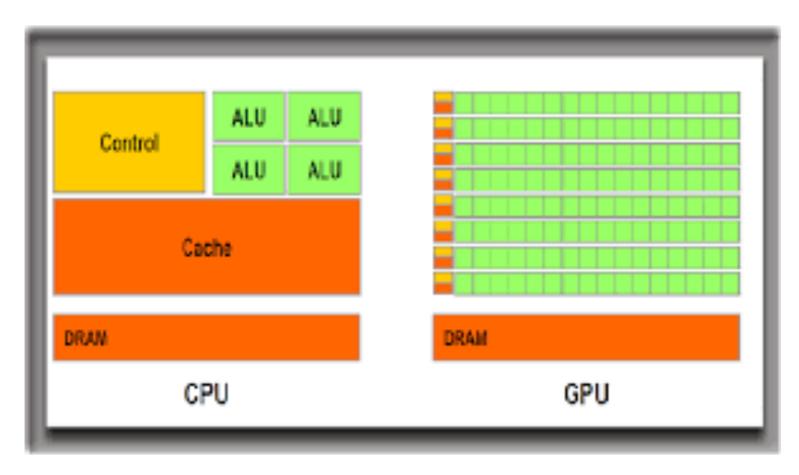
Introduction to Cuda

Olivier Mattelaer UCLouvain CP3 & CISM

Program of this lecture

- Difference between CPU and GPU
 - ➡ Why and when to use a GPU?
- What is CUDA?
 - ➡ When/where can I use cuda?
- Structure of a GPU program
 - Nomenclature
- First example of CUDA programming
- First step in optimisation of a CUDA program
 - Managing memory transfer

CPU versus GPU



CPU	GPU
Central Processing Unit	Graphics Processing Unit
Several cores	Many cores
Low latency	High throughput
Good for serial processing	Good for parallel processing
Can do a handful of operations at once	Can do thousands of operations at once

Speed versus Latency

- Speed: number of operation per second
- Latency: delay in the first operation
 - $\Rightarrow T = L + vD$
- How amazon transfer data from one cluster to another

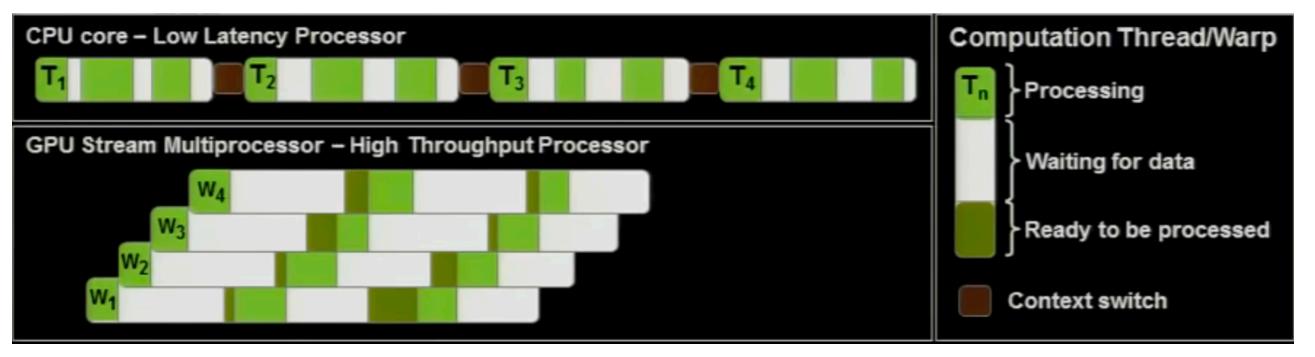


- Speed: Large bandwidth
 - Fiber connection: Gb
- Latency: time of the travel between the two cluster.

• Latency is "reactivity"

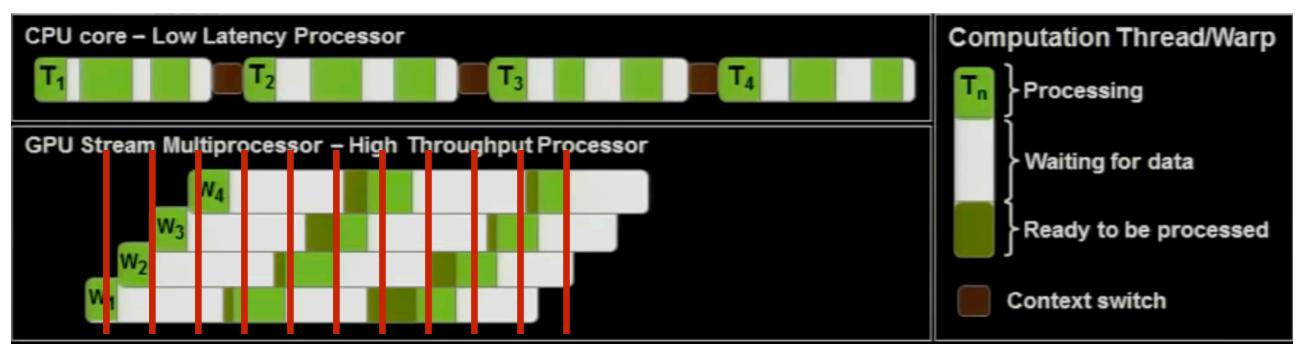
GPU versus CPU

- CPU minimizes latency $\rightarrow T = L + NvD$
- GPU hides latency by overlapping computation



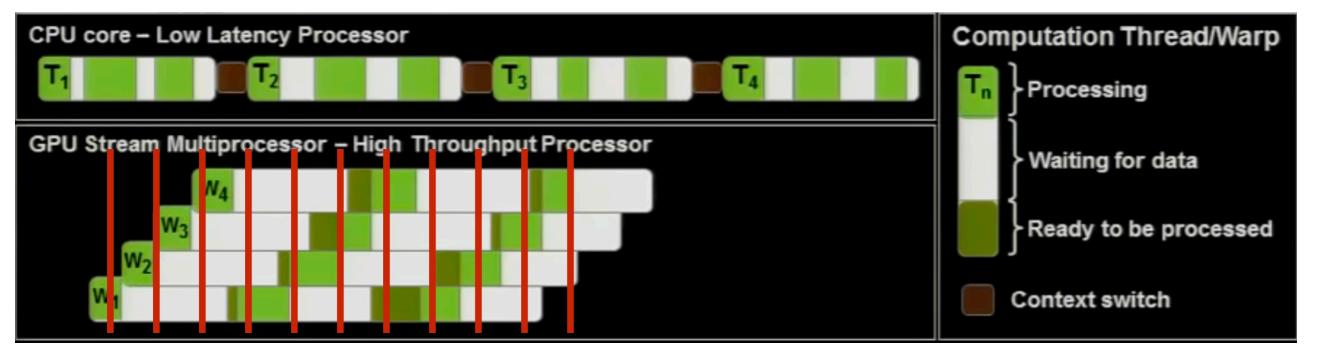
GPU versus CPU

- CPU minimizes latency \rightarrow T = L + NvD
- GPU hides latency by overlapping computation



GPU versus CPU

- CPU minimizes latency \Rightarrow T = L + NvD
- GPU hides latency by overlapping computation

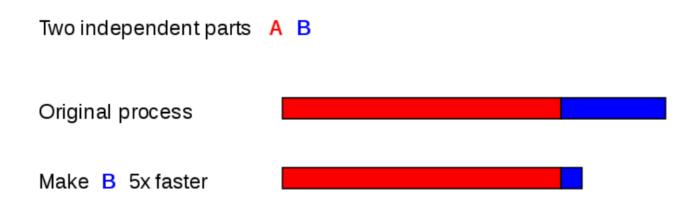




CECI training: Cuda

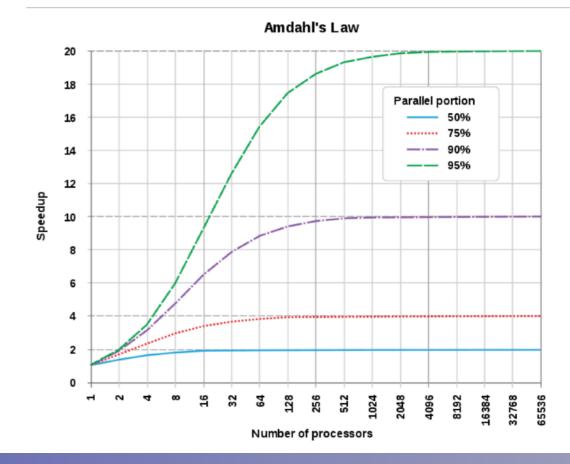
Amdahl's law

- A cpu has 8 core a GPU 2056 core
 - Should my code should be 200 faster?



 It depends which fraction of your code can use parallelism

 This is Amdahl's law given theoretical speed-up of your code

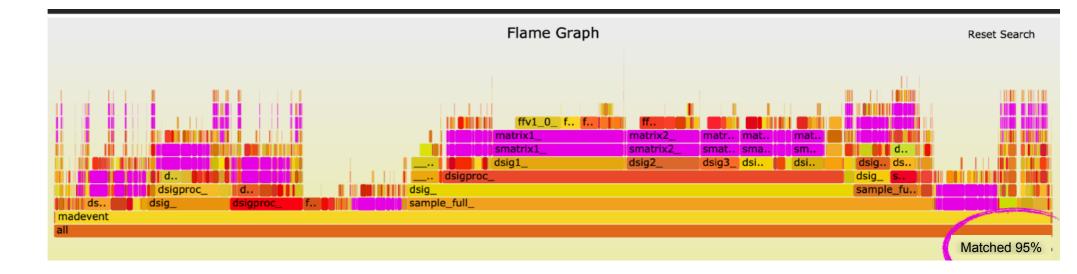


Speed-up in practise

- Comparing speed of code between cpu and gpu are not really fair
 - Cost of the GPU/CPU
 - Huge speed-up typically means "bad" denominator
- A "normal" is around 5-20
 - Much higher number reported in some cases.
- GPU clock is slower than CPU clock
 - ➡ GPU ~ mhz
 - ➡ CPU ~ Ghz

Real case example

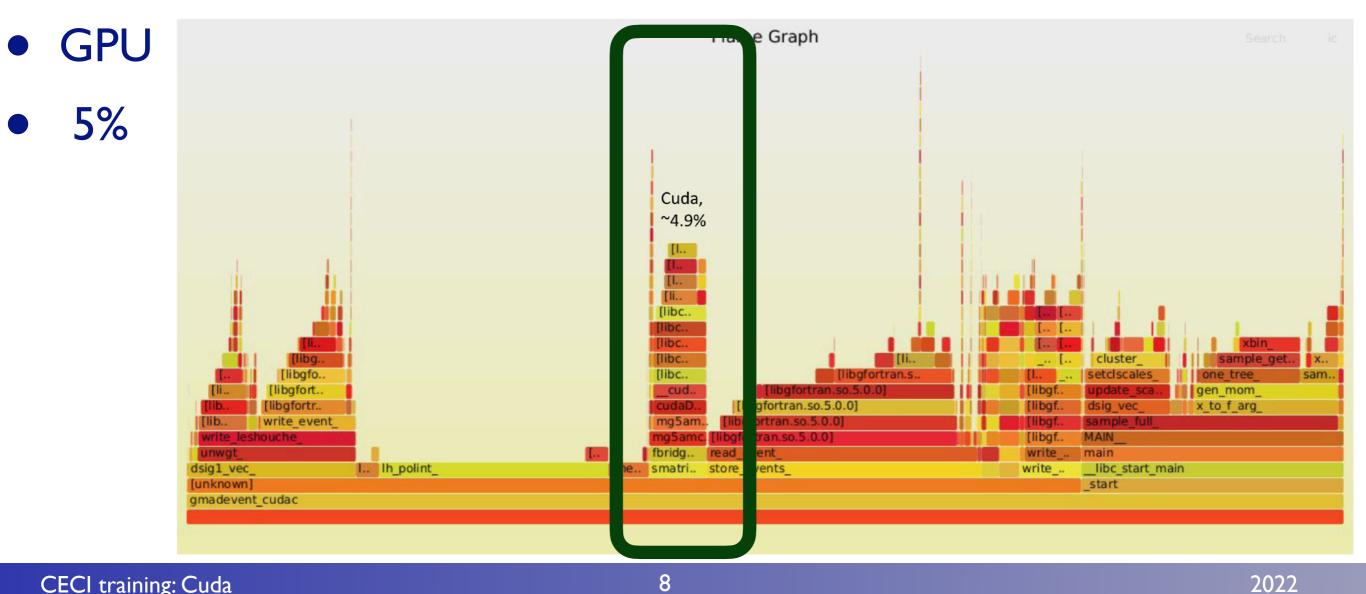
- CPU
- 95%



Real case example

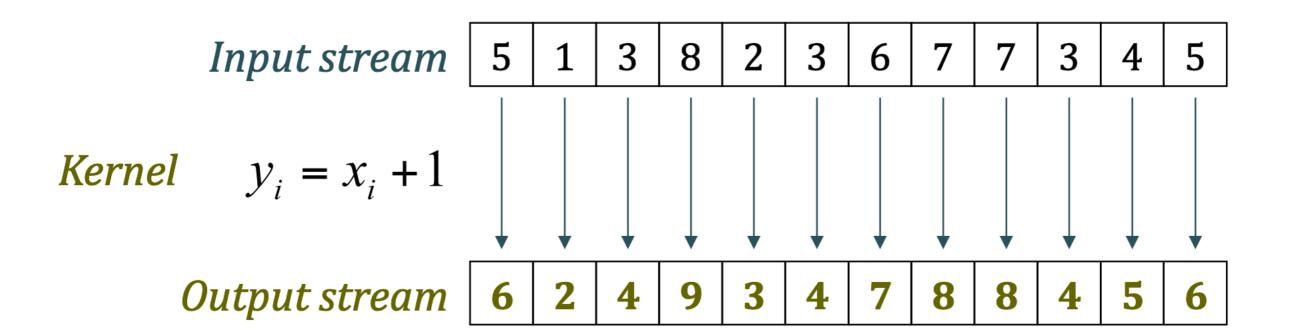
- CPU
- 95%

Flame Graph	Reset Search
dsigproc_	d dsig., ds lsig_ s ample_fu Matched 95%

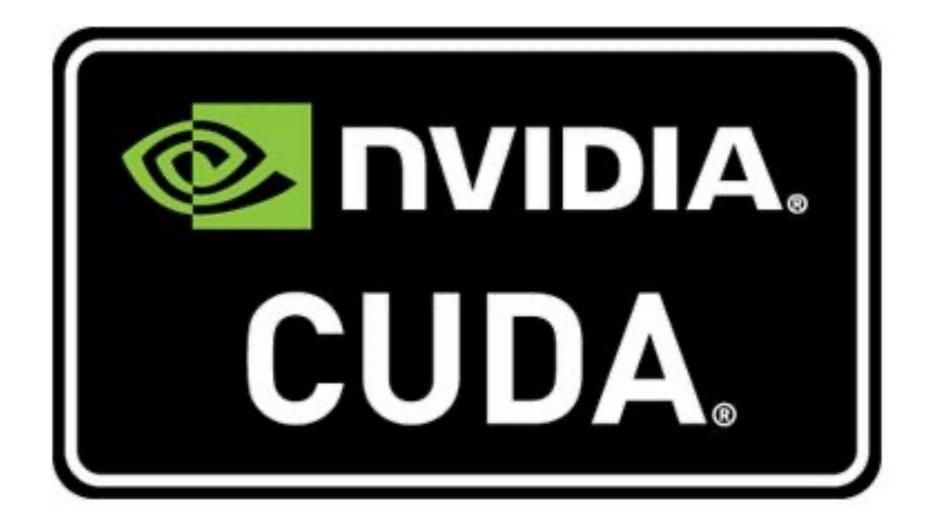


Stream computing

- The idea of GPU are
 - "multiple data"
 - ➡ SAME operation
- Same as vectorisation on CPU (but different scale)



