EE382V (17325): Principles in Computer Architecture
Parallelism and Locality
Fall 2007

Lecture 14 – CUDA (and architecture loose ends)

Mattan Erez

The University of Texas at Austin

Outline

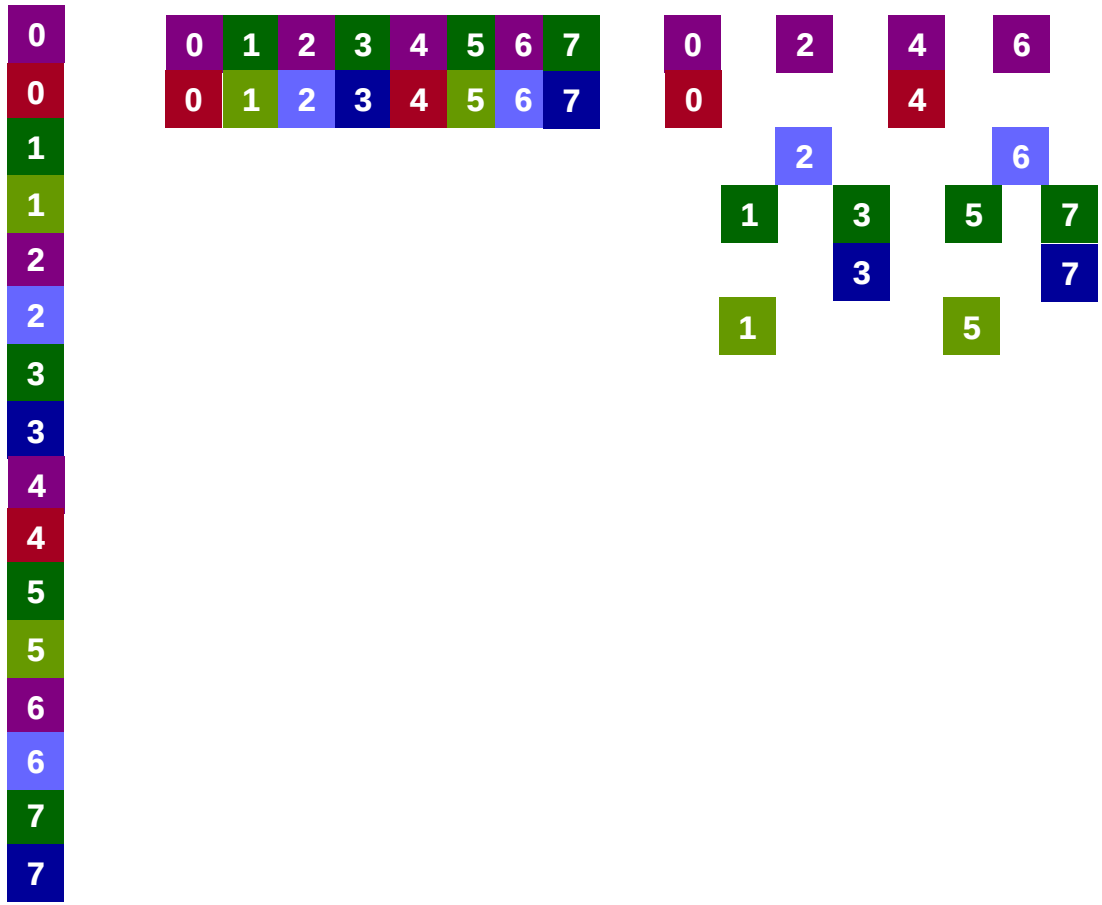
- Control Flow
- Bandwidths

Control Flow

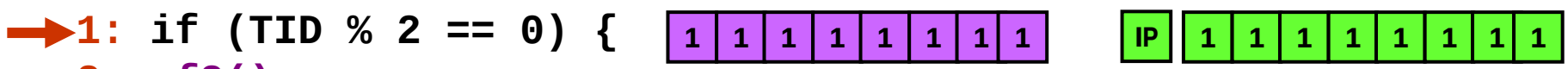
- Recap:
  - 32 threads in a warm are executed in SIMD (share one instruction sequencer)
  - Threads within a warp can be disabled (masked)
    - For example, handling bank conflicts
  - Threads contain arbitrary code including conditional branches
- How do we handle different conditions in different threads?
  - No problem if the threads are in different warps
  - Control ***divergence***
  - ***Predication***

Control Flow Divergence

```
if (TID % 2 == 0) {
 f2();
 if (TID % 4 == 0) {
   f4();
 }
 else {
   f2'();
 }
}
else {
 f(1);
 if (TID % 3 == 0) {
   f3();
 }
 else {
   f1'();
 }
}
```

