CUDA Fortran 2013

Mathew Colgrove
The Portland Group
brent.leback@pgroup.com

PGI
What Does CUDA Fortran Look Like?

```
real, device, allocatable, dimension(:,,:) ::
  Adev, Bdev, Cdev

allocate (Adev(N,M), Bdev(M,L), Cdev(N,L))
Adev = A(1:N,1:M)
Bdev = B(1:M,1:L)

call mm_kernel <<<dim3(N/16,M/16),dim3(16,16)>>>
  ( Adev, Bdev, Cdev, N, M, L)
C(1:N,1:L) = Cdev
deallocate ( Adev, Bdev, Cdev )
```

```
attributes(global) subroutine mm_kernel
  ( A, B, C, N, M, L )
real :: A(N,M), B(M,L), C(N,L), Cij
integer, value :: N, M, L
integer :: i, j, kb, k, tx, ty
real, shared :: Asub(16,16), Bsub(16,16)

tx = threadidx%x
ty = threadidx%y
i = blockIdx%x * 16 + tx
j = blockIdx%y * 16 + ty
Cij = 0.0

do kb = 1, M, 16
  Asub(tx,ty) = A(i,kb+tx-1)
  Bsub(tx,ty) = B(kb+ty-1, j)
  call syncthreads()
  do k = 1,16
    Cij = Cij + Asub(tx,k) * Bsub(k,ty)
  enddo
  call syncthreads()
enddo
C(i,j) = Cij
end subroutine mmul_kernel
```
Declaring Fortran Device Data

- Variables / arrays with device attribute are allocated in device memory
  
  real, device, allocatable :: a(:)
  real, allocatable :: a(:)
  attributes(device) :: a

- In a host subroutine or function
  - device allocatables and automatics may be declared
  - device variables and arrays may be passed to other host subroutines or functions (explicit interface)
  - device variables and arrays may be passed to kernel subroutines
Declaring Fortran Module Data

- Variables / arrays with device attribute are allocated in device memory

```fortran
module mm
  real, device, allocatable :: a(:)
  real, device :: x, y(10)
  real, constant :: c1, c2(10)
  integer, device :: n
contains
  attributes(global) subroutine s(b)
  ...
```

- Module data must be fixed size, or allocatable
Allocating Data

- Fortran allocate / deallocate statement
  
  ```fortran
  real, device, allocatable :: a(:,,:), b
  allocate( a(1:n,1:m), b )
  ....
  deallocate( a, b )
  ```

- Arrays or variables with device attribute are allocated in device memory
  - Allocate is done by the host subprogram
  - Memory is not virtual, you can run out
  - Device memory is shared among users / processes, you can have deadlock
  - STAT=ivar clause to catch and test for errors
Copying Data to / from Device

- Assignment statements
  ```fortran
  real, device, allocatable :: a(:,:), b
  allocate( a(1:n,1:m), b )
  a(1:n,1:m) = x(1:n,1:m)  ! copies to device
  b = 99.0
  
  x(1:n,1:m) = a(1:n,1:m)! copies from device
  y = b
  deallocate( a, b )
  ```

- Data copy may be noncontiguous, but will then be slower (multiple DMAs)
- Data copy to / from host pinned memory will be faster
- Asynchronous copies currently require API interface
Launching Kernels

- Subroutine call with chevron syntax for launch configuration
  
  ```fortran
  call vaddkernel <<< (N+31)/32,32 >>> (A,B,C,N)
  ```

  ```fortran
  type(dim3) :: g, b
  g = dim3((N+31)/32, 1, 1)
  b = dim3( 32, 1, 1 )
  call vaddkernel <<< g, b >>> ( A, B, C, N )
  ```

- Interface must be explicit
  - In the same module as the host subprogram
  - In a module that the host subprogram uses
  - Declared in an interface block

- The launch is asynchronous
  - Host program continues, may issue other launches
Writing a CUDA Kernel (1)