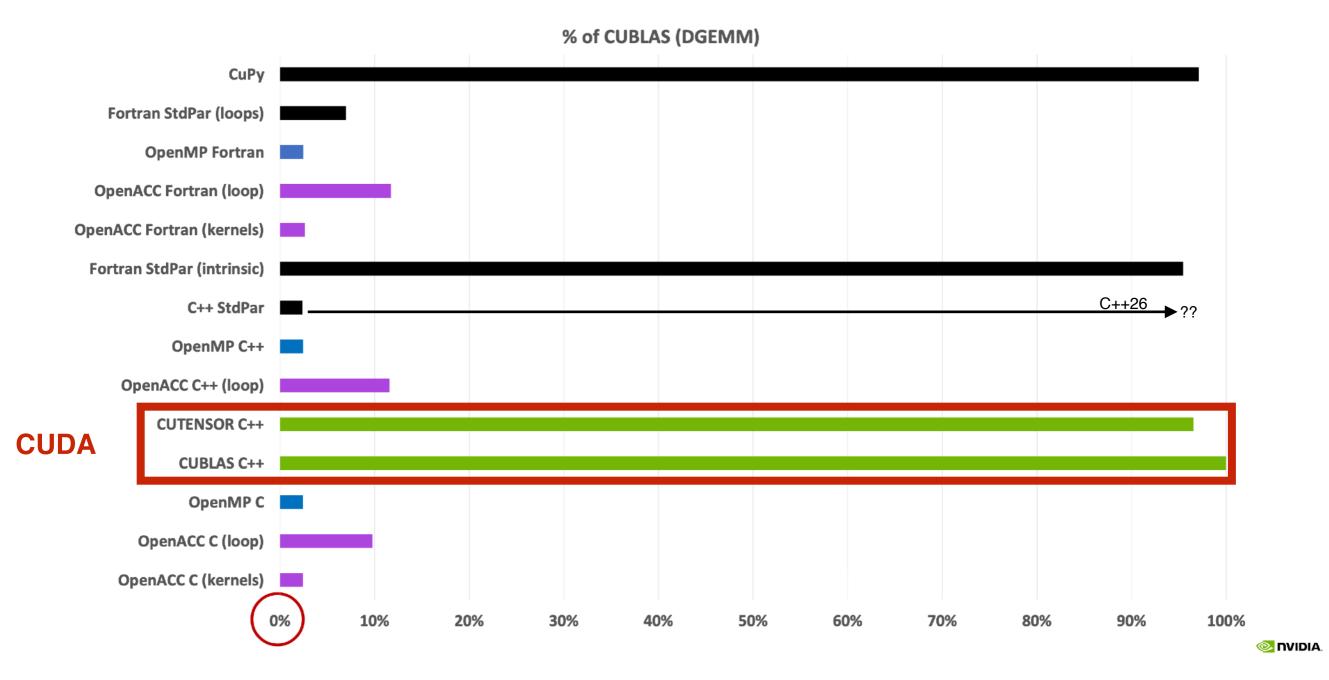
• You can have synchronisation between threads



- As for CPU, you do not want to code at assembler level
- First released in 2006
 - Restricted to nvidia GPU
 - Expose the raw computation power
 - No need of graphical knowledge

Do you need cuda?

Matrix Multiplication: C = C + A * B



https://youtu.be/nlo_8x54Syo?t=1304

GPU availability

- Dragon 2:
 - Two machines with two Nvidia V100
- Namur
 - ➡ Gaming GPU
- Manneback (UCL only)
 - Nvidia V100/ A100
 - Bunch of gaming GPU
- (Future) Lumi European computer (EUROHPC)
 - Not Nvidia GPU machine
 - Cuda code need to be converted to HIP
 - Alternative: OpenACC, OneAPI, Sycl, kokos,...

SLURM FOR GPU

• Check ressource

sinfo --format="%N %.6D %P %G"

mb-bro080	1 cp3-gpu gpu:TeslaK80:2,localscratch:156
mb-cas101	1 gpu gpu:TeslaV100:2,mps:TeslaV100:100,localscratch:172
mb-cas102	1 gpu gpu:TeslaV100:2,mps:TeslaV100:100,localscratch:411
mb-sab040	1 gpu gpu:TeslaM10:4,localscratch:46

- First run interactively
 - srun -p gpu --gres=gpu:TeslaV100:1 --pty bash
- Check module on the machine
 - module av
- Check that you have access to the GPU
 - nvidia-smi

SLURM FOR GPU

- Check ressource
 - ➡ sinfo --format="%N %.6D %P %G"

NODELIST NO	ODES PARTIT	ION GRES
drg2-w[001-0	017] 17	batch* (null)
drg2-w[001-0	017] 17	long (null)
drg2-w[018-0	019] 2	gpu gpu:2
drg2-w[001-0	017] 17	debug (null)

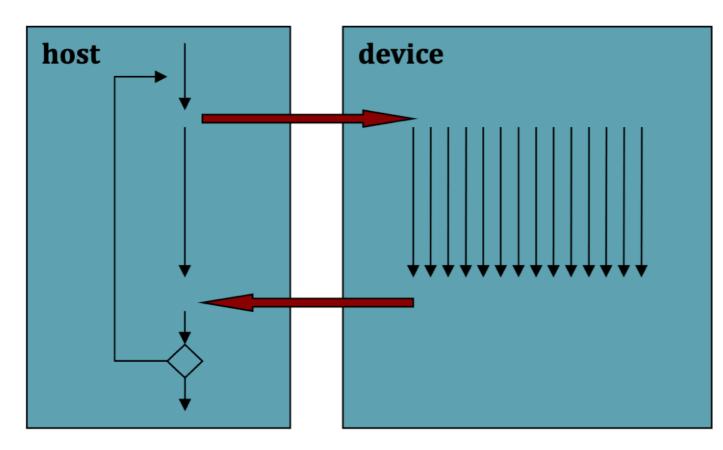
- First run interactively
 - srun -p gpu --gres=gpu: I --pty bash
- Check module on the machine
 - module av
- Check that you have access to the GPU
 - nvidia-smi

Virtual GPU : MIG (A100)

- If you say "A100 is too big for me to fill it. I prefer to have multiple small GPU"
- Then MIG is for you.
 - MIG split your A100 in multiple virtual smaller GPU that are seen as different independent GPU
 - Many splitting are technically possible but we offer only one (splitting in 7)
 - Add in slurm "--comment=MIG"
- Only for A100 GPU (so not dragon2)

Cuda Programming model

- A GPU needs to be controlled by a CPU.
 - ➡ All programs start by the CPU
 - Data are prepared on the CPU and moved to the GPU
 - ➡ GPU is crunching data
 - Data moved back to the cpu
 - Programs end



Kernel execution is asynchronous

Asynchronous memory transfers also available

Cuda Programming model

- The cpu is called the "host"
- The gpu is called the **"device"**
 - Viewed as a co-processor
- Function executed on gpu are called kernel
 - Executed in parallel on different data element
- Both the host/device have their own memory
 - Memory management is handle by the host
 - Automatic management is possible

Multi-processor/block/thread



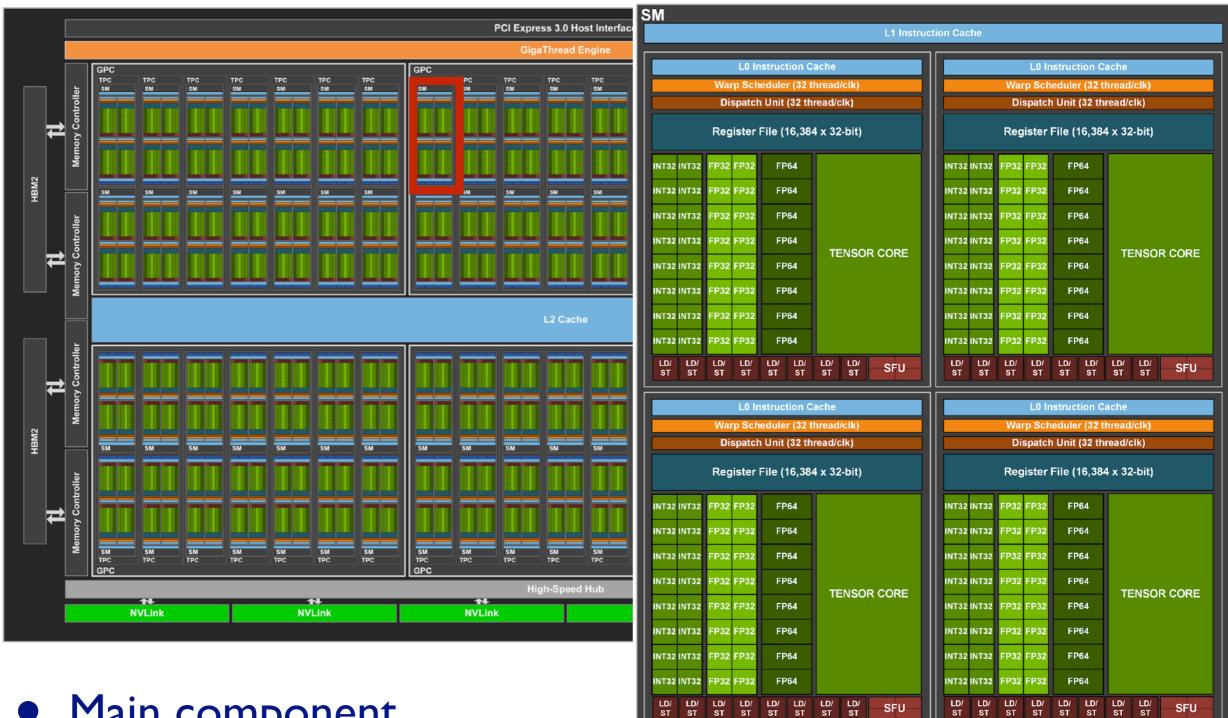
- Main component
 - Memory
 - Streaming Multiprocessor (84 of them here)

Multi-processor/block/thread



- Main component
 - Memory
 - Streaming Multiprocessor (84 of them here)

Multi-processor/block/thread



- Main component
 - Memory
 - Streaming Multiprocessor (84 of them here)

Tex

SFU

Tex

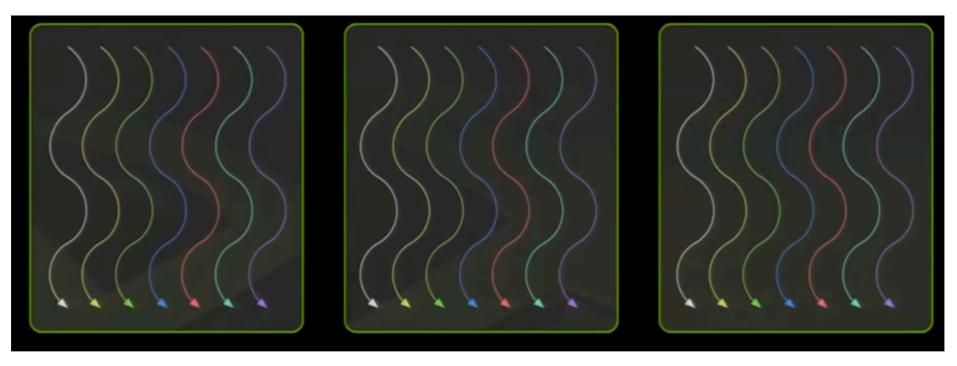
SFU

Tex

192KB L1 Data Cache / Shared Memory

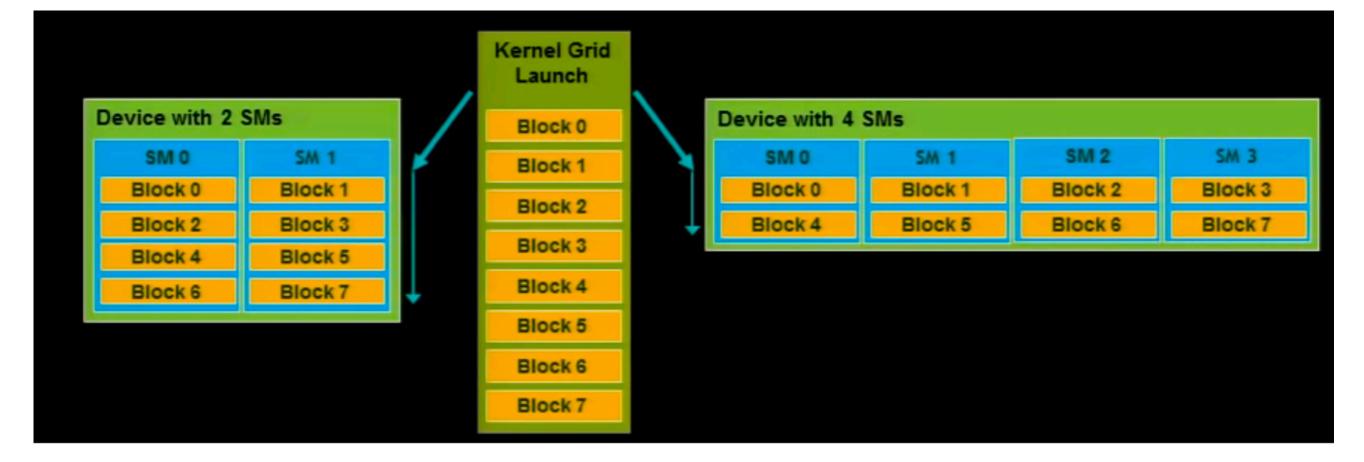
Tex

Block



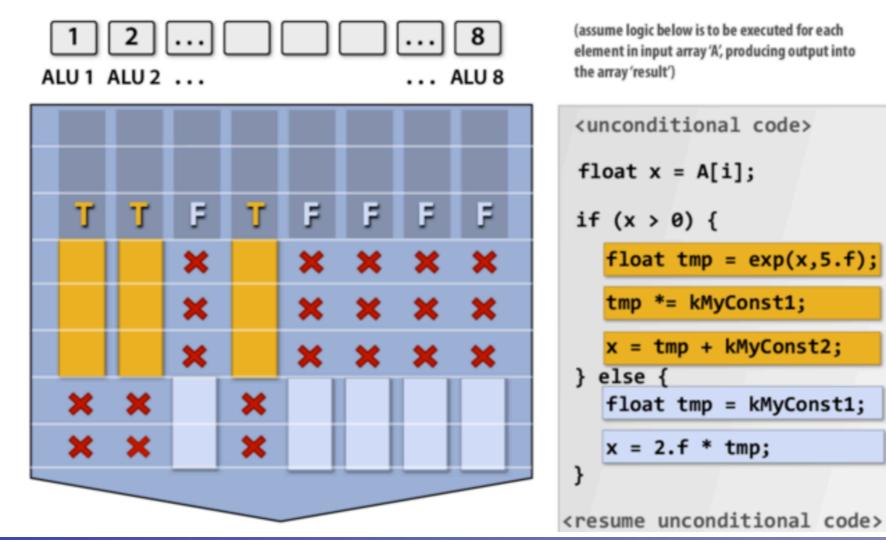
- Thread are grouped by block
 - Collaboration of thread (syncronization, shared memory)
- Up to 2048 thread per block
- Block are fully independent
 - Can be executed in any order
 - Can be executed on different GPU

• Separation into block allow you to adapt to various GPU in an easy way.



