HPC: GPU

UROŠ LOTRIČ

History

Graphics accelerators

• 2D

- bit-blit (block image transfer without blinking),
- line graphics (Bresenham),
- Clipping of unvisible part of image

• 3D

- object is modelled with polygons which are projected to the screen
- wire frame, invisible edges, painting, lightning, textures, shading
- lots of mathematical operations: projections and rotations









History

Pixel shaders

- 2D acceleration
- Determine pixels colours
- Vertex shaders
- 3D acceleration
- Perform geometric transformations to map a vertex to screen coordinates

Problem

- Image editing involves pixel shaders
- CAD applications involve mainly vertex shaders

Solution

- CUDA (Compute Unified Device Architecture)
- Streaming processors: general shaders which can be used in 2D and 3D
- Added floating point operations, additional general purpose instructions

CPU vs GPU

Performance increase in last decade



CPU vs GPU

Complex control logic

Large caches

Optimized for serial operations

All types of applications, also tree operations, recursion



High portion of simpler processing units (ALUs), highly parallel

Build for parallel operations

High latency tolerance

Applications with high level of parallelism



GPU programming

Unusual programming model, conceptually different from CPU

Re-coding

• Some new approaches fight against it

Philosophy

- Create unlimited number of threads
- Threads are dynamically scheduled on hardware

Applications which excel on GPU

- High level of data parallelism
- Huge quantities of data
- 2D/3D structures with limited dependencies

Designed to execute thousands of arithmetic operations simultaneously

- To put a lot of processing power to one die
- We need a slimmer design
- All complex and large units are removed
 - Cache, branch predictor, out-of-order logic
- Control logic (fetch/decode) shared among ALUs
 - ALUs process the same instruction on different data
- Memory shared among ALUs to be able to exchange data

Compute unit (multiprocessor)

- Basic computational building block
- Is equivalent to cores in CPU
- Is composed of many processing elements
- Entirely new instruction set, simpler for compiler, more constant performance
- SIMD parallelism
- Do not support branch prediction and speculative execution
- Have less cache then CPU
- Terminology: stream multiprocessor (Nvidia), SIMD engine (AMD)

Processing element (core, shader processor)

- Is equivalent to ALU in CPU
- They share fetch/decode logic
- ALUs run the same instruction on different data
- Terminology: stream processor (Nvidia), ALU (AMD)

Tens of compute units

Striving towards large number of PEs to efficiently hide memorylatency

 Completely different from CPUs where caches and out-of-order execution is used for latency hiding

Example

• Nvidia tesla K40m has 15 CUs with 192 PEs each

Tesla

- 8 PE (SP Streaming processor)
- 2 SFU Special Function Units
- 1 warp scheduler

Fermi

- 32 Pes (Cores)
- 2 warp schedulers
- 16 LD/ST units

м	Streaming Multiprocessor (SM)										
	I cache										
	MT issue										
	C cache										
	SP	SP									
	SP	SP									
	SP	SP									
	SP	SP									
	SFU	SFU									
	Shared Memory										

SM												
Instruction Cache												
War	p Sched	uler	Warp Scheduler									
Dis	spatch U	nit	Dispatch Unit									
÷												
Register File (32,768 x 32-bit)												
	-+	-+-	-		-							
Core	Core	Core	Core	LD/ST	SELL							
Core	Core	Core	Core	LD/ST	aru							
core	Core	Core	Core	LD/ST								
Core	Core	Core	Core	LD/ST								
				LD/ST	SEU							
Core	Core	Core	Core	LD/ST								
				LD/ST								
Core	Core	Core	Core	LD/ST								
				LD/ST	SFU							
Core	Core	Core	Core	LD/ST								
				LD/ST	<u> </u>							
Core	Core	Core	Core	LD/ST								
		Щ		LD/ST	SFU							
Core	Core	Core	Core	LD/ST								
				LD/ST								
	int (erconne	ct Netwo	rk	8889							
64 KB Shared Memory / L1 Cache												
Uniform Cache												
Tex Tex Tex Tex												
	Texture Cache											
	PolyMorph Engine											
Verte	Fetch	Tesse	llator	Transfo	orm							
	Attribut	te Setup	Stream	Output								

