# CSC4005 Parallel Programming Tutorial 5

Bokai Xu, 119010355@link.cuhk.edu.cn

## **Outline of Tutorial 5**

#### More about Project 2

- Distribute data points in advance (Static Scheduling)
- Dynamically schedule jobs using minibatch (you may need a thread pool)
- Modify compute function to improve performance

#### • GPU & CUDA

- SM core
- Multithreading
- Memory: Registers, Local shared memory (L1 cache), L2 cache, Global memory
- Blocks and threads

#### Utility for Writing Technical Report

The idea is to do data parallelization.



acknowledgement : Zhou Yutao (118010459)

- The idea is to do data parallelization.
- The basic approach is to distribute all data points to workers in advance.
- However, workers will not terminate at the same time.
- Algorithms needed.



acknowledgement : Zhou Yutao (118010459)



Time consumed



acknowledgement : Zhou Yutao (118010459)



How about randomly allocate data points?

- Another approach is to utilize a thread pool (fixed capacity), with which we can dynamically schedule the computational task.
- Data points are divided into many minibatches.
- Batch size and capacity of thread pool are possible experimental variables.



acknowledgement : Zhu Chuyan (119010486)



= minibatch

#### **Discussion from Tutorial 4**

- Is multithreading always faster than multiprocessing?
  - Computational intensive / I/O intensive
- How about using enormous amount of degenerated CPU cores (simpler instructions, slower), which can only execute simple tasks ----GPU.



Up to 1024 threads can be reside on a single SM core at a specific time. But only 32 of them can be simultaneously executed.

Tesla V100 GPU has 80 SM cores.

#### CUDA

#### CUDA thread block (software concept)

- Up to **1024** CUDA threads form a **CUDA thread block**.
- Data can be distributed to **multiple CUDA blocks** (depending on experiment result).
- Each **CUDA thread block** has an unique BlockIdx.
- Each thread in a **CUDA thread block** has an unique ThreadIdx.
- We use BlockIdx and ThreadIdx to **identify** all threads.

Let's go to hardware part.

### **GPU** structure

		PCI Express 3.0 Host Interface GigaThread Engine		
₽				Hanza Memory Controller Memory Controller
₽ ₽				TL TL TL TL
				HBM2 1
	NVLink NVLink	High-Speed Hub +++ NVLInk NVLInk	NVLink NVLink	

• Tesla V100 GPU has **80 SM** (streaming multiprocessor).

