OpenACC User FeedBack
July 24th, 2017

Ronald M. Caplan
Predictive Science Inc.
www.predsci.com

Programming Challenges

1) For efficiency, had to make some temporary arrays global
   to avoid setting initial values in kernel or using GPU-CPU transfers of initial values
   for every call of the subroutine

```
real(r_typ), dimension(nr,nt,np) :: x_ax
```

```
x_ax=0.
     call set_boundary_points_inner (x_ax,one)
```

Is overhead of GPU “enter data create” and “delete” more/significant compared to CPU stacked/automatic temporary arrays?

If not, this is only an issue with temporary arrays that need an initialization (e.g. summation arrays)
FIX: see init suggestion later…

Programming Challenges cont.

2) For efficiency, had to create new scalars when constant-indexed values of array were being used in a loop, in order to avoid having to transfer the whole array to the GPU.

```
          x(1,2:ntm1,2:npm1)= x(2,2:ntm1,2:npm1)
&                            -vmask*br0(2:ntm1,2:npm1)*dr(1)
```

```
!$acc kernels present(x,br0)
        x( 1,2:ntm1,2:npm1)= x(2,2:ntm1,2:npm1)
     &                      -vmask*br0(2:ntm1,2:npm1)*dr1
!$acc end kernels
```

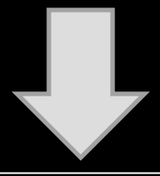
FIX?

- Have compiler detect situation and pass in as scalar?
- Have user direct compiler to do this by adding:
  !$acc kernels loop … scalars(dr(1),…)

Programming Challenges cont.

3) Loop order generated here was not great, but that could have been the case on the CPU as well…

```
do i=1,nr
  x(i, 1,2:npm1)=two*sum0(i)-x(i,2,2:npm1)
enddo
```



```
!$acc kernels loop gang worker(8) private(i,k) present(x,sum0)
        do k=2,npm1
!$acc loop gang vector(32)
          do i=1,nr
            x(i,1,k)=two*sum0(i)-x(i,2,k)
          enddo
        enddo
```

Other Challenges

1) The HPC center I had access to had bugs in its CUDA-aware MPI

2) The HPC center had outdated versions of PGI

3) Some HPC centers refuse to install PGI when requested ☹

4) Using MPI+OpenACC x86 gave bad performance and OpenMP/mpiexec flags/ENV to control affinity/bindings did not seem to work when using OpenACCx86

My OpenACC Wishlist (1) acc memcopy

```
!$acc memcopy (a,b)
      a=b
!$acc end memcopy
```

```
!$acc memcopy (a,b)
      do i=1,N
         a(i)=b(i)
      end do
!$acc end memcopy
```

- This would perform a device-to-device memcopy from array b to array a
- The pragmas would surround the original program's copy code (such as a loop) and would tell the compiler to ignore the code in the region, and do the memcopy instead.
- This would avoid having inefficient copy kernels being generated instead of fast memcopys.
- Would be great alongside deep-copy

My OpenACC Wishlist (2) acc ignore

```
!$acc ignore
      call DEBUG_IO()
      call COMPLICATED_STUFF_I DON'T_NEED_NOW()
      call NEED_TO_THINK_ABOUT_THIS_ONE()
!$acc end ignore
```

- This pragma would tell the compiler to ignore the code in the region

- This would be very useful for incremental development, avoiding not-so-necessary debugging IO/complicated calls and/or using OpenACC for a subset of program modes/options
(Allows commenting out code without effecting the CPU code)

My OpenACC Wishlist (3) acc enter data create(x) init(a)

```
!$acc enter data create(x,y) init(a)
```

- Add an “init(a)” option to the create data pragma.
- This would initialize arrays “x,y” to the scalar “a”.
- This would avoid having to copy an initialized CPU array to the GPU.
- USE CASE: Arrays of summations, where the local array of the sum values needs to be set to 0 each subroutine call.

(PGI has compile flag for init to 0 but is not portable,
and accidently not supplying the flag would lead to errors.)

My OpenACC Wishlist (4) acc kernels gang,worker,vector

- Allow gang,worker,vector options in "acc kernels" pragma
- For implicit loops in fortran using array-syntax, allow the user to map dimensions to gang,worker,vector (somehow…).

```
!$acc kernels present(x,br0) dim(1:gang,vector;2:gang,worker;3:gang)
        x( 1,2:ntm1,2:npm1)= x(2,2:ntm1,2:npm1)
     &                      -vmask*br0(2:ntm1,2:npm1)*dr1
!$acc end kernels
```

My OpenACC Wishlist (5) New ENV

- Add compiler-helper environment variables:

  ACC_ENABLE=1
  ACC_DEVICE_TYPE=nvidia_gpu
  ACC_DEVICE_MODEL=v100

  This would allow the compiler to set the proper gang,worker,vector, instruction set, optimizations, etc. in a portable way without relying on compiler-specific flags.

OpenACC SPECS Documentation

1. Concept of gang, worker, vector is explained but more concrete examples for known architectures would be nice.
   (I am never sure which to use, where, on which nested loops, and especially which numbers for each).

2. Would like more clarity on when an "acc end" is needed in Fortran and when it is not

3. Would like more clarity on when "acc declare" is needed in Fortran and when it is not
   (e.g. when can one use just a “data enter/create”?)

OpenACC Implementations

1. Really only one option for most users (PGI)

2. Implementation seems to lags specification somewhat (e.g. acc set device_num() was missing in PGI 16.10) .

3. Maybe require (at least one) full implementation before releasing spec as official (or have two kinds of releases)?

4. Maybe some code/driver sharing and/or funding and/or marketing campaign to get GNU OpenACC on a competitive track?