CUDA Fortran

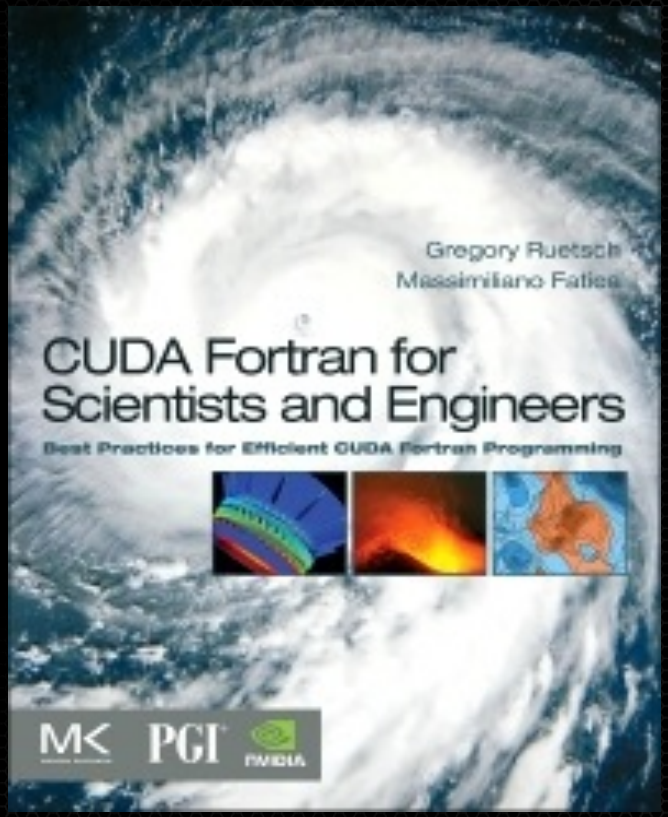
Trieste Feb 2017

Supplemental Materials

Some source code and examples are from this book which is available for download from this website:

booksite.elsevier.com/9780124169708

Also /opt/pgi/linux8664/2014/examples/CUDA-Fortran/CUDA-Fortran-Book



Introduction

- CUDA is a scalable model for parallel computing
- CUDA Fortran is the Fortran analog to CUDA C
  - Program has host and device code similar to CUDA C
  - Host code is based on the runtime API
  - Fortran language extensions to simplify data management
- CUDA Fortran implemented in the PGI compilers
- New community edition (latest is 16.10) freely available

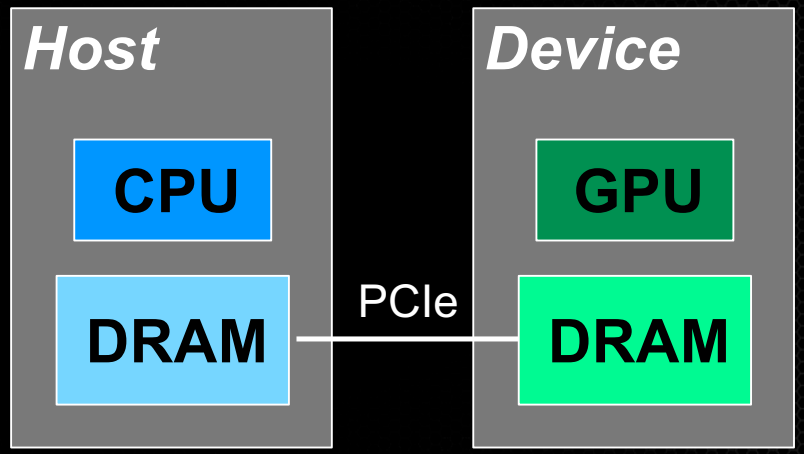
CUDA Programming

- Heterogeneous programming model
  - CPU and GPU are separate devices with separate memory spaces
  - Host code runs on the CPU
    - Handles data management for both host and device
    - Launches kernels which are subroutines executed on the GPU
  - Device code runs on the GPU
    - Executed by many GPU threads in parallel
  - Allows for incremental development

