ESCAPE 824064

Laboratoire d'Annecy de Physique des Particules

GPU Architecture

Pierre Aubert

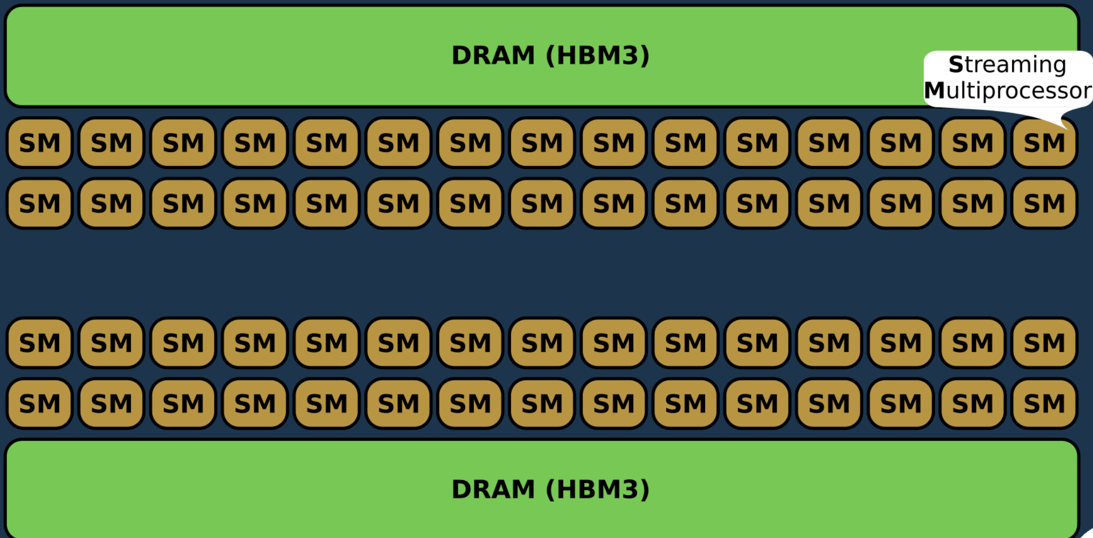
GPU Architecture

GPU Architecture

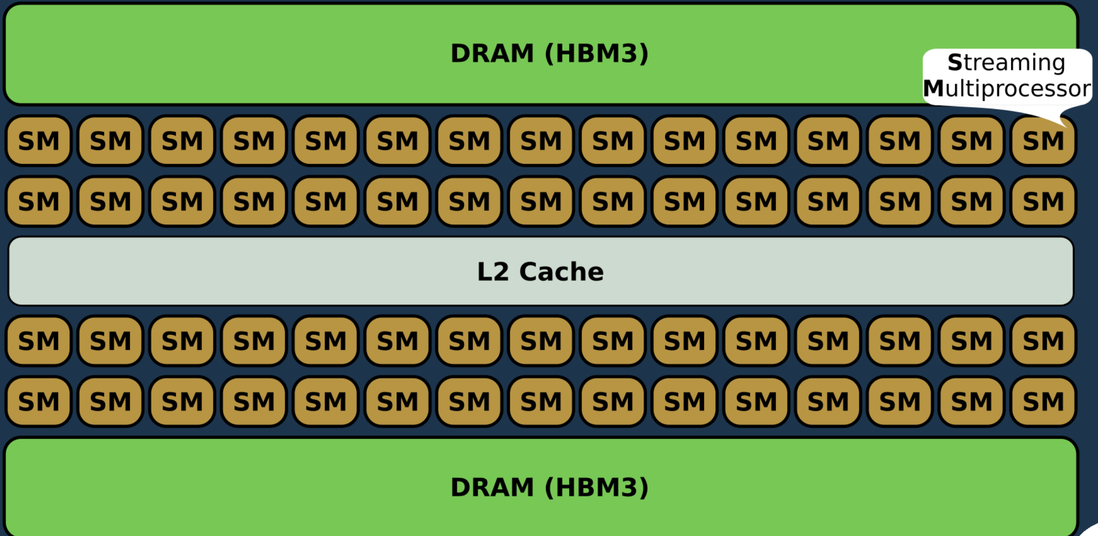
DRAM (HBM3)

DRAM (HBM3)

GPU Architecture



GPU Architecture

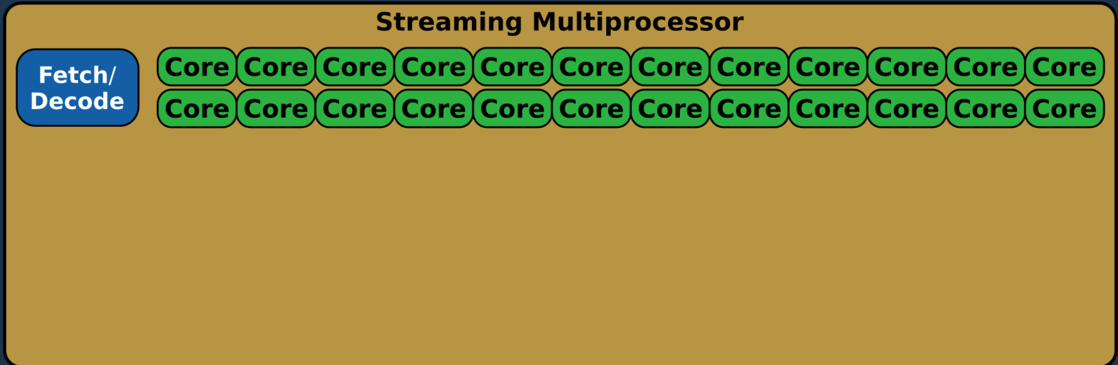


Streaming Multiprocessor

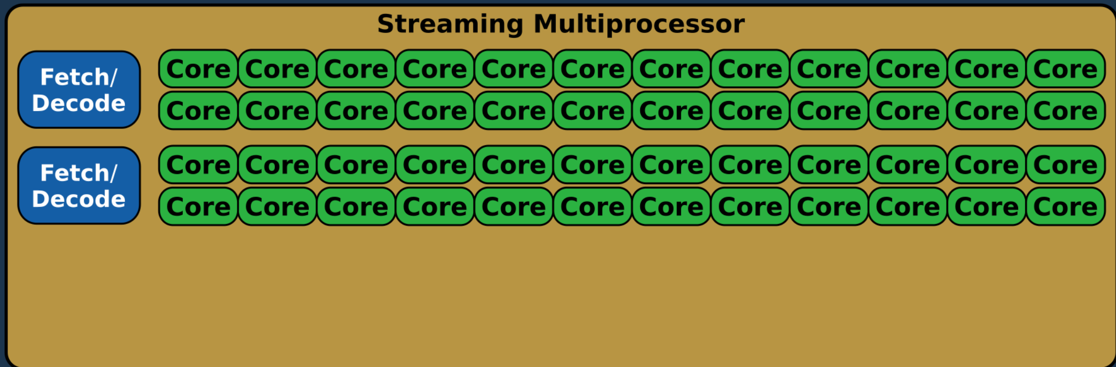
Streaming Multiprocessor

Streaming Multiprocessor

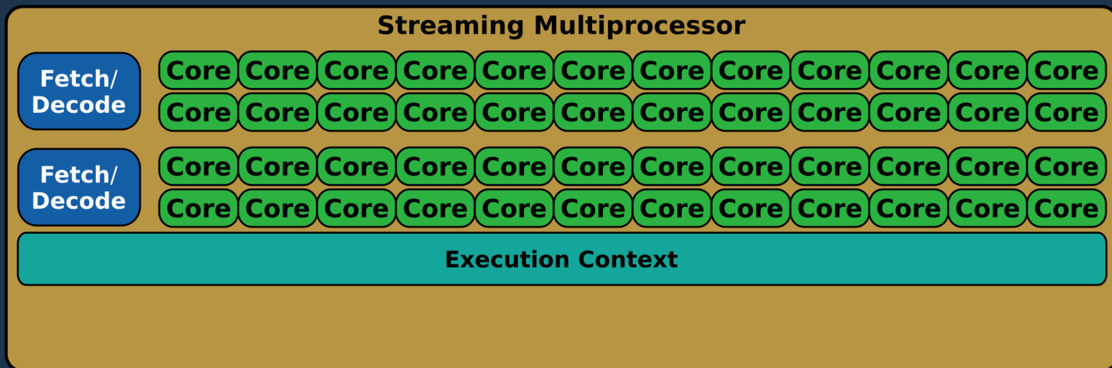
Streaming Multiprocessor



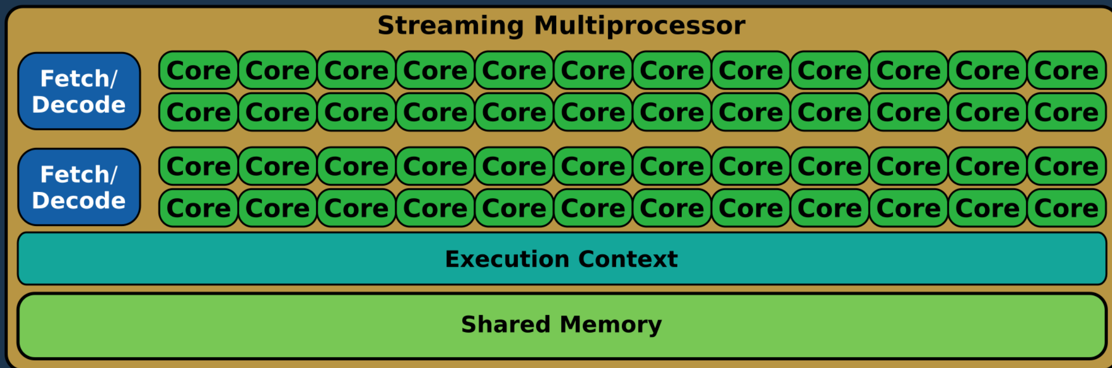
Streaming Multiprocessor



Streaming Multiprocessor



Streaming Multiprocessor



Streaming Multiprocessor H100

L1 Instruction Cache

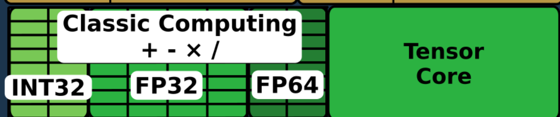
Tensor Memory Accelerator

L1 Data Cache / Shared Memory

Tex | Tex | Tex | Tex

Streaming Multiprocessor H100

L1 Instruction Cache



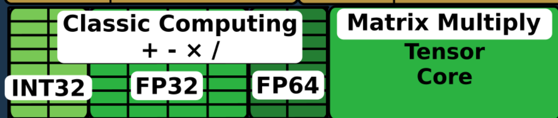
Tensor Memory Accelerator

L1 Data Cache / Shared Memory

Tex	Tex	Tex	Tex

Streaming Multiprocessor H100

L1 Instruction Cache



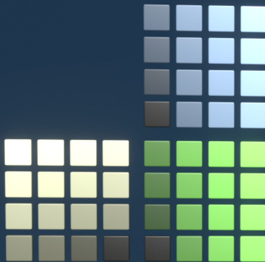
Tensor Memory Accelerator

L1 Data Cache / Shared Memory

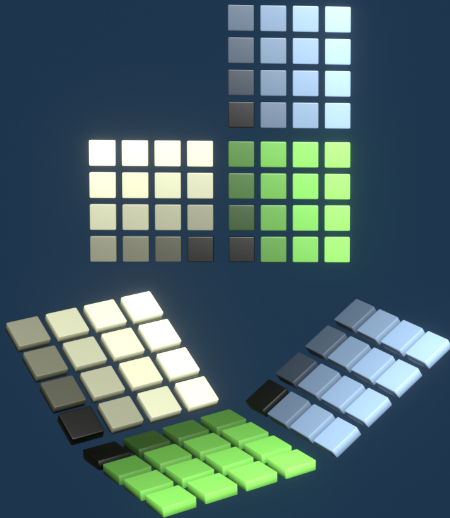
Tex | Tex | Tex | Tex

Tensor Core

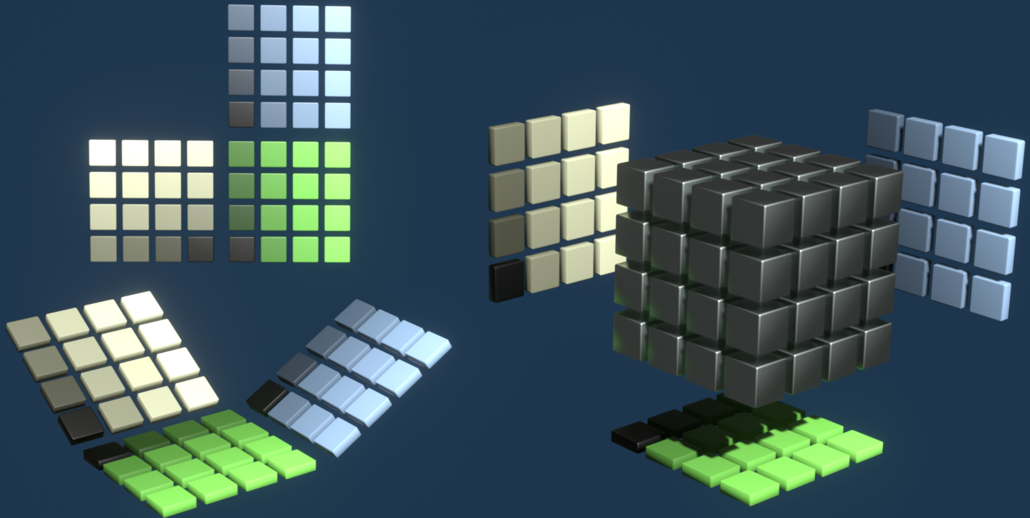
Tensor Core



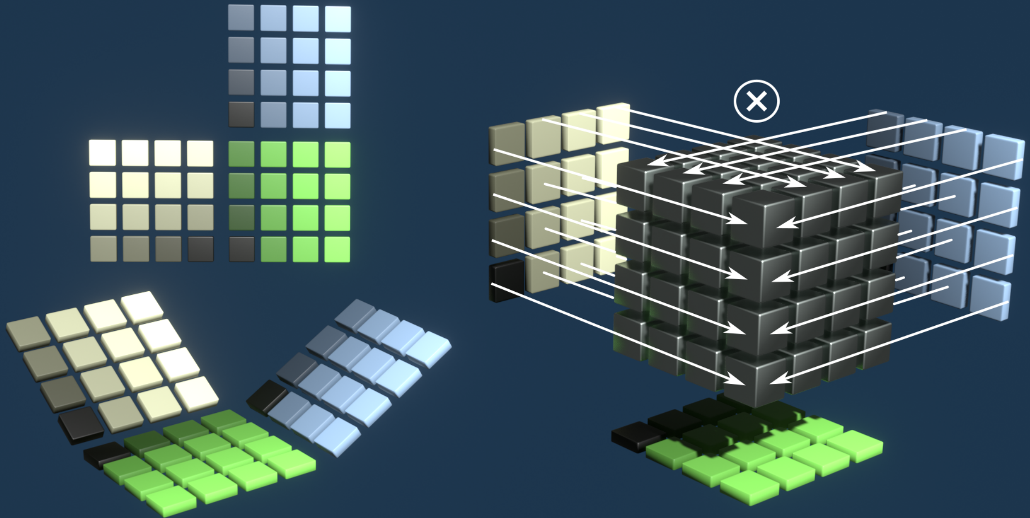
Tensor Core



Tensor Core

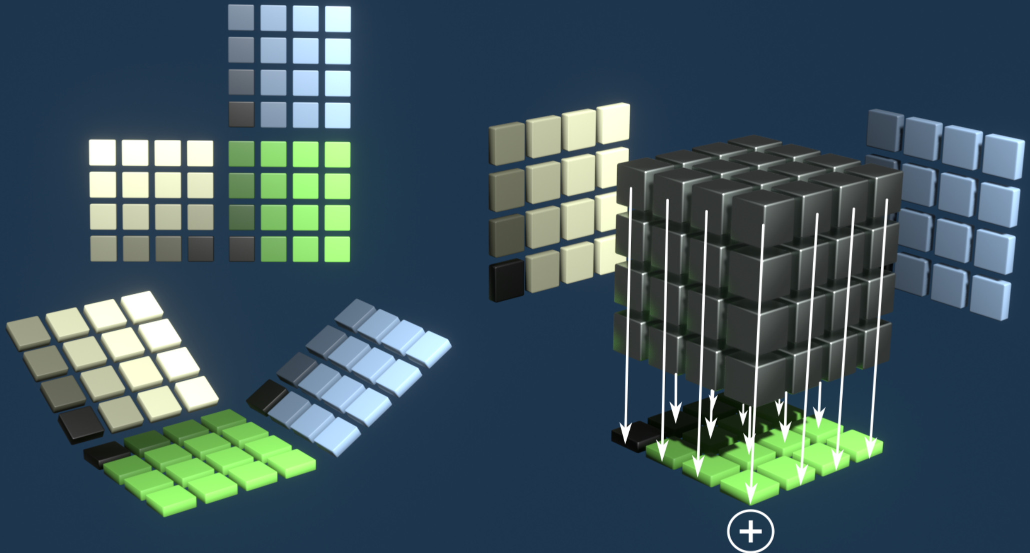


Tensor Core

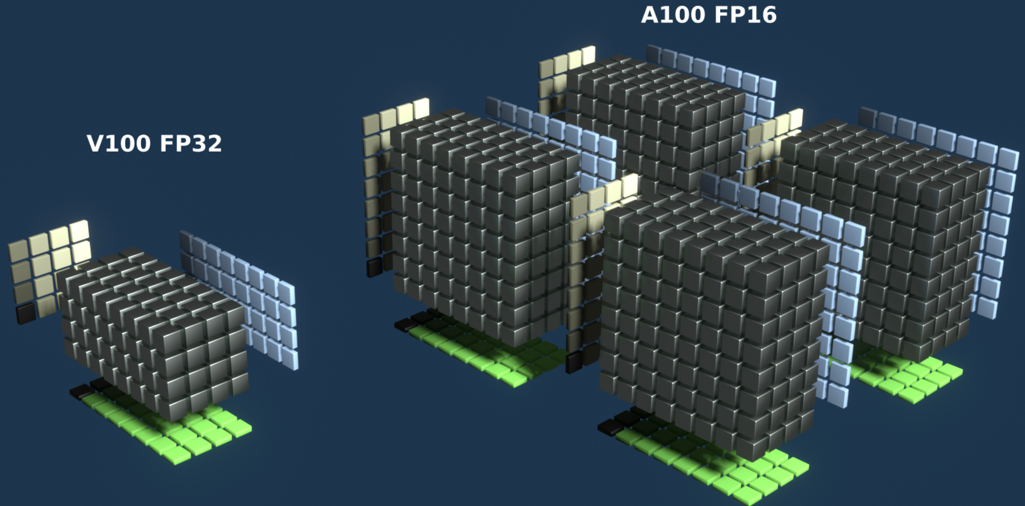


LAPP
Laboratoire d'Annecy de Physique des Particules

Tensor Core

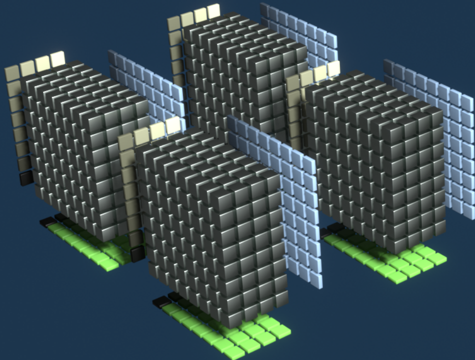


Tensor Core

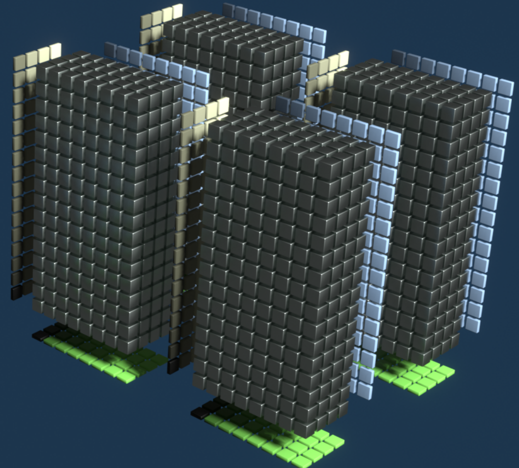


Tensor Core

A100 FP16

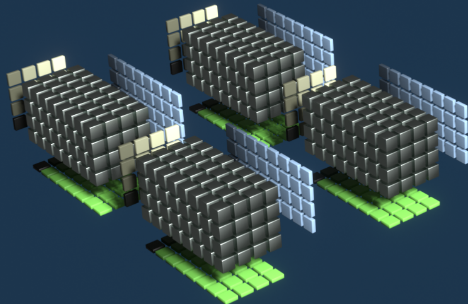


H100 FP16

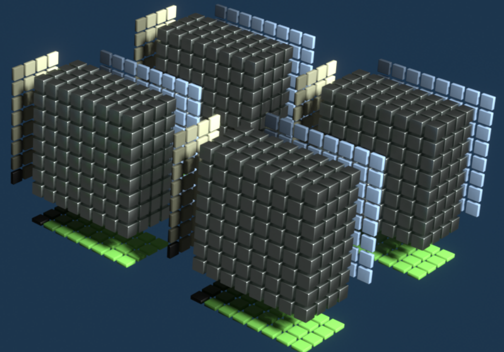


Tensor Core

A100 TF32

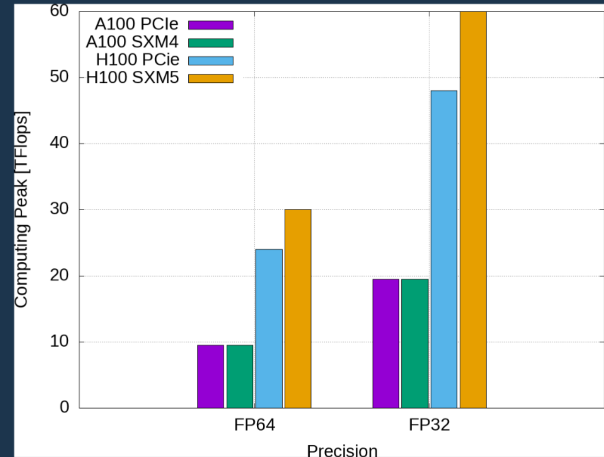


H100 TF32

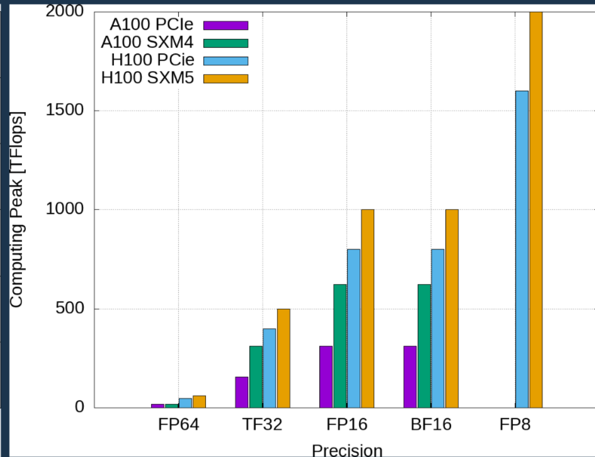


Why using Tensor Cores ?

