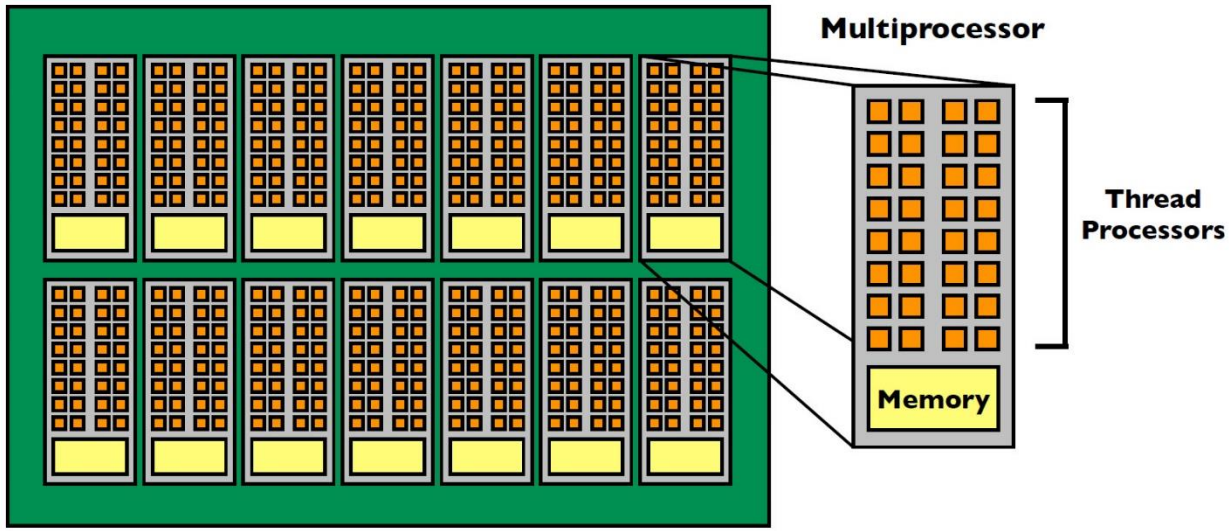
CUDA Fortran

Danh-Tai HOANG,
"Design Principles of Cellular Networks" Group,
Asia Pacific Center for Theoretical Physics (APCTP),
Pohang, KOREA