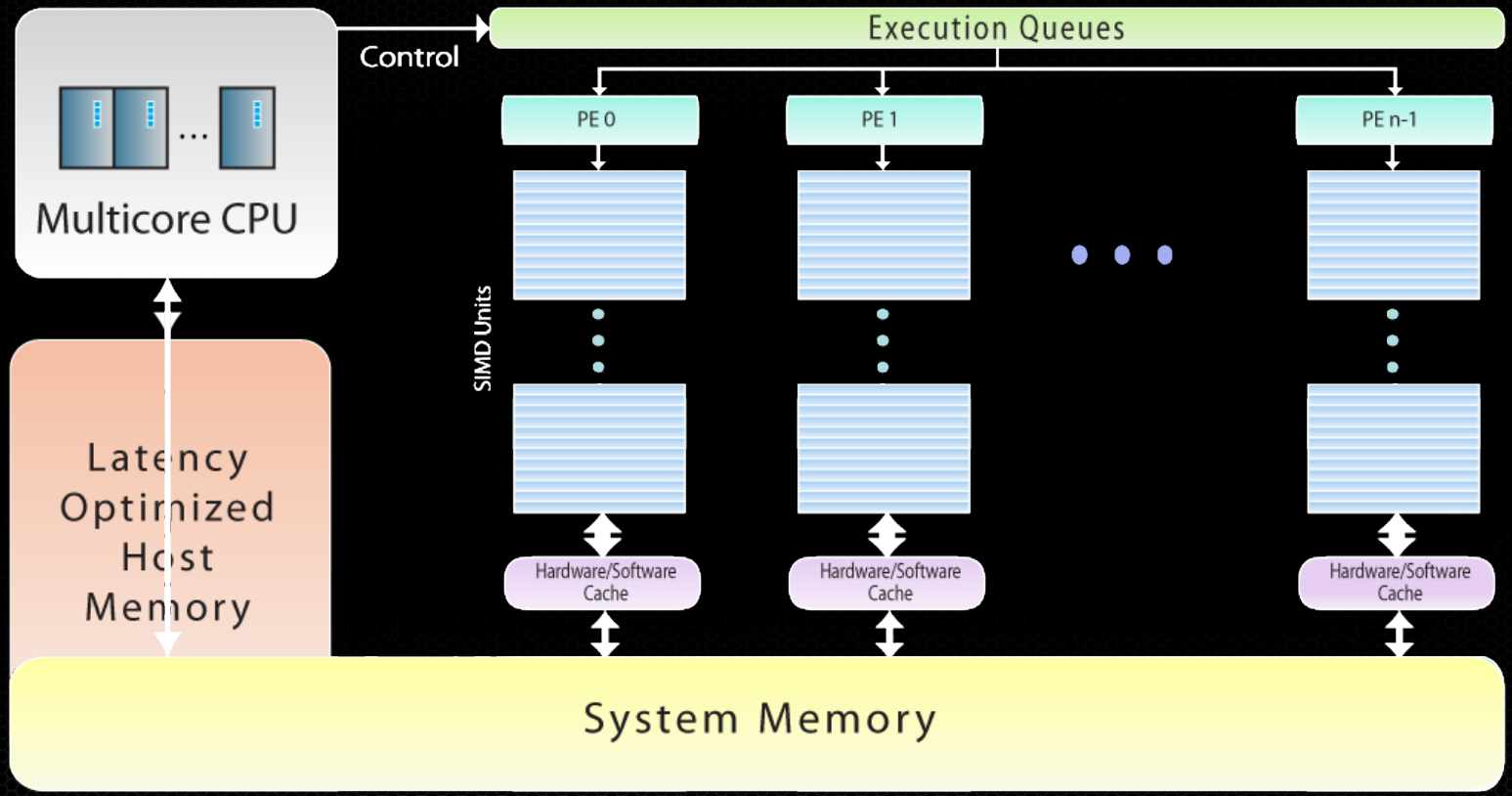
Heterogeneous Programming

- Host = CPU and its memory
- Device = GPU and its memory

- Typical code progression
  - Allocate memory on host and device
  - Transfer data from host to device
  - Execute kernel (device computation)
  - Transfer result from device to host
  - Deallocate memory



Accelerator Abstract Machine Architecture



What is CUDA Fortran?

```
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 )

. . .
```

Host Code

```
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
```

Device Code

F90 Array Increment

```
module simpleOps_m
contains
  subroutine inc(a, b)
    implicit none
    integer :: a(:)
    integer :: b
    integer :: i, n

    n = size(a)
    do i = 1, n
       a(i) = a(i)+b
    enddo

  end subroutine inc
end module simpleOps_m
```

```
program incTest
  use simpleOps_m
  implicit none
  integer :: b, n = 256
  integer, allocatable:: a(:)

  allocate (a(n))
  a = 1   ! array assignment
  b = 3
  call inc(a, b)

  if (all(a == 4)) &
     write(*,*) 'Test Passed'

  deallocate(a)
end program incTest
```

CUDA Fortran - Host Code

F90

```
program incTest

  use simpleOps_m
  implicit none
  integer :: b, n = 256
  integer, allocatable:: a(:)


  allocate (a(n))
  a = 1   ! array assignment
  b = 3


  call inc(a, b)


  if (all(a == 4)) &
     write(*,*) 'Test Passed'
  deallocate(a)

end program incTest
```