CUDA Cores

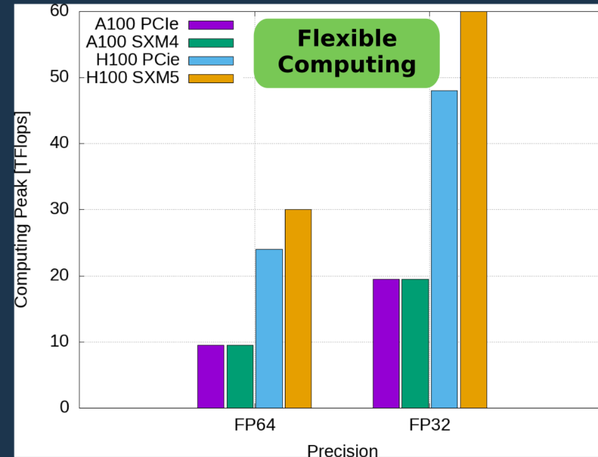


TensorCores

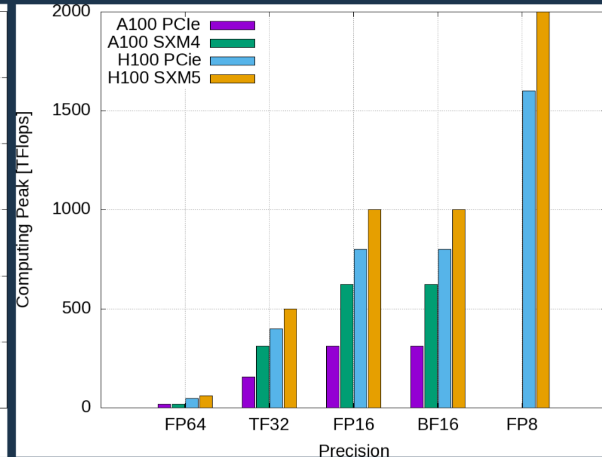


Why using Tensor Cores ?

CUDA Cores

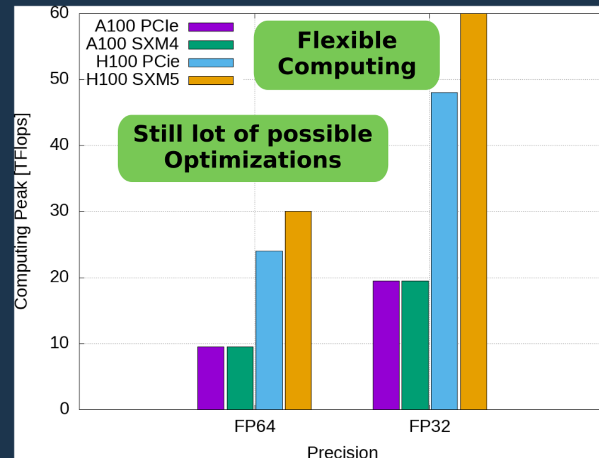


TensorCores

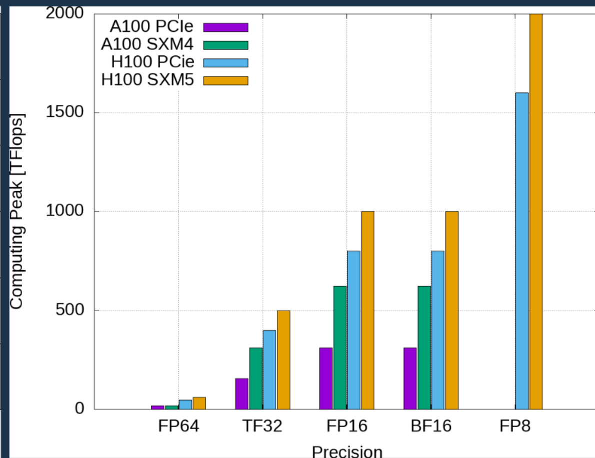


Why using Tensor Cores ?

CUDA Cores

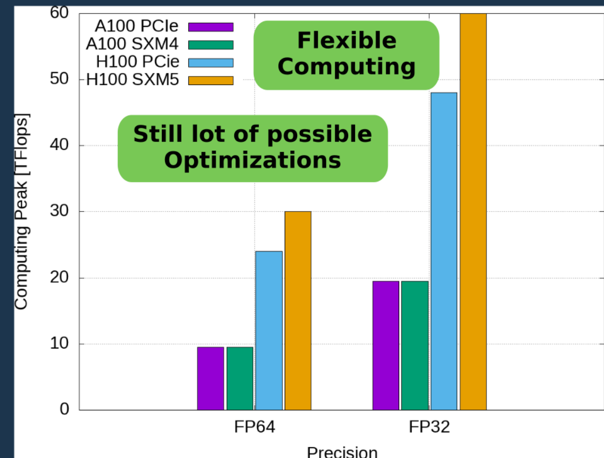


TensorCores

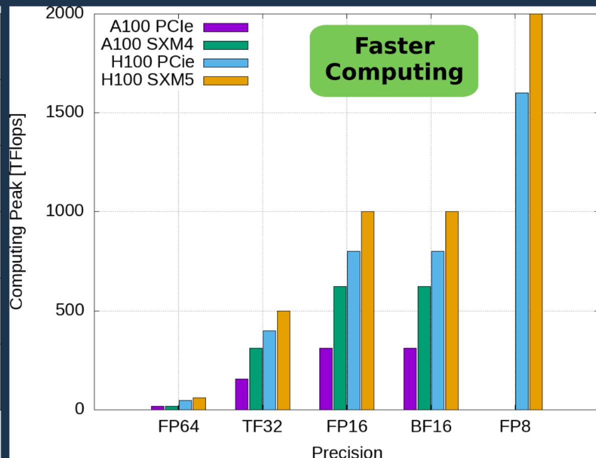


Why using Tensor Cores ?

CUDA Cores

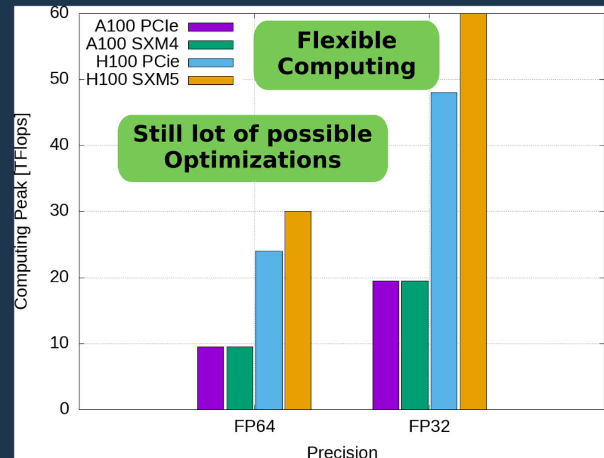


TensorCores

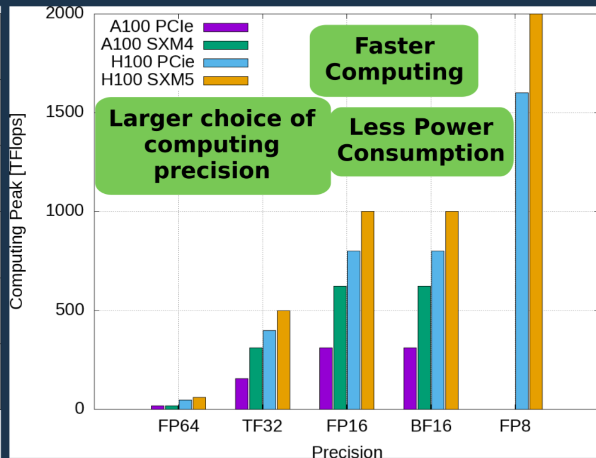


Why using Tensor Cores ?

CUDA Cores

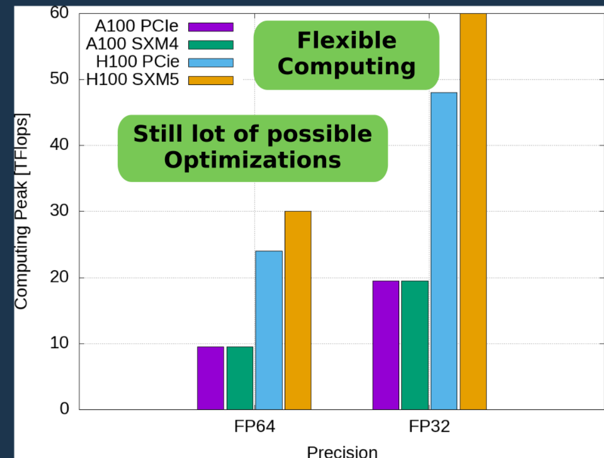


TensorCores

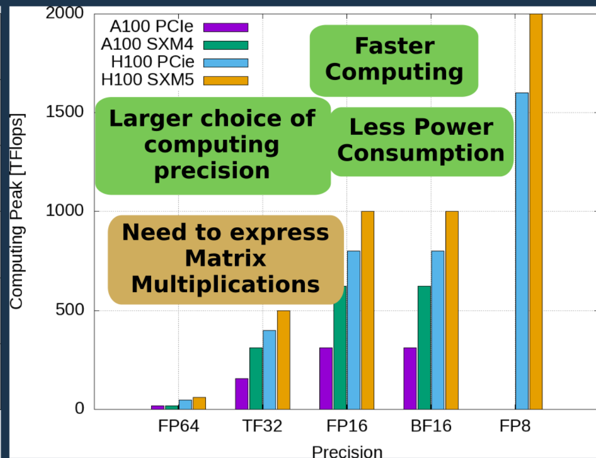


Why using Tensor Cores ?

CUDA Cores

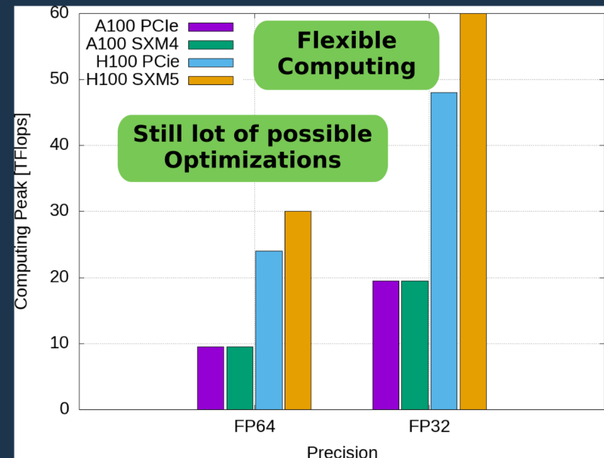


TensorCores

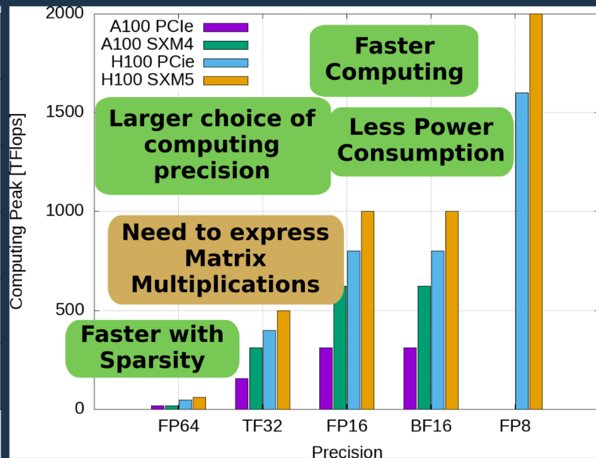


Why using Tensor Cores ?

CUDA Cores



TensorCores



The CUDA ecosystem

The CUDA ecosystem

Kernel Development

CUDA | cu++ | CUTLASS

The CUDA ecosystem

Computing Libraries

BLAS | LAPACK | PBLAS | SCALAPACK

TENSOR | SPARSE | RAND | FFT

Kernel Development

CUDA | cu++ | CUTLASS

The CUDA ecosystem

Computing Libraries

BLAS | LAPACK | PBLAS | SCALAPACK
TENSOR | SPARSE | RAND | FFT

Kernel Development

CUDA | cu++ | CUTLASS

The CUDA ecosystem

Computing Libraries

BLAS	LAPACK	PBLAS	SCALAPACK
TENSOR	SPARSE	RAND	FFT

Computing + I/O

RAPIDS

cunumeric	legate	cuDF

Kernel Development

CUDA	cu++	CUTLASS

The CUDA ecosystem

Computing Libraries

BLAS | LAPACK | PBLAS | SCALAPACK
TENSOR | SPARSE | RAND | FFT

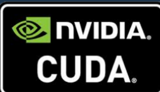
Computing + I/O

RAPIDS

cunumeric | legate | cuDF

Kernel Development

CUDA | cu++ | CUTLASS



I/O and communication

MagnumIO

NVShmem | GPUDirect | NCCL

The CUDA ecosystem

Computing Libraries

BLAS | LAPACK | PBLAS | SCALAPACK

TENSOR | SPARSE | RAND | FFT

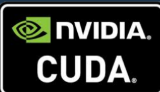
Computing + I/O

RAPIDS

cunumeric | legate | cuDF

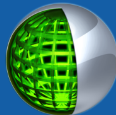
Kernel Development

CUDA | cu++ | CUTLASS



Profiler / Debugger

NSight



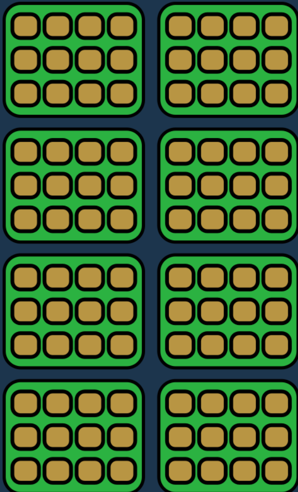
I/O and communication

MagnumIO

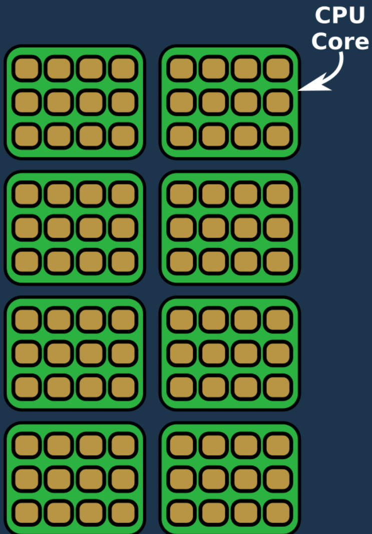
NVShmem | GPUDirect | NCCL

Threads ?

Threads ?

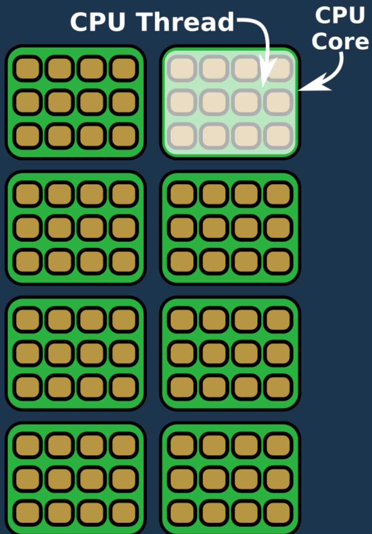


Threads ?

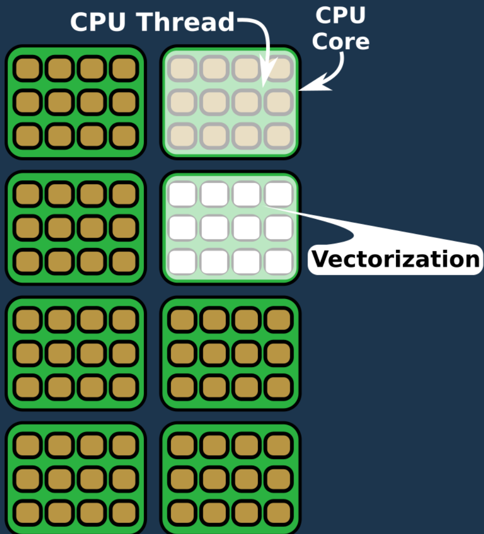


Laboratoire d'Annecy de Physique des Particules

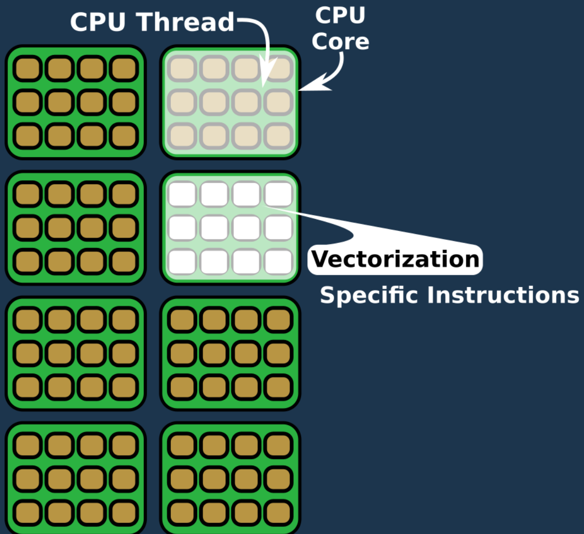
Threads ?



Threads ?

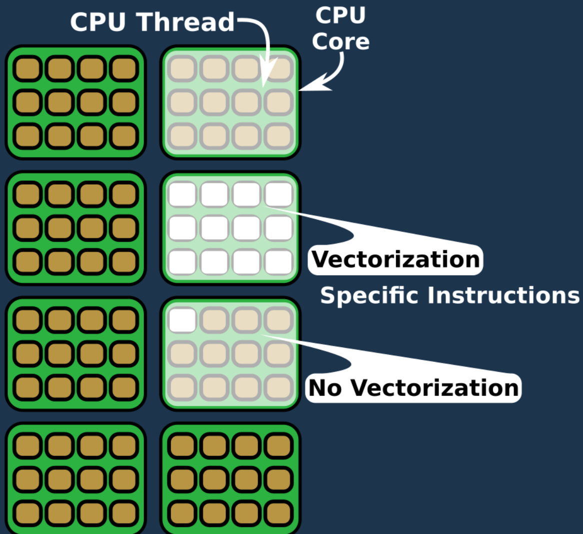


Threads ?