- C: global attribute on the function header, must be void type
  - `__global__ void kernel ( ... ){...}
- F: global attribute on the subroutine statement
  - `attributes(global) subroutine kernel ( A, B, C, N )`
- May declare scalars, fixed size arrays in local memory
- May declare shared memory arrays
  - C: `__shared__ float sm(16,16);`
  - F: `real, shared :: sm(16,16)`
    - Limited amount of shared memory available (16KB, 48KB)
    - shared among all threads in the same thread block
- May declare assumed-size shared memory arrays when the size in bytes is passed in the kernel execution configuration
  - `call kernel<<< grid, block, shared memory size>>> (args...)`
- Data types allowed
  - int (long, short, char), float, double, struct, union, ...
  - integer(1,2,4,8), logical(1,2,4,8), real(4,8), complex(4,8), derivedtype
Writing a CUDA Kernel (2)

- **Predefined variables**
  - blockIdx, threadIdx, blockDim, gridDim, blockDim, warpSize

- **Executable statements in a kernel**
  - assignment
  - for, do, while, if, goto, switch
  - function call to device function
  - intrinsic function call
  - most intrinsics implemented in header files
Writing a CUDA Kernel (3)

- Fortran disallowed statements include:
  - read, write, print, open, close, inquire, format, other IO except now some limited support for list-directed (print *) ENTRY statement, optional arguments, alternate return
  - data initialization, SAVEd data
  - assigned goto, ASSIGN statement
  - stop, pause

- These features available with PGI 13.3 or later:
  - allocate, deallocate (needs CC3.5, CUDA 5.0)
  - Fortran pointer assignment, pointers in general
  - recursive procedure calls, direct or indirect (CC3.5, 5.0)
Using the CUDA API

use cudafor
real, allocatable, device :: a(:)
real :: b(10), b2(2), c(10)
integer(kind=cuda_stream_kind) :: istrm
...
istat = cudaMalloc( a, 10 )
istat = cudaMemcpy( a, b, 10 )
istat = cudaMemcpy( a(2), b2, 2 )

istat = cudaMemcpy( c, a, 10 )
istat = cudaFree( a )

istat = cudaMemcpyAsync(a, x, 10, istrm)
CUDA Errors

- Out of memory
- Launch failure (array out of bounds, ...)
- No device found
- Invalid device code (compute capability mismatch)

Test for error:

```c
ir = cudaGetLastError();
if( ir ) printf( "%s
", cudaGetErrorString(ir) );
```
Calling CUDA C Kernels

```fortran
interface
    subroutine myCkern(arr, N) bind(c,name='cudaCkernel')
        use iso_c_binding
        integer(c_float), dimension(:,,:), device :: arr
        integer, value :: N ! pass by value
    end subroutine myCkern
end interface

... 
real, allocatable, device:: d_Arr(:,:)
allocate(d_Arr(N,N))
call myCkern <<< g, b >>> (d_Arr, N)
```
Generic interfaces and overloading

Allows programmers to define Fortran-like operations:

```fortran
module dev_transpose
  interface transpose
    module procedure real4devxspose
    module procedure int4devxspose
  end interface
  contains
  function realdevxpose(adev) result(b)
    real, device :: adev(:,:)
    real b(ubound(adev,2),ubound(adev,1))
    <add your choice of transpose kernel>
    return
  end
end module dev_transpose
```

At the site of the function reference, the look is normal Fortran:

```fortran
subroutine s1(a,b,n,m)
  use dev_transpose
  real, device :: a(n,m)
  real b(m,n)
  b = transpose(a)
end
```
module cublas
  ! isamax
  interface isamax
    integer function isamax &
      (n, x, incx)
    integer :: n, incx
    real(4) :: x(*)
  end function
end interface

use cublas

real(4), device :: xd(N)
real(4) x(N)
call random_number(x)

! On the device
xd = x
j = isamax(N, xd, 1)

! On the host, same name
k = isamax(N, x, 1)
Building a CUDA Fortran Program

% `pgfortran a.cuf`

.cuf suffix implies CUDA Fortran (free form)

.CUF suffix runs preprocessor

Use the `–Mfixed` option for F77-style fixed format

% `pgfortran –Mcuda a.f90`

–Mcuda=[emu|cc10|cc13|cc20|cc30|cc35]

