OpenACC - gaining experience

Luis Kornblueh and Uwe Schulweida, MPIM
Jörg Behrens and Hendryk Bockelmann, DKRZ
Guo Xiaolin, ETHZ
Will Sawyer, CSCS
Stephane Chauveau, NVIDIA

Max Planck Institute for Meteorology
German Climate Computing Centre

Climate model characteristics

- Fortran 2003 code base
- using derived types for memory management
- using modules with parameters and variables
- accessing module variables in routines
- using FFT (select from several optimized libraries or the fastest version: fft992 (Fortran)
- using DGEMM from BLAS (select from several optimized libraries or the slowest reference implementation (Fortran)
- hybrid OpenMP - MPI code

Modern Fortran - a strange encounter?

Concepts

- Type parameters
- Procedure pointers
- Type extension
- Polymorphism

More keywords

generic procedures, type extension, polymorphic variables and type-bound procedures

Other keywords

Fortran pointer and `noalias/contiguous` directive

Fortran Array Syntax

PGI parse error: stored value and pointer type do not match

```
!$acc kernels
WHERE (ABS(xim1(:,:,jrow)) < zlimit)
  xim1(:,:,jrow) = 0.0_wp
END WHERE }
```

At Linktime: Undefined reference to `cudaMalloc`, `cudaFree`

```
!$acc parallel
WHERE (ABS(xim1(:,:,jrow)) < zlimit)
  xim1(:,:,jrow) = 0.0_wp
END WHERE
```

How to handle memory

data clause

- `!$acc enter data <create|copyin> (<var>)`
- `!$acc exit data delete (<var>)`

Fortran pointer are only sometimes a problem
(Cray: `object%array`) !
Fortran module parameter arrays are sometimes a problem
(PGI: temporary name `a$ac` instead of `a` and `copyin`)

Initialize and copy

initialize and copy directive

- `!$acc update device(a=<val>)`
- `!$acc update device(a=host(b))`

Getting the FFT interface and handling right

PGI is not in-line with the OpenACC spec: it must be `int` not `long` for the arg of `acc_get_cuda_stream`

```
interface
  function acc_get_cuda_stream(async) result(stream) &
           bind(c,name='acc_get_cuda_stream')
    import :: c_ptr, c_int
    type(c_ptr) :: stream
    integer(c_int), value :: async
   end function acc_get_cuda_stream
end interface

interface
  integer(c_int) function cufftSetStream(plan, stream) &
           bind(c,name='cufftSetStream')
    import :: c_ptr, c_int
    integer(c_int), value :: plan
    type(c_ptr), value :: stream
  end function cufftSetStream
end interface
```

Getting the FFT interface and handling right

Set before using the planned transform!

```
integer(c_int) :: iret

iret = cufftSetStream(forward_plan, &
                      acc_get_cuda_stream(acc_async_sync))
iret = cufftSetStream(backward_plan, &
                      acc_get_cuda_stream(acc_async_sync))
```

Do not be to smart with small examples the `async` default behavior might or might not hit you.

General remarks:

Add an intrinsic scaling property as is available in MKL!

Usability first (safe run mode), speed up later (change of mode by user)!

Getting the Fortran DGEMM interface and handling right

- CRAY and PGI compilers provide optimized BLAS for OpenACC.
- Both do magic such as automatic detection of arrays on device.
- Standard APIs: dgemm may or not be translated to an accelerated call.
- OpenACC specific APIs: cublasDgemm vs dgemm_acc.
- PGI cublasDgemm and dgemm on device are asynchronous (also CUDA Fortran).
- CRAY dgemm_acc and dgemm on device are synchronous.
- No easy and portable way to launch multiple concurrent small dgemms.
- Currently writing a custom binding to cublasDgemm : −(
- CRAY and PGI should agree on a common BLAS interface to OpenACC.

