

# CUDA Threads

Bedřich Beneš, Ph.D.  
Purdue University  
Department of Computer Graphics

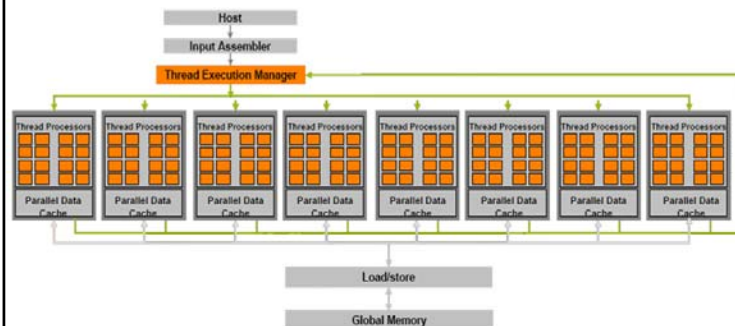


## Terminology

- *Streaming Multiprocessor (SM)*
- *Streaming Processors (SP)*  
also called *CUDA Cores*
- A SP processes threads belonging to a block (shared resources)

© Bedřich Beneš

## Terminology





© Bedřich Beneš

## How it works

- 1) Grid is launched
- 2) Blocks are assigned to streaming multiprocessors (SM) on block-by-block basis in arbitrary order (*scalability*) (Each SM can process more blocks)  
e.g., GT200 can do max 8 blocks or max 1024 threads per SM



© Bedřich Beneš

## How it works

- 3) An assigned block is partitioned into *warps*.  
Their execution is interleaved
- 4) Warps are assigned to SM  
(one thread to one SP)
- 5) Warps can be delayed if idle for some  
reason (waiting for memory)



© Bedrich Benes

## Basic Considerations

- the size of a block is limited to 512 threads  
`blockDim(512,1,1)`  
`blockDim(8,16,2)`  
`blockDim(16,16,2)`
- kernel can handle up to  
65,536x65,536 blocks



© Bedrich Benes

## G80 Architecture

has **16 SMs**  
each can process  
 $\leq 8$  blocks  
or  
 $\leq$  **768 threads**  
max:  $8 \times 16 = 128$  CUDA Cores (SPs)  
max:  $16 \times 768 = 12,288$  threads