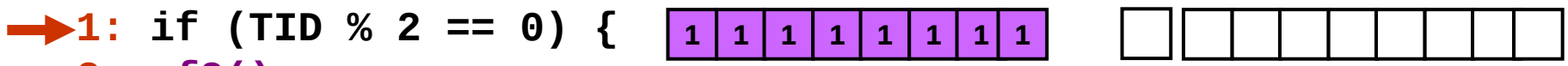


Mask Stack Enables Divergence



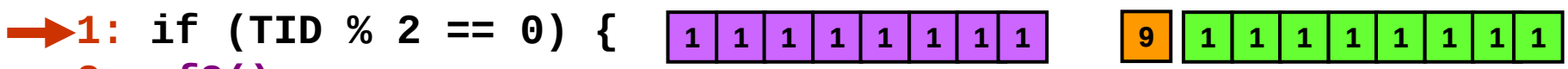

```
1: if (TID % 2 == 0) {
2:  f2();
3:  if (TID % 4 == 0) {
4:   f4();
5:  }
6:  else {
7:   f2'();
8:  }
9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

Mask Stack Enables Divergence




```
1: if (TID % 2 == 0) {
2:  f2();
3:  if (TID % 4 == 0) {
4:   f4();
5:  }
6:  else {
7:   f2'();
8:  }
9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

Mask Stack Enables Divergence



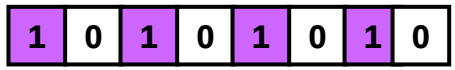

```
1: if (TID % 2 == 0) {
2:  f2();
3:  if (TID % 4 == 0) {
4:   f4();
5:  }
6:  else {
7:   f2'();
8:  }
9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

Mask Stack Enables Divergence

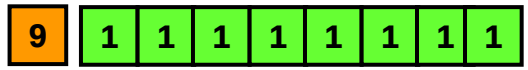
IP

```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

enable mask



stack

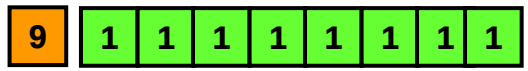


Mask Stack Enables Divergence

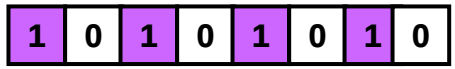
IP

enable mask

stack




```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

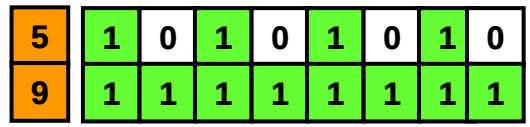


Mask Stack Enables Divergence

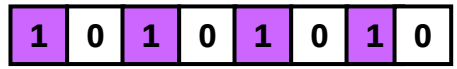
IP

enable mask

stack




```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```



Mask Stack Enables Divergence

IP

```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

enable mask

1	0	0	0	1	0	0	0

stack

5	1	0	1	0	1	0	1	0
9	1	1	1	1	1	1	1	1

Mask Stack Enables Divergence

IP

```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

enable mask

1	0	0	0	1	0	0	0

stack

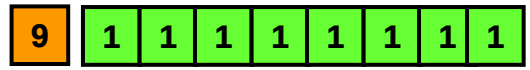
5	1	0	1	0	1	0	1	0
9	1	1	1	1	1	1	1	1

Mask Stack Enables Divergence

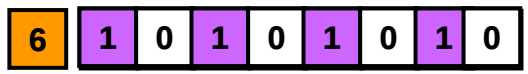
IP

enable mask

stack




```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```



Mask Stack Enables Divergence

IP

```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

enable mask

1	0	1	0	1	0	1	0

stack

8	1	0	1	0	1	0	1	0
9	1	1	1	1	1	1	1	1

Mask Stack Enables Divergence

IP

```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

enable mask

0	0	1	0	0	0	1	0

stack

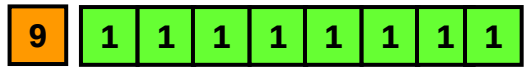
8	1	0	1	0	1	0	1	0
9	1	1	1	1	1	1	1	1

Mask Stack Enables Divergence

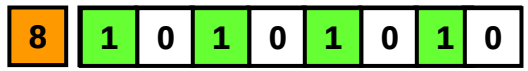
IP

enable mask

stack