							L1 Instru	ictio	on Cache							
		L0 lr	nstruc	tion C	ache			7	_		L0 lr	nstruc	tion C	ache		
	War	no Sch	nedule	r (32 t	hread	/clk)	_	Warp Scheduler (32 thread/clk)								
	Di	spatcl	h Unit	(32 th	read/c	:lk)		Dispatch Unit (32 thread/clk)								
	Reg	ister	File ('	16,384	4 x 32	!-bit)		Register File (16,384 x 32-bit)								
FP64	INT	INT	FP32	FP32	F				FP64	INT	INT	FP32	FP32	F		
FP64	INT	INT	FP32	FP32					FP64	INT	INT	FP32	FP32			
FP64	S	M	I	٦l	0	-k			FP64	INT	INT	FP32	FP32	Ħ		
FP64	5						TENSOR		FP64	INT	INT	FP32	FP32	TEN	ISOR	TENSOR
FP64	INT	INT	FP32	FP32	co	RE	CORE		FP64	INT	INT	FP32	FP32	C	DRE	CORE
FP64	INT	INT	FP32	FP32	$\vdash$		$\mapsto$		FP64	INT	INT	FP32	FP32	$\vdash$		+ + + + + + + + + + + + + + + + + + +
FP64	INT	INT	FP32	FP32	E				FP64	INT	INT	FP32	FP32	H		
FP64	INT	INT	FP32	FP32	$\vdash$	H	++++		FP64	INT	INT	FP32	FP32	$\vdash$	++	
LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU	Ľ	LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU
L0 Instruction Cache																
		L0 lr	nstruc	tion C	ache						L0 Ir	nstruc	tion C	ache		
	War	L0 Ir p Sch	nstruc nedule	tion C r (32 t	ache hread	/clk)				War	L0 Ir rp Sch	nstruc Iedulei	tion C r (32 ti	ache hread	l/clk)	
	War Di	L0 Ir p Sch spatcl	nstruc nedule h Unit	tion C r (32 ti (32 th	ache hread read/c	/clk) :lk)				War Di:	L0 Ir rp Sch spatcl	nstruc Iedulei n Unit	tion C r (32 tl (32 th	ache hread read/	l/clk) clk)	
	War Di Reg	L0 Ir p Sch spatcl ister	nstruc hedule h Unit File ('	tion C r (32 ti (32 th 16,384	ache hread read/c 4 x 32	/clk) :lk) !-bit)				War Di Reg	L0 Ir rp Sch spatcl jister	nstruc Iedule h Unit File (1	tion C r (32 tl (32 th 16,384	ache hread read/ 4 x 32	l/clk) clk) 2-bit)	
FP64	War Di Reg	L0 Ir rp Sch spatcl ister INT	nstruc nedule h Unit File (' FP32	tion C r (32 ti (32 th 16,384 FP32	ache hread read/c 4 x 32	/clk) :lk) !-bit)			FP64	War Di: Reg	L0 Ir rp Sch spatcl jister INT	nstruct edule n Unit File (1 FP32	tion C r (32 th (32 th I 6,384 FP32	ache hread read/ 4 x 32	l/clk) clk) 2-bit)	
FP64 FP64	Wai Di Reg	L0 Ir rp Sch spatcl ister INT INT	nstruc hedule h Unit File (' FP32 FP32	tion C r (32 th (32 th 16,384 FP32 FP32	ache hread, read/c 4 x 32	/clk) :lk) !-bit)			FP64 FP64	Wai Di Reg	L0 Ir rp Sch spatcl jister INT INT	FP32	tion C (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th) (32 th	ache hread read/ 4 x 32	l/clk) clk) 2-bit)	
FP64 FP64 FP64	War Di Reg INT INT	L0 In p Sch spatcl ister INT INT	nstruc nedule h Unit File (' FP32 FP32 FP32	tion C r (32 t (32 th 16,384 FP32 FP32 FP32	ache hread/c 4 x 32	/clk) :lk) !-bit)			FP64 FP64 FP64	War Di Reg INT INT	L0 Ir rp Sch spatcl jister INT INT	FP32	tion C (32 th (32 th (6,384 FP32 FP32 FP32	ache hread read/ 4 x 32	l/clk) clk) 2-bit)	
FP64 FP64 FP64 FP64	Wai Di Reg INT INT INT	L0 In p Sch spatcl ister INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32	tion C r (32 th (32 th 16,384 FP32 FP32 FP32 FP32	ache hread/c 4 x 32	/clk) :lk) !-bit)	TENSOR		FP64 FP64 FP64 FP64	War Di Reg INT INT INT	L0 Ir rp Sch spatcl jister INT INT INT	edule o Unit File (1 FP32 FP32 FP32 FP32	tion C r (32 th (32 th 16,384 FP32 FP32 FP32 FP32	ache hread/ 4 x 32	I/clk) clk) 2-bit)	TENSOR
FP64 FP64 FP64 FP64 FP64	War Di: Reg INT INT INT INT	L0 In rp Sch spatcl ister INT INT INT INT	File (* FP32 FP32 FP32 FP32 FP32 FP32	tion C r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32	ache hread/c 4 x 32 TEN CO	/clk) :lk) !-bit) !SOR DRE	TENSOR		FP64 FP64 FP64 FP64 FP64	War Di Reg INT INT INT INT	L0 Ir rp Sch spatcl jister INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32	tion C (32 th (32 th (32 th (32 th (32 th FP32 FP32 FP32 FP32	ache hread// 4 x 32	I/clk) clk) 2-bit) 2-bit 2-bit	TENSOR
FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Reg INT INT INT INT INT	LO In p Sch spatcl ister INT INT INT INT INT	nstruc nedule h Unit File (1 FP32 FP32 FP32 FP32 FP32 FP32	tion C r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread/ read/c 4 x 32	/clk) clk) t-bit) SOR DRE	TENSOR		FP64 FP64 FP64 FP64 FP64 FP64	Wan Dia Reg INT INT INT INT INT	LO Ir rp Sch spatcl jister INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C r (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th) (32	ache hread/ 4 x 32	I/clk) clk) 2-bit)	TENSOR
FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Reg INT INT INT INT INT	LO II p Sch spatcl spatcl iister INT INT INT INT INT	nstruc nedule h Unit File (' FP32 FP32 FP32 FP32 FP32 FP32	tion C r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread. tread/c t x 32	/clk) clk) t-bit) SOR RE	TENSOR		FP64 FP64 FP64 FP64 FP64 FP64 FP64	War Di: Reg INT INT INT INT INT	LO Ir rp Sch spatcl spatcl ster INT INT INT INT INT	ISTRUCT In Unit File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C (32 th (32 th (32 th 6,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread/ 4 x 32	I/clk) clk) 2-bit) ISOR DRE	TENSOR
FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Reg INT INT INT INT INT INT	LO II p Sch spatcl ister INT INT INT INT INT INT INT	nstruc nedule h Unit File (' FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C (32 t (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread fread/c 4 x 32	/clk) :lk) t-bit) SOR	TENSOR		FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	War Di Reg INT INT INT INT INT INT	LO Ir rp Sch spatcl sister INT INT INT INT INT INT INT	ISTRUC In Unit File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C (32 th (32 th (32 th 6,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread 4 x 32 TEN CC	I/clk) clk) 2-bit) ISOR DRE	TENSOR
FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Reg INT INT INT INT INT INT INT	LO II rp Sch spatcl ister INT INT INT INT INT INT INT INT INT	nedule nedule h Unit File (' FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C r (32 t (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread/c 4 x 32 TEN CO	/clk) clk) t-bit) SOR DRE	TENSOR CORE		FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wan Di Reg INT INT INT INT INT INT INT INT	LO In p Schu spatcl ister INT INT INT INT INT INT INT INT INT	Istruction In Unit File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	(32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th) (32 th	ache hread 4 x 32 TEN CC	I/clk) clk) 2-bit) ISOR DRE LD/ ST	TENSOR CORE
FP64 FP64 FP64 FP64 FP64 FP64 FP64 L0/ ST L0/	Wan Di Reg INT INT INT INT INT INT INT INT INT	LO III P Sch ispatcl ister INT INT INT INT INT INT INT INT	nstruc 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 LD/ ST	ache hread/c 4 x 32 TEN CO	/cik) ::ik) :-bit) SOR DRE	TENSOR CORE SFU	iche	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Wan Di: Reg INT INT INT INT INT INT INT INT INT INT	LO In rp Sch jister INT INT INT INT INT INT INT INT INT	Istruction In Unit File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tion C (32 th (32 th 6,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ache hread// 4 x 3/ TEN CC	l/clk) clk) 2-bit) USOR USOR LD/ ST	TENSOR CORE