March 27, 2014

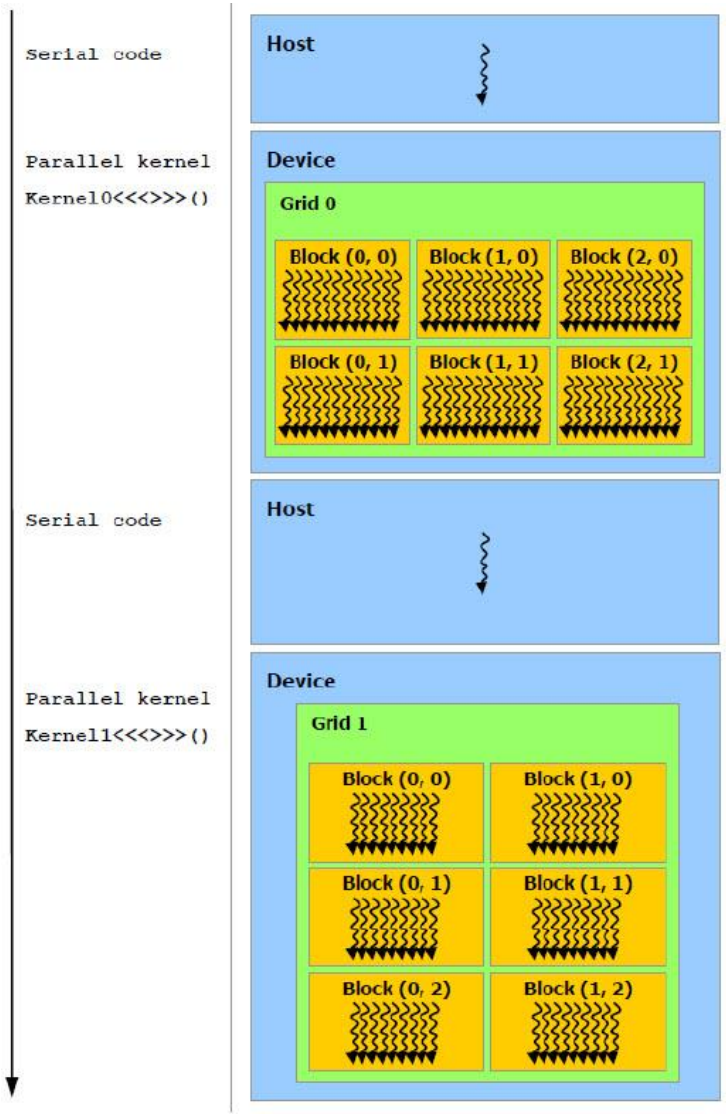
GPU Architecture

Tesla C2075:



- Compute capatibility: 2.0
- Number of multiprocessors: 14 (~ 14 CPU)
- Number of cores: 448 (14 x 32)

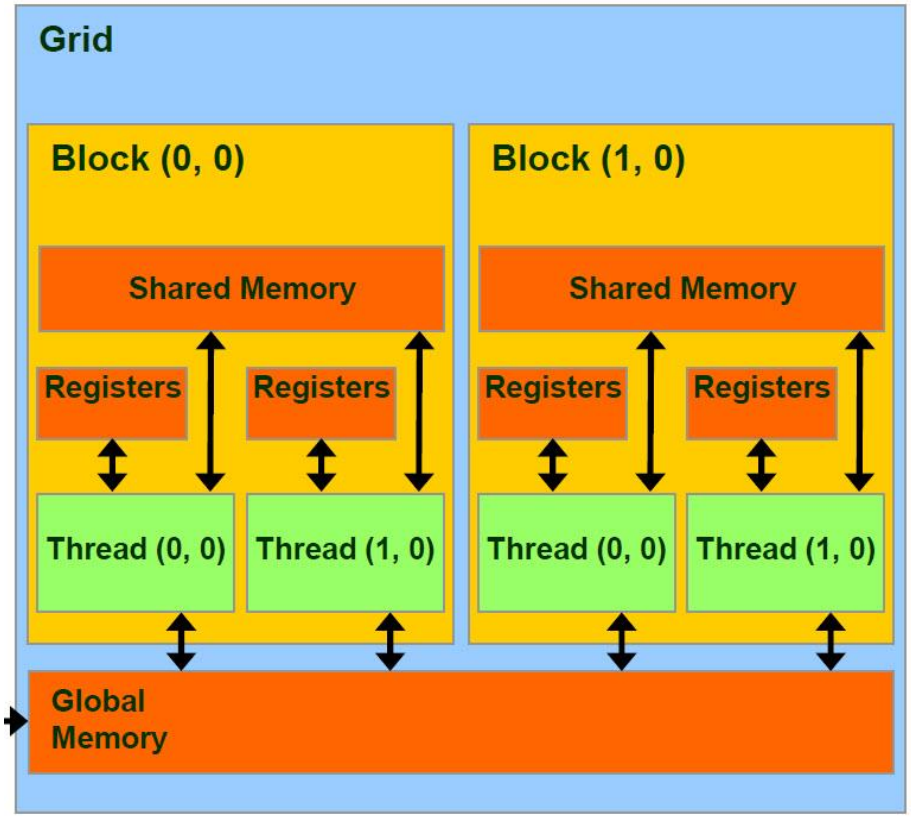
CUDA programming



Execution Model

* Host = CPU
* Device = GPU
* Kernel = function called from the host that runs on the device
- One kernel is executed at a time
- Many threads execute each kernel

- Each kernel is launched in one grid.
- Maximum number of kernel: 16



1. Registers
- Small
- Fast

2. Shared Memory
- Shared among threads in a single block
- On-chip, small, as fast as registers

3. Global Memory
- Kernel input and output data reside here
- Off-chip (host), large
- Shared by all threads
- Inter-Grid communication
- Inter-Kernel sychronization.

