmpt3)
!dir$ NOBLOCKING
      do e = 1,nelt
         do k = 1,nz1
         do j = 1,ny1
         do i = 1,nx1
            tmpr1 = 0.0
            tmpr2 = 0.0
            tmpr3 = 0.0
            tmps1 = 0.0
            tmps2 = 0.0
            tmps3 = 0.0
            tmpt1 = 0.0
            tmpt2 = 0.0
            tmpt3 = 0.0
!$ACC LOOP SEQ
            do l=1,nx1
               tmpr1=tmpr1+d(i,l)*u1(l,j,k,e)
               tmpr2=tmpr2+d(i,l)*u2(l,j,k,e)
               tmpr3=tmpr3+d(i,l)*u3(l,j,k,e)
               tmps1=tmps1+d(j,l)*u1(i,l,k,e)
               tmps2=tmps2+d(j,l)*u2(i,l,k,e)
               tmps3=tmps3+d(j,l)*u3(i,l,k,e)
               tmpt1=tmpt1+d(k,l)*u1(i,j,l,e)
               tmpt2=tmpt2+d(k,l)*u2(i,j,l,e)
               tmpt3=tmpt3+d(k,l)*u3(i,j,l,e)
            enddo
            u1r(i,j,k,e) = tmpr1
            u2r(i,j,k,e) = tmpr2
            u3r(i,j,k,e) = tmpr3
            u1s(i,j,k,e) = tmps1
            u2s(i,j,k,e) = tmps2
            u3s(i,j,k,e) = tmps3
            u1t(i,j,k,e) = tmpt1
            u2t(i,j,k,e) = tmpt2
            u3t(i,j,k,e) = tmpt3
         enddo
         enddo
         enddo
      enddo
!$ACC END PARALLEL LOOP
```

```
 Host |   Host |    Acc | Acc Copy | Acc Copy |  Events |Calltree
  Time% |   Time |   Time |       In |      Out |         | Thread=HIDE
        |        |        | (MBytes) | (MBytes) |         |

 100.0% | 43.875 | 38.897 |    19827 |    15450 | 1077662 |Total
|-------------------------------------------------------------------------------------------------------------------------------
| 100.0% | 43.875 | 38.897 |    19827 |    15450 | 1077662 |nekton_
|        |        |        |          |          |         | cem_solve_
3        |        |        |          |          |         |  cem_solver_
4        |        |        |          |          |         |   time_advancing_pde_
5        |        |        |          |          |         |    time_advancing_pde_.ACC_DATA_REGION@li.931
||||||-------------------------------------------------------------------------------------------------------------------------
6|||||  95.7% | 41.993 | 37.662 |    15450 |    15450 | 1055000 |cem_
7|||||        |        |        |          |          |         | cem_maxwell_op_rk_acc_
|||||||||-------------------------------------------------------------------------------------------------------------------------
8|||||||  93.5% | 41.006 | 34.509 |    15450 |    15450 |  875000 |cem_maxwell_op_acc_
|||||||||-------------------------------------------------------------------------------------------------------------------------