CUDA Fortran

```
program incTest
  use cudafor
  use simpleOps_m
  implicit none
  integer :: b, n = 256
  integer, allocatable :: a(:)
  integer, allocatable, device :: a_d(:)

  allocate (a(n),a_d(n))
  a = 1
  b = 3

  a_d = a
  call inc<<<1,n>>>(a_d, b)
  a = a_d

  if (all(a == 4)) &
     write(*,*) 'Test Passed'
  deallocate (a,a_d)

end program incTest
```

CUDA Fortran - Device Code

F90

```
module simpleOps_m
contains
  subroutine inc(a, b)
    implicit none
    integer :: a(:)
    integer :: b
    integer :: i, n

    n = size(a)
    do i = 1, n
       a(i) = a(i)+b
    enddo

  end subroutine inc
end module simpleOps_m
```

CUDA Fortran

```
module simpleOps_m
contains
  attributes(global) subroutine inc(a, b)
    implicit none
    integer :: a(:)
    integer, value :: b
    integer :: i

    i = threadIdx%x
    a(i) = a(i)+b


  end subroutine inc
end module simpleOps_m
```

Compile and run increment.cuf

- Compile with pgf90
- Files with **`.cuf`** or **`.CUF`** extensions automatically enable CUDA Fortran compilation

```
$ pgf90 increment.cuf
$ ./a.out
 Program Passed
$
```

Extending to Larger Arrays

- Previous example works with small arrays

```
call inc<<<1,n>>>(a_d,b)
```

  - Limit of `n=1024` (Fermi) or `n=512` (pre-Fermi)

- For larger arrays, change the first execution parameter (`<<<1,n>>>`)

Execution Model

Software

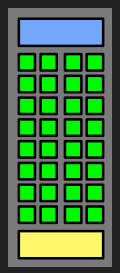
Thread

Hardware

Thread Processor

Threads are executed by thread processors

Thread Block

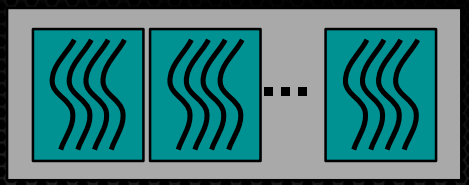


Multiprocessor

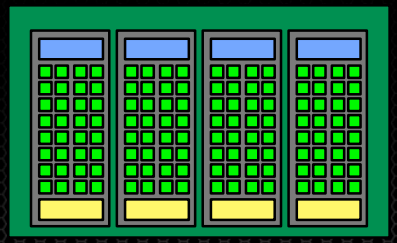
Thread blocks are executed on multiprocessors

Thread blocks do not migrate

Several concurrent thread blocks can reside on a multiprocessor



Grid



Device

A kernel is launched on a device as a grid of thread blocks

Execution Configuration

- Execution configuration specified in host code

```
call inc<<<blocksPerGrid, threadsPerBlock>>>(a_d,b)
```

- Previous example used a single thread block

```
call inc<<<1,n>>>(a_d,b)
```

- Multiple thread blocks

```
tPB = 256
call inc<<<ceiling(real(n)/tPB),tPB>>>(a_d,b)
```

Large Array - Host Code

```
program incrementTest
  use cudafor
  use simpleOps_m
  integer, parameter :: n = 1024*1024
  integer, allocatable :: a(:)
  integer, device, allocatable :: a_d(:)
  integer :: b, tPB = 256

  allocate(a(n), a_d(n))

  a = 1; b = 3
  a_d = a
  call increment<<<ceiling(real(n)/tPB),tPB>>>(a_d, b)
  a = a_d

  if (any(a /= 4)) then
     write(*,*) '**** Program Failed ****'
  else
     write(*,*) 'Program Passed'
  endif
  deallocate(a, a_d)
end program incrementTest
```

Built-in Variables for Device Code

- Predefined variables in device subroutines
  - Grid and block dimensions - **gridDim**, **blockDim**
  - Block and thread indices - **blockIdx**, **threadIdx**
  - Of type **dim3**