#### 1 SM has 4 SM blocks (different from CUDA thread block).

#### Each has

- 16 FP32 Cores
- 8 FP64 Cores
- 16 INT32 Cores
- 128KB L1 / Shared memory
- 64 KB Register File

32 concurrent threads (real concurrent) can run on 1 SM (also called a warp)



Warp

acknowledgement : Prof.Wes Armour (Oxford)

## GPU design philosophy: SM (Streaming Multiprocessor)

### Key feature of SM: *Single Instruction Multiple Data*.

**32** threads on **1 SM (a warp)** execute the *same instructions (kernel function)* simultaneously, but with *different data*.

// Kernel - Adding two matrices MatA and MatB
\_\_global\_\_ void MatAdd(float MatA[N][N], float MatB[N][N],
float MatC[N][N]) {
 int i = blockIdx.x \* blockDim.x + threadIdx.x;
 int j = blockIdx.y \* blockDim.y + threadIdx.y;
 if (i + N & 0.0 i + N) MatC[i][i] = MatA[i][i] + MatD[i][i];

if (i < N && j < N) MatC[i][j] = MatA[i][j] + MatB[i][j];

Single instruction (single kernel function) Multiple data (retrieve unique data by using index)

}



Warp

# Wait! So what is the relationship between 32 and 1024... quite confused!

1 CUDA thread block has up to 1024 threads, and they will be divided into up to 1024/32=32 warps. All warps will be executed within the same SM.