Specific Instructions

Threads ?



Threads ?

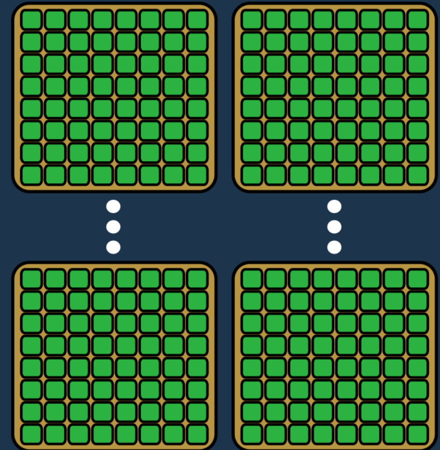
CPU Thread

CPU Core

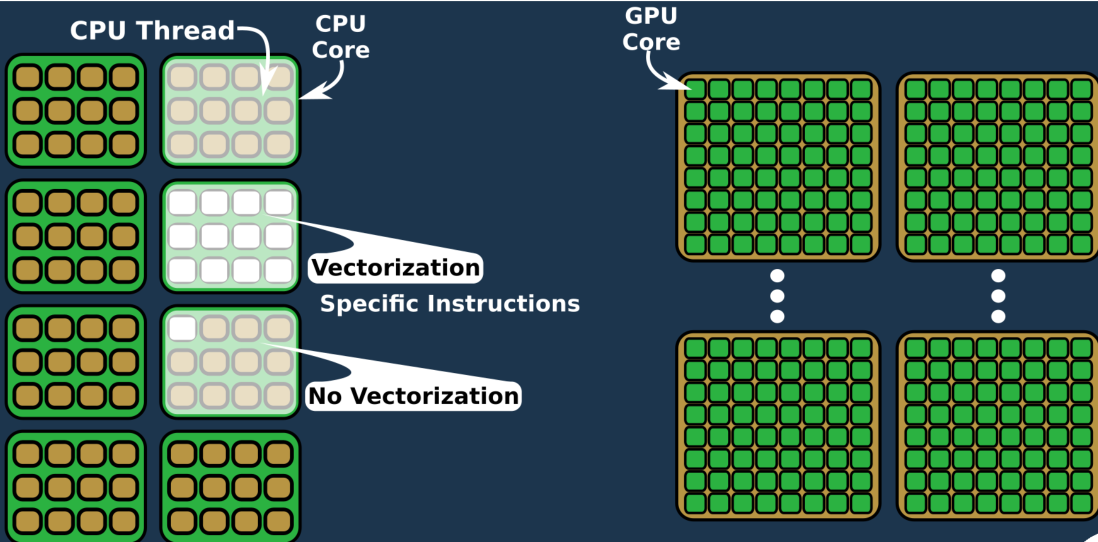
Vectorization

Specific Instructions

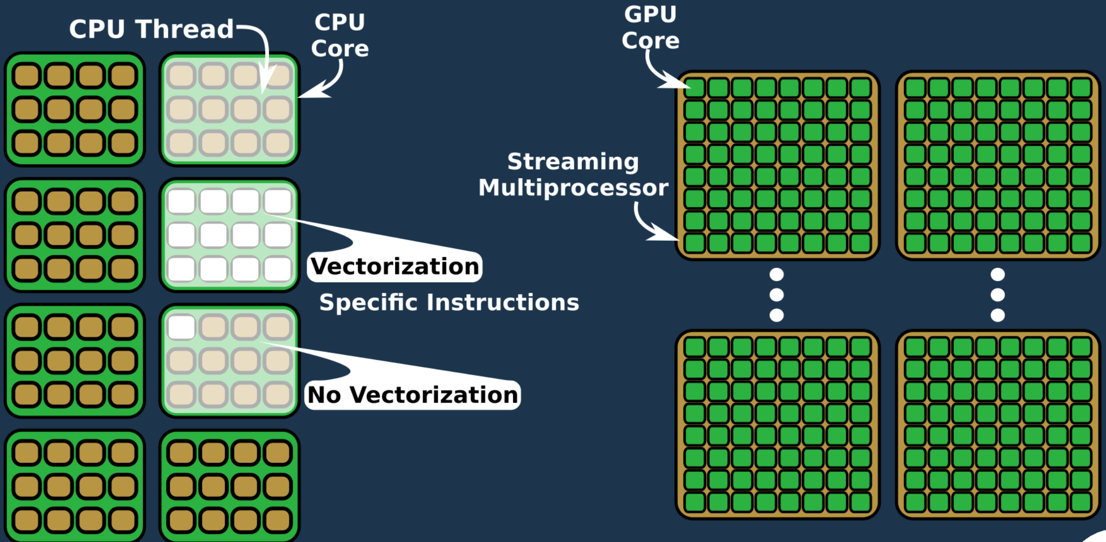
No Vectorization



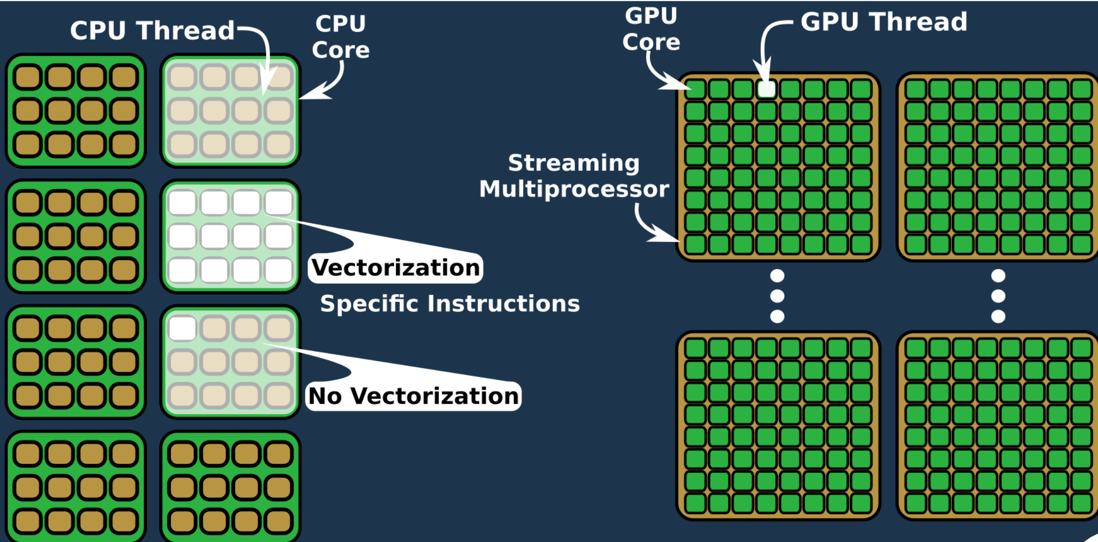
Threads ?



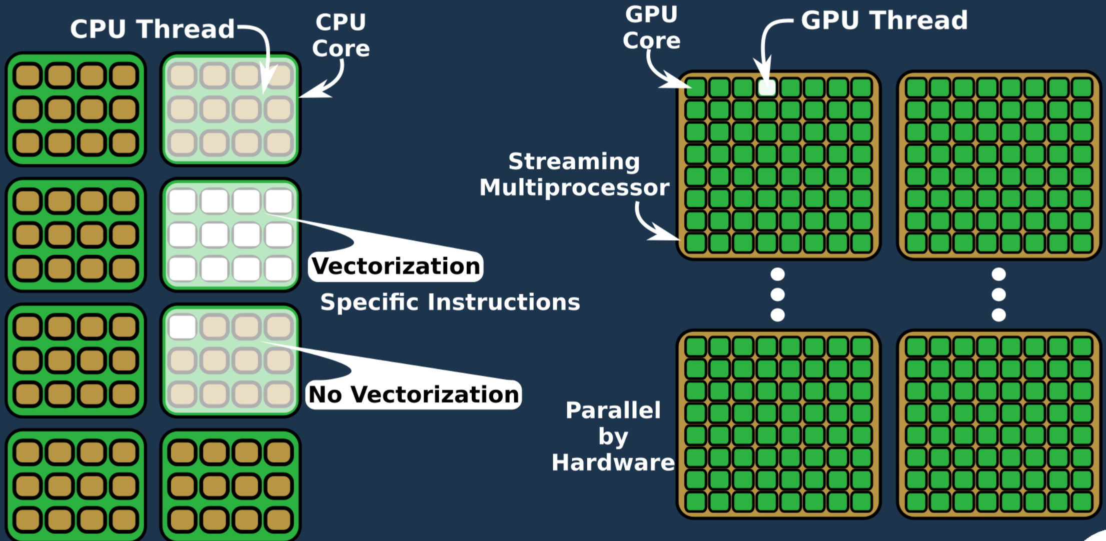
Threads ?



Threads ?



Threads ?



Threads Hierarchy

Threads Hierarchy

Thread

Threads Hierarchy

Thread Core

Threads Hierarchy

Thread Core Executes Computing Kernel

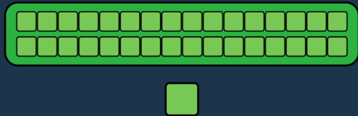
LAPP
Laboratoire d'Annecy de Physique des Particules

Threads Hierarchy

Warp

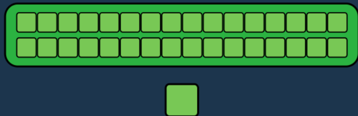
Thread Core Executes Computing Kernel

Threads Hierarchy



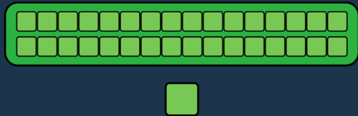
Warp	Hardware Scheduler	
Thread	Core	Executes Computing Kernel

Threads Hierarchy



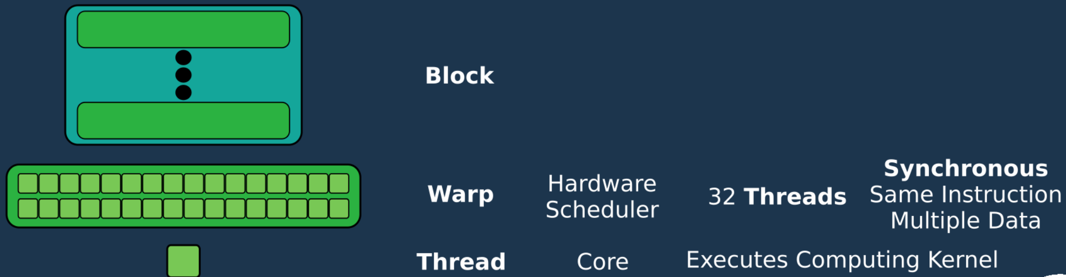
Warp	Hardware Scheduler	32 **Threads**
Thread	Core	Executes Computing Kernel

Threads Hierarchy

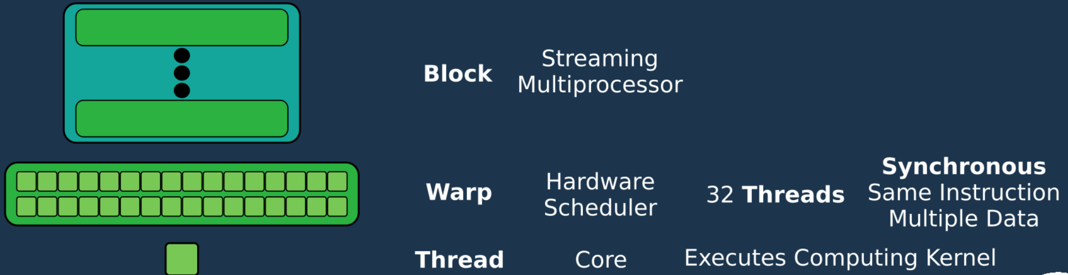


Warp	Hardware Scheduler	32 **Threads**	**Synchronous** Same Instruction Multiple Data
Thread	Core	Executes Computing Kernel	

Threads Hierarchy



Threads Hierarchy

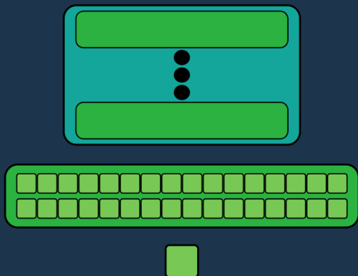


LAPP
Laboratoire d'Annecy de Physique des Particules

Threads Hierarchy

Block	Streaming Multiprocessor	2 - 4 **Warps** 64 - 128 **Cores**	
Warp	Hardware Scheduler	32 **Threads**	**Synchronous** Same Instruction Multiple Data
Thread	Core	Executes Computing Kernel	

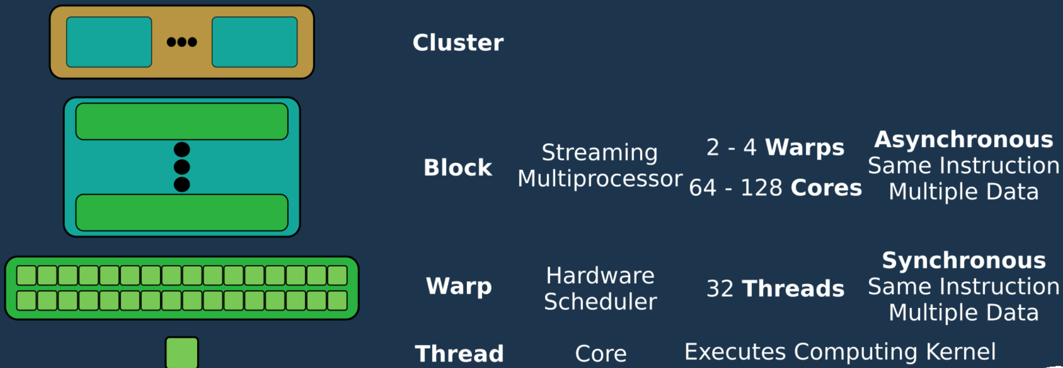
Threads Hierarchy



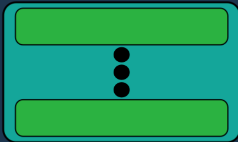
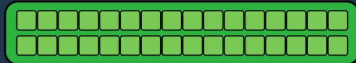
Block	Streaming Multiprocessor	2 - 4 **Warps** 64 - 128 **Cores**	**Asynchronous** Same Instruction Multiple Data
Warp	Hardware Scheduler	32 **Threads**	**Synchronous** Same Instruction Multiple Data
Thread	Core	Executes Computing Kernel	

Laboratoire d'Annecy de Physique des Particules

Threads Hierarchy



Threads Hierarchy

	Cluster	Blocks Communication		
	**Block**	Streaming Multiprocessor	2 - 4 **Warps** 64 - 128 **Cores**	**Asynchronous** Same Instruction Multiple Data
	**Warp**	Hardware Scheduler	32 **Threads**	**Synchronous** Same Instruction Multiple Data
	Thread	Core	Executes Computing Kernel	

Threads Hierarchy

Cluster	Blocks Communication	Since : - CUDA **12** - **Hopper**	
Block	Streaming Multiprocessor	2 - 4 **Warps** 64 - 128 **Cores**	**Asynchronous** Same Instruction Multiple Data
Warp	Hardware Scheduler	32 **Threads**	**Synchronous** Same Instruction Multiple Data
Thread	Core	Executes Computing Kernel	

Threads Hierarchy

	Cluster	Blocks Communication	Since : - CUDA **12** - **Hopper**	**Hopper** : 132 SMs **Ampere** : 108 SMs
	Block	Streaming Multiprocessor	2 - 4 **Warps** 64 - 128 **Cores**	**Asynchronous** Same Instruction Multiple Data
	Warp	Hardware Scheduler	32 **Threads**	**Synchronous** Same Instruction Multiple Data
	Thread	Core	Executes Computing Kernel	

Threads Hierarchy

Grid

Cluster — Blocks Communication — Since : - CUDA **12** - **Hopper** — **Hopper** : 132 SMs **Ampere** : 108 SMs

Block — Streaming Multiprocessor — 2 - 4 **Warps** 64 - 128 **Cores** — **Asynchronous** Same Instruction Multiple Data

Warp — Hardware Scheduler — 32 **Threads** — **Synchronous** Same Instruction Multiple Data

Thread — Core — Executes Computing Kernel

Threads Hierarchy

Grid — Full **GPU**

Cluster — Blocks Communication — Since :
- CUDA **12**
- **Hopper**

Hopper : 132 SMs
Ampere : 108 SMs

Block — Streaming Multiprocessor — 2 - 4 **Warps**, 64 - 128 **Cores** — **Asynchronous** Same Instruction Multiple Data

Warp — Hardware Scheduler — 32 **Threads** — **Synchronous** Same Instruction Multiple Data

Thread — Core — Executes Computing Kernel

Threads Hierarchy

Grid	Full **GPU**	**Hopper** : 8448 Cores **Ampere** : 6912 Cores	
Cluster	Blocks Communication	Since : - CUDA **12** - **Hopper**	**Hopper** : 132 SMs **Ampere** : 108 SMs
Block	Streaming Multiprocessor	2 - 4 **Warps** 64 - 128 **Cores**	**Asynchronous** Same Instruction Multiple Data
Warp	Hardware Scheduler	32 **Threads**	**Synchronous** Same Instruction Multiple Data
Thread	Core	Executes Computing Kernel	