Wrap

- block are organised in wrap of 32 thread
 - Correspond to an hardware configuration
- Those 32 threads are working in lock step
 - Run the same command at the same time
 - If statement slows down the code



CECI training: Cuda

include <stdio.h>

#include <stdlib.h>
#include <algorithm>
#include <cmath>

```
void saxpy_cpu(float* vecX, float* vecY, float alpha, int n)
  for (int i=0; i < n; i++){</pre>
   vecY[i] = alpha * vecX[i] + vecY[i];
  }
}
int main(){
  int N = 1<<20; // 2^20 = 1,048,576
  float* x;
  float* y;
  x = (float *) malloc(N*sizeof(float));
  y = (float *) malloc(N*sizeof(float));
  for (int i = 0; i < N; i++) {
   x[i] = 1.0f;
   y[i] = 2.0f;
  }
  saxpy_cpu(x, y, 2.f, N);
```

```
float maxError = 0.0f;
for (int i = 0; i < N; i++)
maxError = fmax(maxError, std::abs(y[i]-4.0f));
printf("Max error: %f\n", maxError);
```

}

include <stdio.h>

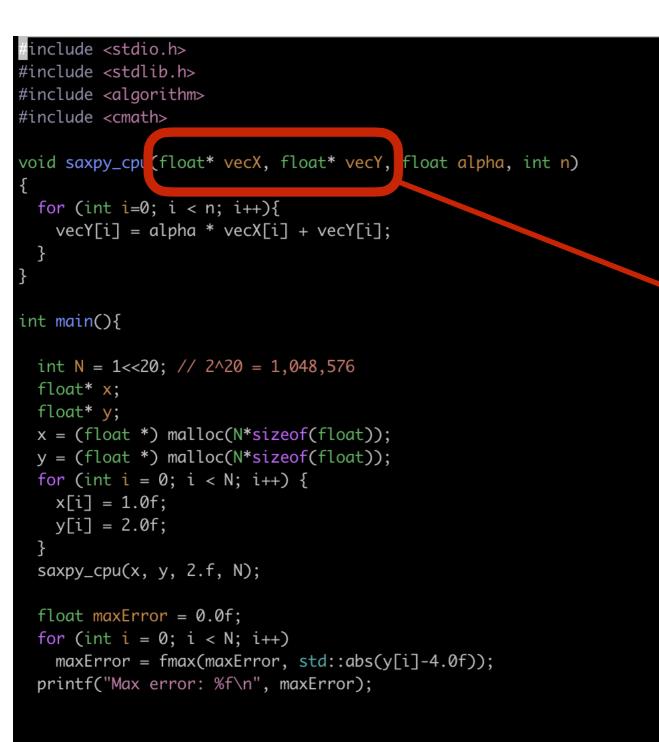
#include <stdlib.h>
#include <algorithm>
#include <cmath>

```
void saxpy_cpu(float* vecX, float* vecY, float alpha, int n)
  for (int i=0; i < n; i++){</pre>
   vecY[i] = alpha * vecX[i] + vecY[i];
  }
}
int main(){
  int N = 1<<20; // 2^20 = 1,048,576
  float* x;
  float* y;
  x = (float *) malloc(N*sizeof(float));
  y = (float *) malloc(N*sizeof(float));
  for (int i = 0; i < N; i++) {
    x[i] = 1.0f;
   y[i] = 2.0f;
  }
  saxpy_cpu(x, y, 2.f, N);
  float maxError = 0.0f;
  for (int i = 0; i < N; i++)
   maxError = fmax(maxError, std::abs(y[i]-4.0f));
```

```
printf("Max error: %f\n", maxError);
```

• $\vec{y} = a\vec{x} + \vec{y}$

CECI training: Cuda



• $\overrightarrow{y} = a\overrightarrow{x} + \overrightarrow{y}$

 float* is used here for passing an array

include <stdio.h>

#include <stdlib.h>
#include <algorithm>
#include <cmath>

```
void saxpy_cpu(float* vecX, float* vecY, float alpha, int n)
{
  for (int i=0; i < n; i++){
    vecY[i] = alpha * vecX[i] + vecY[i];
  }
}
int main(){
  int N = 1<<20; // 2^20 = 1,048,576
  float* x;
  float* x;</pre>
```

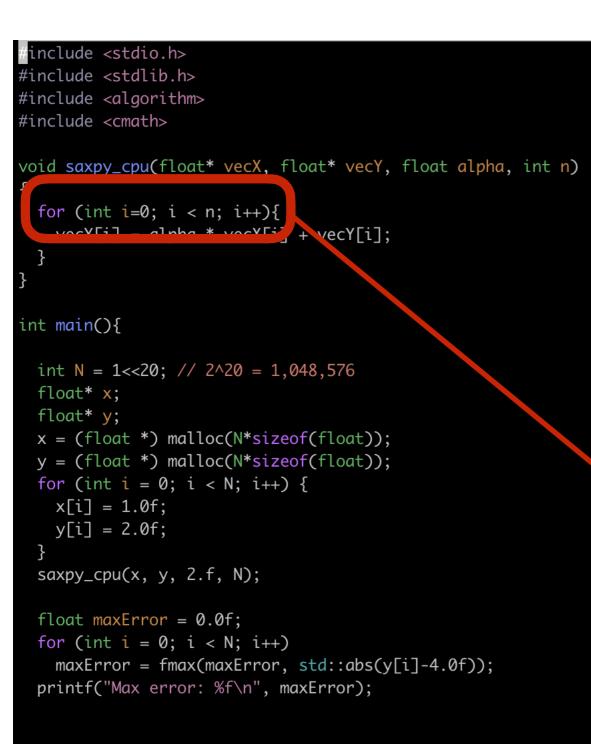
```
x = (float *) malloc(N*sizeof(float));
y = (float *) malloc(N*sizeof(float));
```

```
x[i] = 1.0f;
y[i] = 2.0f;
}
saxpy_cpu(x, y, 2.f, N);
```

```
float maxError = 0.0f;
for (int i = 0; i < N; i++)
maxError = fmax(maxError, std::abs(y[i]-4.0f));
printf("Max error: %f\n", maxError);
```

• $\overrightarrow{y} = a\overrightarrow{x} + \overrightarrow{y}$

- float* is used here for passing an array
 - that array is assigned dynamically (malloc)



• $\overrightarrow{y} = a\overrightarrow{x} + \overrightarrow{y}$

- float* is used here for passing an array
- that array is assigned dynamically (malloc)
- We explicitly loop over the data element

Vector Addition: Z = a * X + Y

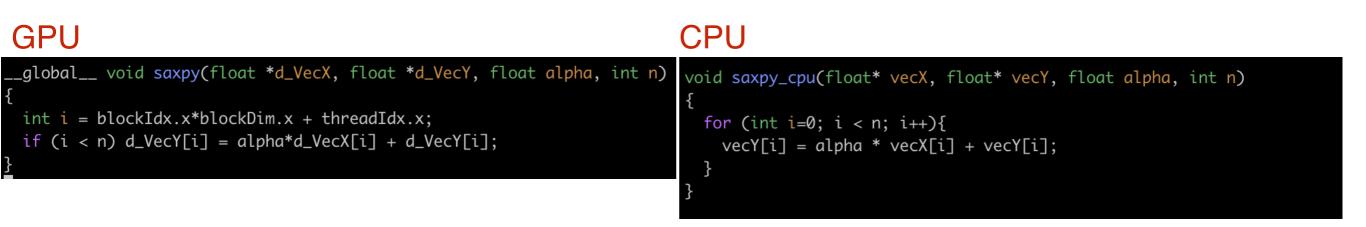
% of CUDA C++ **CUDA Python** CuPy Fortran StdPar **OpenMP Fortran OpenACC Fortran (loop) OpenACC Fortran (kernels) CUDA Fortran** C++ StdPar OpenMP C++ **OpenACC C++ (loop)** CUDA C++ CUBLAS C++ **OpenMP C OpenACC C (loop)** 55% 70% 50% 60% 65% 75% 80% 85% 90% 95% 100%

The limitation here is the **time to move data**, all parallelism language are efficient for this.

-> Fast code can be achieved with StdPar

2022

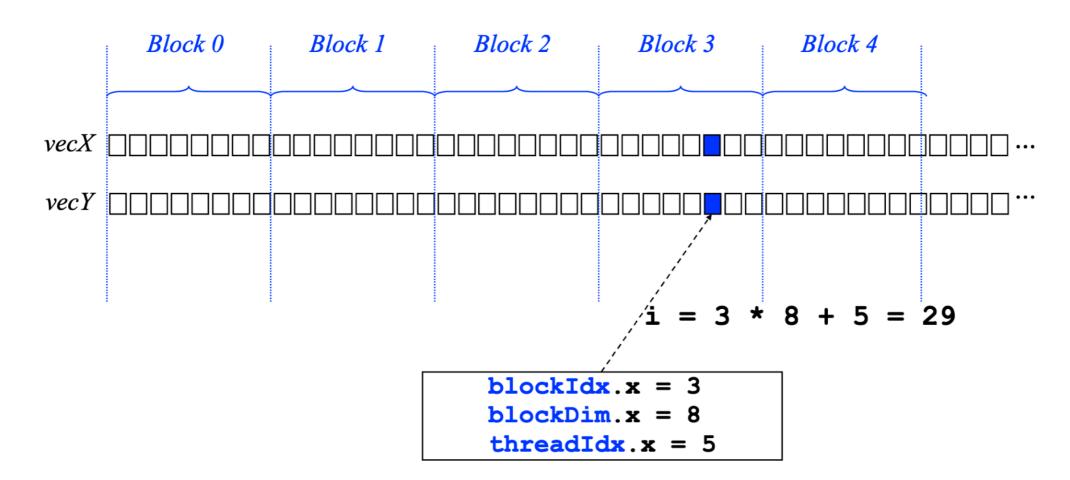
Cuda version: kernel



- No loop anymore !!
 - Each thread will take care of one data
 - Need to compute which element each thread has to handle.
 - Various variable defined for that
 - blockldx.x (.y/ .z if 2D and 3D): id of the current block
 - blockDim.x: number thread in Block (for that dimension)
 - threadIdx.x: id of the current thread inside the block

Index

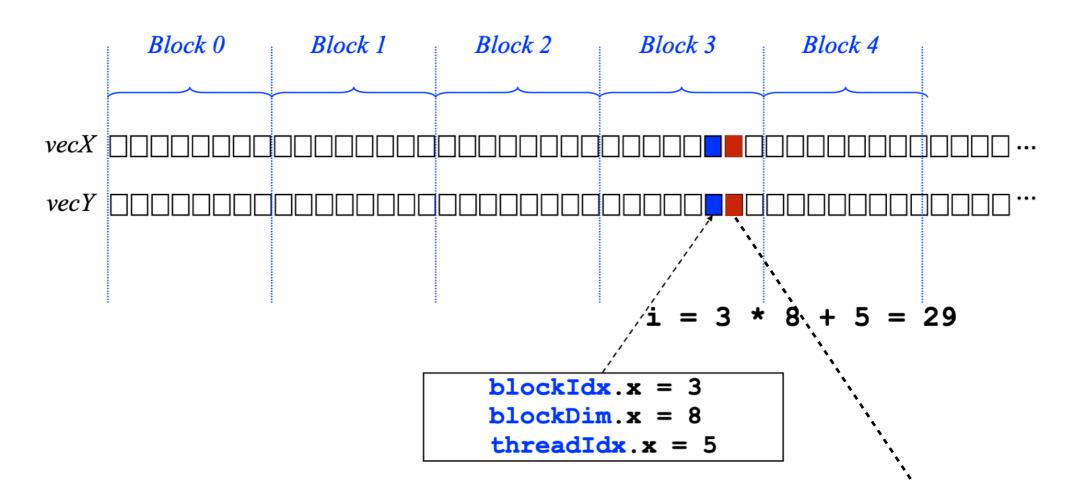
• Let's give an example:



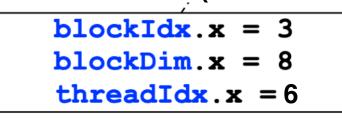
- Super Important coalesced memory:
 - Reading (global) memory should be from adjacent memory address for the threads

Index

• Let's give an example:

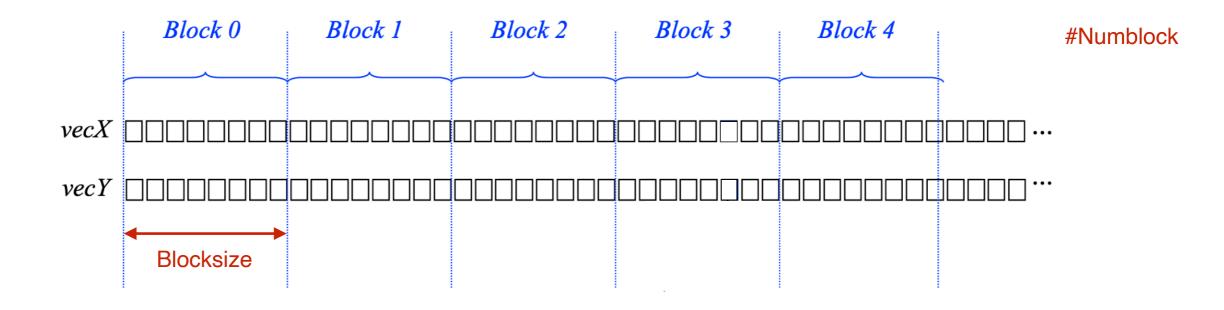


- Super Important coalesced memory:
 - Reading (global) memory should be from adjacent memory address for the threads



Kernel call

- How do you call a kernel?
 - saxpy<<<numblock, blocksize>>>(d_x, d_y, a, n)



- blocksize: number of thread in a block
 - Should be multiple of 32 (due to wrap)
 - Maximum of 2048
 - depends of the GPU capabilities

- Code steps in more details:
 - I. Initialise GPU
 - 2. Initialise variable on the host (cpu)
 - 3. Allocate memory on the device (gpu)
 - 4. Move data from host to device
 - 5. Execute kernel on device
 - 6. Move back results
 - 7. Clean up (deallocation)

- I. Initialise GPU
- 2. Initialise variable on the host (cpu)



- culnit(0) is NOT required for the code to work
 - Will be called automatically at first cuda function call
 - Nice to use for profiling
 - Otherwise first call much slower than expected

3. Allocate memory on the device (gpu)

float *d_x, *d_y; cudaMalloc(&d_x, N*sizeof(float)); cudaMalloc(&d_y, N*sizeof(float));

- cudaMalloc does NOT follow the exact same syntax as a malloc:
 x = (float*)malloc(N*sizeof(float));
 - The cuda rule for any function is to return an error code
 - So the cuda malloc does not return a pointer but has one more argument (pointer of pointer)
- Here we use "d_" prefix to indicated device pointer.
 - Useful convention for code clarity

4. Move data from host to device

cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice); cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

4. Move data from host to device

Device pointer

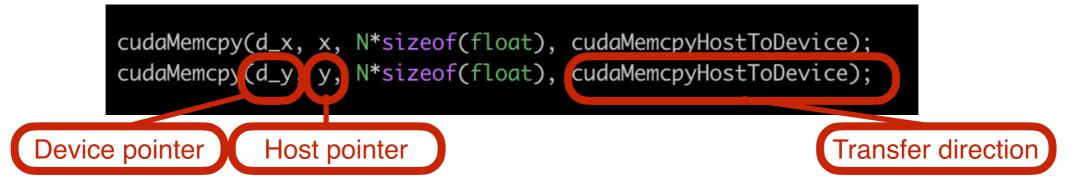
cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice); cudaMemcpy(d_y) y, N*sizeof(float), cudaMemcpyHostToDevice);



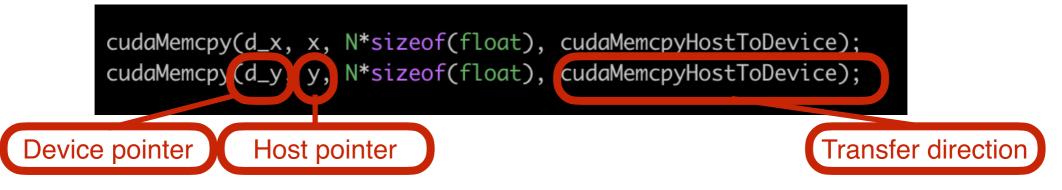
4. Move data from host to device

cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice); cudaMemcpy(d_y y, N*sizeof(float), cudaMemcpyHostToDevice); Device pointer Host pointer

