Data Movement Tips and Tricks

David Appelhans

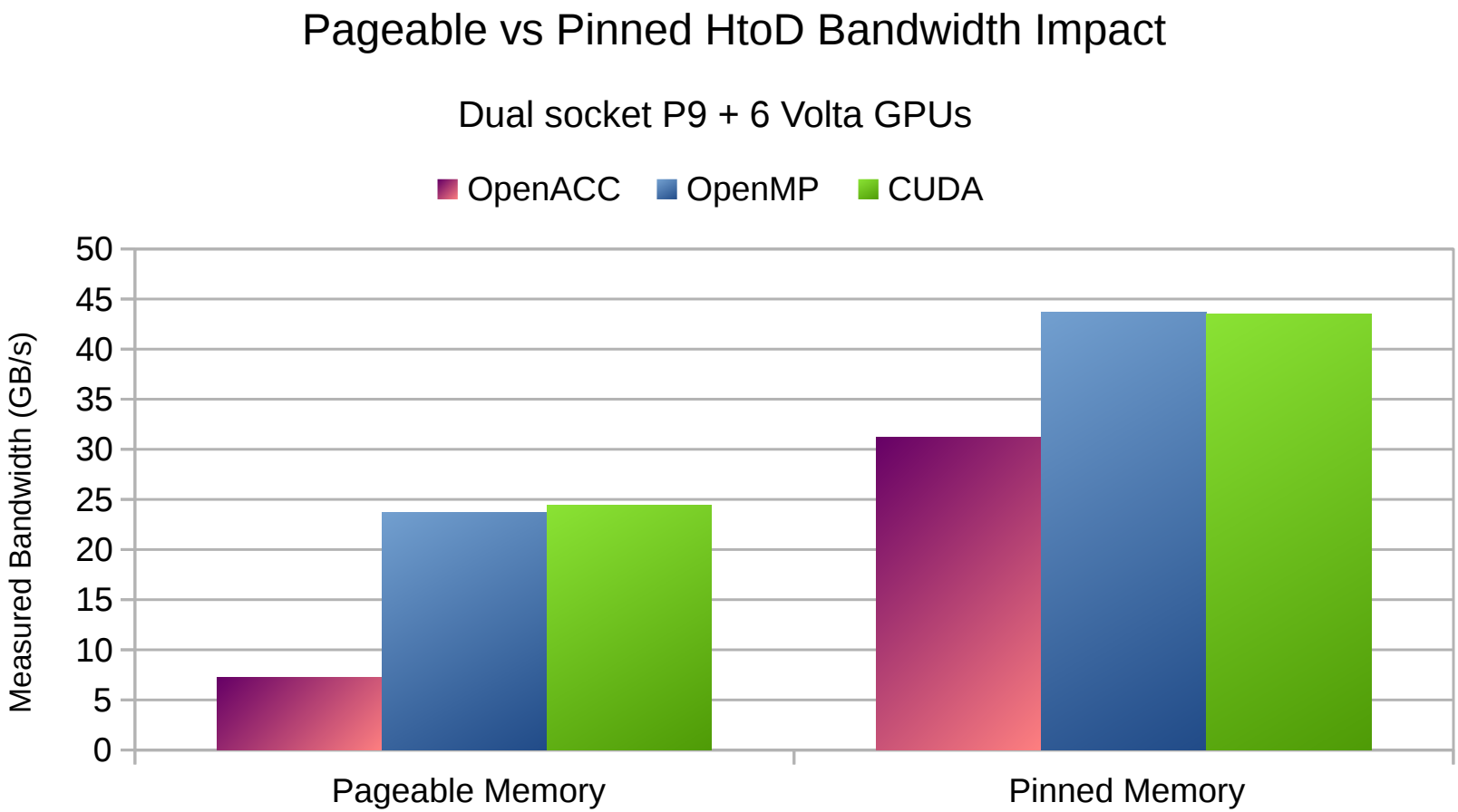
IBM Research,

ORNL SUMMIT workshop
March 7, 2018

Introduction

- This is an early version of 50 min talk for GTC 2018–For more complete details, please see that talk.
- Importance of pinned memory. (Interoperability, CUDA+OpenMP+OpenACC)
- Zero-copy tricks. (Interoperability, CUDA+OpenMP)
- Dealing with nested data structures. (Efficiency, CUDA Fortran)

Motivation: Why you should pin your memory



Example code runable on SUMMIT is availabe at https://github.com/dappelha/gpu-tips/tree/master/pinned-memory

Pinned Memory Option 1:

Use CUDA Fortran[1] pinned attribute to pin at allocation time,

```
1  real(kind=8), pinned, allocatable :: p_A(:)
2  allocate ( p_A(N) )
3  !$omp target data map(alloc:p_A)
4  do i=1,samples
5     !$omp target update to(p_A)
6     ...
7  enddo
```

Can also check success of pinning:

```
1  logical :: pstat
2  allocate ( p_A(N), pinned=pstat)
3  if (.not. pstat ) print *, "ERROR: p_A was not pinned"
```

[1] PGI and XLF compilers both support CUDA Fortran, so the pinned attribute can easily be combined with directives.