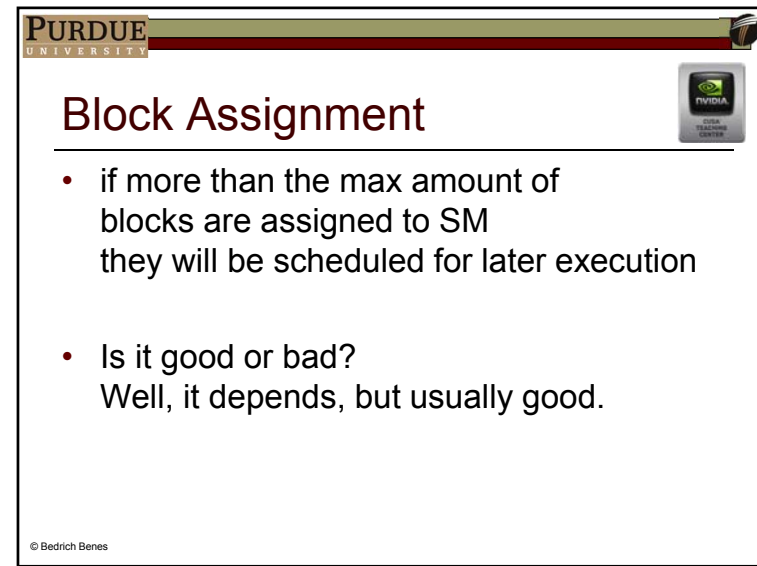
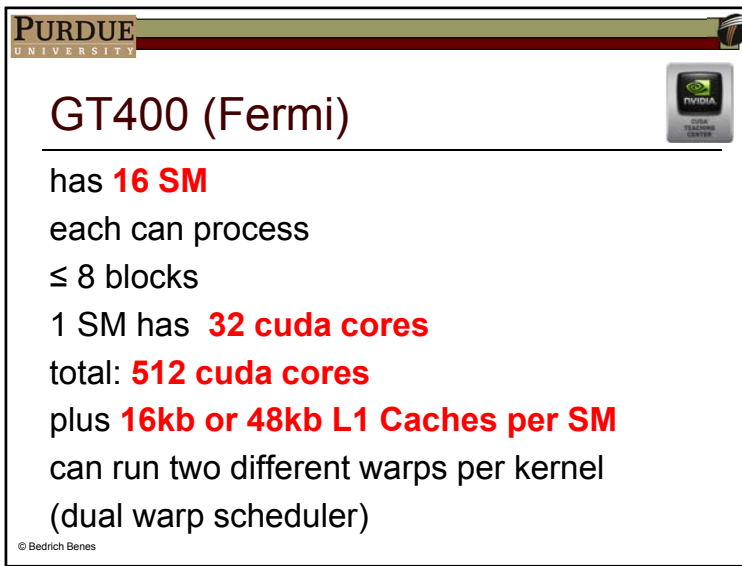
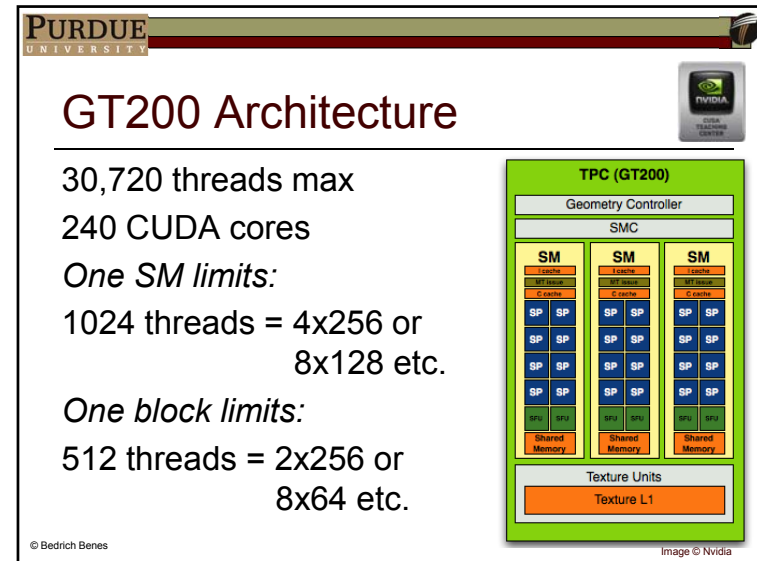
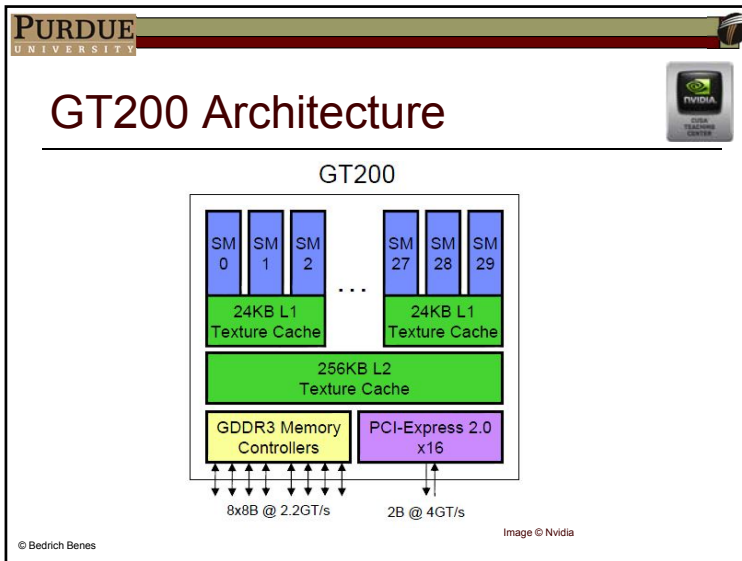
© Bedrich Benes






## GT200 Architecture

has **30 SMs**  
each can process  
 $\leq 8$  blocks  
or  
 $\leq$  **1024 threads**  
max:  $8 \times 30 = 240$  CUDA Cores (SPs)  
max:  $30 \times 1,024 = 30,720$  threads

© Bedrich Benes





## Comparison

	G80	GT200	GT500
Transistors	681 milions	1.4 billions	3.0 billions
CUDA Cores	128	240	512
Warp scheduler per SM	1	1	2
Shard Memory per SM	16kB	16kB	16 or 48kB
L1 cache per SM	None	None	16 or 48kB
L2 cache per SM	None	None	768 kB
Load/store address width	32b	32b	64b



© Bedrich Benes

## Warps

- A thread block is divided into *warps*
- A block of 32 threads (hw dependent and can change)
- Warps are scheduling units of SM**
- $\text{warp}_0: t_0, t_1, \dots, t_{31}$   
 $\text{warp}_1: t_{32}, t_{32}, \dots, t_{63}$



© Bedrich Benes

## Warps

- Example:  
3 blocks assigned to SM, each with 128 threads.  
*How many warps we have in the SM?*
- $128 \text{ threads} / 32 \text{ (warp length)} = 4 \text{ warps}$
- $4(\text{warps}) \times 3 \text{ (blocks)} = 12 \text{ warps at the same time}$

© Bedrich Benes


## Warps

- Example2:  
*How many warps in the GT200?*
- $1024 \text{ threads} / 32 \text{ (warp length)} = 32 \text{ warps}$

© Bedrich Benes

PURDUE UNIVERSITY

## Warp Assignment




- one thread is assigned to one SP
- SM has 8 SPs
- warp has 32 threads
- so a **warp is executed in four steps**

© Bedrich Benes

PURDUE UNIVERSITY

## Warps – latency hiding



- Why do we need so many warps if there are just 8 CUDA cores in SM (GT200)?