4. Move data from host to device



4. Move data from host to device



- Quite slow transfer but 2 tricks:
 - I. For simple initialisation/value
 - cudaMemSet(d_x, 0, N*sizeof(xxxxx))
 - 2. Used hosted pinned memory for host
 - cudaMallocHost(&&x_host, size)
 - Slower allocation on host

5. Execute kernel on device

```
// Perform SAXPY on 1M elements
int blocksize = 512;
int nblock = N/blocksize + ( n % blocksize > 0 ? 1: 0 );
saxpy<<<nblock, blocksize>>>( d_x, d_y, 2.0f, N);
```

5. Execute kernel on device

```
// Perform SAXPY on 1M elements
int blocksize = 512;
int nblock = N/blocksize + ( n % blocksize > 0 ? 1: 0 );
saxpy<<<nblock, blocksize>>>( d_x, d_y, 2.0f, N);
```

- Computing the number of block needed
 - Special <<<A, B, C, D >>> syntax
 - A: number of block
 - B: number of thread per block
 - C: dynamically allocated shared memory
 - D: which stream to use

5. Execute kernel on device

```
// Perform SAXPY on 1M elements
int blocksize = 512;
int nblock = N/blocksize + ( n % blocksize > 0 ? 1: 0 );
saxpy<<<nblock, blocksize>>>( d_x, d_y, 2.0f, N);
```

- Computing the number of block needed
 - Special <<<A, B, C, D >>> syntax
 - A: number of block
 - B: number of thread per block
 - C: dynamically allocated shared memory
 - D: which stream to use

__global__ void saxpy(float *d_VecX, float *d_VecY, float alpha, int n)

- __global___ to use for kernel called from the host
- __device___ for GPU function call from a kernel

- 6. Move back results
- 7. Clean up (deallocation)

```
cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);
float maxError = 0.0f;
for (int i = 0; i < N; i++)
  maxError = max(maxError, abs(y[i]-4.0f));
printf("Max error: %f\n", maxError);
cudaFree(d_x);
cudaFree(d_y);
free(x);
free(y);
```

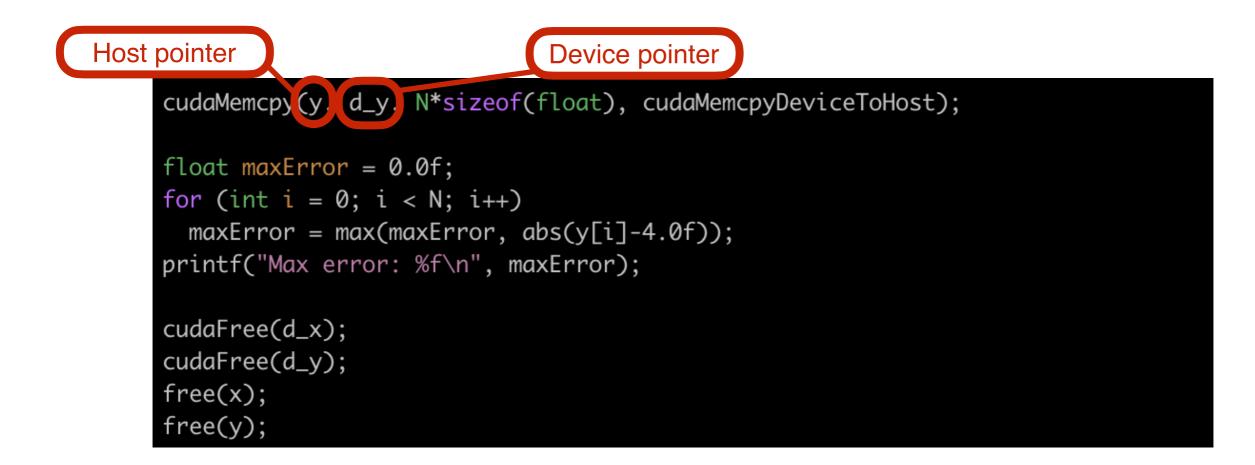
- 6. Move back results
- 7. Clean up (deallocation)

Device pointer

```
cudaMemcpy(y, d_y. N*sizeof(float), cudaMemcpyDeviceToHost);
```

```
float maxError = 0.0f;
for (int i = 0; i < N; i++)
  maxError = max(maxError, abs(y[i]-4.0f));
printf("Max error: %f\n", maxError);
cudaFree(d_x);
cudaFree(d_y);
free(x);
free(x);
free(y);
```

- 6. Move back results
- 7. Clean up (deallocation)



Full code

```
#include <stdio.h>
__global__ void saxpy(float *d_VecX, float *d_VecY, float alpha, int n)
 int i = blockIdx.x*blockDim.x + threadIdx.x;
 if (i < n) d_VecY[i] = alpha*d_VecX[i] + d_VecY[i];
int main(void)
 int N = 1 << 20;
 float *x, *y;
 x = (float*)malloc(N*sizeof(float));
 y = (float*)malloc(N*sizeof(float));
  cuInit(0);
  float *d_x, *d_y;
  cudaMalloc(&d_x, N*sizeof(float));
  cudaMalloc(&d_y, N*sizeof(float));
 for (int i = 0; i < N; i++) {
   x[i] = 1.0f;
   y[i] = 2.0f;
 }
 cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);
 // Perform SAXPY on 1M elements
 int blocksize = 512;
 int nblock = N/blocksize + ( n % blocksize > 0 ? 1: 0 );
  saxpy<<<nblock, blocksize>>>( d_x, d_y, 2.0f, N);
  cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);
  float maxError = 0.0f;
  for (int i = 0; i < N; i++)
   maxError = max(maxError, abs(y[i]-4.0f));
 printf("Max error: %f\n", maxError);
  cudaFree(d_x);
```

noold vi

• How to compile it?

Compilation of cuda code

- Module load CUDA
- nvcc -arch=sm_70 saxpy.cu -o saxpy
 - You can have additional flags for C++ par of the code (library linking, -O3,...)
 - Arch allows to have a minimum target gpu
 - No dedicated flag for additional GPU optimisation
 - ➡ GPU does support multiple file source code
 - But seriously limit optimisation
 - Cudal I starts supports for that but still limited.

Is GPU always faster?

CPU

• GPU

[omatt@mb-c Max error:	cas102 omatt]\$ t 0.000000	ime ./saxpy		@mb-cas102 omat ror: 0.000000	t]\$ time	./saxpy_cpu
user Øm@	2.401s 0.752s 0.555s		real user sys	0m0.803s 0m0.704s 0m0.097s		
• 10 ke	ernel • I	00 kernel	• 10) kernel	• 10	0 kernel
user Øm2	.029s real .284s user .091s sys	0m25.084s 0m23.164s 0m1.302s	real user sys	0m5.265s 0m5.202s 0m0.057s	real user sys	0m45.636s 0m45.524s 0m0.067s

- The gpu initialisation is large for simple problem like this one
- You need to optimise your GPU code to hide the latency, data transfer, ...
 - This GPU code can be speed-up quite a lot

• You have to manage memory: Plenty of type of memory on the GPU

6M							L1 Instr	ucti	on Car	:he								
			L0 In	struct	tion Ca	ache						L0 In	struc	tion C	ache			
			-			read/clk)						-	edule					
		Di	spatch	n Unit ((32 thr	ead/clk)					Di	spatch	ı Unit	(32 th	read/o	:lk)		
		Reg	jister	File (1	16,384	x 32-bit)					Reg	ister	File (1	16,384	4 x 32	-bit)		
INT32	INT32	FP32	FP32	FP	64				INT32	INT32	FP32	FP32	FP	64				
INT32	INT32	FP32	FP32	FP	64				INT32	INT32	FP32	FP32	FP	64				
INT32	INT32	FP32	FP32	FP	64			INT32	INT32	FP32	FP32	FP	64					
INT32	INT32	FP32	FP32	FP64	64	TENSO	R CORE		INT32	INT32	FP32	FP32	FP	64	т	NSO	R CO	RE
INT32	INT32	FP32	FP32	FP	64	TEN30			INT32	INT32	FP32	FP32	FP	64		1100	K 001	
INT32	INT32	FP32	FP32	FP	64				INT32	INT32	FP32	FP32	FP	64				
INT32	INT32	FP32	FP32	FP	64				INT32	INT32	FP32	FP32	FP	64				
INT32	INT32	FP32	FP32	FP	64				INT32	INT32	FP32	FP32	FP	64				
LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ LD/ ST ST	SFU		LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SF	:U
																		_
			L0 In	struct	tion Ca	iche		ןר ורי					istruct		ache			
		Wa				ache hread/clk)					Wai	L0 In		tion C	_			
			rp Sch	eduler	r (32 th							L0 In rp Sch	struct	tion Ca r (32 tl	hread	/clk)		
		Di	rp Sch spatch	eduler 1 Unit (r (32 th (32 thr	nread/clk)					Di	L0 In rp Sch spatch	istruci iedule	tion Ca r (32 tl (32 th	hread read/c	/clk) :lk)		
INT32	INT32	Di Reg	rp Sch spatch jister	eduler 1 Unit (r (32 th (32 thr 16,384	nread/clk) ead/clk)				INT32	Di	L0 In rp Sch spatch ister	istruci iedule i Unit	tion C: r (32 tl (32 th (32 th 16,384	hread read/c	/clk) :lk)		
	INT32 INT32	Di Reg FP32	rp Sch spatch jister FP32	eduler 1 Unit (File (1	r (32 th (32 thr 16,384 64	nread/clk) ead/clk)			INT32		Di Reg	L0 In rp Sch spatch ister FP32	edule Unit File (1	tion C: r (32 th (32 th (32 th 16,384	hread read/c	/clk) :lk)		
INT32		Di Reg FP32	rp Sch spatch jister FP32 FP32	File (1	r (32 th (32 thr 16,384 64 64	nread/clk) ead/clk)			INT32	INT32	Di Reg	L0 In rp Sch spatch ister FP32 FP32	nstruct nedule n Unit File (1 FP	tion C r (32 th (32 th (32 th (32 th (32 th (32 th) (32 th) (3	hread read/c	/clk) :lk)		
INT32 INT32	INT32	Di Reg FP32 FP32 FP32	rp Sch spatch jister FP32 FP32 FP32	eduler 1 Unit (File (1 FP	r (32 th (32 thr 16,384 64 64 64	nread/clk) ead/clk) x 32-bit)	R CORE		INT32 INT32	INT32 INT32	Dis Reg FP32 FP32 FP32	L0 In rp Sch spatch jister FP32 FP32 FP32	nstruct edule 1 Unit File (1 FP FP	tion Ci r (32 th (32 th (32 th (32 th (32 th) (32 th)(hread read/c 4 x 32	/clk) :lk) !-bit)	R COI	
INT32 INT32 INT32	INT32 INT32	Di Reg FP32 FP32 FP32 FP32	rp Sch spatch jister FP32 FP32 FP32 FP32	eduler 1 Unit (File (1 FP FP	r (32 th (32 thr 16,384 64 64 64 64	nread/clk) ead/clk) x 32-bit)	R CORE		INT32 INT32 INT32 INT32	INT32 INT32 INT32	Dis Reg FP32 FP32 FP32 FP32	L0 In rp Sch spatch ister FP32 FP32 FP32 FP32	istruci edule 1 Unit File (1 FP FP	tion Ci r (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th) (32	hread read/c 4 x 32	/clk) :lk) !-bit)	R COI	
INT32 INT32 INT32 INT32	INT32 INT32 INT32	Di Reg FP32 FP32 FP32 FP32 FP32	rp Sch spatch jister FP32 FP32 FP32 FP32 FP32	eduler Unit (File (1 FP FP FP	r (32 th (32 thr 16,384 64 64 64 64 64	nread/clk) ead/clk) x 32-bit)	R CORE		INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32 INT32	Dis Reg FP32 FP32 FP32 FP32	L0 In rp Sch spatch ister FP32 FP32 FP32 FP32 FP32	struct Unit File (1 FP FP FP	tion C r (32 th (32 th (32 th (32 th (32 th) (32 th) (hread read/c 4 x 32	/clk) :lk) !-bit)	R CO	
INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32	Di Reg FP32 FP32 FP32 FP32 FP32 FP32	rp Sch spatch jister FP32 FP32 FP32 FP32 FP32 FP32	eduler 1 Unit (File (1 FP FP FP FP	r (32 th (32 thr 16,384 64 64 64 64 64 64	nread/clk) ead/clk) x 32-bit)	R CORE		INT32 INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32 INT32 INT32	Dis Reg FP32 FP32 FP32 FP32 FP32	L0 In rp Sch spatch ister FP32 FP32 FP32 FP32 FP32 FP32 FP32	edule Unit File (1 FP FP FP FP	tion C: r (32 th (32 th) 16,384 64 64 64 64 64 64	hread read/c 4 x 32	/clk) :lk) !-bit)	R COI	
INT32 INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32 INT32 INT32	Di Reg FP32 FP32 FP32 FP32 FP32 FP32 FP32	rp Sch spatch ister FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	eduler n Unit (File (1 FP FP FP FP FP FP	r (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th) (32	rread/clk) ead/clk) x 32-bit) TENSO	R CORE		INT32 INT32 INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32 INT32 INT32 INT32	Dii Reg FP32 FP32 FP32 FP32 FP32 FP32 FP32	L0 In rp Sch spatch ister FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	istruct edule 1 Unit File (1 FP FP FP FP FP FP	tion C. r (32 th (32 th) 16, 384 64 64 64 64 64 64 64 64	hread read/o 4 x 32	/clk) ilk) I-bit)	R CO	
INT32 INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32 INT32 INT32	Di Reg FP32 FP32 FP32 FP32 FP32 FP32	rp Sch spatch fP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	eduler I Unit (File (1 FP FP FP FP	r (32 th (32 thr 64 64 64 64 64 64 64	nread/clk) ead/clk) x 32-bit)	R CORE		INT32 INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32 INT32 INT32	Dis Reg FP32 FP32 FP32 FP32 FP32 FP32	L0 In rp Sch spatch fF932 FP32 FP32 FP32 FP32 FP32 FP32 FP32	istruct edule 1 Unit File (1 FP FP FP FP FP	tion C. (32 th (32 th (32 th (32 th (32 th (34 th) (34	hread read/c 4 x 32	/clk) :lk) !-bit)	R COI	RE
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32 INT32 INT32 INT32 LD/	Di Reg FP32 FP32 FP32 FP32 FP32 FP32 FP32 LD/	rp Sch spatch ister FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	eduler a Unit (File (1 FPI FPI FPI FPI FPI FPI FPI FPI	r (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th (32 th) (32	rread/clk) ead/clk) x 32-bit) TENSO			INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	Dit Reg FP32 FP32 FP32 FP32 FP32 FP32 FP32 LD/ ST	L0 In rp Sch spatch fP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 F	istruct edule 1 Unit File (1 FP FP FP FP FP FP FP	tion C: r (32 th (32 th) (32 th) (32 th) (32 th) (64 (64 (64 (64 (64 (64 (64 (64) (64)	hread read/c 4 x 32 TE LD/	/clk) :lk) I-bit)		RE

Inside each SM:

CECI training: Cuda

• You have to manage memory: Plenty of type of memory on the GPU

M				L1 Instr	uctio	on Cac	:he							
LC	Instruct	ion Cach	e						L0 In	struct	tion C	ache		
Warp S	cheduler	r (32 thre	ad/clk)					War	p Sch	edule	r (32 t	hread	/clk)	
								-						
		6,384 x	32-bit)					Reg	ister I	File (1	16,384	4 x 32	2-bit)	
NISZINISZI FOZI F						INTOE	1141-52	FF 92	FFJE		0 4			
INT32 INT32 FP32 FP3	2 FP	64				INT32	INT32	FP32	FP32	FP	64			
INT32 INT32 FP32 FP3	2 FP	64				INT32	INT32	FP32	FP32	FP64				
INT32 INT32 FP32 FP3	2 FP	64				INT32 INT32 FP32 FP32 F		FP	64					
INT32 INT32 FP32 FP3	2 FP		TENSO	SOR CORE		INT32	INT32	FP32	FP32	FP	64	TENSOR CORE		
INT32 INT32 FP32 FP3	2 FP	64				INT32	INT32	FP32	FP32	FP	64			
INT32 INT32 FP32 FP3	2 FP	64				INT32	INT32	FP32	FP32	FP	64			
INT32 INT32 FP32 FP3		64						FP32		FP	64			
LD/ LD/ LD/ LD	/ LD/		-					LD/	LD/	LD/	LD/	LD/		
ST ST ST ST	ST	LD/ LC ST S ⁻	r st	SFU		LD/ ST	LD/ ST	ST	ST	ST	ST	ST	LD/ ST	SFU
ST ST ST ST ST	ST Instruct cheduler		r st le ad/clk)	SFU				Wai	ST	ST Istruct edule	ST tion C r (32 t	ST ache thread	ST /clk)	SFU
ST ST ST ST ST	ST Instruct cheduler	ST S	r st le ad/clk)	SFU				Wai	ST LO In rp Sch	ST Istruct edule	ST tion C r (32 t	ST ache thread	ST /clk)	SFU
ST ST ST ST ST	ST Instruct cheduler er File (1	st s ion Cach (32 thre 6,384 x	r st le ad/clk)	SFU		ST	ST	ST War Reg	ST LO In p Sch ister I	st struct edule File (1	ST tion C (32 t (33 th 16,38	ST ache thread	ST /clk)	SFU
ST ST ST ST ST Warp S Piene Register INT32 INT32 FP32 FP3	ST Instruct cheduler er File (1 2 FP	ST S' ion Cach (32 three (32 three (32 three (6,384 x (32 three 64 (34 three	r st le ad/clk)	SFU		ST	ST	ST War Reg	ST LO In p Sch ister I FP32	struct edule File (1	ST tion C (32 t (33 th (33 th (33 th (33 th (33 th (33 th) (33 th) (33 th) (33 th) (33 th) (33 th) (33 th) (33 th) (33 th) (34 th) (35	ST ache thread	ST /clk)	SFU
ST ST ST ST ST Warp S Register INT32 INT32 FP32 FP3 INT32 INT32 FP32 FP3	st Instruct cheduler er File (1 2 FP 2 FP	ST S' ion Cach (32 three 6,384 x (32 three 64 (42 three	r st le ad/clk)	SFU		ST	ST INT32	ST War Reg FP32 FP32	ST LO In rp Sch ister I FP32 FP32	struct edule File (1 FP FP	ST tion C (32 t (23 t) 16,384 64 64	ST ache thread	ST /clk)	SFU
ST S	sT Instruct cheduler r File (1 2 FPI 2 FPI 2 FPI	ST S' ion Cach (32 three (32 three (32 three 6,384 x (32 three 64 (34 three 64 (34 three	e ad/clk) at/clb) 32-bit)	SFU		ST	ST INT32 INT32 INT32	ST War Reg FP32 FP32 FP32	ST L0 In rp Sch ister I FP32 FP32 FP32	ST edule File (1 FP FP	ST iion C r (32 t 16,38 64 64 64	st ache hread 4 x 32	st /clk) 2-bit)	R CORE
ST ST ST ST ST ST Warp S Register INT32 INT32 FP32 FP3 INT32 INT32 FP32 FP3 INT32 INT32 FP32 FP3	sT Instruct cheduler er File (1 2 FPI 2 FPI 2 FPI 2 FPI	ST S' ion Cach (32 three (32 three (32 three 6,384 x (32 three 64 (34 three 64 (34 three	e ad/clk) at/clb) 32-bit)			ST	ST INT32 INT32 INT32 INT32	ST Wan Pii Reg FP32 FP32 FP32 FP32	st L0 In p Sch iister I FP32 FP32 FP32 FP32	ST edule File (1 FP FP FP	ST (ion C ((32 t) (32	st ache hread 4 x 32	st /clk) 2-bit)	
ST ST ST ST ST Warp S Register INT32 INT32 FP32 FP3 INT32 INT32 FP32 FP3	st Instruct cheduler	ST S' ion Cach (32 three 6,384 x (32 three 64 (32 three 64 (32 three 64 (32 three 64 (32 three	e ad/clk) at/clb) 32-bit)			ST IN132 IN132 IN132 IN132 IN132	ST INT32 INT32 INT32 INT32	ST Wan Reg FP32 FP32 FP32 FP32 FP32	ST LO In p Sch iister I FP32 FP32 FP32 FP32 FP32	st istruct edulei File (1 FP FP FP FP	ST iion C (22 t 16,38 64 64 64 64 64	st ache hread 4 x 32	st /clk) 2-bit)	
ST ST ST ST ST ST Warp S Register INT32 INT32 FP32 FP3 INT32 INT32 FP32 FP3	ST Instruct cheduler cheduler 2 FPI 2 FPI 2 FPI 2 FPI 2 FPI	ST S' ion Cach (32 three (32 three (32 three 6,384 x (32 three 64 (34 three	e ad/clk) at/clb) 32-bit)			ST INT32 INT32 INT32 INT32 INT32	ST INT32 INT32 INT32 INT32 INT32	ST Wan Reg FP32 FP32 FP32 FP32 FP32 FP32	ST LO Im p Sch ister I FP32 FP32 FP32 FP32 FP32 FP32 FP32	ST edule File (1 FP FP FP FP	ST (32 t) (32 t) (32 t) (32 t) (32 t) (32 t) (32 t) (42 t)	st ache hread 4 x 32	st /clk) 2-bit)	
ST ST ST ST ST ST UU Warp S Warp S Regista INT32 INT32 FP32 FP32 INT32 INT32 FP32 FP32 FP32 INT32 INT32 INT32 FP32 FP32 INT32 INT32 INT32 INT32 FP32 FP32 INT32 INT32 INT32 INT32 FP32 FP32 INT32 INT32 INT32 INT32 INT32 INT32	ST Instruct cheduler	ST S' ion Cach (32 three (32 three (32 three 6,384 x (32 three 64 (34 three	r st e ad/clk) d/cl/d 32-bit) TENSO	R CORE		ST INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	ST INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	ST War Reg FP32 FP32 FP32 FP32 FP32 FP32 FP32 LD/	ST LO Im p Sch ister I FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ST istruct edule: File (1 FP FP FP FP FP FP FP	ST tion C (32 t (32 t) (32 t) (32 t) (32 t) (32 t) (42 t)	st iache hread 4 x 32 TE	st /clk) 2-bit) ENSO	R CORE
ST ST ST ST ST ST UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	ST Instruct cheduler	ST S' ion Cach (32 three (32 three (32 three 6,384 x (32 three 64 (34 three	r st e ad/clk) 32-bit) TENSO	IR CORE		ST INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	ST INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	ST War Reg FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ST LO Im p Sch ister I FP32 FP32 FP32 FP32 FP32 FP32 FP32 LD/ ST	st edulei File (1 FP FP FP FP FP	ST (321) (321) (321) (321) (321) (321) (321) (42) (42) (42) (42) (42) (42) (42) (42	st iache hread 4 x 32	st /clk) 2-bit)	
ST ST <thst< th=""> ST ST ST<!--</td--><td>ST Instruct cheduler</td><td>ST S' ion Cach (32 three (32 three (32 three 6,384 x (32 three 64 (34 three</td><td>r st e ad/clk) 32-bit) TENSO</td><td>R CORE</td><td></td><td>ST INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32</td><td>ST INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32</td><td>ST War Reg FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32</td><td>ST LO Im p Sch ister I FP32 FP32 FP32 FP32 FP32 FP32 FP32 LD/ ST</td><td>ST istruct edule: File (1 FP FP FP FP FP FP FP</td><td>ST tion C (32 t (32 t) (32 t) (32 t) (32 t) (32 t) (42 t)</td><td>st iache hread 4 x 32 TE</td><td>st /clk) 2-bit) ENSO</td><td>R CORE</td></thst<>	ST Instruct cheduler	ST S' ion Cach (32 three (32 three (32 three 6,384 x (32 three 64 (34 three	r st e ad/clk) 32-bit) TENSO	R CORE		ST INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	ST INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	ST War Reg FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	ST LO Im p Sch ister I FP32 FP32 FP32 FP32 FP32 FP32 FP32 LD/ ST	ST istruct edule: File (1 FP FP FP FP FP FP FP	ST tion C (32 t (32 t) (32 t) (32 t) (32 t) (32 t) (42 t)	st iache hread 4 x 32 TE	st /clk) 2-bit) ENSO	R CORE

Inside each SM:

- Register
 - Fastest memory
 - Thread specific
 - Very limited amount
 - Overflow goes to L1

CECI training: Cuda

• You have to manage memory: Plenty of type of memory on the GPU

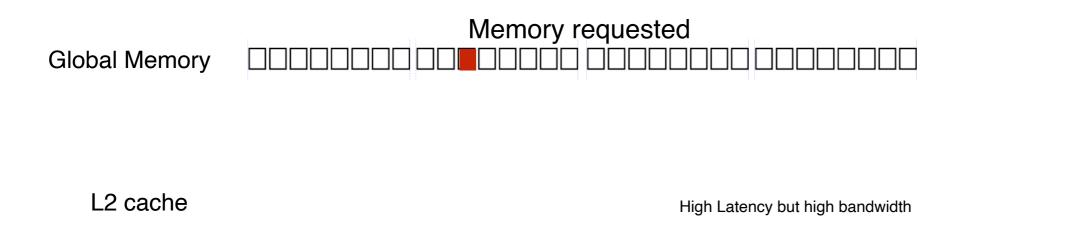
			L1 Instruc	tion Cache				
	L0 I	nstruction C	ache	LOI	nstruction C	ache		
	Warp Scl	neduler (32 t	hread/clk)	Warp Scl	neduler (32 t	hread/clk)		
	Dispatc	h Unit (32 th	read/clk)	Dispatc	h Unit (32 th	read/clk)		
	Register	File (16,38	4 x 32-bit)	Register	File (16,38	4 x 32-bit)		
INT32 INT32	FP32 FP32	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	751000 0005		
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			
LD/ LD/ ST ST	LD/ LD/ ST ST	LD/ LD/ ST ST	LD/ LD/ ST ST SFU	LD/ LD/ LD/ LD/ ST ST ST ST	LD/ LD/ ST ST	LD/ LD/ SFU		
	Warp Scl Dispatc	nstruction C neduler (32 t h Unit (32 th File (16,38	hread/clk) read/clk)	Warp Scl Dispatc	nstruction C neduler (321 h Unit (321h File (16,38	thread/clk) nread/clk)		
INT32 INT32	Warp Scl Dispatc Register	neduler (32 t h Unit (32 th	hread/clk) read/clk)	Warp Scl Dispatc	nstruction C neduler (32 t h Unit (32 th	thread/clk) nread/clk)		
INT32 INT32	Warp Scl Dispatc Register FP32 FP32	neduler (32 t h Unit (32 th File (16,384	hread/clk) read/clk)	Warp Scl Dispatc Register	nstruction C neduler (32 t h Unit (32 th File (16,38	thread/clk) nread/clk)		
	Warp Scl Dispatc Register FP32 FP32 FP32 FP32	neduler (32 t h Unit (32 th File (16,38 FP64	hread/clk) read/clk)	Warp Scl Dispatc Register	nstruction C neduler (32 th h Unit (32 th File (16,38 FP64	thread/clk) nread/clk)		
INT32 INT32	Warp Scl Dispatc Register FP32 FP32 FP32 FP32 FP32 FP32	heduler (32 t h Unit (32 th File (16,38 FP64 FP64	hread/clk) read/clk) 4 x 32-bit)	Warp Scl Dispatc Register INT32 INT32 FP32 FP32 INT32 INT32 FP32 FP32	nstruction C neduler (32 th h Unit (32 th File (16,38 FP64 FP64	thread/clk) aread/clk) 4 x 32-bit)		
INT32 INT32 INT32 INT32	Warp Scl Dispatc Register FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	heduler (32 th h Unit (32 th File (16,38 FP64 FP64 FP64	hread/clk) read/clk)	Warp Scl Dispatc Register INT32 INT32 FP32 FP32 INT32 INT32 FP32 FP32 INT32 INT32 FP32 FP32	nstruction C neduler (32 th h Unit (32 th File (16,38 FP64 FP64 FP64	thread/clk) nread/clk)		
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	Warp Scl Dispatc Register FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	heduler (32 th h Unit (32 th File (16,38 FP64 FP64 FP64 FP64	hread/clk) read/clk) 4 x 32-bit)	Warp Scl Dispate Register INT32 INT32 FP32 FP32	nstruction C neduler (32 1 h Unit (32 th File (16,38 FP64 FP64 FP64 FP64	thread/clk) aread/clk) 4 x 32-bit)		
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	Warp Scl Dispatc Register FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	neduler (32 t h Unit (32 th File (16,38 FP64 FP64 FP64 FP64 FP64	hread/clk) read/clk) 4 x 32-bit)	Warp Scl Dispate Register INT32 INT32 FP32 FP32	nstruction C neduler (32 th h Unit (32 th File (16,38 FP64 FP64 FP64 FP64 FP64	thread/clk) aread/clk) 4 x 32-bit)		
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	Warp Scl Dispatc Register FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	neduler (32 t h Unit (32 th File (16,38 FP64 FP64 FP64 FP64 FP64 FP64	hread/clk) read/clk) 4 x 32-bit)	Warp Scl Dispate INT32 INT32 FP32 FP32 INT32 INT32 FP32 FP32 FP32 INT32 INT32 FP32 FP32 FP32 INT32 INT32 FP32 FP32 FP32 INT32 INT32 FP32 FP32 FP32 INT32 INT32 FP32 FP32 FP32 INT32 INT32 FP32 FP32 FP32 INT32 INT32 FP32 FP32 FP32 INT32 INT32 FP32 FP32 FP32	nstruction C neduler (32 th h Unit (32 th File (16,38 FP64 FP64 FP64 FP64 FP64 FP64 FP64	thread/clk) aread/clk) 4 x 32-bit)		
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	Warp Scl Dispatc Register FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	neduler (32 t h Unit (32 th File (16,38 FP64 FP64 FP64 FP64 FP64 FP64 FP64	hread/clk) read/clk) 4 x 32-bit)	Warp Scl Dispate Dispate Register INT32 INT32 FP32 FP32	nstruction C neduler (32 th h Unit (32 th File (16,38 FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	thread/clk) aread/clk) 4 x 32-bit)		
INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32 INT32	Warp Scl Dispatc Register FP32 FP3	reduler (32 th h Unit (32 th File (16,38 FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	hread/clk) read/clk) 4 x 32-bit) TENSOR CORE	Warp Scl Dispate Dispate Register INT32 INT32 FP32 FP32 INT32 INT32 INT32 FP32 INT32 INT32 INT32 FP32	nstruction C neduler (32 th h Unit (32 th File (16,38 FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	thread/clk) aread/clk) 4 x 32-bit) TENSOR CORE		

Inside each SM:

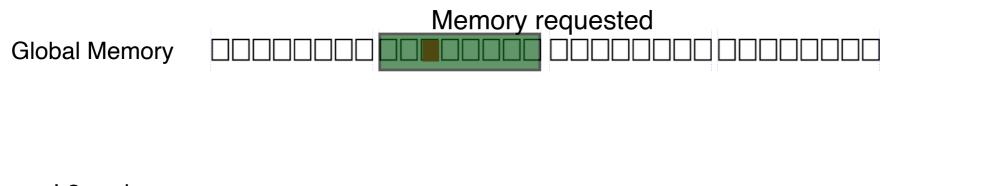
- Register
 - Fastest memory
 - Thread specific
 - Very limited amount
 - Overflow goes to L1
- Shared memory
 - Limited amount
 - Block wide memory
 - __shared_

CECI training: Cuda

- You have to manage memory: Plenty of type of memory on the GPU
- Outside the SM
 - Global memory
 - High bandwidth (900Gb/s) but High latency
 - High number of thread need to hide this latency
 - Default memory for cpu/gpu pointer

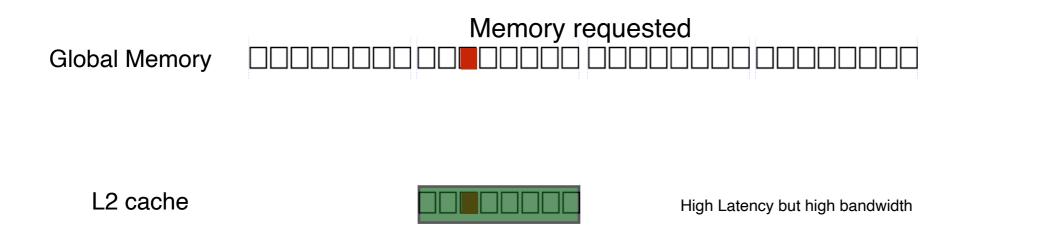


- You have to manage memory: Plenty of type of memory on the GPU
- Outside the SM
 - Global memory
 - High bandwidth (900Gb/s) but High latency
 - High number of thread need to hide this latency
 - Default memory for cpu/gpu pointer

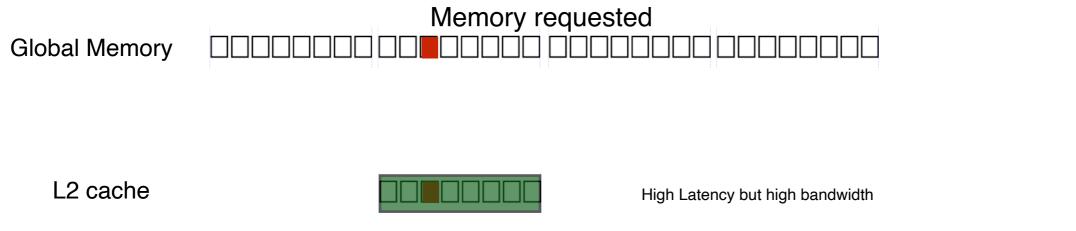


High Latency but high bandwidth

- You have to manage memory: Plenty of type of memory on the GPU
- Outside the SM
 - Global memory
 - High bandwidth (900Gb/s) but High latency
 - High number of thread need to hide this latency
 - Default memory for cpu/gpu pointer



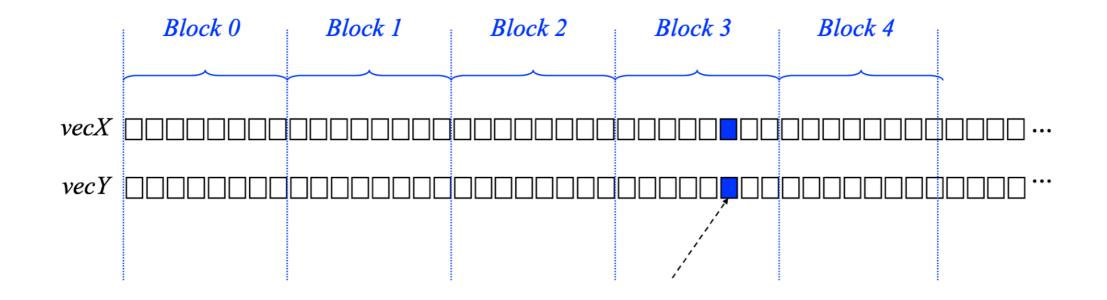
- You have to manage memory: Plenty of type of memory on the GPU
- Outside the SM
 - Global memory
 - High bandwidth (900Gb/s) but High latency
 - High number of thread need to hide this latency
 - Default memory for cpu/gpu pointer



Efficiency in memory transfer = maximise the usage of the data transfer, need to use all data in a block before needing a new block transfer

Index

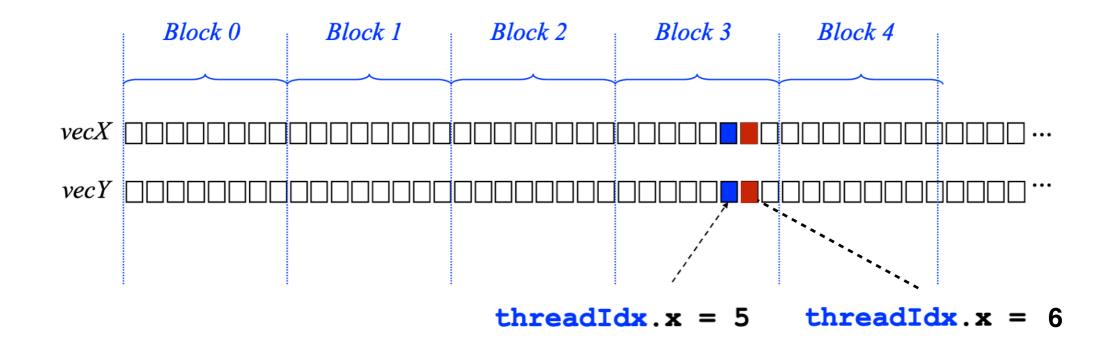
• This is how the memory should be read/write by the various thread



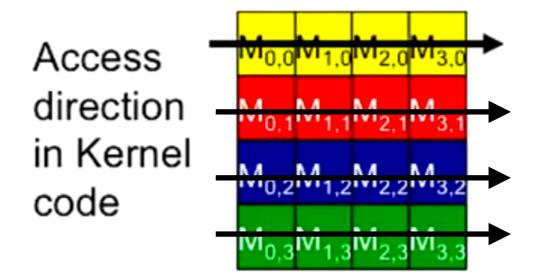
• You need to be careful with 2D array to be sure that you follow that pattern

Index

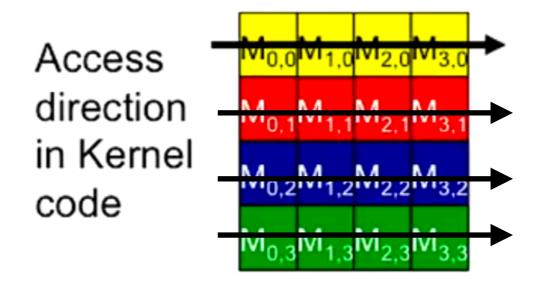
• This is how the memory should be read/write by the various thread

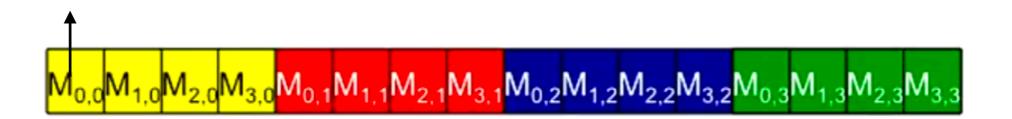


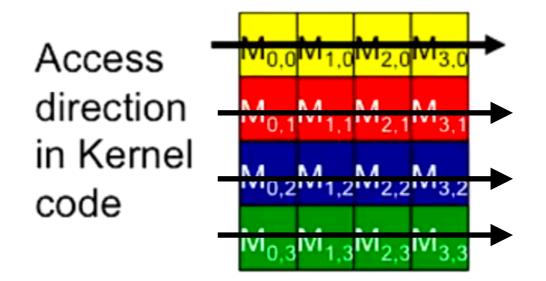
• You need to be careful with 2D array to be sure that you follow that pattern

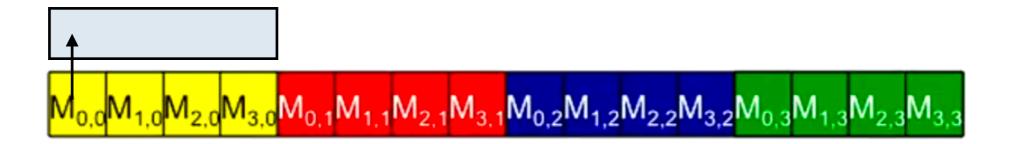


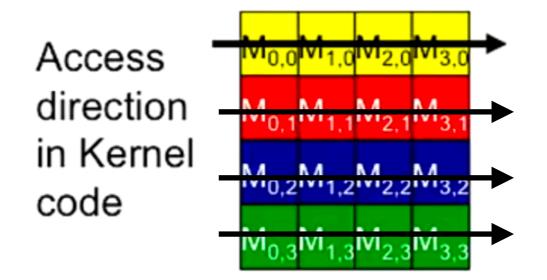
 $M_{0,0}M_{1,0}M_{2,0}M_{3,0}M_{0,1}M_{1,1}M_{2,1}M_{3,1}M_{0,2}M_{1,2}M_{2,2}M_{3,2}M_{0,3}M_{1,3}M_{2,3}M_{3,3}$