Pinned Memory Option 2:

Pin already allocated memory,[2]

```
1  use, intrinsic :: iso_c_binding
2  use cudafor
3  real, pointer, contiguous :: phi(:,:)
4  allocate( phi(dim1, dim2) ) ! phi can also be pointer passed from C++
5  istat = cudaHostRegister(C_LOC(phi(1,1)), sizeof(phi), cudaHostRegisterMapped)
6
7  !$acc enter data create(phi)
8  do i=1,samples
9      !$acc update self(phi)
10     ...
11 enddo
```

Warning: pinning memory is very slow. Memory should only be pinned if it is going to be used for many transfers.

[2] This technique is especially useful if the memory was allocated outside the developers control (for example in a C++ calling routine).

OpenACC Interoperabilty Warning 1

You must use the flag -ta=tesla:pinned in order for OpenACC to benefit from pinned memory. Without this flag you will see the same bandwidth as pageable memory. There are two ways to use the flag:

1. *Compiling* with the flag -ta=tesla:pinned forces **all** memory to be pinned memory. If you have many arrays that are not used by the GPU, you will unnecessarily be paying the expensive cost to pin them.
2. *Linking* the final executable with -ta=tesla:pinned causes the OpenACC runtime to check if an array is already pinned. This gives fine grain user control to manually pin arrays as shown in the previous examples.

OpenACC Interoperabilty Warning 2

The OpenACC runtime uses a memory pool on the device to save from repeated allocation/deallocation of device memory. This is good for exclusive OpenACC codes but it can cause trouble when mixing programming models:

```
1  integer :: N = 8*gigabyte
2  real(kind=8), allocatable :: A(:)
3  real(kind=8), device, allocatable :: d_A(:)
4  allocate ( A(N) )
5  !$acc enter data create(A)
6  !$acc exit data delete(A) ! <--not truly free'd unless PGI_ACC_MEM_MANAGE=0
7  allocate ( d_A(N) ) ! <---- can then run out of device memory
```

To disable this optimization, set the environment flag PGI_ACC_MEM_MANAGE=0 and the runtime will free the data at the exit data.

Uses of Zero Copy

Zero copy refers to accessing host resident pinned memory directly from a GPU without having to copy the data to the device beforehand (i.e. there are zero device copies).

- Quick overlap of data movement and kernel compute (mostly bad idea because it hides compute/movement distinction, and unified/managed memory now serves this purpose)
- Reads or writes of non unit stride arrays.
  - Example 1: Large arrays where only small percent of data is accessed in random pattern.
  - Example 2: All data is accessed, but read/write pattern is strided/not coalesced.
  - Example 3: Efficiently populating elements of a structure, avoiding the overhead of many copy api calls by using GPU threads to fetch data directly.

CUDA Zero Copy Setup

To set up zero copy of a basic array in Fortran, use a CUDA API to get a device pointer that points to the pinned host array, and then associate a fortran array with that C device pointer, specifying the Fortran array attributes.

```
1  use iso_c_binding   ! provides c_f_pointer and C_LOC
2  ! zero copy pointers for psib
3  type(C_DEVPTR)                    :: d_psib_p
4  real(adqt), device, allocatable   :: pinned_psib (:,:,:)
5
6  ! sets up zero copy of psib needed for snrefelctD on device.
7   istat = cudaHostGetDevicePointer(d_psib_p, C_LOC(psib(1,1,1)), 0)
8  ! Translate that C pointer to the fortran array with given dimensions
9  call c_f_pointer(d_psib_p, pinned_psib, [QuadSet%Groups, Size%nbelem,
       QuadSet%NumAngles] )
```

OpenMP Zero Copy Example

Only requires CUDA pinned array and OpenMP is_device_ptr clause.

```
1  real(kind=8), pinned, allocatable :: A(:,:),At(:,:)
2  ...
3  allocate ( A(nx,ny), At(ny,nx) )
4
5  ! Transpose in the typical way:
6  !$omp target enter data map(alloc:A,At)
7  call transpose(A,At,nx,ny)
8  !$omp target update from(At)
9  !$omp target exit data map(delete:At)
10
11 ! Ensure device has finished for accurate benchmarking
12 ierr = cudaDeviceSynchronize()
13
14 ! Transpose using zero copy for At.
15 ! At is no longer mapped--is_device_ptr(At) will
16 ! allow addressing host pinned memory (zero copy)
17 call transpose_zero_copy(A,At,nx,ny)
```

continued on next slide

OpenMP Zero Copy Example Continued

```
1  subroutine transpose_zero_copy(A,At,nx,ny)
2    ! example of strided writes to an array that lives on the host
3    ! variable declarations
4    implicit none
5    real(kind=8), intent(in) :: A(:,:)
6    real(kind=8), intent(out) :: At(:,:)
7    integer, intent(in) :: nx, ny
8    integer :: i,j
9    !$omp target teams distribute parallel do is_device_ptr(At)
10   do j=1,ny
11       do i=1,nx
12           At(j,i) = A(i,j)
13       enddo
14   enddo
15   return
16 end subroutine transpose_zero_copy
```

OpenMP Zero Copy Transpose

Table : Power8 + P100 results of doing a traditional matrix transpose and then copying back from GPU vs doing the transpose directly into pinned host memory.