–Mcuda=[cuda4.2,cuda5.0]

–Mcuda=fastmath,flushz,keep[bin|gpu|ptx]

–Mcuda=rdc

Must use –Mcuda when linking from object files, compiler driver pulls in correct path and libraries
Demo

- Simple Monte-Carlo algorithm
  - Writing a basic kernel
  - Sum reductions
  - Call CUDA C kernels
PGI CUDA Fortran 2013 New Features

- Texture memory support
- CUDA 5.0 Dynamic Parallelism
  - Chevron launches within global subroutines
  - Support for allocate, deallocate within global subroutines
    ```fortran
    real, device, allocatable :: a_d_local(:)
    ```
  - Pre-defined Fortran interfaces to supported CUDA API, cublas
- CUDA 5.0 support for separate compilation, creating and linking static device objects and libraries. Relocatable device code.
Texture Memory as a Read-only Cache

CUDA Fortran module

real, texture, pointer :: q(:)

CUDA Fortran host code

real, device, target :: p(:)
allocate(p(n))
q => p

CUDA Fortran device code

i = threadIdx%x
j = (blockIdx%x-1)*blockDim%x + i
s(j) = s(j) + q(i)

CUDA C/C++ global

texture<float, 1,
cudaReadModeElementType> tq;

cudaMalloc((void **) d_p, size);
cudaChannelFormatDesc desc =
    cudaCreateChannelDesc(32, 0, 0,
                          0, cudaFormatKindFloat);
cudaBindTexture(0, tq, d_p,
                desc, size);

CUDA C/C++ host code

float *d_p;
allocate(p(n))
cudaMalloc((void **) d_p, size);
cudaChannelFormatDesc desc =
    cudaCreateChannelDesc(32, 0, 0,
                          0, cudaFormatKindFloat);
cudaBindTexture(0, tq, d_p,
                desc, size);

CUDA C/C++ device code

i = threadIdx.x;
j = blockIdx.x*blockDim.x + i;
s[j] = s[j] + tex1Dfetch(tq, i);
Texture Memory Performance (measured Gbytes/sec)

module memtests
integer, texture, pointer :: t(:)
contains
  attributes(global) subroutine stridem(m,b,s)
    integer, device :: m(*), b(*)
    integer, value :: s
    i = blockDim%x*(blockIdx%x-1) + threadIdx%x
    j = i * s
    b(i) = m(j) + 1
  return
end subroutine

  attributes(global) subroutine stridet(b,s)
    integer, device :: b(*)
    integer, value :: s
    i = blockDim%x*(blockIdx%x-1) + threadIdx%x
    j = i * s
    b(i) = t(j) + 1
  return
end subroutine
end module memtests

<table>
<thead>
<tr>
<th>blockDim = 32</th>
<th>blockDim = 128</th>
<th>blockDim = 512</th>
</tr>
</thead>
<tbody>
<tr>
<td>stridem</td>
<td>stridet</td>
<td>stridem</td>
</tr>
<tr>
<td>stride</td>
<td>53.45</td>
<td>141.93</td>
</tr>
<tr>
<td>2</td>
<td>50.94</td>
<td>135.43</td>
</tr>
<tr>
<td>3</td>
<td>46.96</td>
<td>75.25</td>
</tr>
<tr>
<td>4</td>
<td>44.25</td>
<td>60.19</td>
</tr>
<tr>
<td>5</td>
<td>39.06</td>
<td>50.09</td>
</tr>
<tr>
<td>6</td>
<td>37.14</td>
<td>42.95</td>
</tr>
<tr>
<td>7</td>
<td>32.88</td>
<td>37.56</td>
</tr>
<tr>
<td>8</td>
<td>32.48</td>
<td>33.38</td>
</tr>
<tr>
<td>9</td>
<td>29.90</td>
<td>33.38</td>
</tr>
<tr>
<td>10</td>
<td>27.43</td>
<td>33.38</td>
</tr>
<tr>
<td>11</td>
<td>25.17</td>
<td>33.38</td>
</tr>
<tr>
<td>12</td>
<td>23.23</td>
<td>33.38</td>
</tr>
<tr>
<td>13</td>
<td>21.57</td>
<td>33.38</td>
</tr>
<tr>
<td>14</td>
<td>20.15</td>
<td>33.38</td>
</tr>
<tr>
<td>15</td>
<td>18.87</td>
<td>33.38</td>
</tr>
<tr>
<td>16</td>
<td>17.78</td>
<td>33.38</td>
</tr>
<tr>
<td>17</td>
<td>16.38</td>
<td>33.38</td>
</tr>
<tr>
<td>18</td>
<td>15.08</td>
<td>33.38</td>
</tr>
<tr>
<td>19</td>
<td>13.78</td>
<td>33.38</td>
</tr>
</tbody>
</table>
attributes(global) subroutine strassen( A, B, C, M, N, K )
real :: A(M,K), B(K,N), C(M,N)
integer, value :: N, M, K