Module parameter arrays

Problem with PGI:

```
method=[ memcpyHtoDasync ] gputime=[ 3.200 ] cputime=[ 14.710 ]
method=[ memcpyHtoDasync ] gputime=[ 2.240 ] cputime=[ 3.664 ]
method=[ const_22_gpu ]    gputime=[ 5.856 ] cputime=[ 19.082 ]
method=[ memcpyDtoHasync ] gputime=[ 2.624 ] cputime=[ 4.991 ]
```

```
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes)
ACC: Transfer 2 items (to acc 12 bytes, to host 0 bytes)
ACC: Execute kernel const_$ck_L22_2 async(auto)
ACC: Transfer 1 items (to acc 0 bytes, to host 12 bytes)
```

Device routine orphaning

```
    !$acc function ...
```

call to cuStreamSynchronize returned error 716: Misaligned address
ACC: craylibs/libcrayacc/acc_hw_nvidia.c:662 CRAY_ACC_ERROR - cuStreamSynchronize returned CUDA_ERROR_LAUNCH_FAILED

Derived types and pointer

Just a comment: It seems there is no real concept available!

Saturation adjustment

Problem

The only (and simplest) physical parameterization (vertical dependencies)

- Removed table lookups in favor of exponential function evaluations
- Use big block size (up to 4608) for best performance on CPU and GPU

```
    zes = f_ua_spline(pt(jl,kk))*zppi
```

Results

CPU (1 core, nproma = 4608): 1.836s GPU (1 node, nproma = 4608): 2.7025s

Vertical transport of passive tracer

- strong loop carried dependencies in k direction prevent from parallelization (algorithmic rewrite needed)
- internal vector blocking (in horizontal i-j) pays out: actualy used nproma=4608
- OpenMP parallelization for j is present, but does not give any speedup
- maybe internal sync on device (although async(tid) used)?
- nasty loop_body call needed for Cray compiler

Vertical transport of passive tracer

```
!$OMP PARALLEL
!$OMP DO ECHAM_GPC_SCHEDULE
   DO j=1,jm
      CALL loop_body(kproma, nq, im, jm, km, q(1,1,1,j), ak,
     bk, ps(1,j), delp(1,1,j))
    ENDDO
!$OMP END DO
!$OMP END PARALLEL
```

Vertical transport of passive tracer

```
SUBROUTINE loop_body(...)
...
    tid = omp_get_thread_num()

!$acc data create(pe1, pe2)
!$acc parallel present(ak, bk) async(tid)
!$acc loop
    DO i=1,im ! profite for long vector length
      pe1(i,1) = ak(1)
      DO k=1,km ! loop seq due to dependencies
        pe1(i,k+1) = pe1(i,k) + delp(i,k)
      ENDDO
      ps(i) = pe1(i,km+1)
    ENDDO
!$acc end parallel
!$acc wait(tid)
...
```

Vertical transport of passive tracer

loop collapsing best (size is km=47, jm=48, im=96)

```
!$acc loop collapse(2)
DO k=1,km
  DO i=1,im
```

- qmap_gp: 12.3s (1 CPU) , 8.3s (1 CPU+GPU), 1.48 speedup
- qmap_gp 60% of tp_core which itself is 40% (without physics)

threads	nproma	times [s]
1 CPU	4608	12.33
2 CPU	2304	6.03
4 CPU	1152	3.17
1 GPU	4608	8.33

Horizontal advection

Problem: Transport multiple (currently 3) quantities in horizontal (independend in vertical).

Status: Conceptually should provide good thread occupancy and performance, however, considerable refactoring needed. Minor kernels tested. No performance numbers

```
!$OMP DO
    merged_loop: DO k_iq = 0, nq*klev-1
      k = MOD(k_iq,klev)+1
      iq = k_iq / klev + 1
      CALL tp2g(q2(1,jfirst-ng,k,iq),    va(1,jfirst,k), &
           cx(1,jfirst-ng,k),  cy(1,jfirst,k),          &
           im,  jm,  iord,     jord,                    &
           ng,  mg,  fx(1,jfirst,k), fy(1,jfirst,k),    &
           ffsl(jfirst-ng,k),    jfirst,   jlast,       &
           delp1(1,jfirst-mg,k),    delp(1,jfirst,k) )

      ! ...  more 2D stuff
    ENDDO merged_loop
!$OMP END DO
```

Conclusion

We will continue $; -)$ . . .