Threads Execution

Threads Execution

Threads Execution

Thread

Warp

LAPP
Laboratoire d'Annecy de Physique des Particules

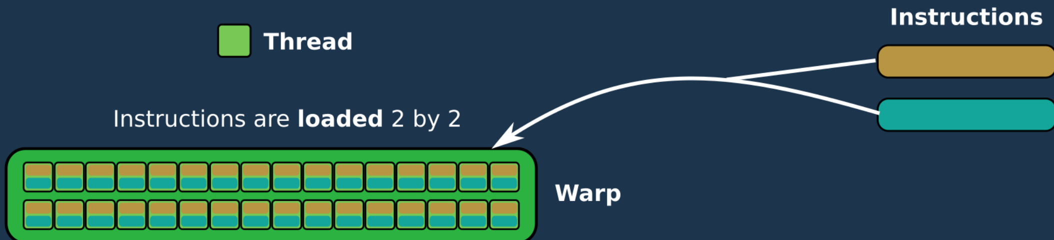
Threads Execution

Thread

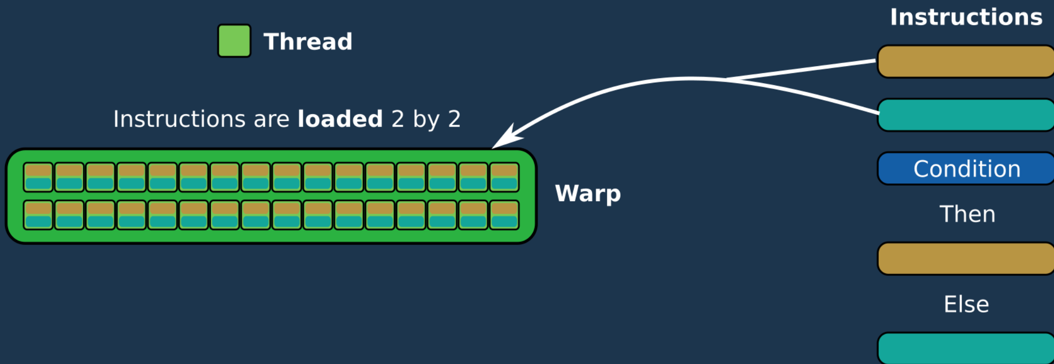
Instructions

Warp

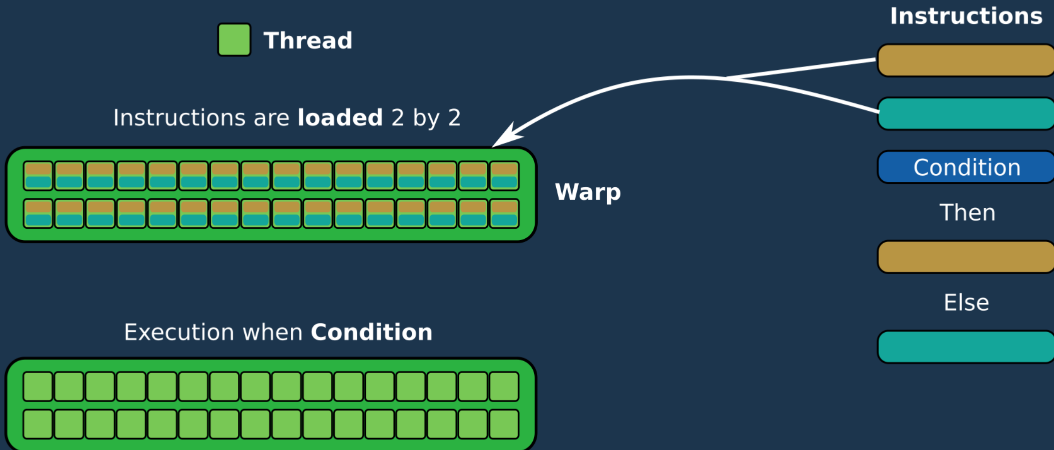
Threads Execution



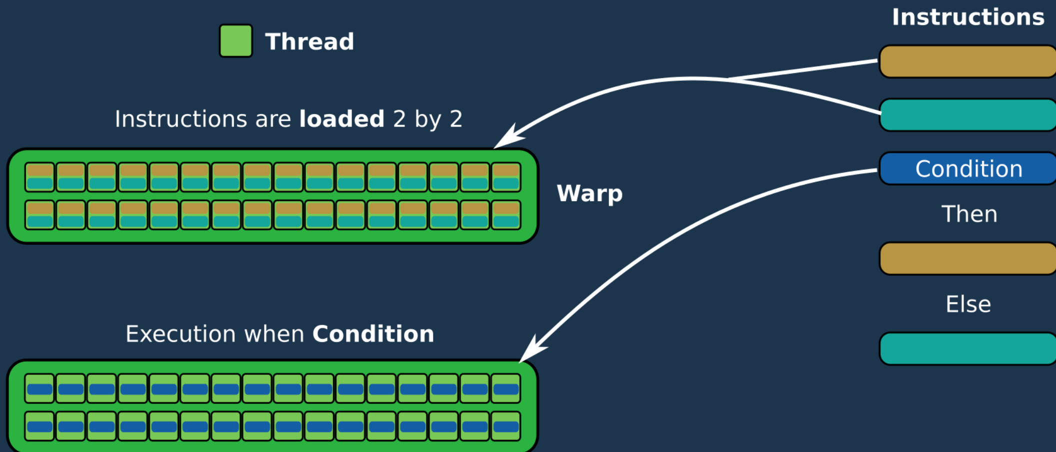
Threads Execution



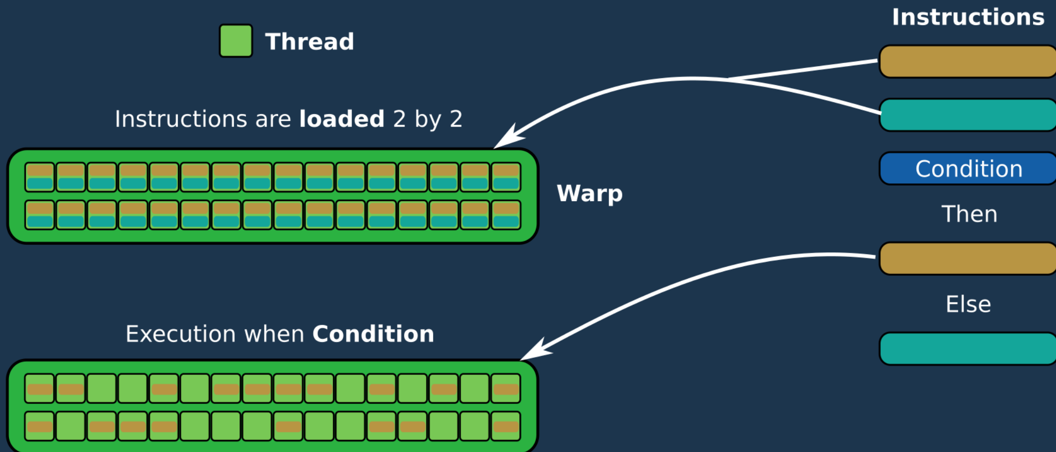
Threads Execution



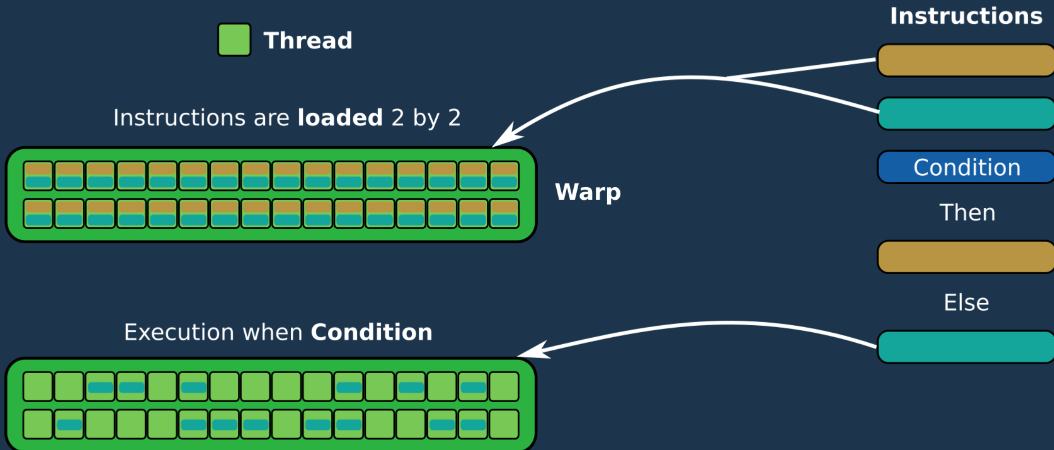
Threads Execution



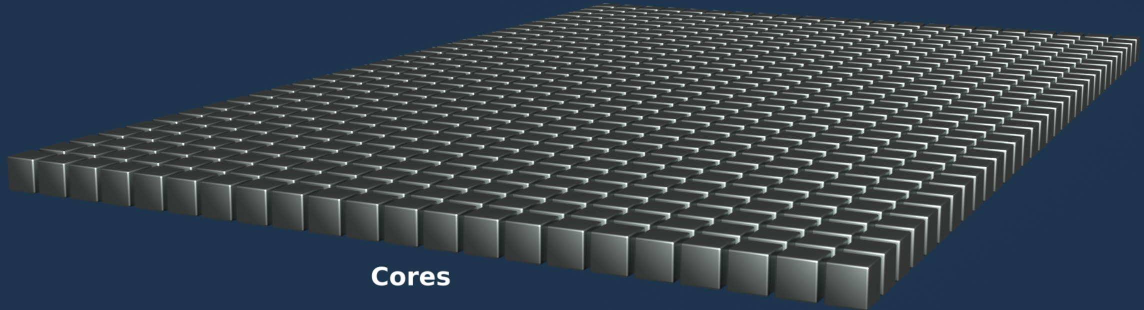
Threads Execution



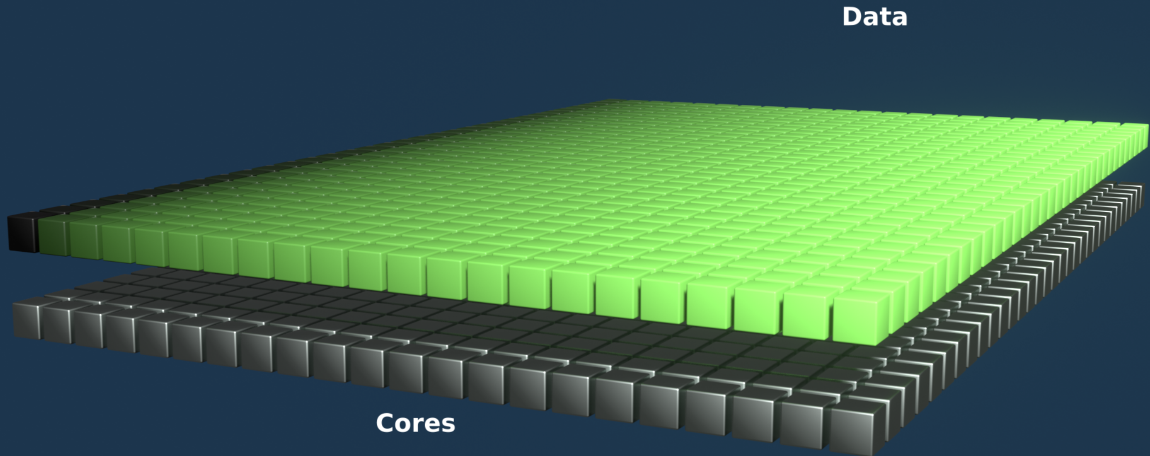
Threads Execution



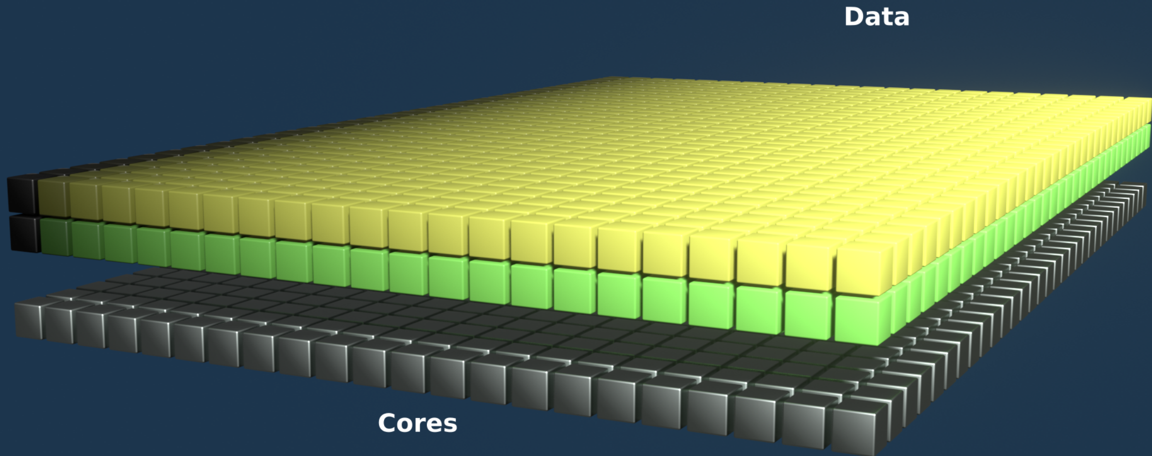
Thread Mapping



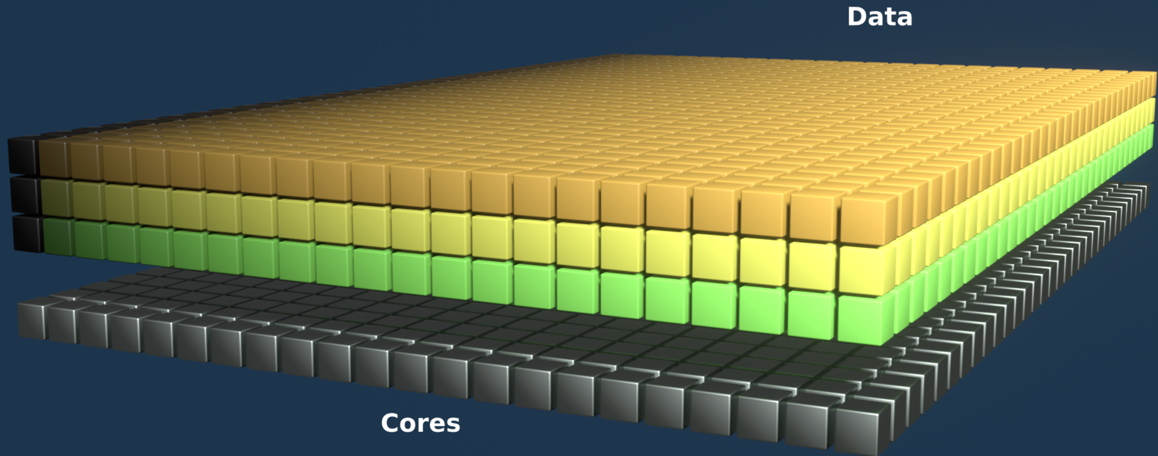
Thread Mapping



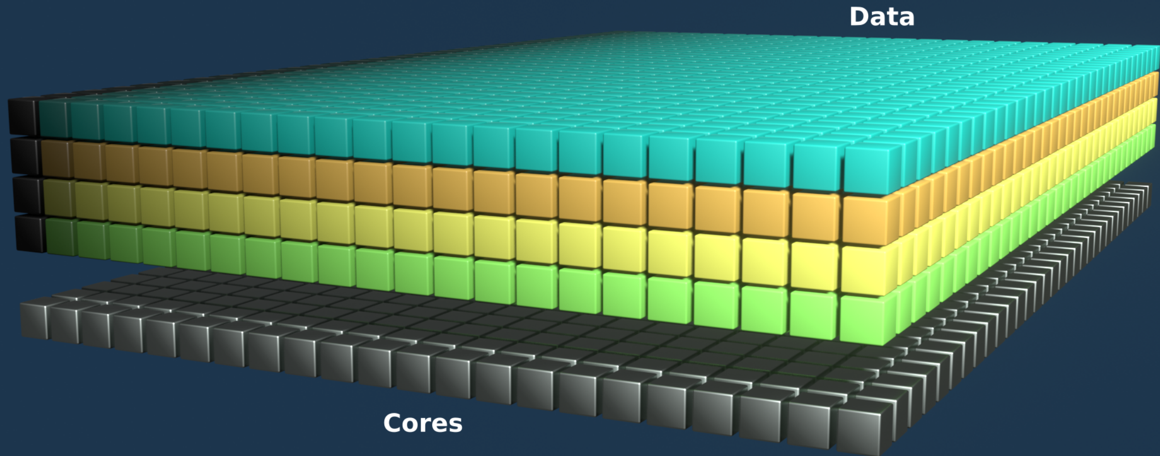
Thread Mapping



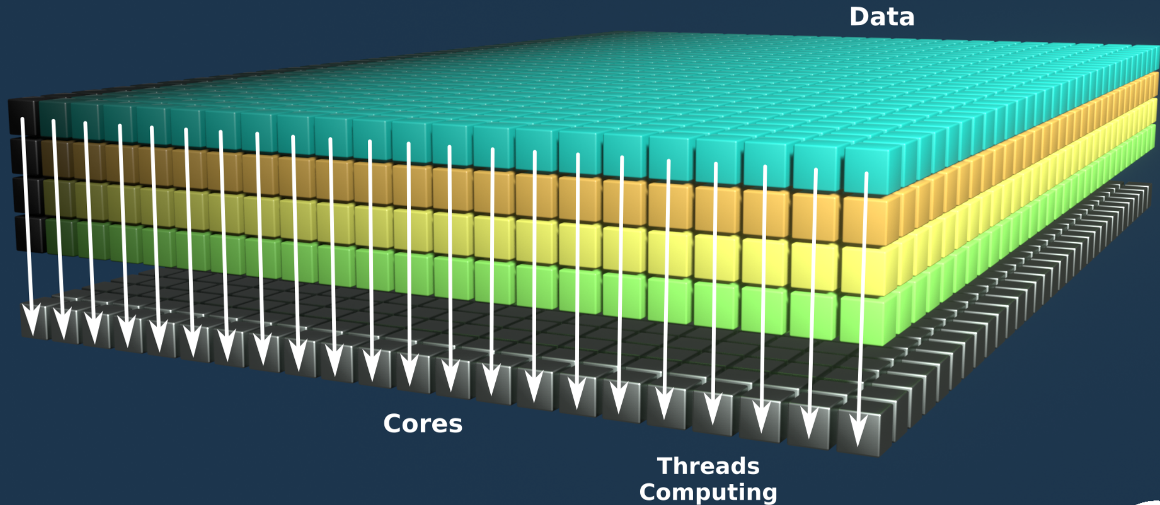
Thread Mapping



Thread Mapping

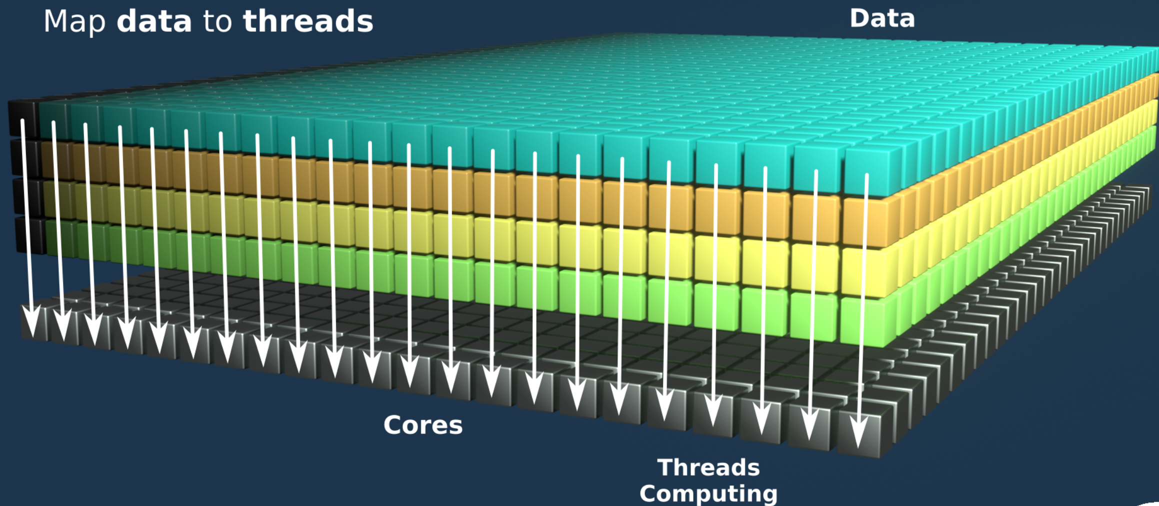


Thread Mapping



Thread Mapping

Map **data** to **threads**



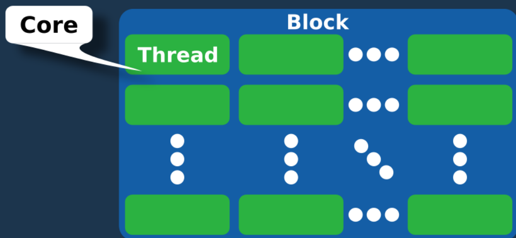
Data / Thread mapping

Data / Thread mapping

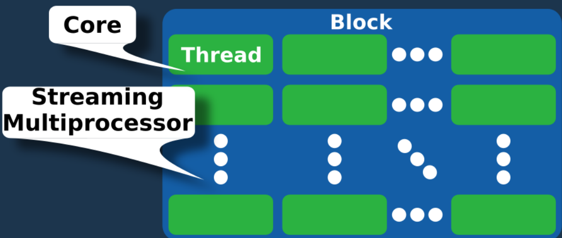
Thread

Data / Thread mapping

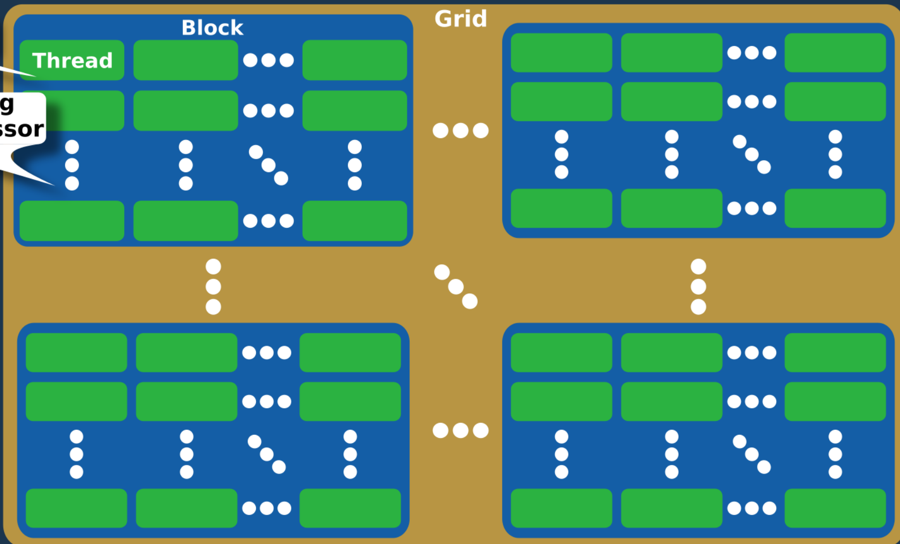
Data / Thread mapping



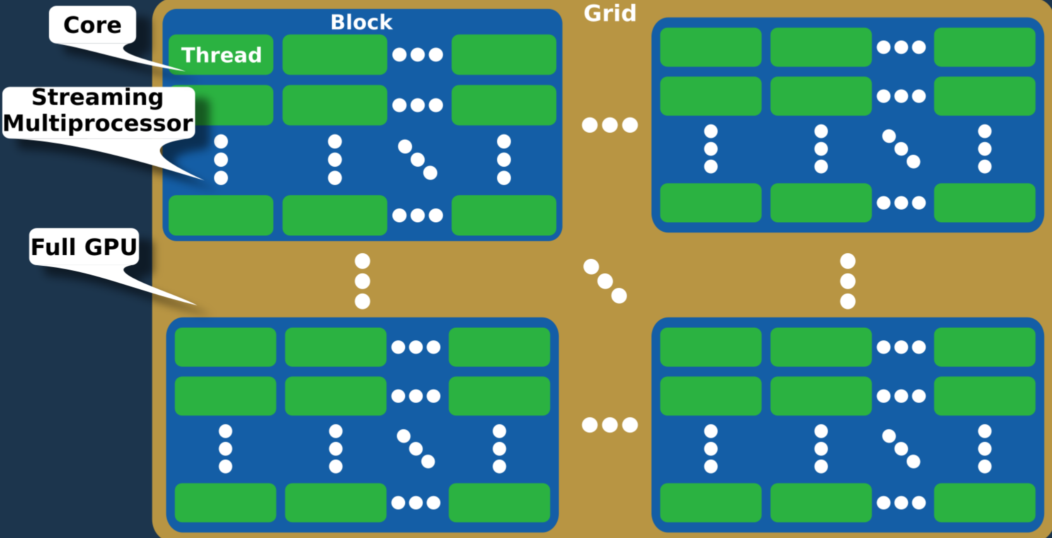
Data / Thread mapping



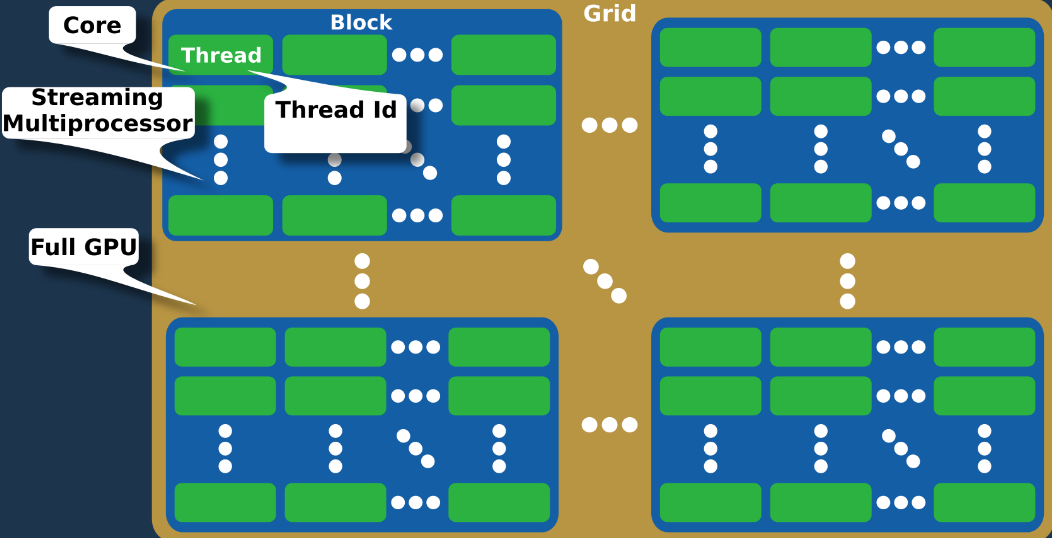
Data / Thread mapping



Data / Thread mapping

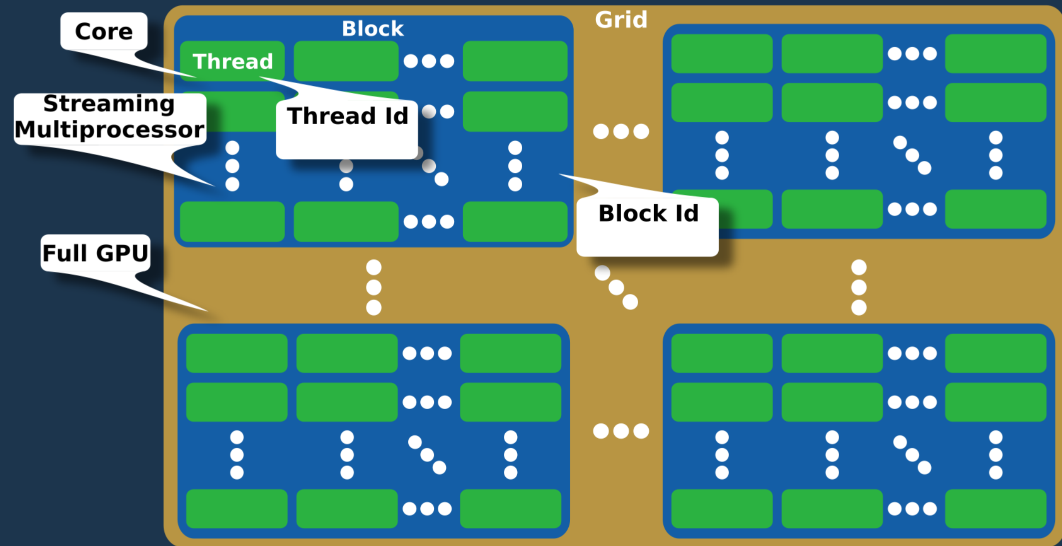


Data / Thread mapping

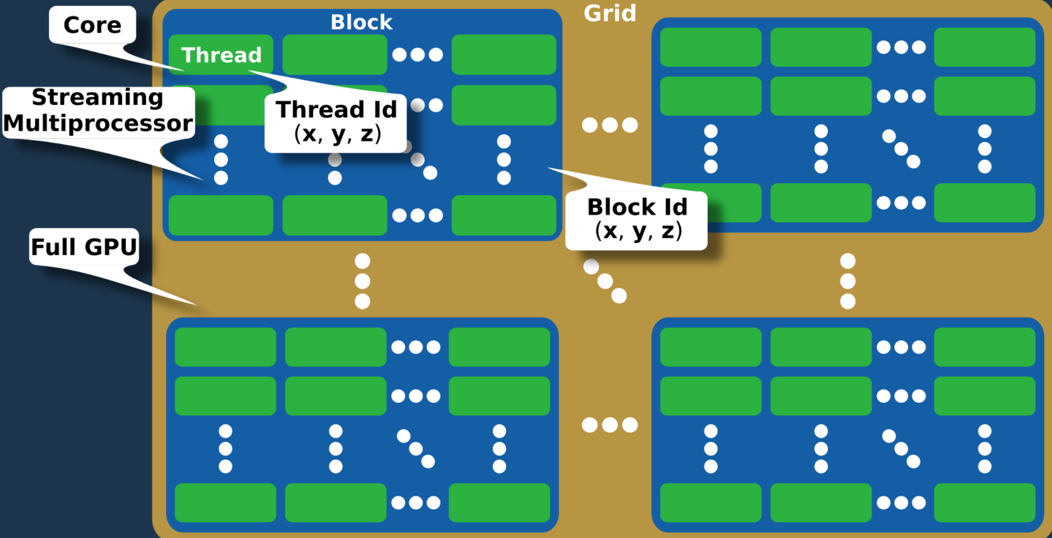


LAPP Laboratoire d'Annecy de Physique des Particules

Data / Thread mapping



Data / Thread mapping



Block Size ?

Block Size ?

Hadamard product on **1000** elements

Block Size ?

Hadamard product on **1000** elements

Scenario #1

10 1d blocks of **size 100**

Block Size ?

Warp of **32 threads**

Hadamard product on **1000** elements

Scenario #1

10 1d blocks of **size 100**

Block Size ?

Warp of **32 threads**

Hadamard product on **1000** elements

Scenario #1

10 1d blocks of **size 100**

Block Size ?

Warp of **32 threads**

Hadamard product on **1000** elements

Use: 78%

Scenario #1

10 1d blocks of **size 100**

Block Size ?

Warp of **32 threads**

Hadamard product on **1000** elements

Use: 78%

Scenario #1

10 1d blocks of **size 100**

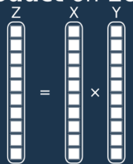
Scenario #2

8 1d blocks of **size 128**

Block Size ?

Warp of **32 threads**

Hadamard product on **1000** elements



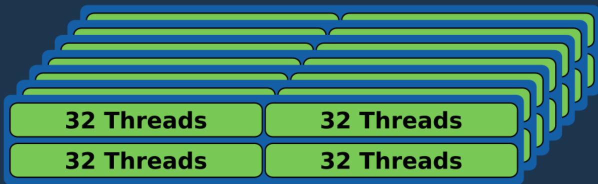
Use: 78%

Scenario #1

10 1d blocks of **size 100**

Scenario #2

8 1d blocks of **size 128**



Block Size ?

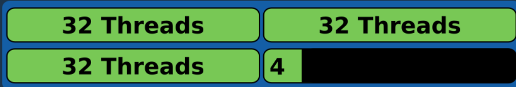
Warp of **32 threads**

Hadamard product on **1000** elements

Use: 78%

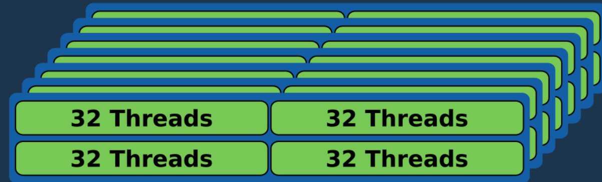
Scenario #1

10 1d blocks of **size 100**



Scenario #2

8 1d blocks of **size 128**



Laboratoire d'Annecy de Physique des Particules

Block Size ?

Warp of **32 threads**

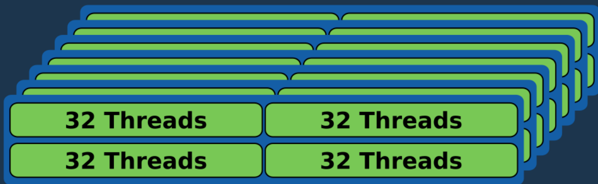
Hadamard product on **1000** elements

Scenario #1 Use: 78%

10 1d blocks of **size 100**

Scenario #2 Use: 98%

8 1d blocks of **size 128**



Warp Read / Write

Laboratoire d'Annecy de Physique des Particules

Warp Read / Write

1 sector : 32 bytes

Cache line size :
- 128 bytes
- 4 sectors

Warp Read / Write

1 sector : 32 bytes

Cache line size :
- 128 bytes
- 4 sectors