Kepler

- 192 PE
- 32 SFU
- 32 LD/ST units
- 64 DP (double precsision) units
- 4 warp schedulers



sмx	MX Instruction Cache																		
	War	n Sch	eduler			Wa	m Scher	ins Juler	tructi	Warp Scheduler Warp Schedu								lulor	
Di	spatcl	h	Dispat	tch	Di	Dispatch Dispatch				Dispatch Dispatch					Dispatch Dispatch				ch
+ +			+ +					+		+			+		+	_			
Register File (65,536 x 32-bit)																			
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
									0511										0511
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	550	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
							CAMP	Inter	conne	ct Net	work								
	64 KB Shared Memory / L1 Cache																		
							48 K	B Rea	ad-Ò	nly D	ata C	ache	•						
	Tex		Tex			Tex		Tex	t.		Tex		Tex	1		Tex		Tex	
	Tex		Tex	ζ		Tex		Tex	ι.		Tex		Tex			Tex		Tex	

• 128 PE

Pascal

- 64 PE
- 32 DP units
- GPU-GPU memory transfers
- Half-precision



SM							Instruct	ion Cacho							
		1	nstructio	on Buffe	r		mstruct			1	nstructi	on Buffe	r		
Warp Scheduler											Warp Se	cheduler	2		
Dispatch Unit Dispatch Unit								Dispato	:h Unit			Dispat	ch Unit		
Register File (32,768 x 32-bit)								Regist	er File (32,768 x	32-bit)				
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU
							Texture	/ L1 Cache	1						
	Te	x			Т	ex			Т	∍x				Гех	
						6	4KB Sha	red Memo	ry						

Turing

- 64 PE
 - int 32 bit, fp 32 bit
- 8 tensor cores
 - matrix operations
 - fp 16 bit
 - int 16, 8, 4 bit
 - RT core
 - Ray Tracing