Method	Time(ms)	Total Time
transpose + memcpy	57.3 + 63.1	**120.4**
zero-copy-transpose	65.4	**65.4**

Efficiently Populating Nested Structures

A niave implementation for getting data structures populated on the GPU usually looks like this:

```
1  % ! allocate GPUelement
2  % do id=1, Nelements
3  %     call construct_GPUelements(GPUelement,id,Nnodes)
4  % enddo
5  %
6  ! still need to poplulate the values from the host version of the data structure :
7  do id=1, Nelements
8      GPUelement(id)%Nnodes = element(id)%Nnodes ! implicit cudaMemcpy
9      GPUelement(id)%x = element(id)%x           ! implicit cudaMemcpy
10     GPUelement(id)%y = element(id)%y           ! implicit cudaMemcpy
11     GPUelement(id)%val = element(id)%val       ! implicit cudaMemcpy
12 enddo
13 %
14 % do id=1,Nelements
15 %     call destruct_GPUelements(GPUelement,id)
16 % enddo
```

This becomes incredibly slow when Nelements is large.

Efficiently Populating Nested Structures

There are two ways to fix the niave approach,

1. loop over cudaMemcpyAsync calls instead of the numerous blocking calls done above,
2. pull the data from the GPU by populating the arrays from within a device kernel using zero copy.

PUSH DATA BY LOOPING OVER CUDAMEMCPYASYNC

CUDA Fortran code would look like this:

```
1  integer (cuda_stream_kind) old_stream, streamid
2  ierr  = cudaStreamCreate(streamid)
3  old_stream  = cudaforGetDefaultStream()        ! save the current default stream
4  ierr  = cudaforSetDefaultStream(streamid)   ! Set the default stream to streamid
5  do id=1, Nelements
6     GPUelement(id)%Nnodes = element(id)%Nnodes ! implicit cudaMemcpyAsync on streamid
7     GPUelement(id)%x = element(id)%x           ! implicit cudaMemcpyAsync on streamid
8     GPUelement(id)%y = element(id)%y           ! implicit cudaMemcpyAsync on streamid
9     GPUelement(id)%val = element(id)%val       ! implicit cudaMemcpyAsync on streamid
10 enddo
11 ierr  = cudaforSetDefaultStream(old_stream) ! restore the original default stream
```

Can do similar in OpenMP 4, with update nowait (no example yet).

PULLING FROM THE GPU

Set up a way to reference host structures from the device (zero copy of a structure):

```
1  ! to use element on the device, we have to make a device valid copy called d_element:
2  istat =cudaMemcpyAsync(d_element, element, size(element), 0)
3  ! similarly with d_GPUelement, we need to transfer the device memory addresses held
       in GPUelement
4  istat =cudaMemcpyAsync(d_GPUelement, GPUelement, size(GPUelement), 0)
5  ! Now we can use these in a CUDA kernel to zero copy
6  ! the member data from d_element into d_GPUelement
7  call set_elements_kernel <<<blocks,threads>>>(d_GPUelement,d_element,Nelements)
```

- line 4 allows deep references within a GPU kernel, such as d_GPUelement(id)%d_member.
- Can also accomplish this by making GPUelement a managed memory variable.

Launch a kernel to have GPU threads pull data from structures on the host into structures on the device:

```
1  attributes(global) subroutine set_elements_kernel(GPUelement,element, Nelements)
2    implicit none
3    ! kernel that uses zero copy to popluate the GPUelement structure.
4    ! I have dropped the d_ prefix for convenience on these dummy variables
5    type(GPUelement_type), device, intent(inout) :: GPUelement(:) ! members are device
6    type(element_type), device, intent(in):: element(:) ! members are pinned host
7    integer, value, intent(in) :: Nelements
8    integer :: id, Nnodes, node
9
10   do id=blockIdx%x,Nelements, gridDim%x
11     Nnodes = element(id)%Nnodes
12     GPUelement(id)%Nnodes = Nnodes
13     do node = threadIdx%x, Nnodes, blockDim%x
14       GPUelement(id)%x(node) = element(id)%x(node)
15       GPUelement(id)%y(node) = element(id)%y(node)
16       GPUelement(id)%val(node) = element(id)%val(node)
17     enddo
18   enddo
19
20 end subroutine set_elements_kernel
```

EXAMPLE CODE

Example code runable on SUMMIT is availabe at
https://github.com/dappelha/gpu-tips/tree/master/pinned-memory

Includes:

- jsrun submission script example,
- affinity checking script to verify,
- device binding helper script,
- pinned memory comparison example.

Questions: David Appelhans - dappelh@us.ibm.com

QUESTIONS?

David Appelhans - dappelh@us.ibm.com