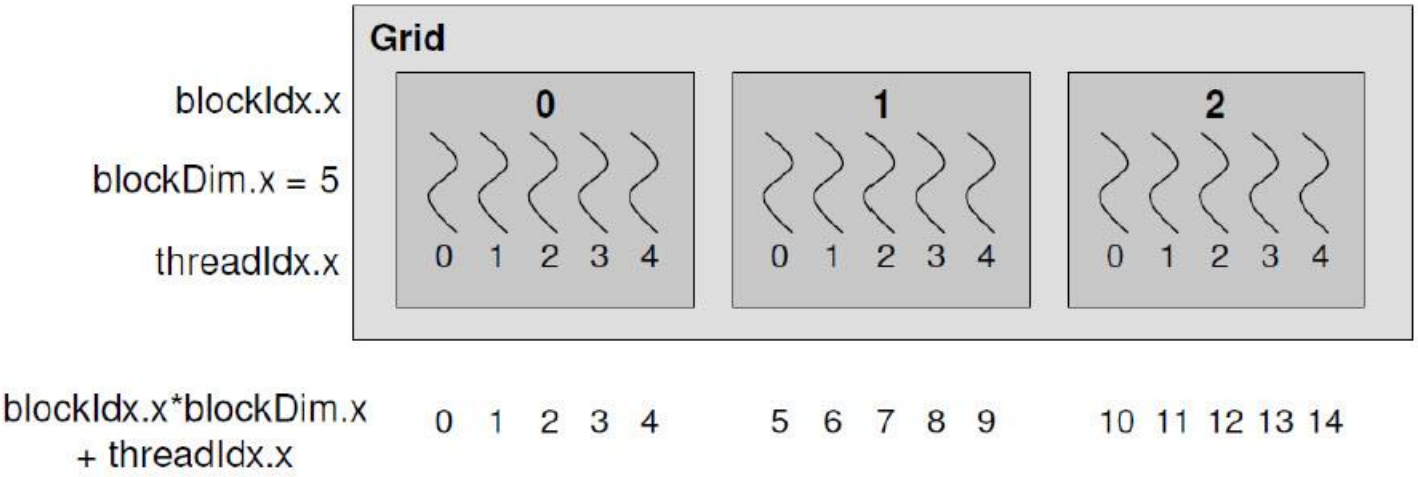
* Host can read & write global memory but not shared memory.

* Threads within a block can:
  + communicate very quickly (shared memory)
  + Synchronize (wait for all threads to catch up)

* Threads in different block cannot cooperate

Thread ID – Block ID

+ threadIdx.x → thread ID within block
+ blockIdx.x → block ID within grid
+ blockDim.x → number of threads/block
+ gridDim.x → number of blocks/grid



Important Numbers

- Compute capatibility: **2.0**
- Number of multiprocessors: **14**
- Number of cores: **448** (= 14 x 32)
- Maximum number of resident blocks / multiprocessor: **8**
- Maximum number of resident warps / multiprocessor: **48**
- Maximum number of resident threads / multiprocessor: **1536**
- Number of 32-bit registers / multiprocessor: **32 KB**
- Maximum number of 32-bit registers / thread: **63**
- Shared memory size / multiprocessor: **48 KB**
- Constant memory size / multiprocessor: **64 KB**
- Maximum Threads / Block: **1024**
- Maximum Block Dimensions: **1024 x 1024 x 64**
- Maximum Grid Dimensions: **65535 x 65535 x 65535** (2^{31}-1)

$\Rightarrow$ All blocks may be not resident at the same time.

Optimizing

- Keep the multiprocessors on the device as busy as possible.
- The number of blocks in a grid should be larger than the number of multiprocessors so that all multiprocessors have at least one block to execute.