#### SM

#### GPUs do not utilize *cpu-style context switching, while* GPU has its own mechanism of context switch.

// Kernel - Adding two matrices MatA and MatB

\_\_global\_\_ void **MatAdd**(float MatA[N][N], float MatB[N][N], float MatC[N][N]) {

int i = blockIdx.x \* blockDim.x + threadIdx.x; int j = blockIdx.y \* blockDim.y + threadIdx.y; if (i < N && j < N) MatC[i][j] = MatA[i][j] + MatB[i][j];</pre>



reading data from memory is slower than registers

						L1 Instru	ction Cache									
	L	.0 Ins	truction	Cache	•				L0 li	nstruc	tion C	ache				
	Warp	Scheo	duler (3	2 threa	d/clk)			Warp Scheduler (32 thread/clk)								
	Disp	atch l	Jnit (32	thread	/clk)			Di	spatc	h Unit	(32 th	read/clk)				
	Regis	ter Fi	ile (16,3	84 x 3	2-bit)			Reg	jister	File (1	16,384	4 x 32-bit)				
FP64	INT I	NT F	P32 FP3	2			FP64	INT	INT	FP32	FP32					
FP64	INT I	NTF	FP32 FP3	2			FP64	INT	INT	FP32	FP32					
FP64	INT I	NTF	P32 FP3	2			FP64	INT	INT	FP32	FP32					
FP64	INT I	NTF	P32 FP3	<sup>2</sup> TE	NSOR	TENSOR	FP64	INT	INT	FP32	FP32	TENSOR	TENSOR			
FP64	INT I	NT F	P32 FP3	12 C	ORE	CORE	FP64	INT	INT	FP32	FP32	CORE	CORE			
FP64	INT I	NTF	P32 FP3	2			FP64	INT	INT	FP32	FP32					
FP64	INT I	NTF	P32 FP3	2			FP64	INT	INT	FP32	FP32					
FP64	INT I	NTF	P32 FP3	2			FP64	INT	INT	FP32	FP32					
LD/ LD/ ST ST	LD/ I ST	LD/ I ST	LD/ LD ST S1	/ LD/ ST	LD/ ST	SFU	LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ LD/ ST ST	SFU			
	L	.0 Ins	truction	Cache	,				L0 l	nstruct	tion C	ache				
	Warp	Scheo	duler (3)	2 threa	d/clk)			War	rp Sch	nedulei	r (32 t	hread/clk)				
	Disp	atch t	onit (32	thread	/CIK)			Di	spate	n Unit	(32 m	read/cik)				
				04 2	2_hit)											
	Regis	ter Fi	ile (16,3	04 X J	2-61()			Reg	jister	File (1	16,384	4 x 32-bit)				
FP64	Regis	ter Fi NT F	ile (16,3 P32 FP3	2	51()		FP64	Reg	ister INT	File (1 FP32	16,384 FP32	4 x 32-bit)				
FP64 FP64	Regis	ter Fi NT F	FP32 FP3	2 2			FP64 FP64	Reg INT INT	INT INT	File (1 FP32 FP32	16,384 FP32 FP32	4 x 32-bit)				
FP64 FP64 FP64	Regis	ter Fi NT F NT F	FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3	12 12			FP64 FP64 FP64	Reg INT INT INT	INT INT INT INT	File (1 FP32 FP32 FP32	16,38 FP32 FP32 FP32	4 x 32-bit)				
FP64 FP64 FP64 FP64	Regis	ter Fi NT F NT F NT F	FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3	12 12 12 12 12 12 12	NSOR	TENSOR	FP64 FP64 FP64 FP64	Reg INT INT INT INT	INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32	16,384 FP32 FP32 FP32 FP32	4 x 32-bit) TENSOR	TENSOR			
FP64 FP64 FP64 FP64 FP64	Regis	ter Fi NT F NT F NT F NT F	FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3	12 12 12 12 12 12 12 12 12 12 12	NSOR ORE	TENSOR	FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT	INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32 FP32	16,384 FP32 FP32 FP32 FP32 FP32	4 x 32-bit) TENSOR CORE	TENSOR CORE			
FP64 FP64 FP64 FP64 FP64 FP64	Regis	ter Fi NT F NT F NT F NT F NT F	FP32 FP: FP32 FP: FP32 FP: FP32 FP: FP32 FP: FP32 FP: FP32 FP:	12 12 12 12 12 12 12 12 12 12 12 12 12	NSOR ORE	TENSOR CORE	FP64 FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT INT	INT INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32 FP32 FP32	16,38 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32-bit) TENSOR CORE	TENSOR CORE			
FP64 FP64 FP64 FP64 FP64 FP64 FP64	Regis	ter Fi NT F NT F NT F NT F NT F	<ul> <li>FP32 FP3</li> </ul>	12   12   12   12   12   12   12   12	NSOR	TENSOR CORE	FP64 FP64 FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT INT INT	INT INT INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32	16,38 FP32 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32-bit) TENSOR CORE	TENSOR			
FP64 FP64 FP64 FP64 FP64 FP64 FP64	Regis	ter Fi NT F NT F NT F NT F NT F NT F	FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3 FP32 FP3	12 12 12 12 12 12 12 12 12 12 12 12	NSOR ORE	TENSOR	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Reg	INT INT INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	16,38 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32-bit) TENSOR CORE	TENSOR			
FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Regis	NT F NT F NT F NT F NT F NT F NT F NT F	FP32         FP:           FP32         FP:	22 22 22 22 22 22 22 22 22 22 22 22 22	NSOR ORE	TENSOR	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT INT INT INT INT	INT INT INT INT INT INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32-bit) TENSOR CORE	TENSOR CORE			
FP64 FP64 FP64 FP64 FP64 FP64 FP64 LD/ LD/ ST LD/	INT         II           INT         II	NT F NT F NT F NT F NT F NT F NT F NT F	Ile (16,3       FP32     FP3       ID32     FP3       LDY     LD	12 12 12 12 12 12 12 12 12 12 12 12 12 1	NSOR ORE	TENSOR CORE	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT INT INT INT Memory	INT INT INT INT INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	6,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32-bit) TENSOR CORE	TENSOR CORE			