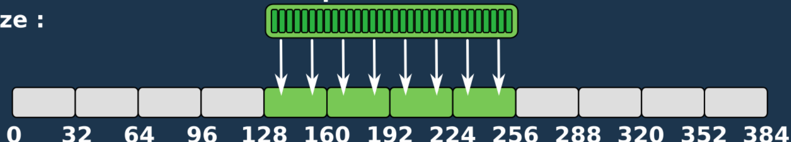
4-bytes elements access -> 4 sectors

Warp of 32 threads

Warp Read / Write

1 sector : 32 bytes

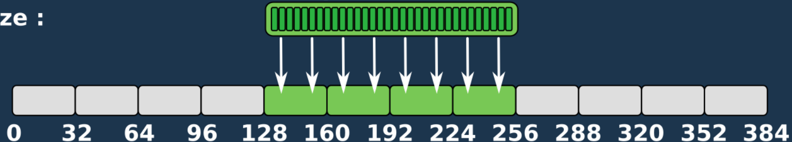
Cache line size :
- 128 bytes
- 4 sectors

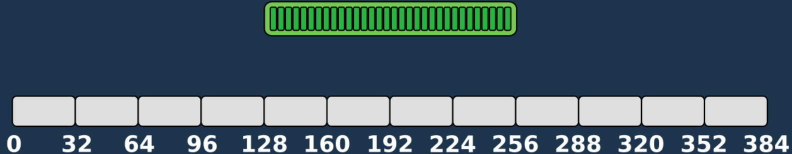


Warp Read / Write

1 sector : 32 bytes

Cache line size :
- 128 bytes
- 4 sectors

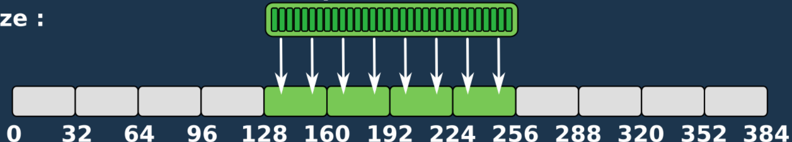


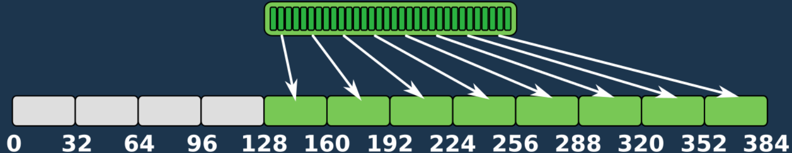


Warp Read / Write

1 sector : 32 bytes

Cache line size :
- 128 bytes
- 4 sectors

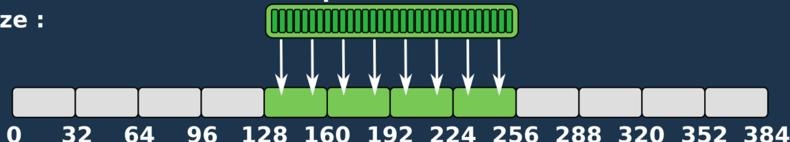




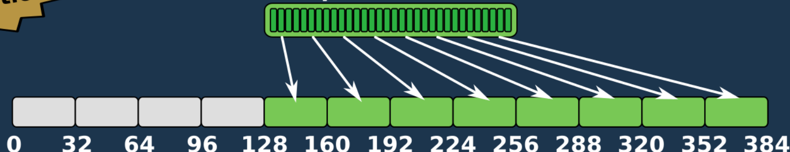
Warp Read / Write

1 sector : 32 bytes

Cache line size :
- 128 bytes
- 4 sectors



Really close to CPU Vectorization



Memory Layout

Memory Layout

Read / Write :
- 1 sector
- 32 bytes

Memory Layout

Read / Write :
- 1 sector
- 32 bytes

Array of Structure (AoS)

```
struct Coeff{
	float u, v, w;
	float x[8], y[8], z;
};
```

80 Bytes

Memory Layout

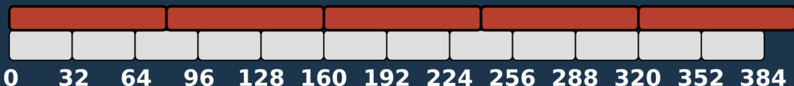
Read / Write :
- 1 sector
- 32 bytes

Array of Structure (AoS)

Warp of **32 threads**

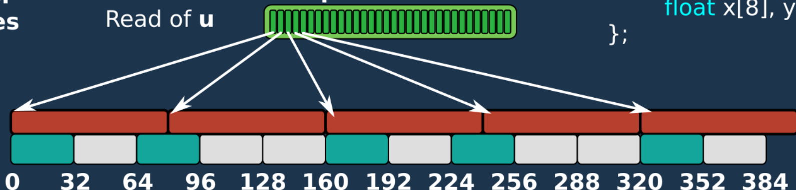
```
struct Coeff{
	float u, v, w;
	float x[8], y[8], z;
};
```

80 Bytes



Memory Layout

Read / Write :
- 1 sector
- 32 bytes

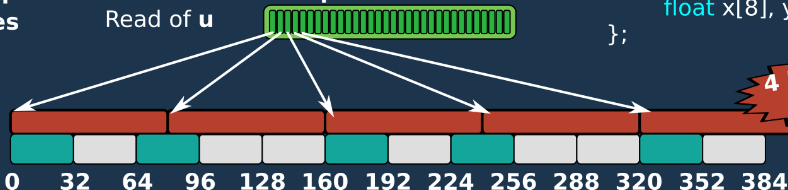



```
struct Coeff{
    float u, v, w;
    float x[8], y[8], z;
};
```

80 Bytes

Memory Layout

Read / Write :
- **1 sector**
- **32 bytes**




```
struct Coeff{
    float u, v, w;
    float x[8], y[8], z;
};
```

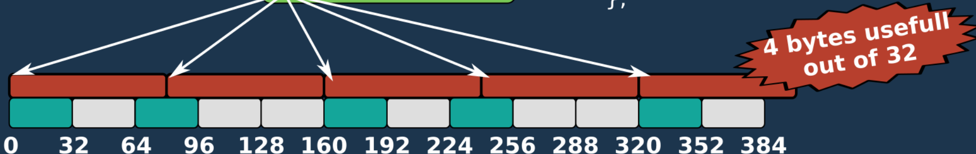
80 Bytes

Memory Layout

Read / Write :
- **1 sector**
- **32 bytes**

Array of Structure (AoS)

Read of **u**




```
struct Coeff{
	float u, v, w;
	float x[8], y[8], z;
};
```

80 Bytes

Strucutre of Array (SoA)

```
struct Coeff{
	float *u, *v, *w;
	float *x0, ... *x7, *y0, ... *y7, *z;
};
```

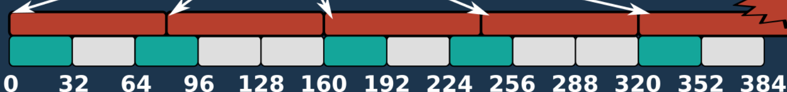
Memory Layout

Read / Write :
- **1 sector**
- **32 bytes**

Array of Structure (AoS)

Read of **u**

Warp of **32 threads**



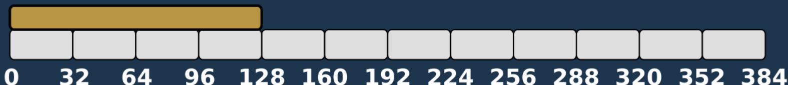

```
struct Coeff{
    float u, v, w;
    float x[8], y[8], z;
};
```

80 Bytes

Strucutre of Array (SoA)

Warp of **32 threads**

```
struct Coeff{
    float *u, *v, *w;
    float *x0, ... *x7, *y0, ... *y7, *z;
};
```



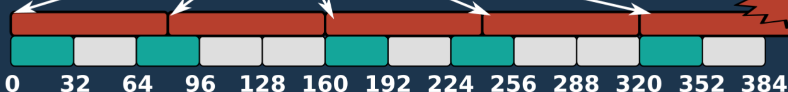
Memory Layout

Read / Write :
- 1 sector
- 32 bytes

Array of Structure (AoS)

Warp of 32 threads

Read of **u**



0 32 64 96 128 160 192 224 256 288 320 352 384

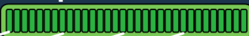
```
struct Coeff{
    float u, v, w;
    float x[8], y[8], z;
};
```

80 Bytes

Strucutre of Array (SoA)

Warp of 32 threads

Read of **u**



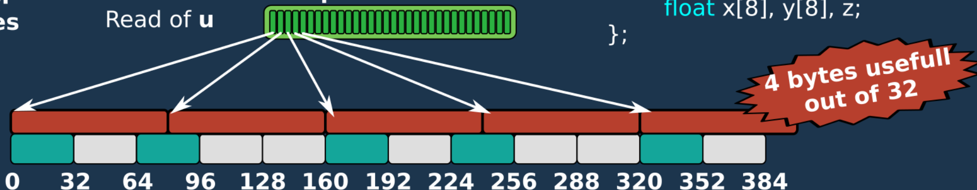
0 32 64 96 128 160 192 224 256 288 320 352 384

```
struct Coeff{
    float *u, *v, *w;
    float *x0, ... *x7, *y0, ... *y7, *z;
};
```

LAPP
Laboratoire d'Annecy de Physique des Particules

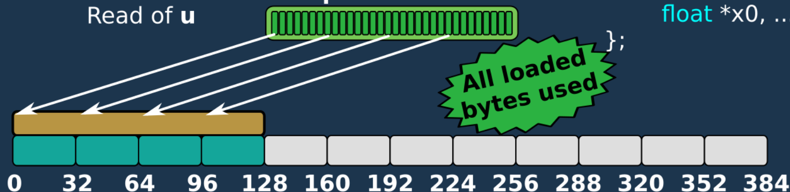
Memory Layout

Read / Write :
- **1 sector**
- **32 bytes**




```
struct Coeff{
    float u, v, w;
    float x[8], y[8], z;
};
```

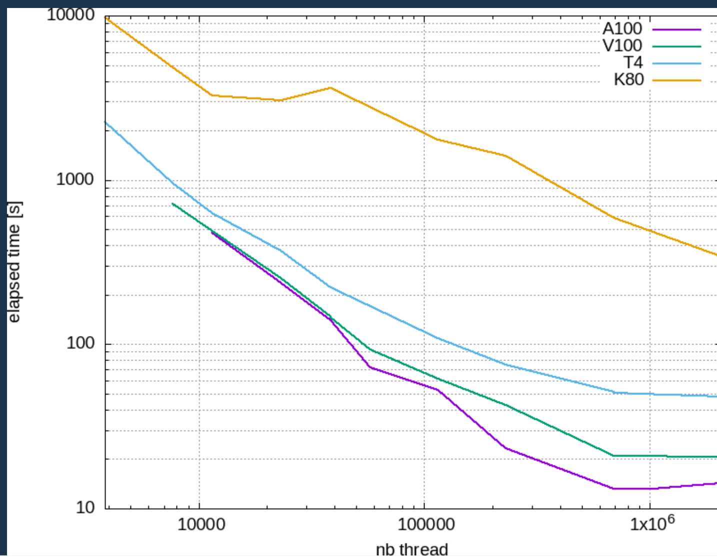
80 Bytes



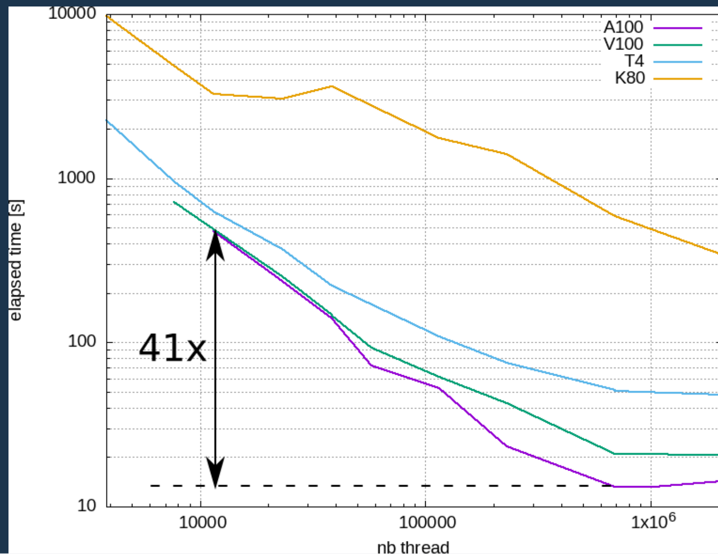

```
struct Coeff{
    float *u, *v, *w;
    float *x0, ... *x7, *y0, ... *y7, *z;
};
```

How many threads ?