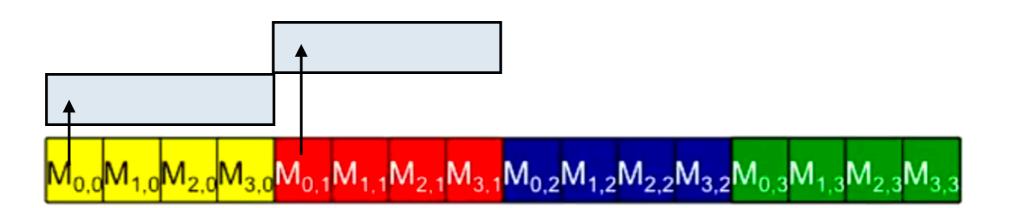


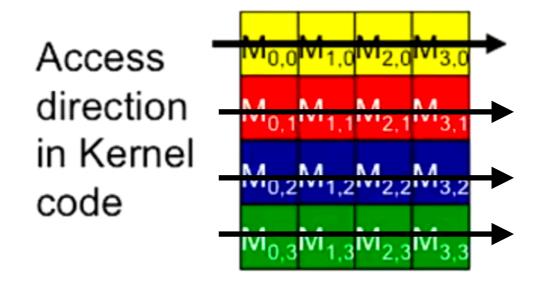


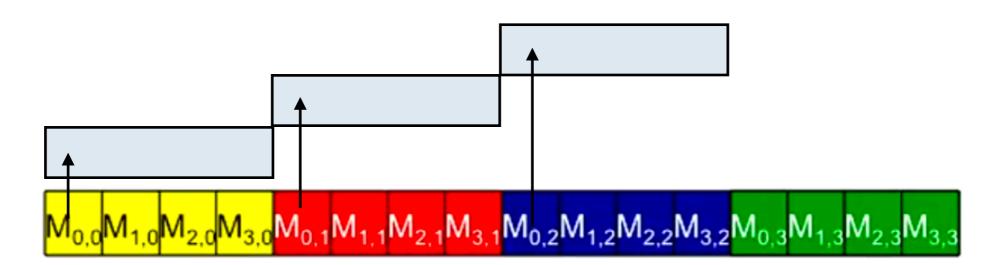


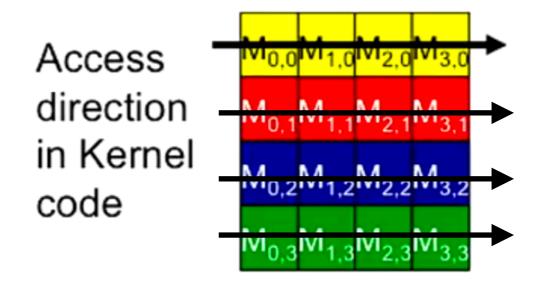


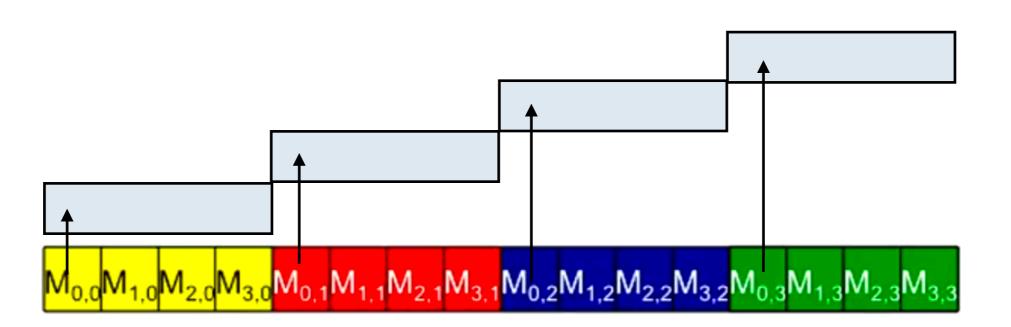


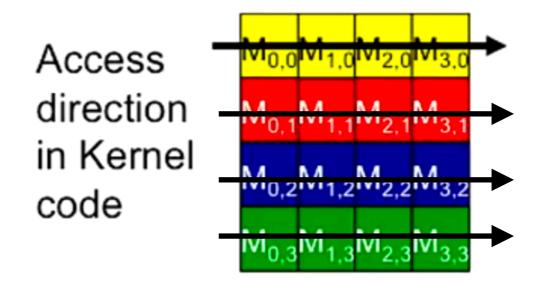


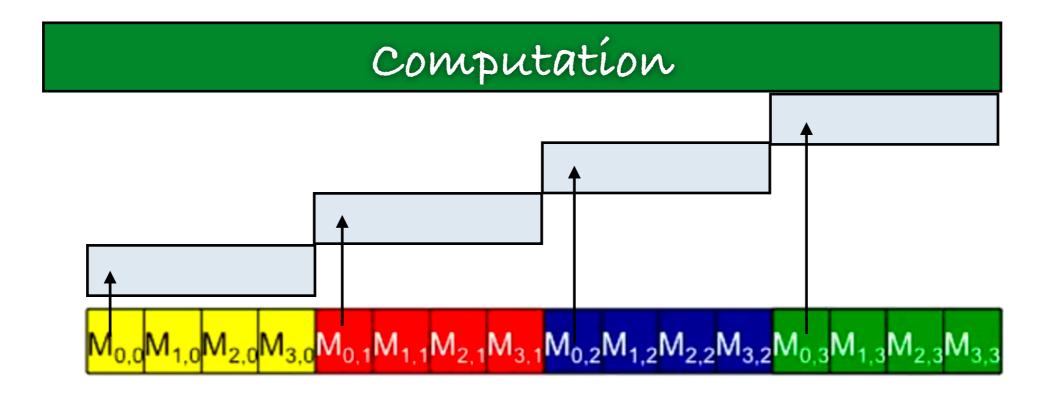


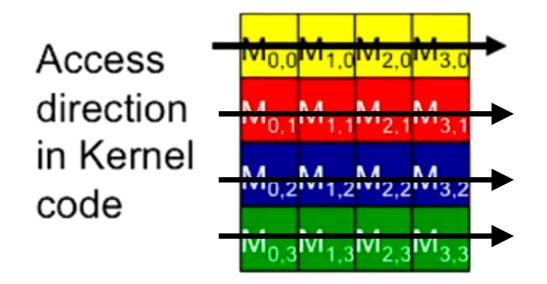


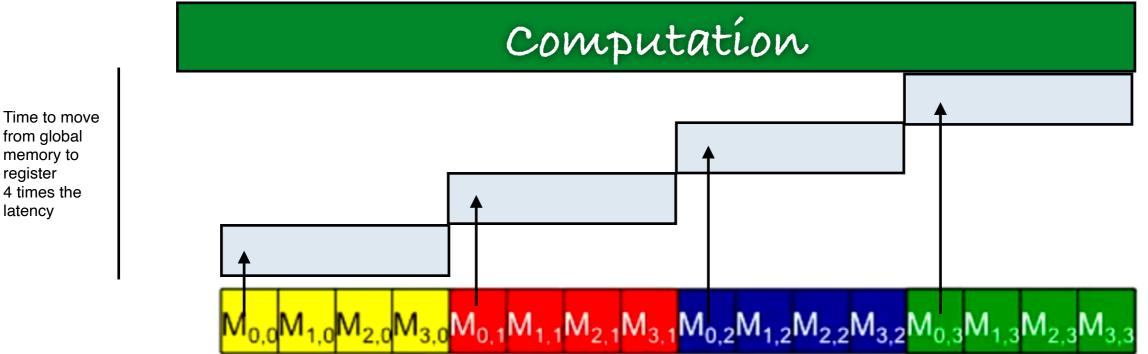




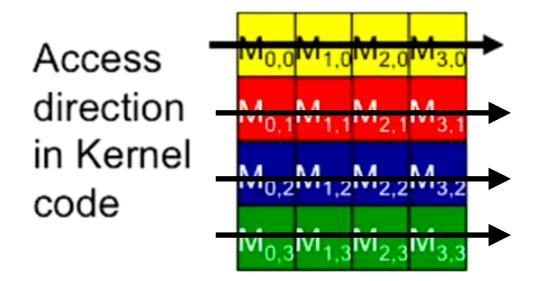






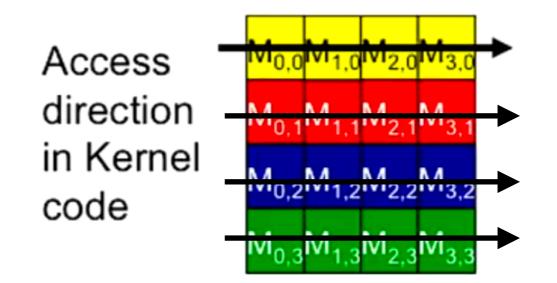


latency

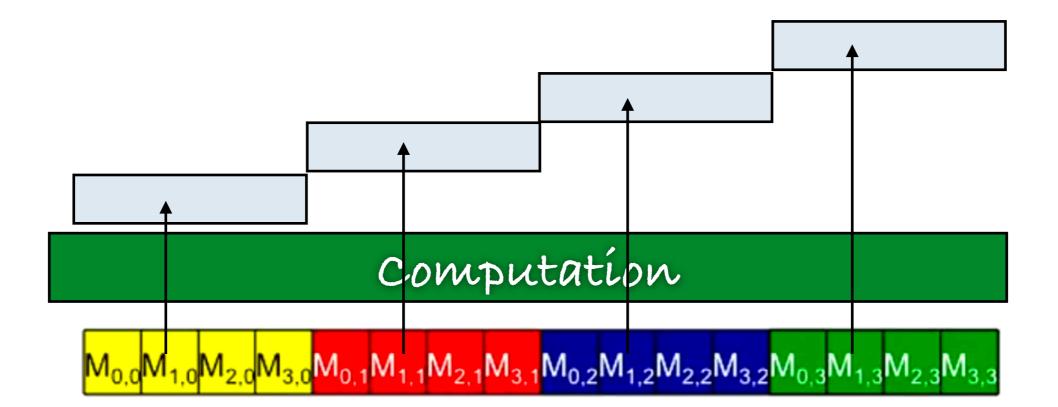


Computation

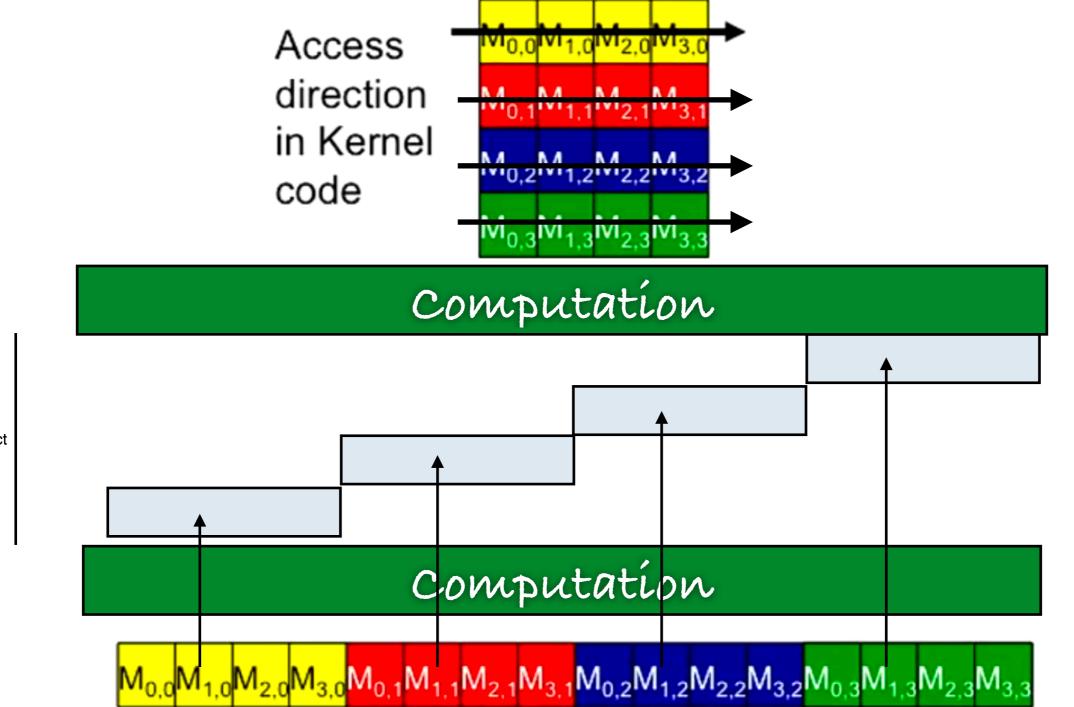
 $M_{0,0}M_{1,0}M_{2,0}M_{3,0}M_{0,1}M_{1,1}M_{2,1}M_{3,1}M_{0,2}M_{1,2}M_{2,2}M_{3,2}M_{0,3}M_{1,3}M_{2,3}M_{3,3}$



Likely not in cache anymore -> need to extract -> four times the latency

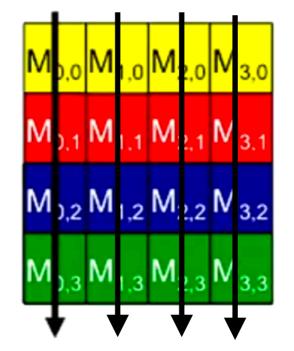


Uncoalesced Memory



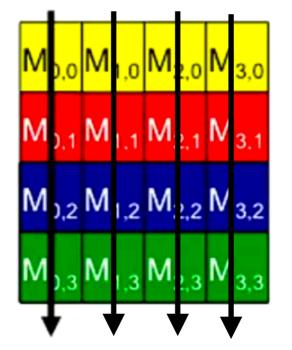
Likely not in cache anymore -> need to extract -> four times the latency