9|||||||  86.7% | 38.038 | 12.498 |    15450 |    15450 |  500000 |cem_maxwell_flux_acc_
10|||||||        |        |        |          |          |         | cem_maxwell_flux3d_acc_
11|||||||        |        |        |          |          |         |  cem_maxwell_flux3d_acc_.ACC_DATA_REGION@li.421
|||||||||||-------------------------------------------------------------------------------------------------------------------------
12||||||||||  85.1% | 37.353 | 10.869 |    15450 |    15450 |  450000 |gs_op_fields_acc_
13||||||||||        |        |        |          |          |         | gs_op_fields_acc_.ACC_DATA_REGION@li.887
||||||||||||-------------------------------------------------------------------------------------------------------------------------
14|||||||||||  24.5% | 10.730 |     -- |       -- |       -- |    5000 |gs_op_fields_acc_.ACC_DATA_REGION@li.887(exclusive)
14|||||||||||  21.5% |  9.450 |  3.993 |       -- |    15450 |   90000 |gs_op_fields_acc_.ACC_UPDATE@li.910
||||||||||||-------------------------------------------------------------------------------------------------------------------------
15|||||||||||  21.4% |  9.397 |  3.993 |       -- |    15450 |   60000 |gs_op_fields_acc_.ACC_ASYNC_COPY@li.910
15|||||||||||   0.1% |  0.053 |     -- |       -- |       -- |   30000 |gs_op_fields_acc_.ACC_UPDATE@li.910(exclusive)
||||||||||||=========================================================================================================================
14|||||||||||  21.2% |  9.284 |  3.005 |       -- |       -- |   90000 |gs_op_fields_acc_.ACC_REGION@li.893
||||||||||||-------------------------------------------------------------------------------------------------------------------------
15|||||||||||  21.0% |  9.231 |  3.005 |       -- |       -- |   60000 |gs_op_fields_acc_.ACC_ASYNC_KERNEL@li.893
15|||||||||||   0.1% |  0.053 |     -- |       -- |       -- |   30000 |gs_op_fields_acc_.ACC_REGION@li.893(exclusive)
||||||||||||=========================================================================================================================
14|||||||||||  10.3% |  4.524 |     -- |       -- |       -- |    5000 |gs_op_fields_acc_.ACC_SYNC_WAIT@li.912
14|||||||||||   2.1% |  0.942 |  0.602 |       -- |       -- |   60000 |rzero_acc_
15|||||||||||        |        |        |          |          |         | rzero_acc_.ACC_DATA_REGION@li.2378
```

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

Implementation of COSMO on Accelerators

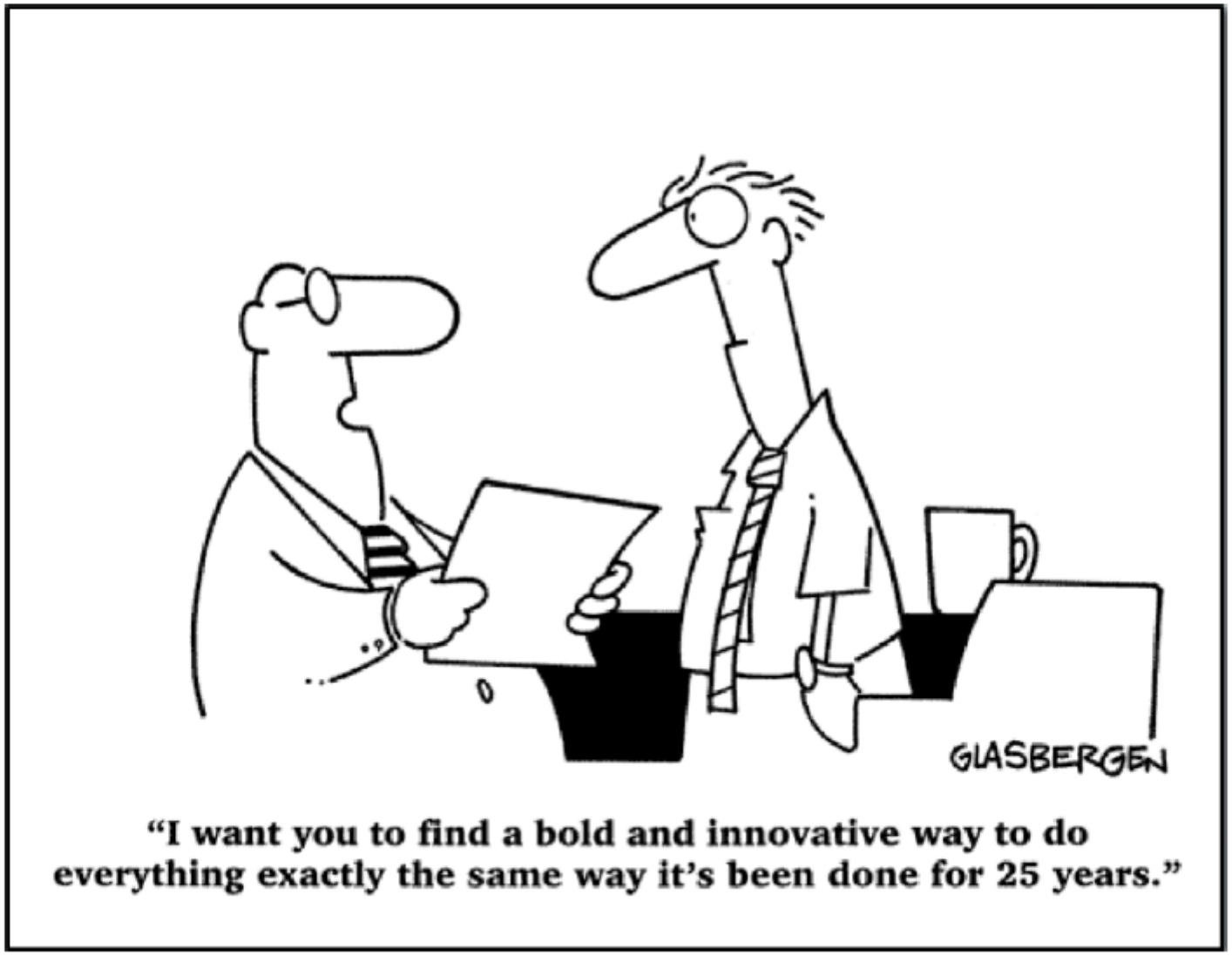
Oliver Fuhrer[1], Tobias Gysi[2], Carlos Osuna[3], Xavier Lapillonne[3], Mauro Bianco[4], Thomas Schulthess[4], et al.