### **Context Switch**

- When read from/write to memory, another warp may occupy a SM (because read/write cost  $\sim 10^2$  cycle (processing unit survival period)).
- Execution alternates between "active" warps, with warps becoming temporarily "inactive" when waiting for data.
- A total of 1024 resident threads (either running (only 32) or ready or blocking) can exist simultaneously within 1 SM.
- Threads in the same **warp** always execute the **same command**.

// Kernel - Adding two matrices MatA and MatB

\_\_global\_\_ void **MatAdd**(float MatA[N][N], float MatB[N][N], float MatC[N][N]) {

```
int i = blockIdx.x * blockDim.x + threadIdx.x;
int j = blockIdx.y * blockDim.y + threadIdx.y;
if (i < N && j < N) MatC[i][j] = MatA[i][j] + MatB[i][j];
Fast_Slow
```

acknowledgement : Prof.Wes Armour (Oxford)

							L1 Instruc		10							
		L0 Ir	nstruct	tion C	ache						L0 In	struc	tion C	ache		
	War	p Sch	edulei	r (32 tl	hread	clk)				War	p Sch	edule	r (32 tl	hread	(cik)	
	Di	spatcl	h Unit	(32 th	read/c	:lk)		Dispatch Unit (32 thread/clk)								
	Reg	ister	File (1	16,384	4 x 32	-bit)		Register File (16,384 x 32-bit)								
FP64	INT	INT	FP32	FP32				FP	64	INT	INT	FP32	FP32	E		
FP64	INT	INT	FP32	FP32	+			FP	54	INT	INT	FP32	FP32	$\vdash$		
FP64	INT	INT	FP32	FP32				FP	54	INT	INT	FP32	FP32	H		
FP64	INT	INT	FP32	FP32	TEN	SOR	TENSOR	FP64		INT	INT	FP32	FP32	TENSOR		TENSOR
FP64	INT	INT	FP32	FP32	co	RE	CORE	FP	64	INT	INT	FP32	FP32	cc	RE	CORE
FP64	INT	INT	FP32	FP32	Ħ			FP	64	INT	INT	FP32	FP32	Ħ		
FP64	INT	INT	FP32	FP32				FP	54	INT	INT	FP32	FP32	Ħ		
FP64	INT	INT	FP32	FP32	Ħ			FP	54	INT	INT	FP32	FP32	Ħ		
LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU
		L0 lr	nstruct	tion C	ache						L0 In	struc	tion C	ache		
	War	p Sch	eduler	r (32 tl	hread	(clk)		Warp Scheduler (32 thread/clk)								
	Di	spatcl	h Unit	(32 th	read/c	:lk)				Dis	spatch	n Unit	(32 th	read/o	:lk)	
	Reg	ister	File (1	16,384	4 x 32	-bit)				Reg	ister	File (1	16,384	4 x 32	-bit)	
FP64	INT	INT	FP32	FP32	$\square$			FP	54	INT	INT	FP32	FP32	F		
FP64	INT	INT	FP32	FP32	$\vdash$			FP	54	INT	INT	FP32	FP32	$\vdash$		
FP64	INT	INT	FP32	FP32				FP	64	INT	INT	FP32	FP32	H		
			5022	ED32			TENSOR	FP	84	INT	INT	EP32	ED32	TEN	SOR	TENSO
FP64	INT	TENSC	CORE CORE		~		in the second se		11.02							
FP64 FP64			FP32	FP32	CO	RE	CORE	FP	54	INT	INT	FP32	FP32	CC	RE	CORE
FP64 FP64 FP64	INT INT INT	INT INT	FP32 FP32 FP32	FP32 FP32	CO	RE	CORE	FP(	54 54	INT	INT	FP32 FP32	FP32 FP32	CC	RE	CORE
FP64 FP64 FP64 FP64	INT INT INT INT	INT INT INT INT	FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32	CO	RE	CORE	FP(	54 54 54 54	INT INT INT	INT INT INT	FP32 FP32 FP32	FP32 FP32 FP32	CC	RE	CORE
FP64 FP64 FP64 FP64 FP64	INT INT INT INT	INT INT INT INT	FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32	CO	RE	CORE	FP( FP( FP(	54 54 54 54	INT INT INT INT	INT INT INT INT	FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32	CC	RE	CORE
FP64 FP64 FP64 FP64 FP64 LD/LD/ ST ST	INT INT INT INT LD/ ST	INT INT INT INT LD/ ST	FP32 FP32 FP32 FP32 FP32 FP32 LD/ ST	FP32 FP32 FP32 FP32 FP32 FP32 LD/ ST	LD/ ST		CORE	FP( FP( FP( LD/ ST	54 54 54 54 54 54 ST	INT INT INT INT LD/ ST	INT INT INT INT LD/ ST	FP32 FP32 FP32 FP32 FP32 LD/ ST	FP32 FP32 FP32 FP32 FP32 LD/ ST	LD/ ST	LD/ ST	SFU
FP64 FP64 FP64 FP64 FP64 LD/ LD/ ST ST	INT INT INT INT LD/ ST	INT INT INT INT LD/ ST	FP32 FP32 FP32 FP32 FP32 LD/ ST	FP32 FP32 FP32 FP32 FP32 LD/ ST	LD/ ST	LD/ ST	CORE SFU	FPC FPC FPC LD/ ST	54 54 54 54 54 54 ST	INT INT INT INT LD/ ST	INT INT INT INT LD/ ST	FP32 FP32 FP32 FP32 FP32 LD/ ST	FP32 FP32 FP32 FP32 FP32 LD/ ST	LD/ ST	LD/ ST	SFU

#### SM

# **1 SM** can execute **several concurrent CUDA thread blocks**, depending on the resources needed by all blocks.

It's about scheduling.