Volta and Ampere

• More advanced tensor cores

							L1 Instru	ctic	on Cache								
		L0 li	nstruc	tion C	ache						L0 li	nstruc	tion C	ache			
	Wa	rp Sch	nedule	r (32 t	hread	/clk)			Warp Scheduler (32 thread/clk)								
	Di	spatc	h Unit	(32 th	read/o	cik)				Di	spatc	h Unit	(32 th	read/o	clk)		
	Reg	jister	File ('	16,384	4 x 32	2-bit)				Reg	ister	File ('	16,38	4 x 32	2-bit)		
FP64	INT	INT	FP32	FP32	H				FP64	INT	INT	FP32	FP32	H			
FP64	INT	INT	FP32	FP32	+				FP64	INT	INT	FP32	FP32	\vdash			
FP64	INT	INT	FP32	FP32	Ħ				FP64	INT	INT	FP32	FP32				
FP64	INT	INT	FP32	FP32	TEN	TENSOR TENSOR			FP64	INT	INT	FP32	FP32	TEN	ISOR	TENSOR	
FP64	INT	INT	FP32	FP32	cc	DRE	CORE		FP64	INT	INT	FP32	FP32	cc	ORE	CORE	
FP64	INT	INT	FP32	FP32	H				FP64	INT	INT	FP32	FP32	-			
FP64	INT	INT	FP32	FP32	H				FP64	INT	INT	FP32	FP32	-			
FP64	INT	INT	FP32	FP32	H				FP64	INT	INT	FP32	FP32				
LD/ LD/	LD/	LD/	LD/	LD/	LD/	LD/	SFU		LD/ LD/	LD/	LD/	LD/	LD/	LD/	LD/	SFU	
		101		1 O											_		
	_	LUII	nstruc	tion C	ache						L0 li	nstruc	tion C	ache			
	Wa	rp Sch	nstruc	uon C r (32 t	ache hread	/clk)				Wa	L0 In rp Sch	nstruc nedule	tion C r (32 t	ache hread	l/clk)		
	Wa Di	rp Sch spatcl	nstruc nedule h Unit	uon C r (32 ti (32 th	ache hread read/o	/ <mark>clk)</mark> clk)				Wa Di	L0 In rp Sch spatc	nstruc nedule h Unit	tion C r (32 t (32 th	ache hread read/e	l/clk) clk)	_	
	Wa Di Reg	rp Sch spatcl jister	nstruc hedule h Unit File ('	16,384	ache hread read/o 4 x 32	/clk) clk) 2-bit)				War Di Reg	LO In rp Sch spatc jister	nstruc nedule h Unit File ('	tion C r (32 t (32 th 16,384	ache hread read/d 4 x 32	l/clk) clk) 2-bit)		
FP64	War Di Reg	rp Sch spatc jister	nstruc hedule h Unit File (' FP32	r (32 th (32 th 16,384 FP32	ache hread read/o 4 x 32	/clk) clk) 2-bit)			FP64	Wai Di Reg	LO In rp Sch spatc jister INT	nstruc nedule h Unit File (' FP32	tion C r (32 t (32 th 16,384 FP32	ache hread read/4 4 x 32	l/clk) clk) 2-bit)		
FP64 FP64	War Di Reg INT	rp Sch spatc jister INT	nedule h Unit File (' FP32 FP32	r (32 th (32 th 16,384 FP32 FP32	ache hread read/o 4 x 32	VcIk) cIk) 2-bit)			FP64 FP64	Wan Di Reg INT	L0 II rp Sch spatc iister INT INT	nstruc nedule h Unit File (' FP32 FP32	tion C r (32 t (32 th 16,38 FP32 FP32	ache hread/ read/ 4 x 32	l/clk) clk) 2-bit)		
FP64 FP64 FP64	Wa Di Reg INT INT	INT	FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32 FP32	ache hread read/d 4 x 32	/clk) clk) 2-bit)			FP64 FP64 FP64	Wa Di Reg INT INT	LO II rp Sch spatc lister INT INT	FP32 FP32 FP32	tion C r (32 t (32 th 16,384 FP32 FP32 FP32	ache hread read/ 4 x 32	l/clk) clk) 2-bit)		
FP64 FP64 FP64 FP64	Wai Di Reg INT INT INT	INT	File (* FP32 FP32 FP32 FP32 FP32	r (32 tl (32 th 16,384 FP32 FP32 FP32 FP32	ache hread read/o 4 x 32	I/clk) clk) 2-bit) ISOR	TENSOR		FP64 FP64 FP64 FP64	Wai Di Reg INT INT INT	LO II rp Sch spatc ister INT INT INT	nstruc nedule h Unit File (' FP32 FP32 FP32 FP32	tion C r (32 t (32 th 16,38 FP32 FP32 FP32 FP32	ache hread read/4 4 x 32	I/clk) clk) 2-bit) ISOR	TENSOR	
FP64 FP64 FP64 FP64 FP64	Wan Di Reg INT INT INT INT	INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32	ache hread fread/d t x 32	//clk) clk) 2-bit) ISOR DRE	TENSOR		FP64 FP64 FP64 FP64 FP64	War Di Reg INT INT INT INT	LO II rp Sch spatci ister INT INT INT INT	nstruc nedule h Unit File (* FP32 FP32 FP32 FP32 FP32	tion C r (32 t (32 th 16,38 FP32 FP32 FP32 FP32 FP32	ache hread read/o 4 x 32 TEN CC	l/clk) clk) 2-bit) ISOR DRE	TENSOR	
FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Reg INT INT INT INT INT	INT INT INT INT INT INT	File (* FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32	acho hread/ 4 x 32 TEN CC	/clk) clk) 2-bit) ISOR DRE	TENSOR		FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Reg INT INT INT INT INT	LO II TP Sch Spatci Spatci IINT INT INT INT INT INT INT	hstruc hedule h Unit File (1 FP32 FP32 FP32 FP32 FP32 FP32	tion C r (32 t (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32	ache hroad read/d 4 x 32	l/clk) clk) 2-bit) ISOR DRE	TENSOR	
FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wai Di Reg INT INT INT INT INT	INT INT INT INT INT INT INT INT	nedule h Unit File (' FP32 FP32 FP32 FP32 FP32 FP32	r (32 ti (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32	acho hread/ 4 x 32	/clk) clk) 2-bit) SSOR	TENSOR		FP64 FP64 FP64 FP64 FP64 FP64 FP64	Waa Di Reg INT INT INT INT INT	INT INT INT INT INT INT INT	netruc nedule h Unit File (' FP32 FP32 FP32 FP32 FP32	tion C (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread/ 4 x 32	i/clk) clk) 2-bit) ISOR	TENSOR	
FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wai Di Reg INT INT INT INT INT INT	INT INT INT INT INT INT INT INT	File (' FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread/ 4 x 32 TEN CC	/clk) clk) 2-bit) ISOR	TENSOR		FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Regg INT INT INT INT INT INT	INT INT INT INT INT INT INT INT INT	Herruc Hedule h Unit File (' FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C (32 th (32 th 16,38 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread/4 4 x 32 TEN CC	I/clk) clk) 2-bit) ISOR DRE	TENSOR	
FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wai Di Reg INT INT INT INT INT INT INT INT INT	INT INT INT INT INT INT INT INT INT INT	redule h Unit File (' FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Ache hread 4 x 32 TEN CC	LD/	TENSOR CORE		FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Regg INT INT INT INT INT INT INT INT INT	LO I p Scl spatc ister INT INT INT INT INT INT INT INT	netruc nedule h Unit File (* FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	Ache hread/ 4 x 32 TEN CC	LD/	TENSOR CORE	
FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wai Di Reg INT INT INT INT INT INT INT INT INT	INT INT INT INT INT INT INT INT INT INT	edule h Unit File (' FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	(32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread/ 4 x 32 TEN CC	/clk) 2-bit) SSOR LD/ ST 128K(TENSOR CORE SFU		FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Regg INT INT INT INT INT INT INT INT INT INT	INT INT INT INT INT INT INT INT INT INT	netruc nedule h Unit File (' FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread/ 4 x 32 TEN CC	LD/	TENSOR CORE	