```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

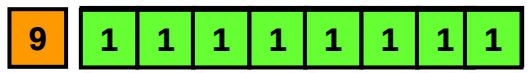


Mask Stack Enables Divergence

IP

```
 1: if (TID % 2 == 0) {
 2:  f2();
 3:  if (TID % 4 == 0) {
 4:   f4();
 5:  }
 6:  else {
 7:   f2'();
 8:  }
 9: }
10: else {
11:  f(1);
12:  if (TID % 3 == 0) {
13:   f3();
14:  }
15:  else {
16:   f1'();
17:  }
18: }
```

enable mask



stack

DirectX 10 specifies 4-deep stack

Predication Eliminates Branches (and Divergence)

```
if (TID % 2 == 0) {
 f2();
 if (TID % 4 == 0) {
   f4();
 }
 else {
   f2'();
 }
}
else {
 f(1);
 if (TID % 3 == 0) {
   f3();
 }
 else {
   f1'();
 }
}
```

Predication Eliminates Branches (and Divergence)

```
    p1 = (TID % 2 == 0)
p1  f2();
```

```
if (TID % 2 == 0) {
 f2();
 if (TID % 4 == 0) {
   f4();
 }
 else {
   f2'();
 }
}
else {
 f(1);
 if (TID % 3 == 0) {
   f3();
 }
 else {
   f1'();
 }
}
```

Predication Eliminates Branches (and Divergence)

```
    p1 = (TID % 2 == 0)
p1  f2();
p1  p2 = (TID % 4 == 0)
p2  f4();
```

```
if (TID % 2 == 0) {
 f2();
 if (TID % 4 == 0) {
   f4();
 }
 else {
   f2'();
 }
}
else {
 f(1);
 if (TID % 3 == 0) {
   f3();
 }
 else {
   f1'();
 }
}
```

Predication Eliminates Branches (and Divergence)

```
    p1 = (TID % 2 == 0)
p1  f2();
p1  p2 = (TID % 4 == 0)
p2  f4();

p1  p3 = !p2
p3  f2'();



    p4 = !p1
p4  f(1);
p4  p5 = (TID % 3 == 0)
p5  f3();

p4  p6 = !p5
p6  f1'();
```

```
if (TID % 2 == 0) {
  f2();
  if (TID % 4 == 0) {
    f4();
  }
  else {
    f2'();
  }
}
else {
  f(1);
  if (TID % 3 == 0) {
    f3();
  }
  else {
    f1'();
  }
}
```

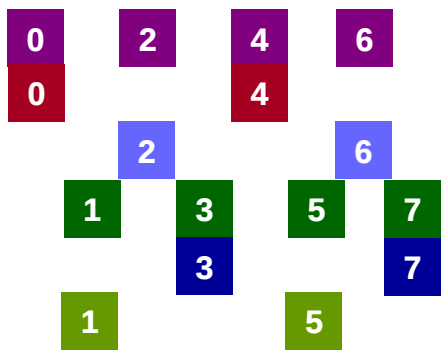
Equivalence of Divergence and Predication

```
   p1 = (TID % 2 == 0)
p1 f2();
p1 p2 = (TID % 4 == 0)
p2 f4();

p1 p3 = !p2
p3 f2'();


   p4 = !p1
p4 f(1);
p4 p5 = (TID % 3 == 0)
p5 f3();

p4 p6 = !p5
p6 f1'();
```




```
if (TID % 2 == 0) {
 f2();
 if (TID % 4 == 0) {
   f4();
 }
 else {
   f2'();
 }
}
else {
 f(1);
 if (TID % 3 == 0) {
   f3();
 }
 else {
   f1'();
 }
}
```

When to Predicate and When to Diverge?

- Divergence
  - No performance penalty if all warp branches the same way
  - Some extra HW cost
  - Static partitioning of stack resources (to warps)

- Predication
  - Always excecute all paths
  - Expose more ILP
  - Add predication registers to instruction encoding

- Selects – software predication
  - Simpler HW and just as flexible mode
  - Simple instruction encoding
  - Need to use more registers and insert select instructions

Outline

- Control Flow
- Bandwidths
- CUDA

- Most slides courtesy Massimiliano Fatica (NVIDIA)

Bandwidths of GeForce 8800 GTX

- Frequency
  - 575 MHz with ALUs running at 1.35 GHz
- ALU bandwidth (GFLOPs)
  - (1.35 GHz) X (16 SM) X ((8 SP)X(2 MADD) + (2 SFU)) = ~388 GFLOPs
- Register BW
  - (1.35 GHz) X (16 SM) X (8 SP) X (4 words) = 2.8 TB/s
- Shared Memory BW
  - (575 MHz) X (16 SM) X (16 Banks) X (1 word) = 588 GB/s
- Device memory BW
  - 1.8 GHz GDDR3 with 384 bit bus: 86.4 GB/s
- Host memory BW
  - PCI-express: 1.5GB/s or 3GB/s with page locking