**Latency hiding:**

- a warp executes a global memory read instruction that delays it for 400 cycles
- any *other warp* can be executed in the meantime
- if more than one is available - priorities

© Bedrich Benes

PURDUE UNIVERSITY

## Warps – processing




- A warp is SIMT (single instruction multiple thread) all run in parallel and the same instruction
- Two warps are MIMD can do branching, loops, etc.
- Threads within one warp do not need synchronization – they run the same time instruction

© Bedrich Benes

PURDUE UNIVERSITY

## Warps – zero-overhead



**Zero-overhead thread scheduling**

- having many warps available, the selection of warps that are ready to go keeps the SM busy (no idle time)
- that is why, caches are not usually necessary

© Bedrich Benes

PURDUE UNIVERSITY

## Example - granularity

- Having GT200 and matrix multiplication.  
Which tiles are the best 4x4, 8x8, 16x16, or 32x32?

© Bedrich Benes

PURDUE UNIVERSITY

## Example - granularity

- 4x4 will need 16 threads per block  
SM can take up to 1024 threads  
We can take  $1024/16=64$  blocks  
BUT! The SM is limited to 8 blocks  
There will be  $8*16=128$  threads in each SM  
 $128/32=4 \rightarrow$  8 warps, but each half full  
heavily underutilized !  
(fewer warps to schedule)

© Bedrich Benes

PURDUE UNIVERSITY

## Example - granularity

- 8x8 will need 64 threads per block  
SM can take up to 1024 threads  
We can take  $1024/64=16$  blocks  
BUT! The SM is limited to 8 blocks  
There will be  $8*64=512$  threads in each SM  
 $512/32=16$  warps  
still underutilized !  
(fewer warps to schedule)

© Bedrich Benes

PURDUE UNIVERSITY

## Example - granularity

- 16x16 will need 256 threads per block  
SM can take up to 1024 threads  
We can take  $1024/256=4$  blocks  
The SM can take it 2x  
There will be  $8*64=512$  threads in each SM  
 $512/32=16$   
full capacity and a lot of warps to schedule

© Bedrich Benes

**PURDUE UNIVERSITY**

## Example - granularity

- 32x32 will need 1024 threads per block  
a block (GT200) can take max 512  
Not even one will fit in the SM

(not true in GT400)

© Bedrich Benes

**PURDUE UNIVERSITY**

## Example - granularity

- granularity does *not* automatically mean a good performance
- depends on using shared memory, branching, loops, etc.
- but it does mean low latency
- Blocks (resp. # of threads in block) should be multiples 32 for better alignment

© Bedrich Benes

**PURDUE UNIVERSITY**

## Warps/block alignment

- 1D Case**  
block of 100 threads – how many warps?  
 $100/32 = 3 + 1/4$

the last warp will be occupied entirely, but only the 8 threads will have meaning

© Bedrich Benes

**PURDUE UNIVERSITY**

## Warps/block alignment

- 2D Case**  
blockDim(9,9)  
81 threads  
 $100/32 = 2$  warps  
and 17 threads

© Bedrich Benes

**Warps/block alignment**

- 3D Case**  
 blockDim(4,4,5)  
 80 threads  
 $100/32=2$  warps  
 and 16 threads

© Bedrich Benes

**Warp execution**

- SIMT – single instruction, multiple threads  
 the same instruction is broadcasted to all threads and execute at the same time in the SM.
- All SPs in the SM  
 execute the same instruction.

© Bedrich Benes

**Thread Divergence**

- How can all threads execute the same instruction if we have the “if” command?

Example:

```
if (threadIdx.x < 10)
    {a[0]=10;}
else {a[1]=10;}
```

Threads [0-9] will do “then”  
 the others will do “else”  
 This is called **thread divergence**

© Bedrich Benes



**Thread Divergence**

- The compiler will unroll both branches and the GPU will perform *both* branches. Then in the first pass, else in the second.
- But not all ifs cause thread divergence!

```
a=tex2D(tex,u,v);
if (a<0.5)
    {a[0]=10;}
else {a[1]=10;}
```

© Bedrich Benes





## Thread Divergence

- What causes thread divergence?

- If statements with functions of threadIdx
- Loops with functions of threadIdx

ifs are expensive anyway...

© Bedrich Benes

## Thread Divergence



Example:

```
for (int i=0;i<threadIdx.x;i++)
    a[i]=i;
```

All loops that should finished will finish, but

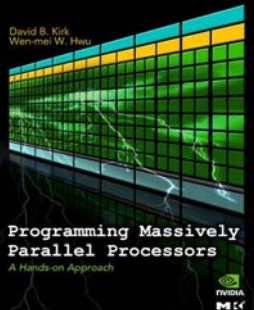
the GPU will iterate for the others till the end

© Bedrich Benes

## Reading

- NVIDIA CUDA Programming Guide
- Kirk, D.B., Hwu, W.W.,  
*Programming Massively  
Parallel Processors*,  
NVIDIA,  
Morgan Kaufmann 2010



© Bedrich Benes