HP2C

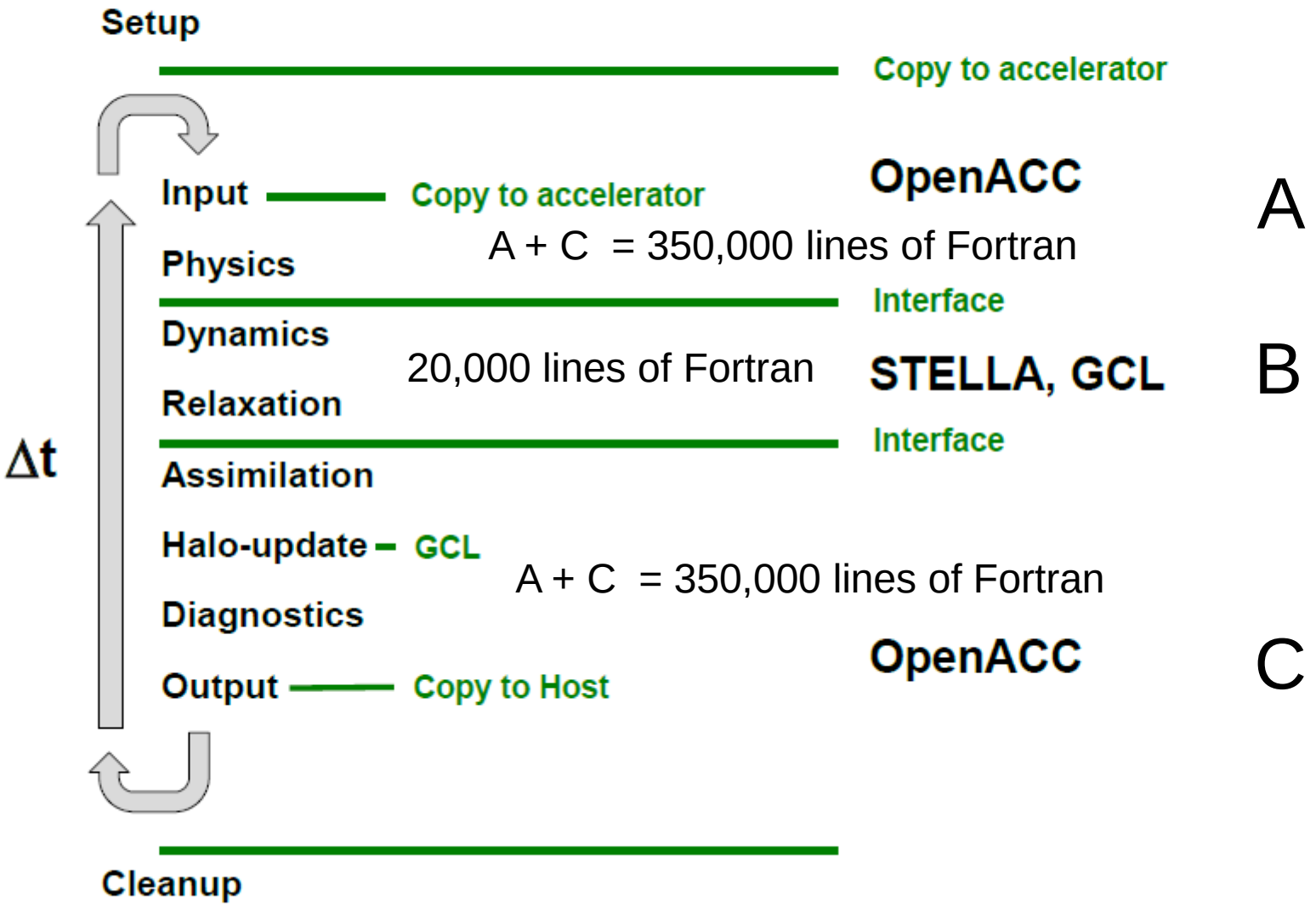
pasc
Platform for Advanced Scientific Computing

[1] Federal Office of Meteorology and Climatology MeteoSwiss, Switzerland
[2] Supercomputing Systems AG, Switzerland
[3] Center for Climate Systems Modeling / ETH, Switzerland
[4] Swiss National Supercomputing Centre CSCS / ETH, Switzerland