							L1 Instru	ctio	n Cache								
		L0 Ir	struc	tion C	ache						L0 lr	nstruc	tion C	ache			
	War	p Sch	edule	r (32 tl	hread	(clk)			Warp Scheduler (32 thread/clk)								
	Di	spatch	n Unit	(32 th	read/c	:lk)			Dispatch Unit (32 thread/clk)								
	Reg	ister	File (1	16,384	4 x 32	-bit)			Register File (16,384 x 32-bit)								
FP64	INT	INT	FP32	FP32	E				FP64	INT	INT	FP32	FP32	E			
FP64	INT	INT	FP32	FP32	H+				FP64	INT	INT	FP32	FP32	Ħ			
FP64	INT	INT	FP32	FP32					FP64	INT	INT	FP32	FP32				
FP64	INT	INT	FP32	FP32	TEN	SOR	TENSOR		FP64	INT	INT	FP32	FP32	TEN	SOR	TENSOF	
FP64	INT	INT	FP32	FP32	cc	RE	CORE		FP64	INT	INT	FP32	FP32	cc	RE	CORE	
FP64	INT	INT	FP32	FP32	H				FP64	INT	INT	FP32	FP32	H			
FP64	INT	INT	FP32	FP32	H				FP64	INT	INT	FP32	FP32	H			
FP64	INT	INT	FP32	FP32	$\vdash$		$\vdash \vdash \vdash$		FP64	INT	INT	FP32	FP32	$\vdash$		++++	
LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU		LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU	
		L0 Ir	struc	tion C	ache		_	Ì٦			L0 lr	nstruc	tion C	ache			
	War	p Sch	edule	r (32 tl	hread	(clk)				Wai	<mark>rp Sc</mark> h	edule	<mark>r (32</mark> t	hread	/clk)		
	Di	spatch	n Unit	(32 th	read/c	:lk)				Di	spatcl	h Unit	(32 th	read/c	:lk)		
	Register File (16,384 x 32-bit)													rouan e			
	Key	ister	File (1	16,384	4 x 32	-bit)				Reg	ister	File ('	16,384	4 x 32	!-bit)		
FP64	INT	ISTER	File (1 FP32	FP32	4 x 32	-bit)			FP64	Reg	ister INT	File (' FP32	16,384 FP32	4 x 32	!-bit)		
FP64 FP64	INT INT	ISTER INT INT	FIIE (1 FP32 FP32	FP32 FP32	4 x 32	-bit)			FP64 FP64	Reg INT INT	ister INT INT	File (* FP32 FP32	16,384 FP32 FP32	4 x 32	!-bit)		
FP64 FP64 FP64	INT INT INT	INT INT INT	FILE (1 FP32 FP32 FP32	FP32 FP32 FP32 FP32	4 x 32	-bit)			FP64 FP64 FP64	Reg INT INT INT	INT INT INT	File (1 FP32 FP32 FP32	16,384 FP32 FP32 FP32	4 x 32	!-bit)		
FP64 FP64 FP64 FP64	INT INT INT INT	INT INT INT INT	FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32	4 x 32	-bit) SOR	TENSOR		FP64 FP64 FP64 FP64	Reg INT INT INT	INT INT INT INT	File (* FP32 FP32 FP32 FP32	16,38 FP32 FP32 FP32 FP32	4 x 32	e-bit)	TENSOF	
FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT	INT INT INT INT INT	FIIe (1 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32	4 x 32	SOR	TENSOR		FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT	INT INT INT INT INT	File (* FP32 FP32 FP32 FP32 FP32	16,38 FP32 FP32 FP32 FP32 FP32	4 x 32	2-bit) SOR DRE	TENSOF	
FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT	INT INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32 FP32	TEN	SOR	TENSOR CORE		FP64 FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT	INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32 FP32	16,38 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32	SOR SOR	TENSOF	
FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT INT	INT INT INT INT INT INT	File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TEN	SOR RE	TENSOR		FP64 FP64 FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT INT	INT INT INT INT INT INT	File (* FP32 FP32 FP32 FP32 FP32 FP32 FP32	16,38 FP32 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32	SOR	TENSOF	
FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT INT INT	INT INT INT INT INT INT INT	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TEN CC	SOR RE	TENSOR		FP64 FP64 FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT INT INT INT	INT INT INT INT INT INT INT	File (* FP32 FP32 FP32 FP32 FP32 FP32 FP32	16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32	e-bit)	TENSOF	
FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT INT INT INT LD/	INT INT INT INT INT INT INT INT LD/ ST	FILE (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TEN CC	-bit)	TENSOR CORE		FP64 FP64 FP64 FP64 FP64 FP64 FP64 ED/ ST	Reg INT INT INT INT INT INT INT INT INT	INT INT INT INT INT INT INT INT LD/	File (* FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	16,38 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32	LD/	TENSOR CORE	
FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	INT INT INT INT INT INT INT LD/ ST	INT INT INT INT INT INT INT INT INT	FIIE (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	TEN CC	LD/ ST	TENSOR CORE SFU	che	FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Reg INT INT INT INT INT INT INT VETOTY	INT INT INT INT INT INT INT INT INT	File (7 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	16,388 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	4 x 32 TEN CC	LD/ ST	TENSOR CORE	

## GPU design philosophy: GPU Memory

- Why does a warp contain 32 threads? (registers)
- Why is the capacity of resident threads is 1024? (discussion question)



SM-0

SM-(N-1)

SM-1

#### **GPU Memory**

- **Registers**—These are private to each thread, which means that registers assigned to a thread are not visible to other threads. The compiler makes decisions about register utilization.
- L1/Shared memory (SMEM)—Every SM has a fast, on-chip scratchpad memory that can be used as L1 cache and shared memory. All threads in a CUDA block can share shared memory, and all CUDA blocks running on a given SM can share the physical memory resource provided by the SM..
- **Read-only memory**—Each SM has an instruction cache, constant memory, texture memory and RO cache, which is read-only to kernel code.
- L2 cache The L2 cache is shared across all SMs, so every thread in every CUDA block can access this memory. The <u>NVIDIA A100 GPU</u> has increased the L2 cache size to 40 MB as compared to 6 MB in V100 GPUs.
- **Global memory**—This is the framebuffer size of the GPU and DRAM sitting in the GPU.