    # blocks = 14 x const

- Threads per block should be a multiple of warp size (32)

    # threads = 32 x const

    128 ≤ # threads ≤ 256

Examples

ex1.f90

```
1 MODULE ex1_mod
2 CONTAINS
3 SUBROUTINE sumation(a,b,d)
4 IMPLICIT NONE
5 INTEGER,INTENT(in) :: a(:)
6 INTEGER,INTENT(in) :: b(:)
7 INTEGER,INTENT(out):: d(:)
8 INTEGER :: i,n
9
10 n = size(a)
11 DO i = 1, n
12 d(i) = a(i)+b(i)
13 END DO
14 END SUBROUTINE sumation
15 END MODULE ex1_mod
16
17 !!=========================================
18 PROGRAM ex1_presentation
19 USE ex1_mod
20 IMPLICIT NONE
21 INTEGER, PARAMETER :: n=1000
22 INTEGER :: a(n),b(n),d(n)
23 a = 10
24 b = 20
25
26 call sumation(a,b,d)
27
28 IF(all(d==30)) THEN
29       WRITE(* ,*)'**** OK ****'
30 ELSE
31       WRITE(* ,*)'Program Failed'
32 END IF
33 END PROGRAM ex1_presentation
```

ex1.cuf

```
1 MODULE ex1_mod
2 CONTAINS
3 ATTRIBUTES(GLOBAL) SUBROUTINE sumation(a,b,d)
4 IMPLICIT NONE
5 INTEGER,INTENT(in) :: a(:)
6 INTEGER,INTENT(in) :: b(:)
7 INTEGER,INTENT(out):: d(:)
8 INTEGER :: i
9
10 i = threadIdx%x
11 d(i) = a(i)+ b(i)
12 END SUBROUTINE sumation
13 END MODULE
14
15 !!=========================================
16 PROGRAM ex1_presentation
17 USE cudafor
18 USE ex1_mod
19 IMPLICIT NONE
20 INTEGER, PARAMETER :: n = 1000
21 INTEGER :: a(n),b(n),d(n)
22 INTEGER, device :: a_d(n),b_d(n),d_d(n)
23 a = 10
24 b = 20
25 a_d=a ; b_d=b
26 CALL sumation <<<1,n>>>(a_d,b_d,d_d)
27 d=d_d
28 IF(all(d==30)) THEN
29       WRITE(* ,*)'**** OK ****'
30 ELSE
31       WRITE(* ,*)'Program Failed'
32 END IF
33 END PROGRAM ex1_presentation
```

ex2.cuf

```
 1 MODULE ex2_mod
 2 CONTAINS
 3 ATTRIBUTES(GLOBAL) SUBROUTINE sumation(a,b,d)
 4 IMPLICIT NONE
 5 INTEGER,INTENT(in) :: a(:)
 6 INTEGER,INTENT(in) :: b(:)
 7 INTEGER,INTENT(out):: d(:)
 8 INTEGER :: i,n
 9
10 i = blockDim%x*(blockIdx%x-1)+threadIdx%x
11 n=size(a)
12 IF (i<=n) THEN
13    d(i) = a(i)+ b(i)
14 END IF
15 END SUBROUTINE sumation
16 END MODULE ex2_mod
17
18 !!=========================================
19 PROGRAM ex2_presentation
20 USE cudafor
21 USE ex2_mod
22 IMPLICIT NONE
23 INTEGER, PARAMETER :: n = 5002
24 INTEGER :: a(n),b(n),d(n)
25 INTEGER, device :: a_d(n),b_d(n),d_d(n)
26 INTEGER :: tpb=1000
27 a = 10
28 b = 20
29 a_d=a ; b_d=b
30 CALL sumation <<<ceiling(real(n)/tpb),tpb>>>(a_d,b_d,d_d)
31 d=d_d
32 IF(all(d==30)) THEN
33       WRITE(* ,*)'**** OK ****'
34 ELSE
35       WRITE(* ,*)'Program Failed'
36 END IF
37 END PROGRAM ex2_presentation
```

Multidimensional arrays

ex3.f90

```
1 MODULE ex3_mod
2 CONTAINS
3 SUBROUTINE sumation(a,b,d)
4 IMPLICIT NONE
5 INTEGER,INTENT(in) :: a(:,:)
6 INTEGER,INTENT(in) :: b(:,:)
7 INTEGER,INTENT(out):: d(:,:)
8 INTEGER :: i,j,nx,ny
9
10 nx = size(a,1) ; ny = size(a,2)
11 DO i=1,nx
12 DO j=1,ny
13 d(i,j) = a(i,j)+b(i,j)
14 END DO
15 END DO
16
17 END SUBROUTINE sumation
18 END MODULE ex3_mod
19
20 !!=====================================
21 PROGRAM ex3_presentation
22 USE ex3_mod
23 IMPLICIT NONE
24 INTEGER, PARAMETER :: nx=30000000, ny=4
25 INTEGER :: a(nx,ny),b(nx,ny),d(nx,ny)
26 a = 10
27 b = 20
28
29 call sumation(a,b,d)
30
31 IF(all(d==30)) THEN
32       WRITE(* ,*)'**** OK ****'
33 ELSE
34       WRITE(* ,*)'Program Failed'
35 END IF
36 END PROGRAM ex4_presentation
```