if (ntimes == 0) then
   allocate(m1(1:m/2,1:k/2))
   allocate(m2(1:k/2,1:n/2))
   allocate(m3(1:m/2,1:k/2))
   allocate(m4(1:k/2,1:n/2))
   allocate(m5(1:m/2,1:k/2))
   allocate(m6(1:k/2,1:n/2))
   allocate(m7(1:m/2,1:k/2))
   flags = cudaStreamNoBlocking
   do i = 1, 7
      istat = cudaStreamCreateWithFlags(istreams(i), flags)
   end do
end if

Support for Fortran allocate and deallocate in device code
CUDA Fortran dynamic parallelism

. . .
! m1 = (A11 + A22) * (B11 + B22)
call dgemm16t1<<<devblocks,devthreads,0,istreams(1)>>>(a(1,1), &
   a(1+m/2,1+k/2), m, &
   b(1,1), b(1+k/2,1+n/2), k, &
   m1(1,1), newn, newn, 1.0d0)

! m2 = (A21 + A22) * B11
call dgemm16t2<<<devblocks,devthreads,0,istreams(2)>>>(a(1+m/2,1), &
   a(1+m/2,1+k/2), m, &
   b(1,1), k, &
   m2(1,1), newn, newn)

. . .

! m7 = (A12 - A22) * (B21 + B22)
call dgemm16t1<<<devblocks,devthreads,0,istreams(7)>>>(a(1,1+k/2), &
   a(1+m/2,1+k/2), m, &
   b(1+k/2,1), b(1+k/2,1+n/2), k, &
   m7(1,1), newn, newn, -1.0d0)

istat = cudaDeviceSynchronize()

. . .
! C11 = m1 + m4 - m5 + m7
  call add16x4<<1,devthreads,0,istreams(1)>>>(m1,m4,m5,m7,m/2,c(1,1),m,n/2)

! C12 = m3 + m5
  call add16<<1,devthreads,0,istreams(2)>>>(m3,m/2,m5,m/2,c(1,1+n/2),m,n/2)

! C21 = m2 + m4
  call add16<<1,devthreads,0,istreams(3)>>>(m2,m/2,m4,m/2,c(1+m/2,1),m,n/2)

! C22 = m1 + m3 - m2 + m6
  call add16x4<<1,devthreads,0,istreams(4)>>>(m1,m3,m2,m6,m/2,c(1+m/2,1+n/2),m,n/2)

...  
  end subroutine strassen
CUDA Fortran Separate Compilation

Compile separate modules independently

% pgf90 -c -O2 -Mcuda=rdc ddfun90.cuf ddmod90.cuf

Object files can be put into a library

% ar rc ddfunc.a ddfun90.o ddmod90.o

Use the modules in device code in typical Fortran fashion

% cat main.cuf

    program main
    use ddmodule

    ...

Link using pgf90 and the rdc option

% pgf90 -O2 -Mcuda=rdc main.cuf ddfunc.a
Copyright Notice

© Contents copyright 2012, The Portland Group, Inc. This material may not be reproduced in any manner without the expressed written permission of The Portland Group.