How many threads ?



How many threads ?



Express Parallelism

Express Parallelism

Element

Express Parallelism

Element Independent Elements => Independent Computing => Parallelism

Express Parallelism

Element

Independent Elements => Independent Computing => Parallelism

Particle

Express Parallelism

Element

Independent Elements => Independent Computing => Parallelism

Particle

p_x

p_y

p_z

Express Parallelism

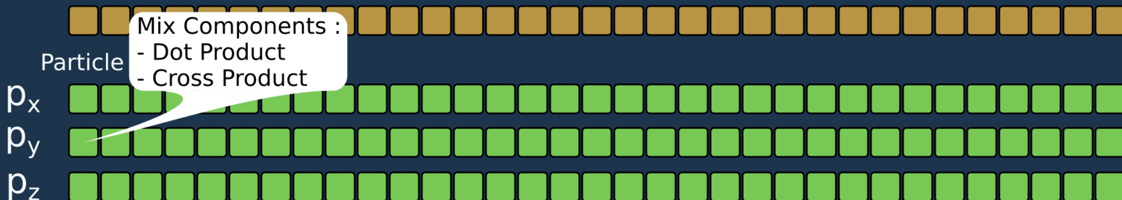
Element

Independent Elements => Independent Computing => Parallelism

Mix Components :
- Dot Product
- Cross Product

Particle

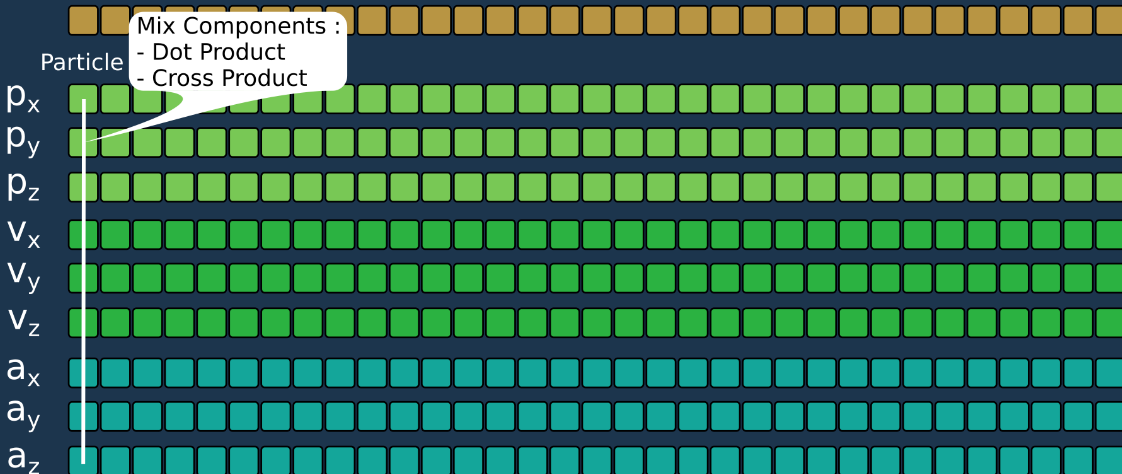
Express Parallelism



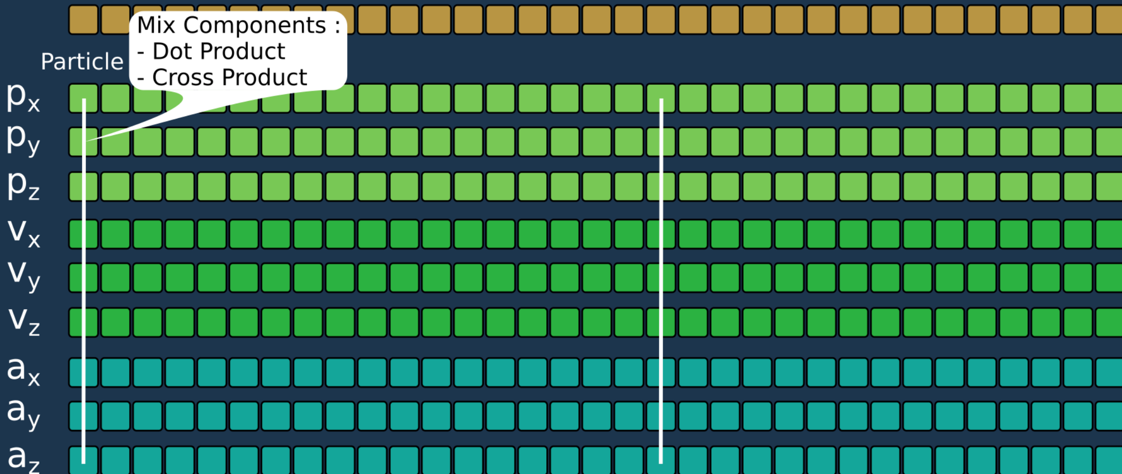
Express Parallelism

Express Parallelism

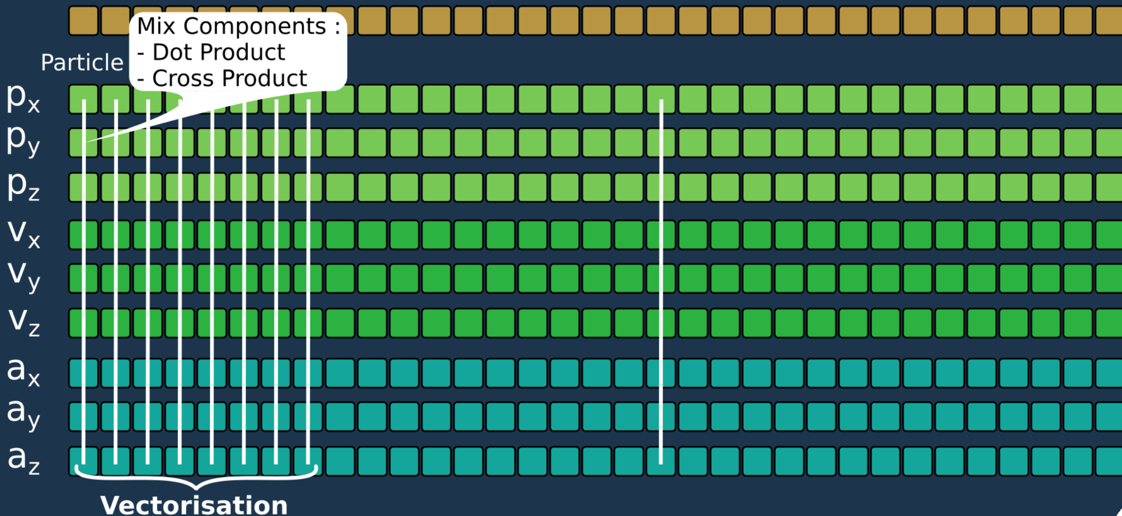
Express Parallelism



Express Parallelism



Express Parallelism



Express Parallelism

Element

Independent Elements => Independent Computing => Parallelism

Particle

Mix Components :
- Dot Product
- Cross Product

p_x

p_y

p_z

v_x

v_y

v_z

a_x

a_y

a_z

Vectorisation

Express Parallelism

Express Parallelism

Event

Express Parallelism

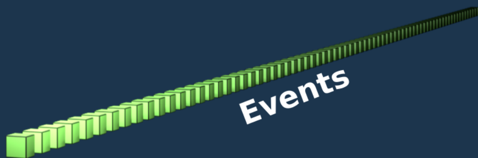
Event

Independent Events => Independent Computing => Parallelism

Express Parallelism

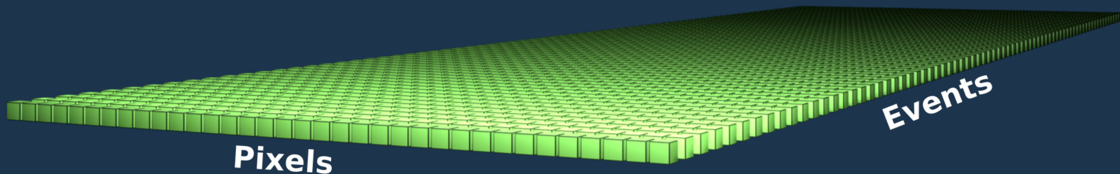
Event

Independent Events => Independent Computing => Parallelism



Express Parallelism

Event Independent Events => Independent Computing => Parallelism



Express Parallelism

Event

Independent Events => Independent Computing => Parallelism

Slices

Pixels

Events

Laboratoire d'Annecy de Physique des Particules

Express Parallelism

Event

Independent Events => Independent Computing => Parallelism

Slices

Pixels

Events

Step 1 : Independent Computing in Calibration

Express Parallelism

Event

Independent Events => Independent Computing => Parallelism

Slices

Pixels

Events

Step 2 :
Independent Computing in Integration

Step 1 : Independent Computing in Calibration

Express Parallelism

Event Independent Events => Independent Computing => Parallelism

Reduction

Slices

Pixels

Events

Step 2 :
Independent Computing in Integration

Step 1 : Independent Computing in Calibration

Express Parallelism

Event Independent Events => Independent Computing => Parallelism

Step 2 : Independent Computing in Integration

Step 1 : Independent Computing in Calibration

Express Parallelism

Event

Independent Events => Independent Computing => Parallelism

Reduction

Allows Vectorization

Contiguous Data

Slices

Pixels

Events

Step 2 :
Independent Computing in Integration

Step 1 : Independent Computing in Calibration

Express Parallelism

Event Independent Events => Independent Computing => Parallelism

Reduction

Allows Vectorization

Even for Integration

Contiguous Data

Slices

Pixels

Events

Step 2 :
Independent Computing in Integration

Step 1 : Independent Computing in Calibration

Express Parallelism

Event   Independent Events => Independent Computing => Parallelism

Reduction

Allows Vectorization

Even for Integration

Contiguous Data

Slices

Pixels

Events

Step 2 :
Independent Computing in Integration

Step 1 : Independent Computing in Calibration

Express Parallelism

Event Independent Events => Independent Computing => Parallelism

Step 2 : Independent Computing in Integration

Step 1 : Independent Computing in Calibration

Express Parallelism

Event   Independent Events => Independent Computing => Parallelism

Reduction

Allows Vectorization

Even for Integration

Contiguous Data

Slices

Parallelisms :
- Cores
- Nodes
- Clusters

Events

Pixels

Step 2 :
Independent Computing in Integration

Step 1 : Independent Computing in Calibration

LAPP
Laboratoire d'Annecy de Physique des Particules

Express Parallelism

Event   Independent Events => Independent Computing => Parallelism

Reduction

Allows Vectorization

Even for Integration

Computing Drives Data Storage

Contiguous Data

Slices

Parallelisms :
- Cores
- Nodes
- Clusters

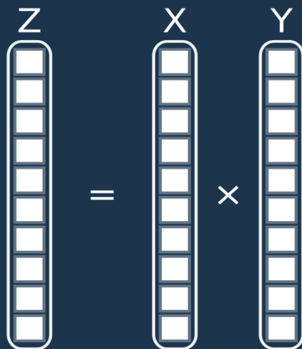
Events

Pixels

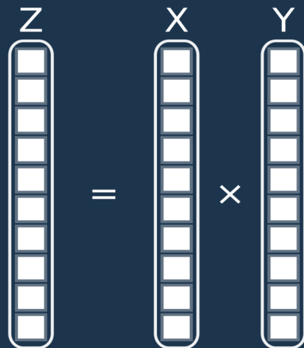
Step 2 :
Independent Computing in Integration

Step 1 : Independent Computing in Calibration

Hadamard Product CUDA

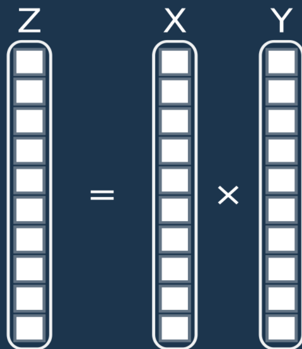


Hadamard Product CUDA



Hadamard Product CUDA

```
__global__ void hadamard_product_kernel(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement){
	int i = blockIdx.x*blockDim.x + threadIdx.x;
	vecResult[i] = vecLeft[i] * vecRight[i];
}
```

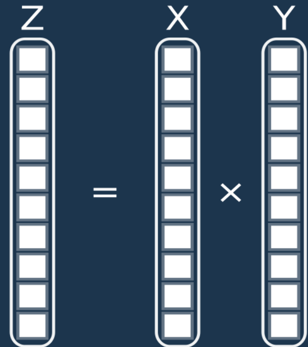


Hadamard Product CUDA

```
__global__ void hadamard_product_kernel(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement){
	int i = blockIdx.x*blockDim.x + threadIdx.x;
	vecResult[i] = vecLeft[i] * vecRight[i];
}
```

Execution on
Device or **Host**

Element Wise
Operation



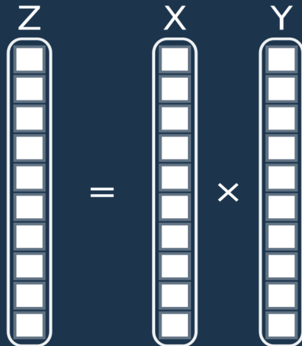
Hadamard Product CUDA

```
__global__ void hadamard_product_kernel(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement){
	int i = blockIdx.x*blockDim.x + threadIdx.x;
	vecResult[i] = vecLeft[i] * vecRight[i];
}
```

Execution on
Device or **Host**

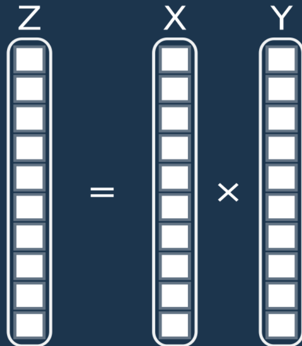
Thread position

Element Wise
Operation



Hadamard Product CUDA

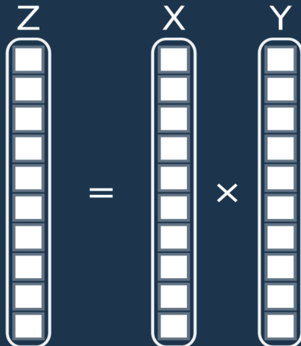
```
void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
                                        int maxNbThreadPerBlockX)
{
        //Allocations
        float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
        int sizeByte = nbElement*sizeof(float);
        cudaMalloc((void**)&vecResultd, sizeByte);
        cudaMalloc((void**)&vecLeftd, sizeByte);
        cudaMalloc((void**)&vecRightd, sizeByte);

        //Block and grid size
        int dimGridX = 1;
        int dimBlockX = 1;
        if(nbElement > maxNbThreadPerBlockX){
                dimBlockX = maxNbThreadPerBlockX;
                dimGridX = nbElement/dimBlockX;
        }else{
                dimBlockX = nbElement;
        }

        //Transfer to Device
        cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
        PLIB_CUDA_CHECK_FILE
        cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
        PLIB_CUDA_CHECK_FILE

        dim3 dimGrid(dimGridX, 1, 1);
        dim3 dimBlock(dimBlockX, 1, 1);
        hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
        PLIB_CUDA_CHECK_FILE
        //Transfer to Host
        cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
        PLIB_CUDA_CHECK_FILE
        //Free memory
        cudaFree(vecRightd);
        cudaFree(vecLeftd);
        cudaFree(vecResultd);
}
```



Hadamard Product CUDA

```
void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
                           int maxNbThreadPerBlockX)
{
	//Allocations
	float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
	int sizeByte = nbElement*sizeof(float);
	cudaMalloc((void**)&vecResultd, sizeByte);
	cudaMalloc((void**)&vecLeftd, sizeByte);
	cudaMalloc((void**)&vecRightd, sizeByte);

	//Block and grid size
	int dimGridX = 1;
	int dimBlockX = 1;
	if(nbElement > maxNbThreadPerBlockX){
		dimBlockX = maxNbThreadPerBlockX;
		dimGridX = nbElement/dimBlockX;
	}else{
		dimBlockX = nbElement;
	}

	//Transfer to Device
	cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE

	dim3 dimGrid(dimGridX, 1, 1);
	dim3 dimBlock(dimBlockX, 1, 1);
	hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
	PLIB_CUDA_CHECK_FILE
	//Transfer to Host
	cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
	PLIB_CUDA_CHECK_FILE
	//Free memory
	cudaFree(vecRightd);
	cudaFree(vecLeftd);
	cudaFree(vecResultd);
}
```

Allocation on **GPU**



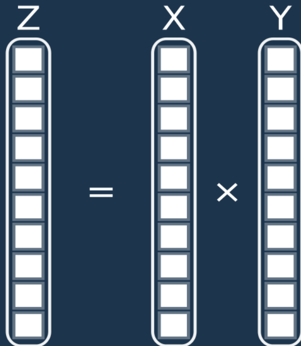
Hadamard Product CUDA

```
void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
                                        int maxNbThreadPerBlockX)
{
	//Allocations
	float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
	int sizeByte = nbElement*sizeof(float);
	cudaMalloc((void**)&vecResultd, sizeByte);
	cudaMalloc((void**)&vecLeftd, sizeByte);
	cudaMalloc((void**)&vecRightd, sizeByte);
	
	//Block and grid size
	int dimGridX = 1;
	int dimBlockX = 1;
	if(nbElement > maxNbThreadPerBlockX){
		dimBlockX = maxNbThreadPerBlockX;
		dimGridX = nbElement/dimBlockX;
	}else{
		dimBlockX = nbElement;
	}
	
	//Transfer to Device
	cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	
	dim3 dimGrid(dimGridX, 1, 1);
	dim3 dimBlock(dimBlockX, 1, 1);
	hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
	PLIB_CUDA_CHECK_FILE
	//Transfer to Host
	cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
	PLIB_CUDA_CHECK_FILE
	//Free memory
	cudaFree(vecRightd);
	cudaFree(vecLeftd);
	cudaFree(vecResultd);
}
```