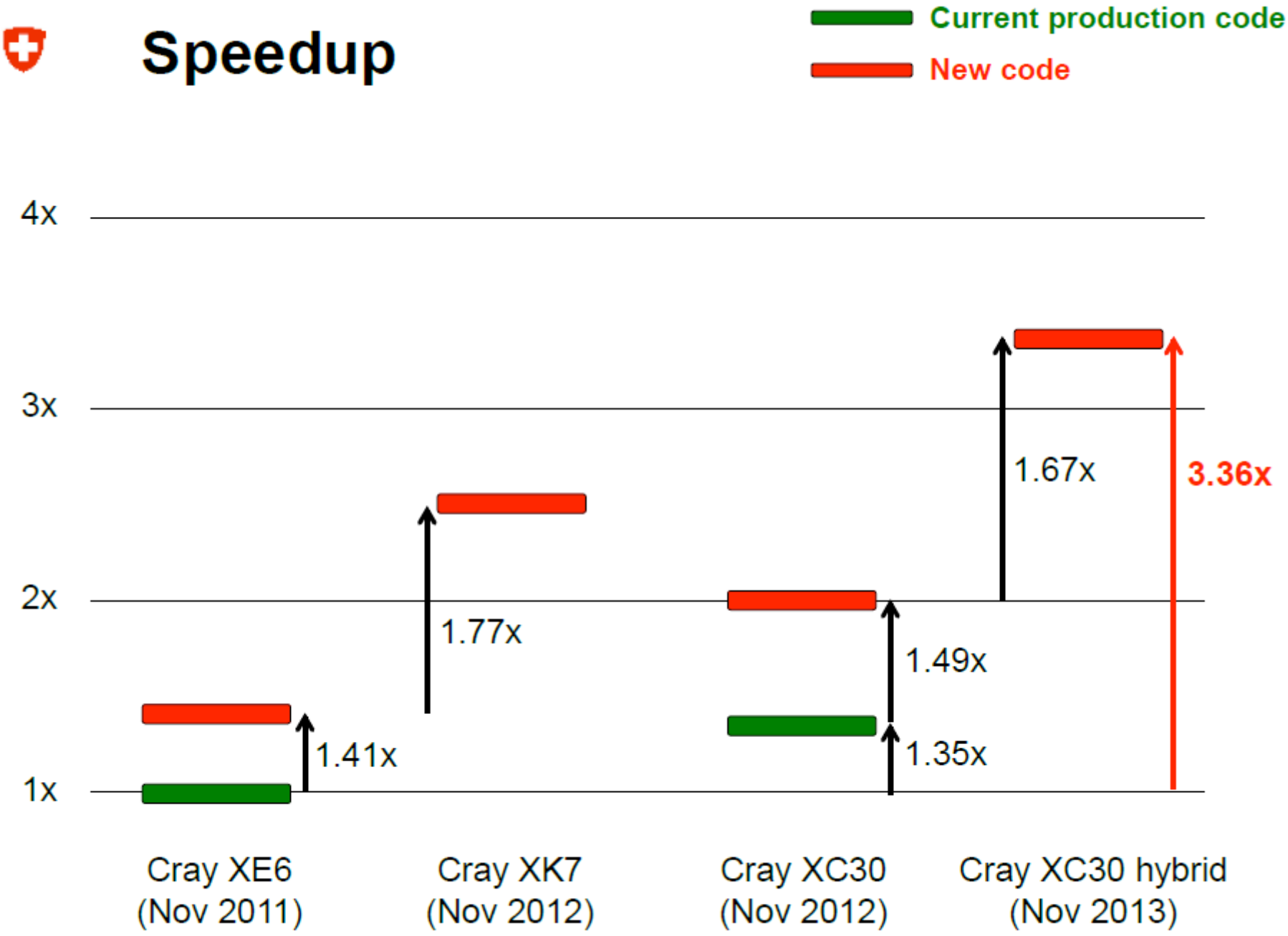
The Fortran + MPI + “X” Approach

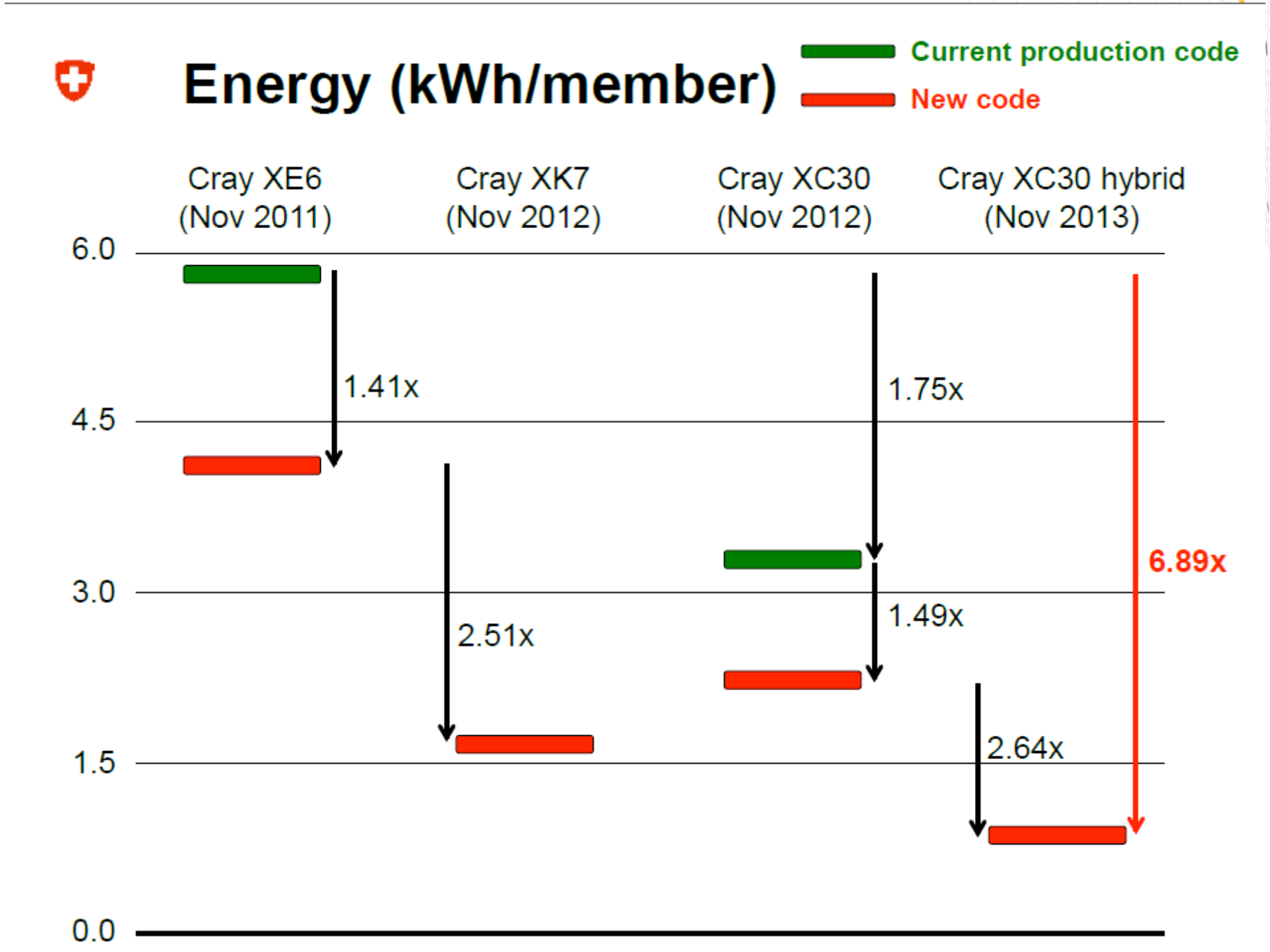


Implementation



Speedup


Current production code
New code
4x
3x
2x
1x
1.41x
1.77x
1.49x
1.35x
1.67x
3.36x
Cray XE6
(Nov 2011)
Cray XK7
(Nov 2012)
Cray XC30
(Nov 2012)
Cray XC30 hybrid
(Nov 2013)


Energy (kWh/member)
Current production code
New code
Cray XE6 (Nov 2011)
Cray XK7 (Nov 2012)
Cray XC30 (Nov 2012)
Cray XC30 hybrid (Nov 2013)
6.0
4.5
3.0
1.5
0.0
1.41x
2.51x
1.75x
1.49x
2.64x
6.89x

"Software is getting slower more rapidly than hardware becomes faster"

–Niklaus Wirth, 1995