SM	L1 Instru	ction Cache									
L0 In	nstruction Cache	L0 Instruction C	ache								
Warp Sch	eduler (32 thread/clk)	Warp Scheduler (32 thread/clk)									
Dispatch	u Unit (32 thread/clk)	Dispatch Unit (32 thread/clk)									
Register I	File (16,384 x 32-bit)	Register File (16,384 x 32-bit)									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64	TENSOR CORE								
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
LD/ LD/ LD/ LD/ ST ST ST ST ST	LD/ LD/ LD/ LD/ ST ST ST ST SFU	LD/ LD/ LD/ LD/ LD/ LD/ LD/ ST ST ST ST ST ST ST	LD/ LD/ ST ST SFU								
L0 In	nstruction Cache	L0 Instruction C	tache								
Warp Sch	eduler (32 thread/clk)	Warp Scheduler (32 thread/clk)									
Dispatch	n Unit (32 thread/clk)	Dispatch Unit (32 thread/clk)									
Register	File (16,384 x 32-bit)	Register File (16,384 x 32-bit)									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64 TENSOR CORE	INT32 INT32 FP32 FP32 FP64	TENSOR CORE								
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
INT32 INT32 FP32 FP32	FP64	INT32 INT32 FP32 FP32 FP64									
LD/ LD/ LD/ LD/ ST ST ST ST	LD/ LD/ LD/ LD/ ST ST ST ST ST	LD/ LD/ LD/ LD/ LD/ LD/ LD/ ST ST ST ST ST ST ST	LD/ LD/ ST ST SFU								
192KB L1 Data Cache / Shared Memory											
		Tex Tex									

Memory hierarchy

Compute unit

- Private memory (registers)
 - Kepler 64k x 32-bit
 - Divided among processing elements
 - Access time 1 cycle
- Local (shared) memory
 - Available to all processing elements
 - Kepler 64k
 - Access times 1-32 cycles



Memory hierarchy

Compute device

- Global Memory
 - Read and write
 - Nvidia K40m 12 GB GDDR5
- Constant Memory
 - Read only
- Both memories
 - Accessible from host and device
 - Cached
 - Access time ~ 500 cycles



Hierarchical organization that suits processor organization

- Thread-block
- a group of threads

Thread

- Exclusive access to private memory (registers) and local memory
- Has access to the dedicated private and shared memory
- Has access to global and constant memory

Threads in a thread-block

- Execute on the same computing unit
- All access shared memory
- Can synchronize at barrier
- Threads in different thread-blocks cannot always synchronize (barrier)