Allocation on **GPU**

Free on **GPU**

Element Wise Operation



Hadamard Product CUDA

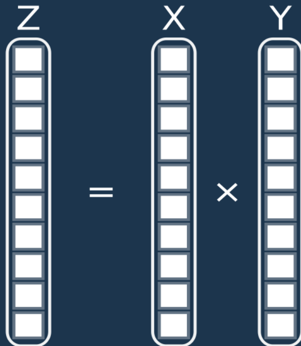
```
void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
                           int maxNbThreadPerBlockX)
{
	//Allocations
	float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
	int sizeByte = nbElement*sizeof(float);
	cudaMalloc((void**)&vecResultd, sizeByte);
	cudaMalloc((void**)&vecLeftd, sizeByte);
	cudaMalloc((void**)&vecRightd, sizeByte);

	//Block and grid size
	int dimGridX = 1;
	int dimBlockX = 1;
	if(nbElement > maxNbThreadPerBlockX){
		dimBlockX = maxNbThreadPerBlockX;
		dimGridX = nbElement/dimBlockX;
	}else{
		dimBlockX = nbElement;
	}

	//Transfer to Device
	cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE

	dim3 dimGrid(dimGridX, 1, 1);
	dim3 dimBlock(dimBlockX, 1, 1);
	hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
	PLIB_CUDA_CHECK_FILE
	//Transfer to Host
	cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
	PLIB_CUDA_CHECK_FILE
	//Free memory
	cudaFree(vecRightd);
	cudaFree(vecLeftd);
	cudaFree(vecResultd);
}
```

Allocation on **GPU**

Free on **GPU**

Out of this function if many calls

Element Wise Operation



Hadamard Product CUDA

```
void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
                           int maxNbThreadPerBlockX)
{
	//Allocations
	float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
	int sizeByte = nbElement*sizeof(float);
	cudaMalloc((void**)&vecResultd, sizeByte);
	cudaMalloc((void**)&vecLeftd, sizeByte);
	cudaMalloc((void**)&vecRightd, sizeByte);

	//Block and grid size
	int dimGridX = 1;
	int dimBlockX = 1;
	if(nbElement > maxNbThreadPerBlockX){
		dimBlockX = maxNbThreadPerBlockX;
		dimGridX = nbElement/dimBlockX;
	}else{
		dimBlockX = nbElement;
	}

	//Transfer to Device
	cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE

	dim3 dimGrid(dimGridX, 1, 1);
	dim3 dimBlock(dimBlockX, 1, 1);
	hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
	PLIB_CUDA_CHECK_FILE
	//Transfer to Host
	cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
	PLIB_CUDA_CHECK_FILE
	//Free memory
	cudaFree(vecRightd);
	cudaFree(vecLeftd);
	cudaFree(vecResultd);
}
```

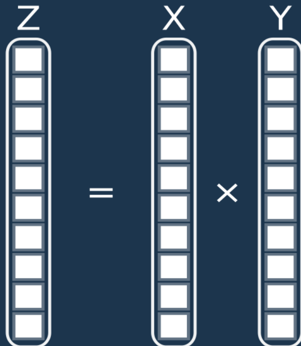
Allocation on **GPU**

Transfer data to **GPU**

Free on **GPU**

Out of this function if many calls

Element Wise Operation



Hadamard Product CUDA

```
void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
                           int maxNbThreadPerBlockX)
{
	//Allocations
	float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
	int sizeByte = nbElement*sizeof(float);
	cudaMalloc((void**)&vecResultd, sizeByte);
	cudaMalloc((void**)&vecLeftd, sizeByte);
	cudaMalloc((void**)&vecRightd, sizeByte);

	//Block and grid size
	int dimGridX = 1;
	int dimBlockX = 1;
	if(nbElement > maxNbThreadPerBlockX){
		dimBlockX = maxNbThreadPerBlockX;
		dimGridX = nbElement/dimBlockX;
	}else{
		dimBlockX = nbElement;
	}

	//Transfer to Device
	cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE

	dim3 dimGrid(dimGridX, 1, 1);
	dim3 dimBlock(dimBlockX, 1, 1);
	hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
	PLIB_CUDA_CHECK_FILE
	//Transfer to Host
	cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
	PLIB_CUDA_CHECK_FILE
	//Free memory
	cudaFree(vecRightd);
	cudaFree(vecLeftd);
	cudaFree(vecResultd);
}
```

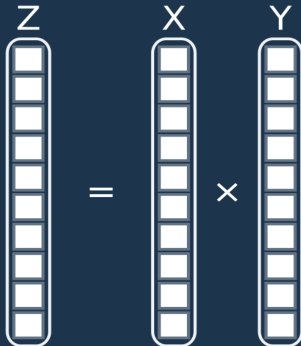
Allocation on **GPU**

Transfer data to **GPU**

Transfer result back to **host**

Free on **GPU**

Out of this function if many calls



Hadamard Product CUDA

```
void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
                           int maxNbThreadPerBlockX)
{
	//Allocations
	float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
	int sizeByte = nbElement*sizeof(float);
	cudaMalloc((void**)&vecResultd, sizeByte);
	cudaMalloc((void**)&vecLeftd, sizeByte);
	cudaMalloc((void**)&vecRightd, sizeByte);
	//Block and grid size
	int dimGridX = 1;
	int dimBlockX = 1;
	if(nbElement > maxNbThreadPerBlockX){
		dimBlockX = maxNbThreadPerBlockX;
		dimGridX = nbElement/dimBlockX;
	}else{
		dimBlockX = nbElement;
	}
	//Transfer to Device
	cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE

	dim3 dimGrid(dimGridX, 1, 1);
	dim3 dimBlock(dimBlockX, 1, 1);
	hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
	PLIB_CUDA_CHECK_FILE
	//Transfer to Host
	cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
	PLIB_CUDA_CHECK_FILE
	//Free memory
	cudaFree(vecRightd);
	cudaFree(vecLeftd);
	cudaFree(vecResultd);
}
```

Out of this function if many calls

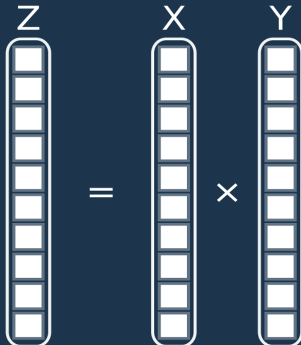
Allocation on **GPU**

Determine Grid and **Block** size

Transfer data to **GPU**

Transfer result back to **host**

Free on **GPU**



Hadamard Product CUDA

```
void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
                           int maxNbThreadPerBlockX)
{
	//Allocations
	float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
	int sizeByte = nbElement*sizeof(float);
	cudaMalloc((void**)&vecResultd, sizeByte);
	cudaMalloc((void**)&vecLeftd, sizeByte);
	cudaMalloc((void**)&vecRightd, sizeByte);
	//Block and grid size
	int dimGridX = 1;
	int dimBlockX = 1;
	if(nbElement > maxNbThreadPerBlockX){
		dimBlockX = maxNbThreadPerBlockX;
		dimGridX = nbElement/dimBlockX;
	}else{
		dimBlockX = nbElement;
	}
	//Transfer to Device
	cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	dim3 dimGrid(dimGridX, 1, 1);
	dim3 dimBlock(dimBlockX, 1, 1);
	hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
	PLIB_CUDA_CHECK_FILE
	//Transfer to Host
	cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
	PLIB_CUDA_CHECK_FILE
	//Free memory
	cudaFree(vecRightd);
	cudaFree(vecLeftd);
	cudaFree(vecResultd);
}
```

Out of this function **if many calls**

Allocation on **GPU**

Determine Grid and **Block** size

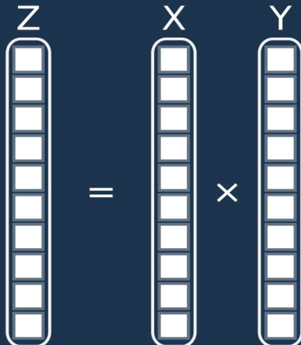
Transfer data to **GPU**

Call Kernel

Transfer result back to **host**

Free on **GPU**

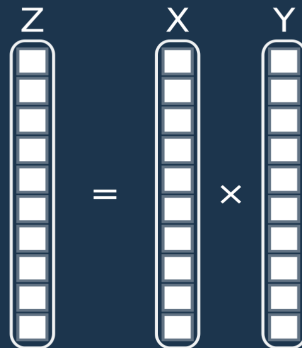
Element Wise Operation



Hadamard Product CUDA

```
#include <cuda.h>
#include <cuda_runtime.h>

__global__ void hadamard_product_kernel(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement){
	int i = blockIdx.x*blockDim.x + threadIdx.x;
	vecResult[i] = vecLeft[i] * vecRight[i];
}

void hadamard_product_cuda(float * vecResult, const float * vecLeft, const float * vecRight, int nbElement,
				int maxNbThreadPerBlockX)
{
	//Allocations
	float *vecResultd = NULL, *vecLeftd = NULL, *vecRightd = NULL;
	int sizeByte = nbElement*sizeof(float);
	cudaMalloc((void**)&vecResultd, sizeByte);
	cudaMalloc((void**)&vecLeftd, sizeByte);
	cudaMalloc((void**)&vecRightd, sizeByte);

	//Block and grid size
	int dimGridX = 1;
	int dimBlockX = 1;
	if(nbElement > maxNbThreadPerBlockX){
		dimBlockX = maxNbThreadPerBlockX;
		dimGridX = nbElement/dimBlockX;
	}else{
		dimBlockX = nbElement;
	}

	//Transfer to Device
	cudaMemcpy(vecLeftd, vecLeft, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE
	cudaMemcpy(vecRightd, vecRight, sizeByte, cudaMemcpyHostToDevice);
	PLIB_CUDA_CHECK_FILE

	dim3 dimGrid(dimGridX, 1, 1);
	dim3 dimBlock(dimBlockX, 1, 1);
	hadamard_product_kernel<<<dimGrid, dimBlock>>>(vecResultd, vecLeftd, vecRightd, nbElement);
	PLIB_CUDA_CHECK_FILE
	//Transfer to Host
	cudaMemcpy(vecResult, vecResultd, sizeByte, cudaMemcpyDeviceToHost);
	PLIB_CUDA_CHECK_FILE
	//Free memory
	cudaFree(vecRightd);
	cudaFree(vecLeftd);
	cudaFree(vecResultd);
}
```

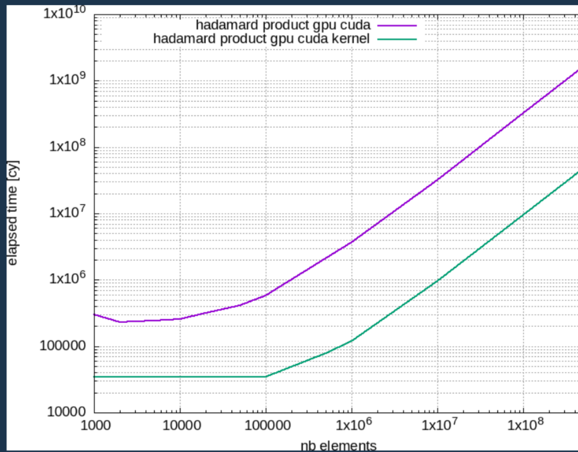
Element Wise
Operation

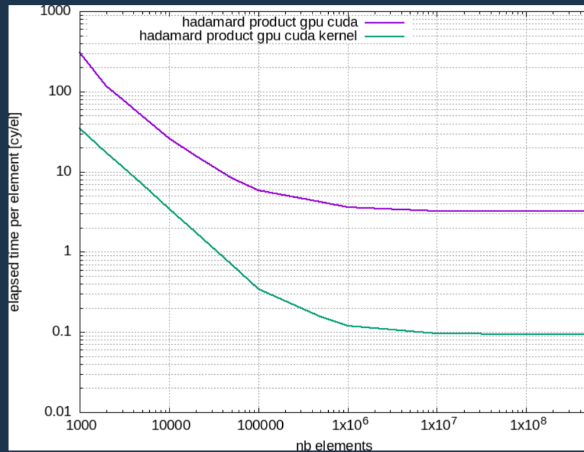


Laboratoire d'Annecy de Physique des Particules

Perf : Hadamard Product CUDA

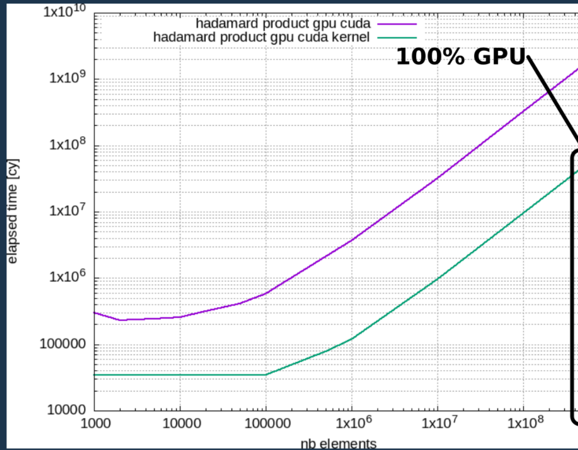
On A3000 GPU

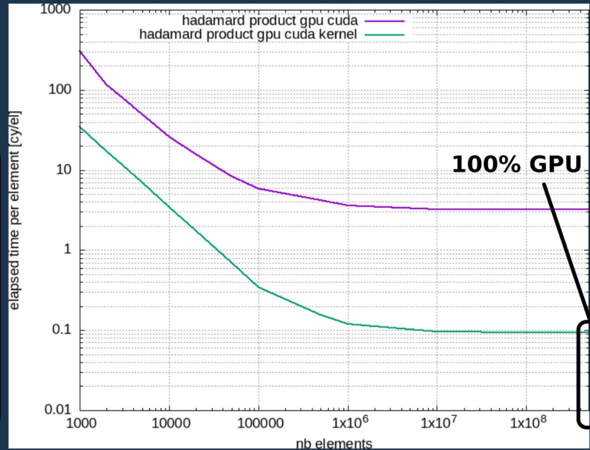




Perf : Hadamard Product CUDA

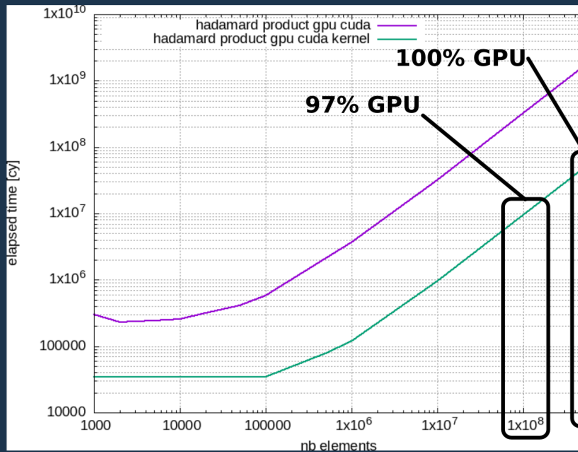
On A3000 GPU

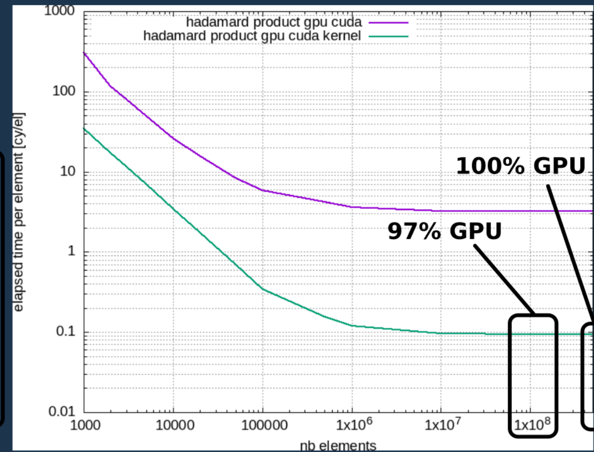




Perf : Hadamard Product CUDA

On A3000 GPU

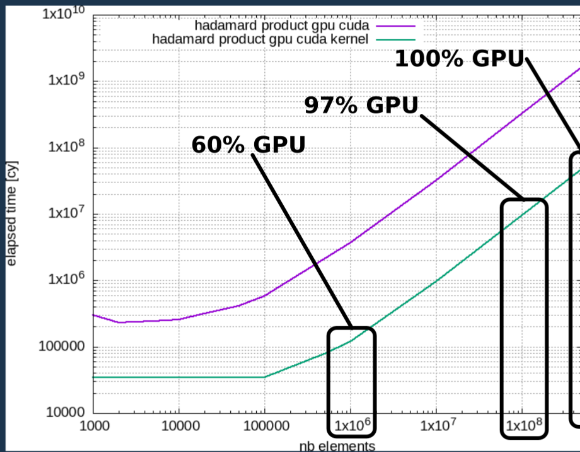


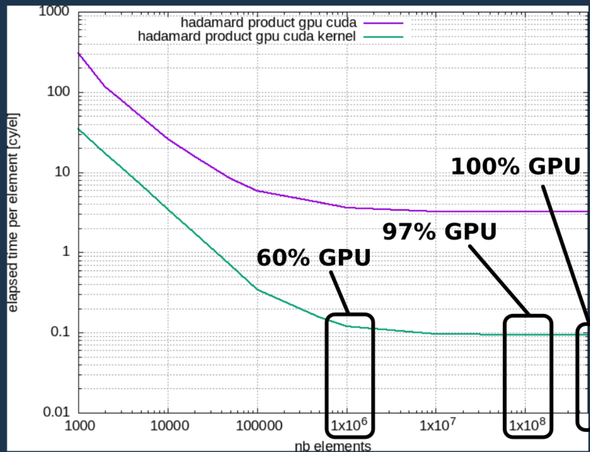


Laboratoire d'Annecy de Physique des Particules

Perf : Hadamard Product CUDA

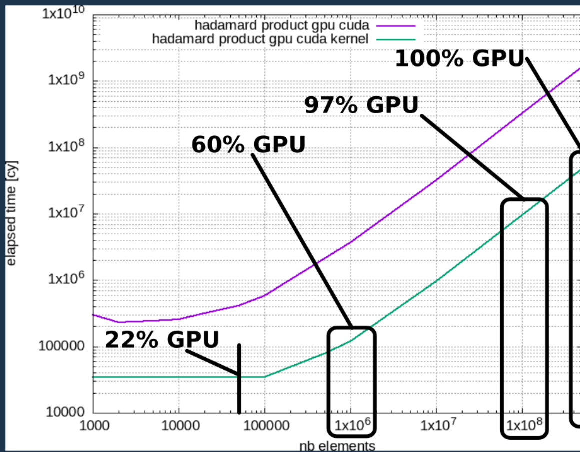
On A3000 GPU

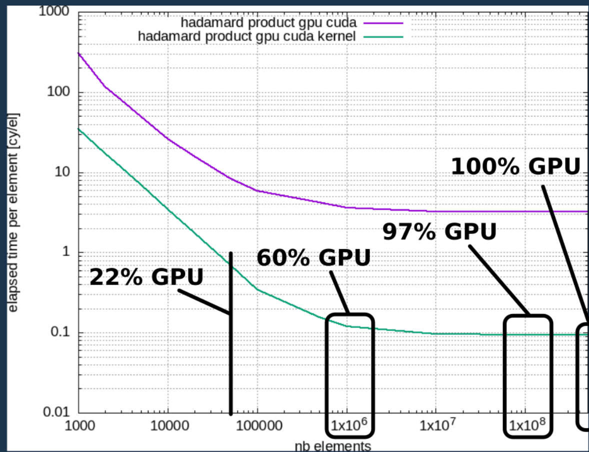




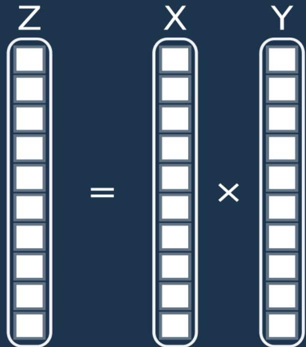
Perf : Hadamard Product CUDA

On A3000 GPU

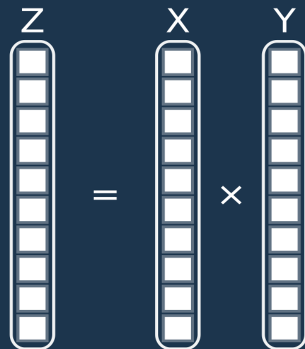




Hadamard Product Thrust



Hadamard Product Thrust



Hadamard Product Thrust

```
int main(int argc, char** argv){
	size_t nbElement(500000000);

	thrust::host_vector<float> h_vecX(nbElement), h_vecY(nbElement);
	for(size_t i(0lu); i < nbElement; ++i){
		h_vecX[i] = i*19lu%11;
		h_vecY[i] = i*27lu%19;
	}
	// Transfer data to the device.
	thrust::device_vector<float> d_vecX = h_vecX, d_vecY = h_vecY, d_vecRes(nbElement);

	thrust::transform(d_vecX.begin(), d_vecX.end(), d_vecY.begin(), d_vecRes.begin(),
		[=] __host__ __device__ (const float& x, const float& y){
			return x * y;
		});

	// Transfer data to the host
	thrust::host_vector<float> h_vecRes = d_vecRes;

	std::cout << "x = " << h_vecX.front() << ", y = " << h_vecY.front() << ", res = " << h_vecRes.front() << std::endl;
	return 0;
}
```