ex3.cuf

```
1 MODULE ex3_mod
2 CONTAINS
3 ATTRIBUTES(GLOBAL) SUBROUTINE sumation(a,b,d)
4 IMPLICIT NONE
5 INTEGER,INTENT(in) :: a(:,:)
6 INTEGER,INTENT(in) :: b(:,:)
7 INTEGER,INTENT(out):: d(:,:)
8 INTEGER :: i,j,nx,ny
9
10 i = blockDim%x*(blockIdx%x-1)+threadIdx%x
11 j = blockDim%y*(blockIdx%y-1)+threadIdx%y
12 nx=size(a,1) ; ny=size(a,2)
13 IF ((i<=nx).and.(j<=ny)) THEN
14    d(i,j) = a(i,j)+ b(i,j)
15 END IF
16 END SUBROUTINE sumation
17 END MODULE ex3_mod
18
19 !!=====================================
20 PROGRAM ex3_presentation
21 USE cudafor
22 USE ex3_mod
23 IMPLICIT NONE
24 INTEGER, PARAMETER :: nx=30000000,ny=4
25 INTEGER :: a(nx,ny),b(nx,ny),d(nx,ny)
26 INTEGER, device :: a_d(nx,ny),b_d(nx,ny),d_d(nx,ny)
27 TYPE(dim3) :: gridDim,blockDim
28 a=10
29 b=20
30 blockDim=dim3(512,2,1)
31 gridDim=dim3(ceiling(real(nx)/blockDim%x),ceiling(real(ny)/blockDim%y),1)
32
33 a_d=a ; b_d=b
34 CALL sumation <<<gridDim,blockDim>>>(a_d,b_d,d_d)
35 d=d_d
36 IF(all(d==30)) THEN
37       WRITE(* ,*)'**** OK ****'
38 ELSE
39       WRITE(* ,*)'Program Failed'
40 END IF
41 END PROGRAM ex3_presentation
```

Shared memory and Synchronization

ex4.cuf

```
 1 MODULE ex_share
 2 IMPLICIT NONE
 3 CONTAINS
 4 ATTRIBUTES(GLOBAL) SUBROUTINE sumation(a,b,e)
 5 IMPLICIT NONE
 6 INTEGER, PARAMETER :: n=1024
 7 INTEGER,INTENT(in) :: a(n)
 8 INTEGER,INTENT(in) :: b(n)
 9 INTEGER,INTENT(out):: e(n)
10 INTEGER :: i,j
11 INTEGER,shared :: d(n)
12 !INTEGER :: d(n)
13
14 i = threadIdx%x
15 d(i) = a(i)+ b(i)
16 CALL syncthreads()
17
18 IF (i/=n) THEN
19     j=i+1
20 ELSE
21     j=1
22 END IF
23
24 e(i) = d(i)+d(j)
25
26 END SUBROUTINE sumation
27 END MODULE ex_share
28 !!==========================================
29 PROGRAM ex_presentation
30 USE cudafor
31 USE ex_share
32 IMPLICIT NONE
33 INTEGER, PARAMETER :: n = 1024
34 INTEGER :: a(n),b(n),e(n)
35 INTEGER, device :: a_d(n),b_d(n),e_d(n)
36
37 a = 10 ; b = 20
38 a_d=a ; b_d=b
39 CALL sumation <<<1,n>>>(a_d,b_d,e_d)
40 e=e_d
41 IF(all(e==60)) THEN
42       WRITE(* ,*)'**** OK ****'
43 ELSE
44       WRITE(* ,*)'Program Failed'
45 END IF
46 END PROGRAM ex_presentation
```