Step 1: thread-block scheduling

- Thread-blocks execute independently of each other
- One or more thread-blocks can be assigned to one compute unit
- The order of thread-block execution
 - Determined by hardware scheduler
 - If there are more thread-blocks than compute units, some thread-blocks may not go to execution before other finish
- Number of threads per thread-block is specified by a programmer
 - Influences number of registers allocated to a thread

Step 2: thread scheduling within a thread-block

- Warps are groups of consecutive threads
 - 32 (Nvidia), 64 (AMD)
- Compute unit schedules warps to processing elements
- Threads in a warp execute the same program
 - SIMT
 - Are free to branch and execute independently, each thread has its own program counter
 - Warp executes one instruction at a time
 - In case of divergence, processing elements execute each path sequentially, masking work-items not in the path
 - For good performance we should avoid branching within a warp

Step 2: thread scheduling within a thread-block

• Latency hiding

- Number of clock-cycles needed to issue next warp for execution
- A warp can wait to get operands (memory) or that all work-items reach a barrier (synchronization)
- Scheduler can execute any warp that is ready
- Full utilization when a warp is ready in each clock-cycle (latency is completely hidden)
- Switching between warps has no cost
 - Warp execution context (PC, registers, ...) is maintained on a compute unit for the entire warp lifetime

Thread scheduling and memory

Private memory (registers)

- Is equally split to all threads executing on a compute unit
- More threads per thread-block we have, less private memory belongs to each

Global memory

- Coalesced access
 - One segment (128 B) can be delivered in one transaction
 - To improve performance, threads in a warp should access contiguous elements in memory to minimize the number of transactions

Constant memory

• Supports broadcasting of a single value to all threads in a warp in one cycle

GPU programming frameworks

Nvidia CUDA

- CUDA C, only for Nvidia hardware
- Firmly tight to Nvidia hardware
- Installed in majority of HPC systems

OpenCL

- Supports GPUs of different vendors
- Supports also CPUs, FPGAs, ...
- Does not have so many features as CUDA
- OpenCL C to write kernels C-like functions executed by work-items

New approaches

- Intel One API
- OpenMP 4.5

Execution model

Offload model

- Host copies data to device
- Host triggers execution on device
- Device executes program (kernel) in parallel
- Host transfers results from device
- Host executes serial code
- Device executes parallel code

Programming models fit to the hardware hierarchy



Thread organization

Problem description

- Number of thread-blocks and number of threads within a thread-block determines problem size – how many threads will execute the kernel
- 1D, 2D, or 3D thread index space
- Each thread executes the same kernel for one point in the index space
- Synchronization is only possible among threads in a thread-block!



Thread organization

Kernel execution

- Kernels are functions executed on a device
- Kernel call is specified as

kernel_function<<<gridSize, blockSize>>>(arguments)

- gridSize number of thread-blocks
- blockSize number of threads in a thread-block
- Declaration of gridSize and blockSize
 - o dim3 gridDim(x,y,z)
 - dim3 blockDim(x,y,z)

Thread organization

Thread organization

- Figure: Example of 1D indexing
- Variables in kernel, which describe thread organization
 - threadIdx.x, threadIdx.y, threadIdx.z
 - blockDim.x, blockDim.y, blockDim.z
 - blockIdx.x, blockIdx.y, blockIdx.z
 - gridDim.x, gridDim.y, gridDim.z
- Warps are formed by consecutive threads in x-dimension, followed by y-dimension and z-dimension



Kernel

Cuda C

- Kernel function must be preceded with qualifier ___global___
 - Kernel function always returns void
 - It cannot return a value to a host (CPU) as it is executed on a different hardware (GPU)
- Functions preceded with the qualifier <u>device</u> execute only on device
 - They can be called by kernel function
- Functions preceded with the qualifier ___host___ execute only on host
- Memory qualifiers
 - ____shared___ defines memory structure shared among all threads in a thread-block
 - ____constant___ for usage of constant memory

Examples

deviceinfo.c

saxpy.c

- sum a*x plus y
- Computation of **y** = a***x** + **y**
- a is scalar, **x** and **y** are vectors

```
for (size_t i = 0; i < n; ++i)
y[i] = a * x[i] + y[i];</pre>
```