#### **GPU Memory**

#### Challenge: how many registers are we using?

Can you guess how many registers are we using in the following vector\_add code?

```
extern "C" __global__ void vector_add(const float * A, const float * B, float * C, const int size) {
    int item = (blockIdx.x * blockDim.x) + threadIdx.x;
    if ( item < size ) {
        C[item] = A[item] + B[item];
    }
}</pre>
```

https://carpentries-incubator.github.io/lesson-gpu-programming/06-global\_local\_memory/index.html

#### **GPU Memory**

• If we want to make registers use more explicit in the vector\_add code, we can try to rewrite it in a slightly different, but equivalent, way.

```
extern "C" __global__ void vector_add(const float * A, const float * B, float * C, const int size) {
    int item = (blockIdx.x * blockDim.x) + threadIdx.x;
    float temp_a, temp_b, temp_c;
    if ( item < size ) {
        temp_a = A[item];
        temp_b = B[item];
        temp_c = temp_a + temp_b;
        C[item] = temp_c;
    }
}</pre>
```

#### Hardware and Software



#### **Question: Data Partitioning**

Based on this architecture, how should we partition data and allocate them to all workers?

• Use multiple-dimensional index: *block index* and *thread index*.

### **Question: Data Partitioning**

#### Example:

We have int x[100000] and int y[100000].

We want to compute x+y.

Let the capacity of each block be 1024 (maximum).

Number of blocks = 100000 / 1024 + 1 = 98.

For each thread, it only compute x[i]+y[i], where i=block\_id \* 1024 + thread\_id.

#### **CUDA** Variables

Variable	Description	Property
gridDim	Total number of blocks	
blockDim	Total number of threads in a block	
blockldx	Block id (x dimensional) of this thread	Threads in a specific block will get the same <b>blockIdx</b>
threadIdx	Thread id (x dimensional) of this thread in its block	From 0 to 1023. Only valid within a block.

**1.** Declare kernel function



2. Allocate memory in host first

// Size of vectors int n = 100000;

// Host input vectors double \*h\_a; double \*h\_b; //Host output vector double \*h\_c;

// Device input vectors double \*d\_a; double \*d\_b; //Device output vector double \*d\_c;

// Size, in bytes, of each vector size\_t bytes = n\*sizeof(double);

// Allocate memory for each vector on host h\_a = (double\*)malloc(bytes); h\_b = (double\*)malloc(bytes); h\_c = (double\*)malloc(bytes);

3. Initialize data and copy data to device (gpu) global memory



4. Determine block size and grid size.

#### int blockSize, gridSize;

// Number of threads in each thread block
blockSize = 1024;

// Number of thread blocks in grid
gridSize = (int)ceil((float)n/blockSize);

5. Launch the kernel function

// Execute the kernel
vecAdd<<<gridSize, blockSize>>>(d\_a, d\_b, d\_c, n);

// Copy array back to host
cudaMemcpy( h\_c, d\_c, bytes, cudaMemcpyDeviceToHost );

**6.** Copy result from device memory back to host memory.

// Execute the kernel
vecAdd<<<gridSize, blockSize>>>(d\_a, d\_b, d\_c, n);
// Copy array back to host

cudaMemcpy( h\_c, d\_c, bytes, cudaMemcpyDeviceToHost );

7. Release device memory and host memory.

<pre>// Sum up vector c and print double sum = 0; for(i=0; i<n; %f\n",<="" +="h_c[i];" i++)="" pre="" printf("final="" result:="" sum=""></n;></pre>	result divided sum/n);	by n, this sho	ould equal 1 within	error
<pre>// Release device memory cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);</pre>				
<pre>// Release host memory free(h_a); free(h_b);</pre>				
<pre>free(h_c); return 0;</pre>				

**Compile:** 

nvcc vec\_add.cu

Run (please use slurm)

salloc -n1 -c1 --gres=gpu:1
srun ./a.out

### Thank you!

We will talk more about CUDA in the following weeks.

### About report

• Integrate flowchart with markdown: **Typora** 



Alice->Bob: Hello Bob, how are you? Note right of Bob: Bob thinks Bob-->Alice: I am good thanks!





• <u>https://support.typora.io/Draw-Diagrams-With-Markdown/</u>