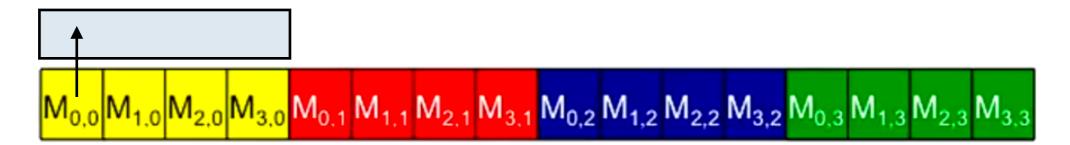
Access direction in Kernel code



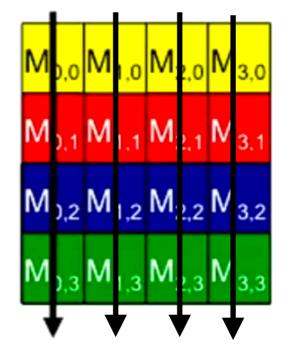


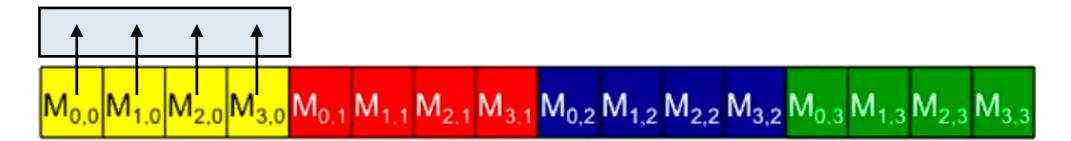
Access direction in Kernel code



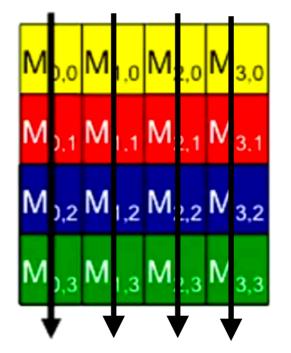


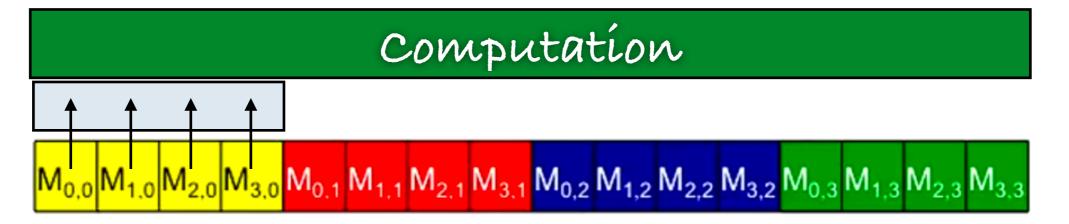
Access direction in Kernel code



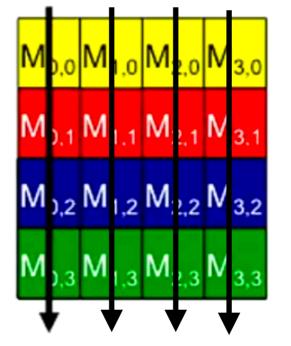


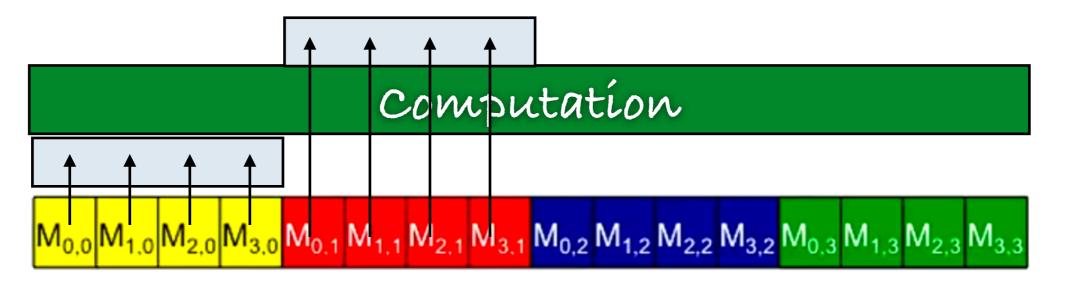
Access direction in Kernel code



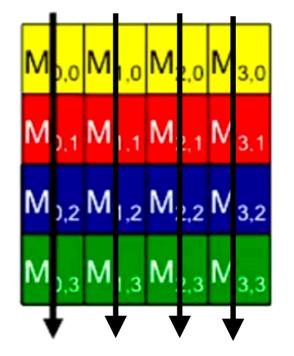


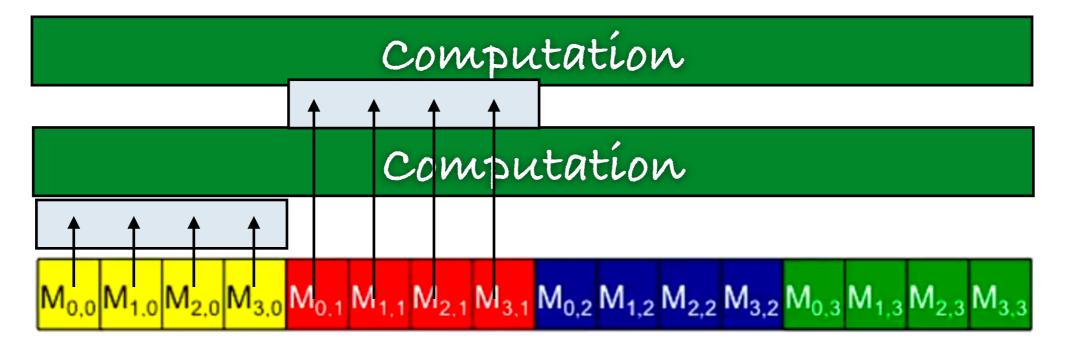




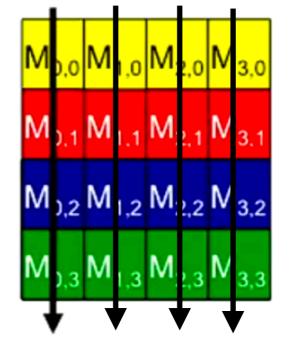


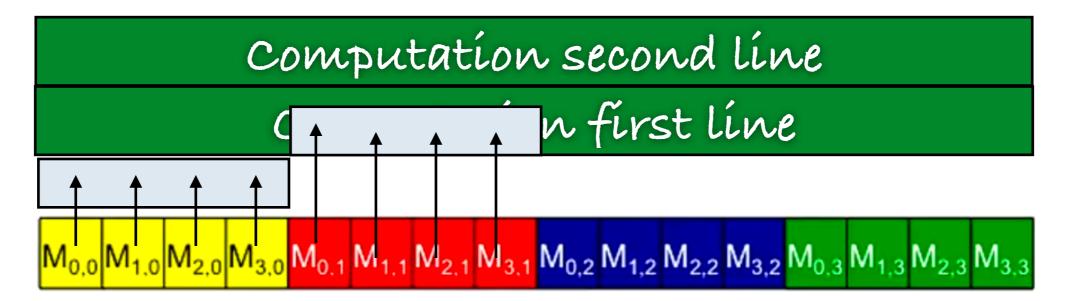
Access direction in Kernel code





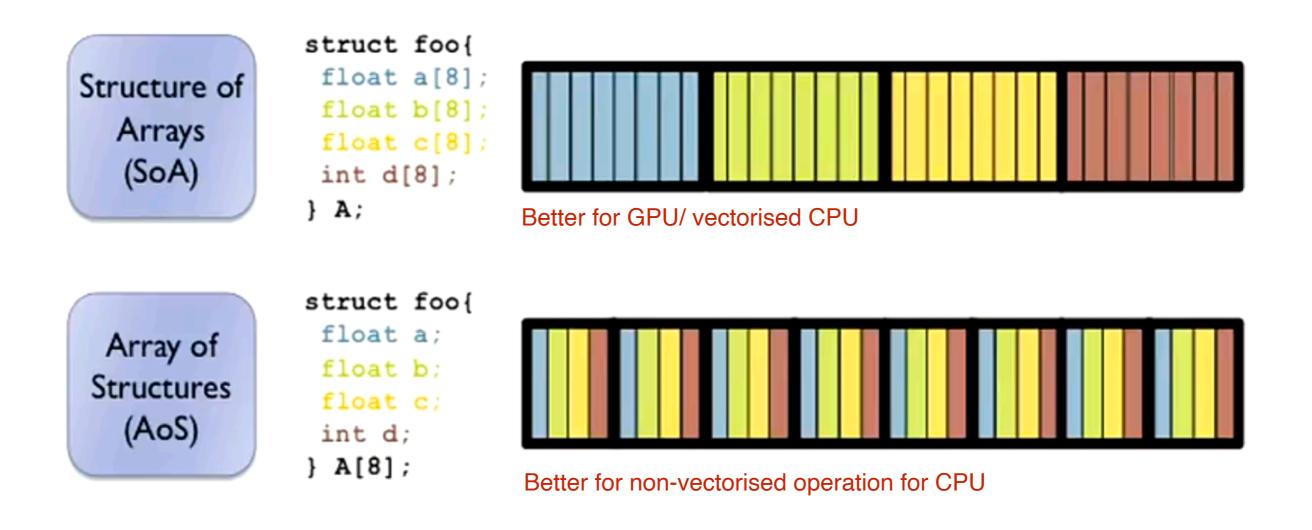






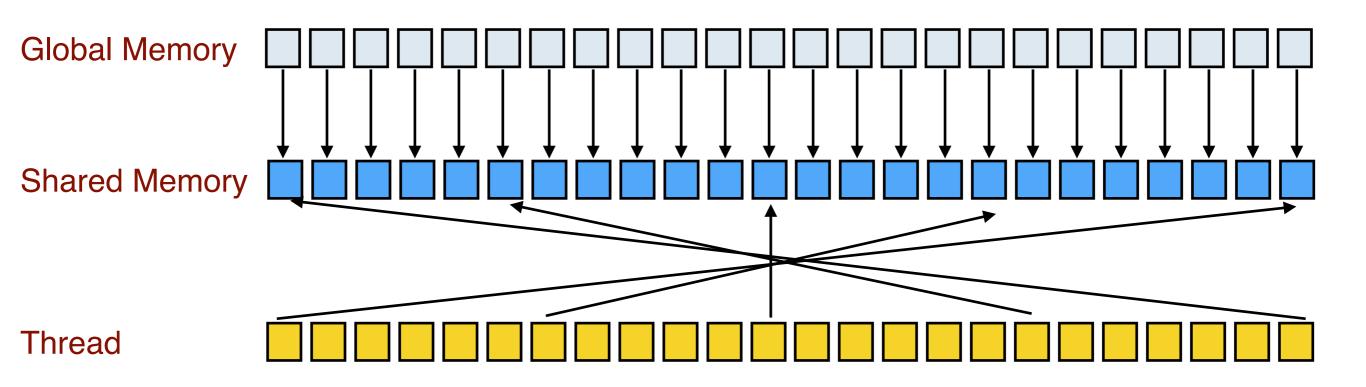
Structure of array

AoS vs. SoA



Coalesced access

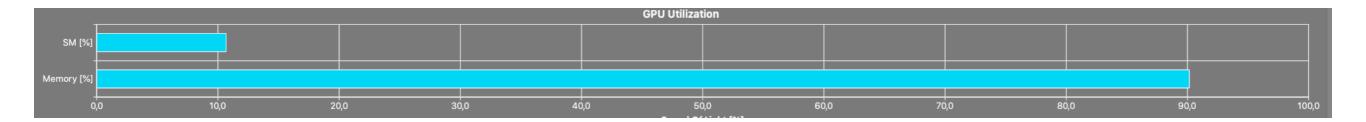
- Coalesced access not possible?
 - Use shared memory as a cache



CUDA profiler

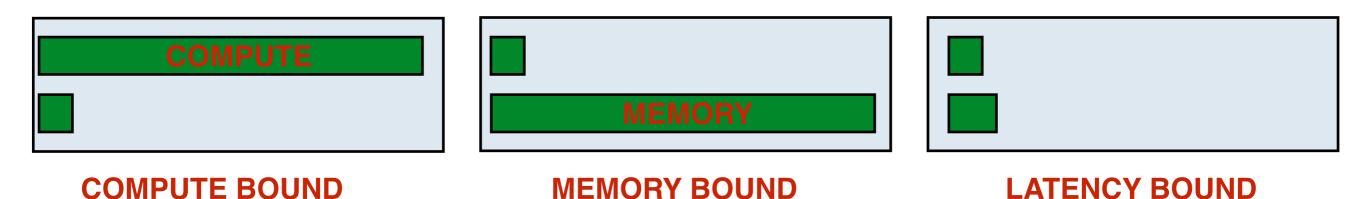
- nv-nsight-cu-cli -o profile --target-processes all ./ saxpy
 - Executable is also sometimes "ncu"
 - The more convenient is to download back that profile on your laptop and use "nsight compute" to visualise the data (do not need a GPU on that machine)
- On cluster mode, you need to be sudo to run those command. Contact us if needed.

What is the limitation of your kernel?

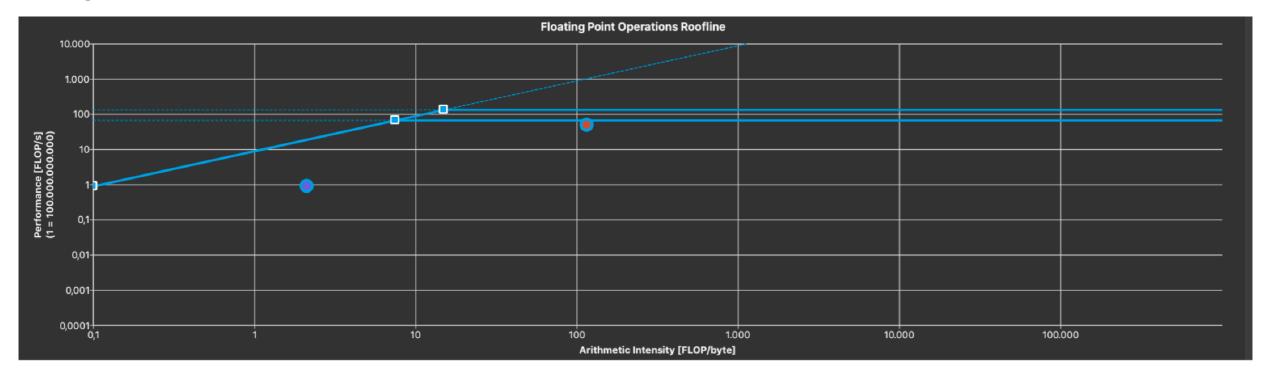


- Here two metric
 - ➡ How much the code compute (here 10%)
 - ➡ How much memory you use (here 90%)
- This indicates what limit your computation
 - ➡ Here we are Memory bound

What is the limitation of your kernel?

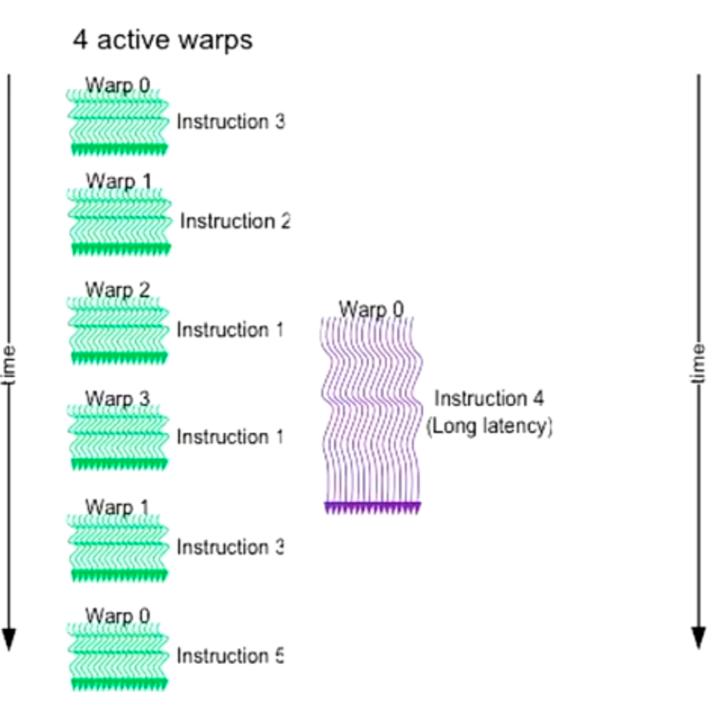


- Ideal case: compute AND memory bound
- If you are latency bound you need to allow more parallelism

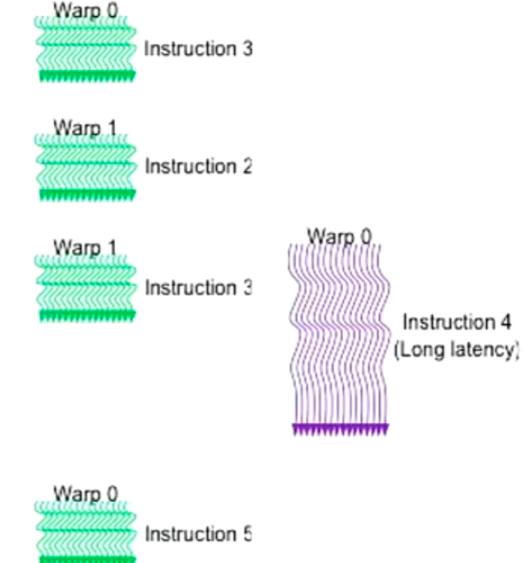


Effect of occupancy

• Hide latency with other wrap

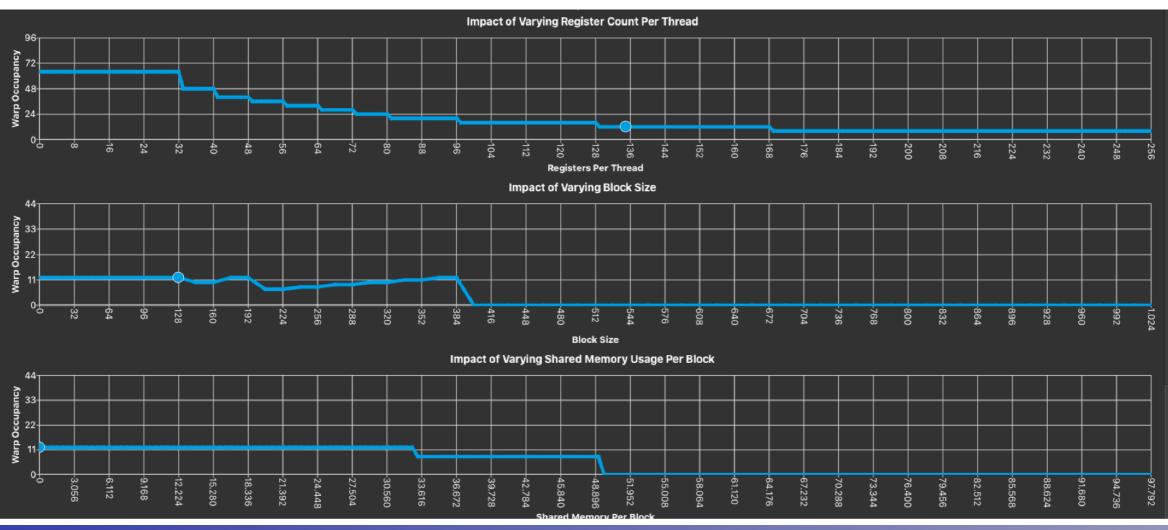


2 active warps



Occupancy

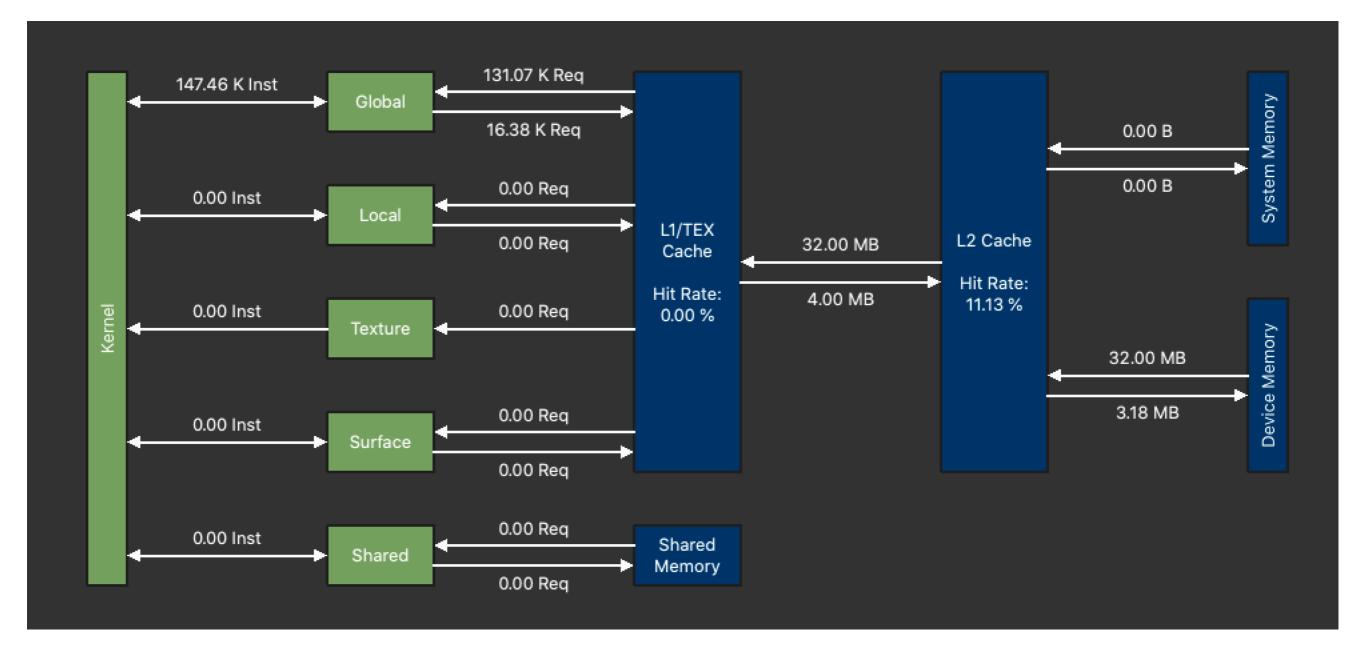
- Occupancy is limited
 - Each SM has limited ressources
 - Maximum number of wrap (64)
 - Maximum number of block (32)
 - Register usage (256Kb)
 - Shared memory usage (64Kb)



CECI training: Cuda

Checking memory

• You should also check where your memory bottleneck are



Nice series of tutorial:

- <u>https://developer.nvidia.com/blog/even-easier-introduction-cuda/</u>
 - At the bottom of the page you have a list of quite progressive tutorial
- Nice video on Cuda 5: https://www.youtube.com/watch? v=irvhW7oSNeQ&list=PLGvfHSgImk4aAt3R3XKvUMIv_RFOzSnWz&index=2
- Nice presentation: https://cac-staff.github.io/ summer-school-2018/files/ cuda_day1_summer_school_2018.pdf

GPU Features	NVIDIA Tesla P100	NVIDIA Tesla V100	NVIDIA A100
GPU Codename	GP100	GV100	GA100
GPU Architecture	NVIDIA Pascal	NVIDIA Volta	NVIDIA Ampere
Compute Capability	6.0	7.0	8.0
Threads / Warp	32	32	32
Max Warps / SM	64	64	64
Max Threads / SM	2048	2048	2048
Max Thread Blocks / SM	32	32	32
Max 32-bit Registers / SM	65536	65536	65536
Max Registers / Block	65536	65536	65536
Max Registers / Thread	255	255	255
Max Thread Block Size	1024	1024	1024
FP32 Cores / SM	64	64	64
Ratio of SM Registers to FP32 Cores	1024	1024	1024
Shared Memory Size / SM	64 KB	Configurable up to 96 KB	Configurable up to 164 KB

Conclusion

- GPU is a high throughput
 - High latency
- Various level of parralelism
 - thread/wrap/block
- Various type of memory
 - register/shared memory/global memory
- Optimization for the hardware is key
 - Coalesced memory ->Array of structure
 - Shared memory