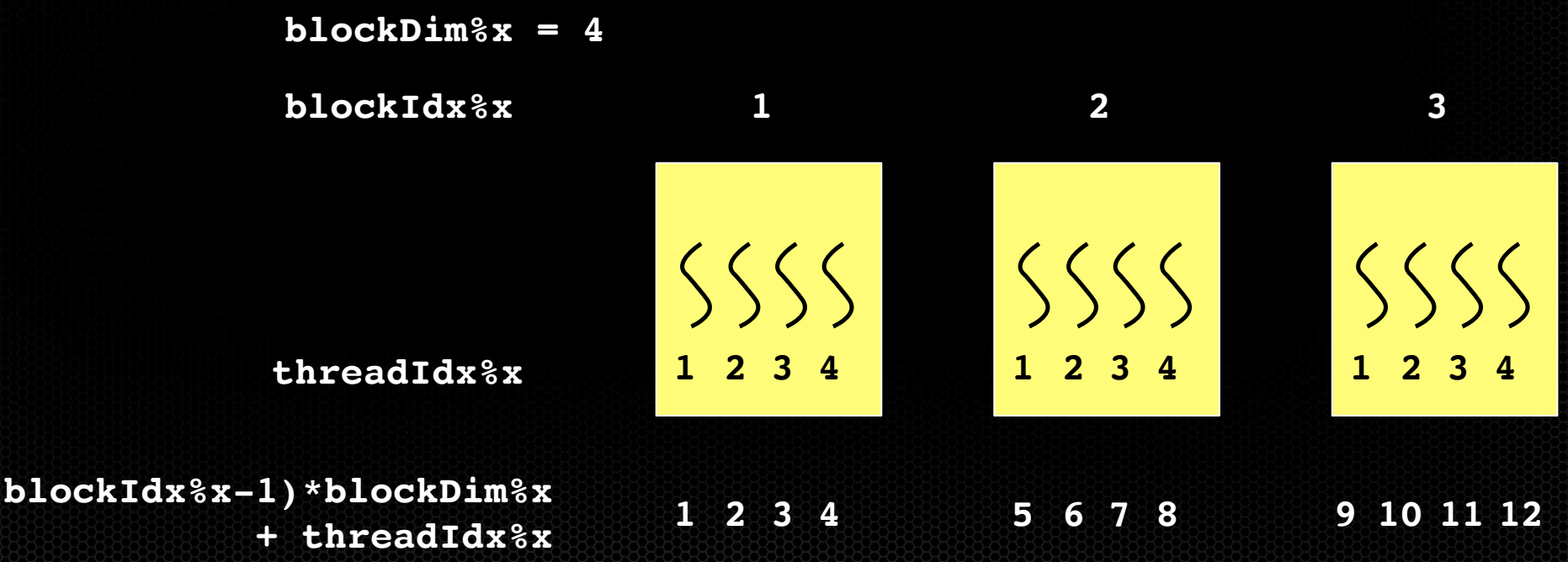
```
type (dim3)
  integer (kind=4) :: x, y, z
end type
```

  - **blockIdx** and **threadIdx** fields have unit offset

```
1 <= blockIdx%x <= gridDim%x
```

Mapping Arrays to Thread Blocks

- `call increment<<<3,4>>>(a_d, b)`



Large Array - Device Code

```
module simpleOps_m
contains
  attributes(global) subroutine increment(a, b)
    implicit none
    integer, intent(inout) :: a(:)
    integer, value :: b
    integer :: i, n

    i = blockDim%x*(blockIdx%x-1) + threadIdx%x
    n = size(a)
    if (i <= n) a(i) = a(i)+b

  end subroutine increment
end module simpleOps_m
```

Multidimensional Example - Host

- Execution Configuration

```
call inc<<<blocksPerGrid, threadsPerBlock>>>(a_d,b)
```

  - Grid dimensions in blocks (**blocksPerGrid**) and block dimensions in threads (**threadsPerBlock**) can be either **integer** or **dim3**

```
type (dim3)
  integer (kind=4) :: x, y, z
end type
```

Exercise - multidim.cuf host code

```
program incrementTest
  use cudafor
  use simpleOps_m
  implicit none
  integer, parameter :: nx=1024, ny=512
  integer :: a(nx,ny), b
  integer, **FIXME** :: a_d(nx,ny)
  type(dim3) :: grid, tBlock

  a = 1; b = 3

  tBlock = dim3(32,8,1)
  grid = **FIXME**
  a_d = a
  call increment<<<**FIXME**>>>(a_d, b)
  a = a_d

  if (any(a /= 4)) then
     write(*,*) '**** Program Failed ****'
  else
     write(*,*) 'Program Passed'
  endif
end program incrementTest
```

Exercise - multidim.cuf device code

```
module simpleOps_m
contains
  **FIXME** subroutine increment(a, b)
    implicit none
    integer :: a(:,:)
    integer, **FIXME** :: b
    integer :: i, j, n(2)

    i = **FIXME**
    j = **FIXME**
    n(1) = size(a,1)
    n(2) = size(a,2)
    if (**FIXME**) a(i,j) = a(i,j) + b
  end subroutine increment
end module simpleOps_m
```