Hadamard Product Thrust

```
int main(int argc, char** argv){
	size_t nbElement(500000000);

	thrust::host_vector<float> h_vecX(nbElement), h_vecY(nbElement);
	for(size_t i(0lu); i < nbElement; ++i){
		h_vecX[i] = i*19lu%11;
		h_vecY[i] = i*27lu%19;
	}
	// Transfer data to the device.
	thrust::device_vector<float> d_vecX = h_vecX, d_vecY = h_vecY, d_vecRes(nbElement);

	thrust::transform(d_vecX.begin(), d_vecX.end(), d_vecY.begin(), d_vecRes.begin(),
		[=] __host__ __device__ (const float& x, const float& y){
			return x * y;
		});

	// Transfer data to the host
	thrust::host_vector<float> h_vecRes = d_vecRes;

	std::cout << "x = " << h_vecX.front() << ", y = " << h_vecY.front() << ", res = " << h_vecRes.front() << std::endl;
	return 0;
}
```

Initialisation

Hadamard Product Thrust

```
int main(int argc, char** argv){
	size_t nbElement(500000000);

	thrust::host_vector<float> h_vecX(nbElement), h_vecY(nbElement);
	for(size_t i(0lu); i < nbElement; ++i){
		h_vecX[i] = i*19lu%11;
		h_vecY[i] = i*27lu%19;
	}
	// Transfer data to the device.
	thrust::device_vector<float> d_vecX = h_vecX, d_vecY = h_vecY, d_vecRes(nbElement);

	thrust::transform(d_vecX.begin(), d_vecX.end(), d_vecY.begin(), d_vecRes.begin(),
		[=] __host__ __device__ (const float& x, const float& y){
			return x * y;
		});

	// Transfer data to the host
	thrust::host_vector<float> h_vecRes = d_vecRes;

	std::cout << "x = " << h_vecX.front() << ", y = " << h_vecY.front() << ", res = " << h_vecRes.front() << std::endl;
	return 0;
}
```

Initialisation

Transfer Host to Device

Hadamard Product Thrust

```
int main(int argc, char** argv){
	size_t nbElement(500000000);

	thrust::host_vector<float> h_vecX(nbElement), h_vecY(nbElement);
	for(size_t i(0lu); i < nbElement; ++i){
		h_vecX[i] = i*19lu%11;
		h_vecY[i] = i*27lu%19;
	}
	// Transfer data to the device.
	thrust::device_vector<float> d_vecX = h_vecX, d_vecY = h_vecY, d_vecRes(nbElement);
	thrust::transform(d_vecX.begin(), d_vecX.end(), d_vecY.begin(), d_vecRes.begin(),
		[=] __host__ __device__ (const float& x, const float& y){
			return x * y;
		});

	// Transfer data to the host
	thrust::host_vector<float> h_vecRes = d_vecRes;

	std::cout << "x = " << h_vecX.front() << ", y = " << h_vecY.front() << ", res = " << h_vecRes.front() << std::endl;
	return 0;
}
```

Initialisation

Transfer Host to Device

Compute Hadamard Product

Hadamard Product Thrust

```
int main(int argc, char** argv){
	size_t nbElement(500000000);

	thrust::host_vector<float> h_vecX(nbElement), h_vecY(nbElement);
	for(size_t i(0lu); i < nbElement; ++i){
		h_vecX[i] = i*19lu%11;
		h_vecY[i] = i*27lu%19;
	}
	// Transfer data to the device.
	thrust::device_vector<float> d_vecX = h_vecX, d_vecY = h_vecY, d_vecRes(nbElement);

	thrust::transform(d_vecX.begin(), d_vecX.end(), d_vecY.begin(), d_vecRes.begin(),
		[=] __host__ __device__ (const float& x, const float& y){
			return x * y;
		});

	// Transfer data to the host
	thrust::host_vector<float> h_vecRes = d_vecRes;

	std::cout << "x = " << h_vecX.front() << ", y = " << h_vecY.front() << ", res = " << h_vecRes.front() << std::endl;
	return 0;
}
```

Initialisation

Transfer Host to Device

Compute Hadamard Product

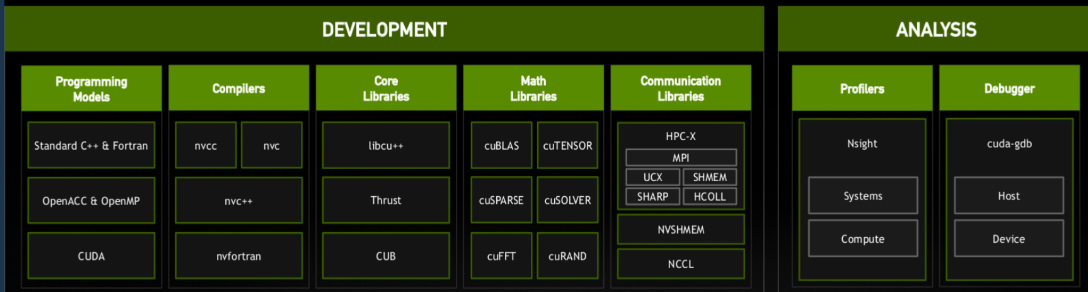
Transfer Device to Host

LAPP
Laboratoire d'Annecy de Physique des Particules

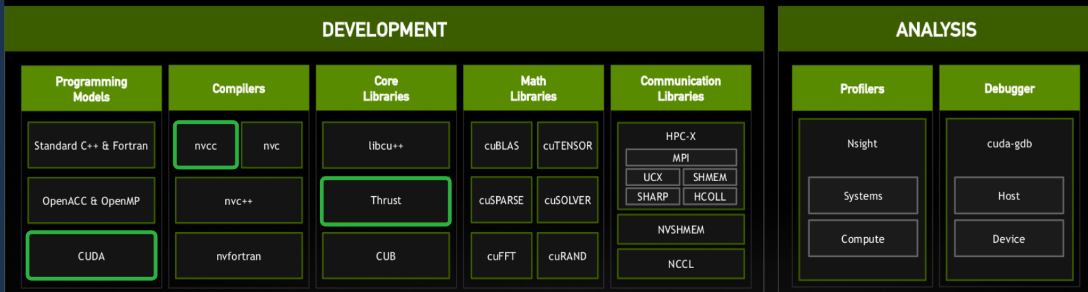
Hadamard Product Thrust

```
#include <iostream>
#include <vector>

#include <thrust/transform.h>
#include <thrust/functional.h>
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>

int main(int argc, char** argv){
	size_t nbElement(500000000);
	
	thrust::host_vector<float> h_vecX(nbElement), h_vecY(nbElement);
	for(size_t i(0lu); i < nbElement; ++i){
		h_vecX[i] = i*19lu%11;
		h_vecY[i] = i*27lu%19;
	}
	// Transfer data to the device.
	thrust::device_vector<float> d_vecX = h_vecX, d_vecY = h_vecY, d_vecRes(nbElement);
	
	thrust::transform(d_vecX.begin(), d_vecX.end(), d_vecY.begin(), d_vecRes.begin(),
		[=] __host__ __device__ (const float& x, const float& y){
			return x * y;
		});
	
	// Transfer data to the host
	thrust::host_vector<float> h_vecRes = d_vecRes;
	
	std::cout << "x = " << h_vecX.front() << ", y = " << h_vecY.front() << ", res = " << h_vecRes.front() << std::endl;
	return 0;
}
```

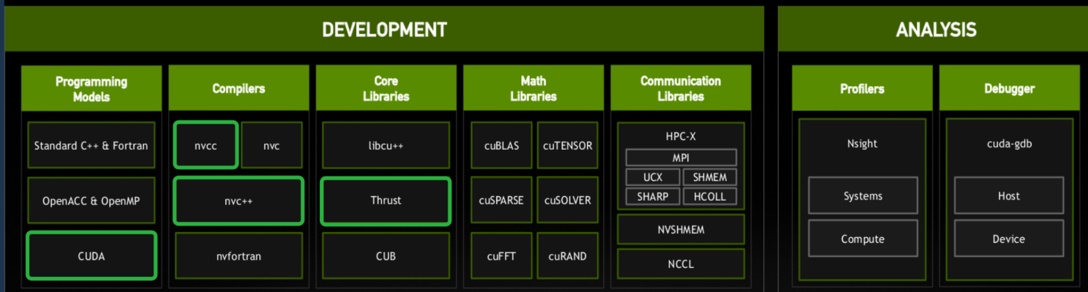
NVidia HPC SDK Introduction



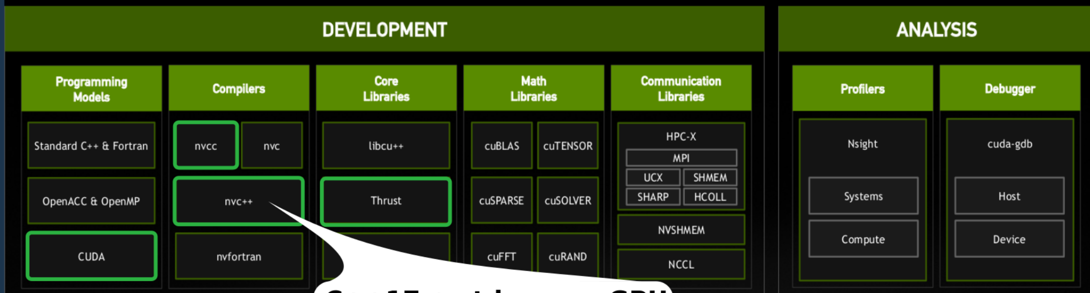
NVidia HPC SDK Introduction



NVidia HPC SDK Introduction

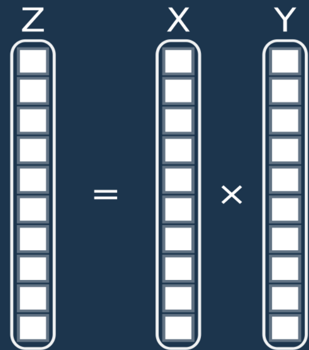


NVidia HPC SDK Introduction

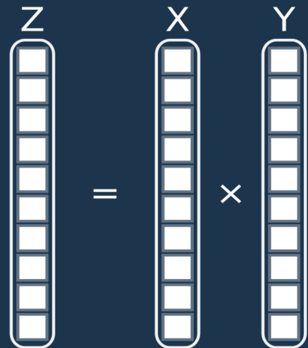


Laboratoire d'Annecy de Physique des Particules

Example : Hadamard Product



Example : Hadamard Product

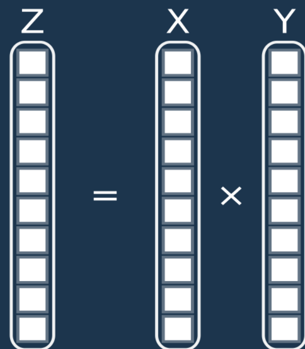


Example : Hadamard Product

C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Element Wise
Operation

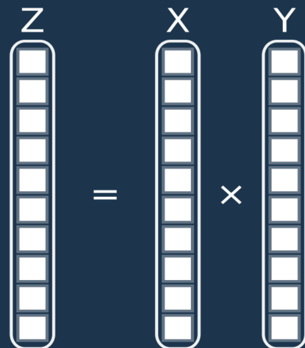


Example : Hadamard Product

C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary



Example : Hadamard Product

C++

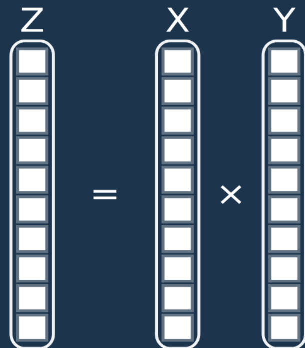
```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary

C++ Algorithm : std::transform

```
std::transform(std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

Element Wise Operation



Example : Hadamard Product

C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary

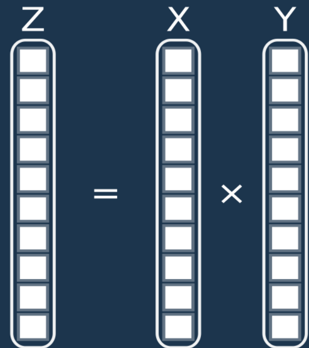
C++ Algorithm : std::transform

```
std::transform(std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

C++ 17 / C++ 20

```
std::transform(std::execution::par_unseq,
	std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

Element Wise
Operation



Example : Hadamard Product

C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary

C++ Algorithm : std::transform

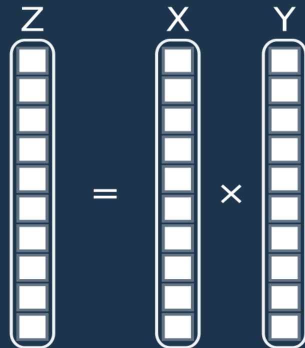
```
std::transform(std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

C++ 17 / C++ 20

Execution Policy

```
std::transform(std::execution::par_unseq,
	std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

Element Wise Operation



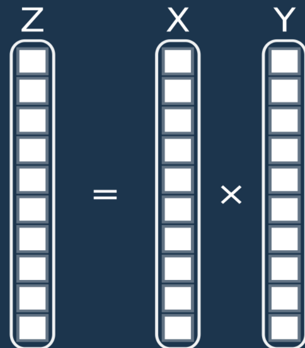
Example : Hadamard Product

C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary

Element Wise Operation



C++ Algorithm : std::transform

```
std::transform(std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

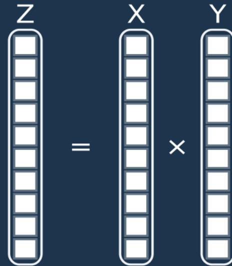
C++ 17 / C++ 20

Execution Policy

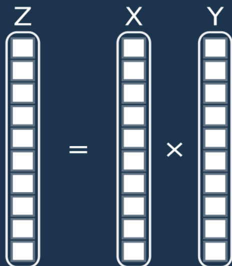
- **seq**
- **unseq**
- **par**
- **par_unseq**

```
std::transform(std::execution::par_unseq,
	std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

Example : Hadamard Product



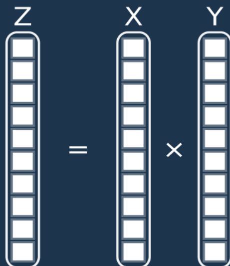
Example : Hadamard Product



With CMake :
- export CC=/path/to/gcc
- export CXX=/path/to/nvc++

Example : Hadamard Product

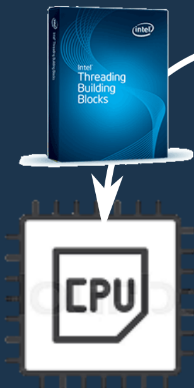
nvc++ -stdpar=cpu

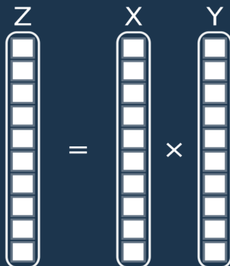


With CMake :
- export CC=/path/to/gcc
- export CXX=/path/to/nvc++

Example : Hadamard Product

nvc++ -stdpar=cpu

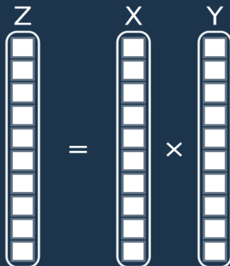




With CMake :
- export CC=/path/to/gcc
- export CXX=/path/to/nvc++

Example : Hadamard Product

nvc++ -stdpar=cpu

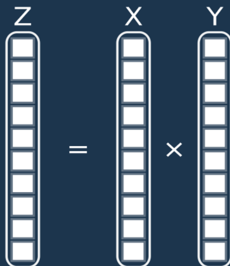


With CMake :
- **export CC=/path/to/gcc**
- **export CXX=/path/to/nvc++**

Do not forget to link with TBB

Example : Hadamard Product

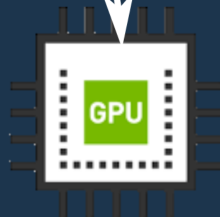
nvc++ -stdpar=cpu



nvc++ -stdpar=gpu

Do not forget to link with TBB

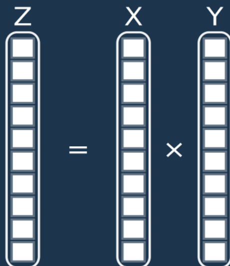
With CMake :
- export CC=/path/to/gcc
- export CXX=/path/to/nvc++



Example : Hadamard Product

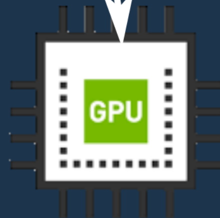
nvc++ -stdpar=cpu

Do not forget to link with TBB



With CMake :
- export CC=/path/to/gcc
- export CXX=/path/to/nvc++

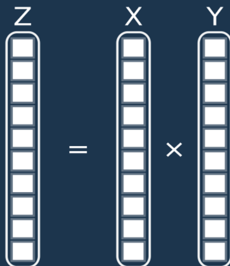
nvc++ -stdpar=gpu



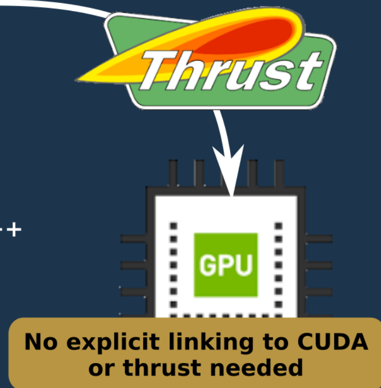
Example : Hadamard Product

nvc++ -stdpar=cpu

Do not forget to link with TBB



nvc++ -stdpar=gpu

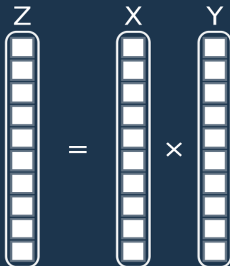


With CMake :
- export CC=/path/to/gcc
- export CXX=/path/to/nvc++

No explicit linking to CUDA or thrust needed

Example : Hadamard Product

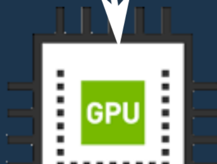
nvc++ -stdpar=cpu



nvc++ -stdpar=gpu

With CMake :
- export CC=/path/to/gcc
- export CXX=/path/to/nvc++

Avoid static Allocation



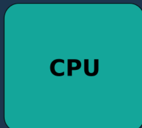
Do not forget to link with TBB

No explicit linking to CUDA or thrust needed

Memory Management

Memory Management

CUDA

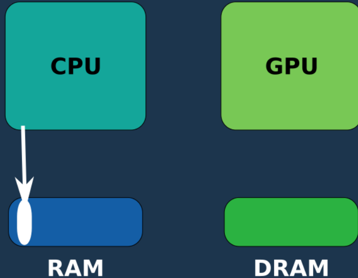


RAM

DRAM

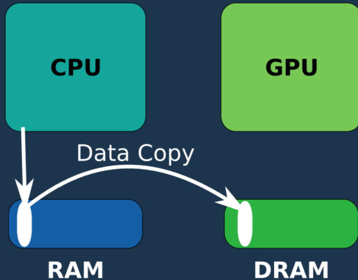
Memory Management

CUDA

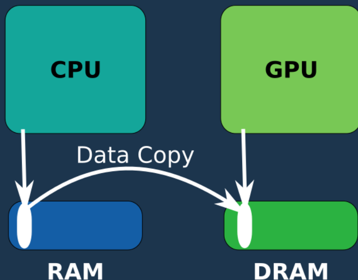


Memory Management

CUDA

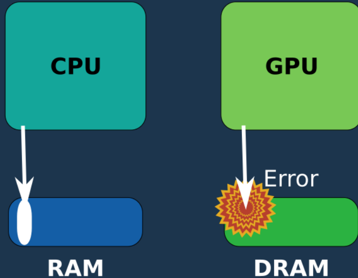


Memory Management

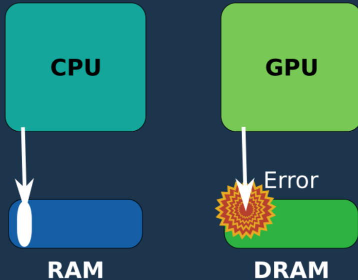


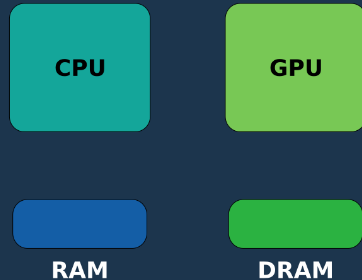
Memory Management

CUDA



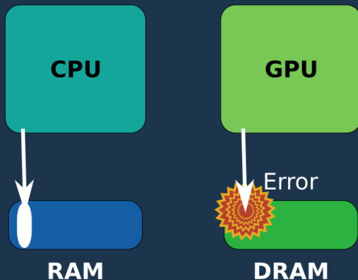
Memory Management



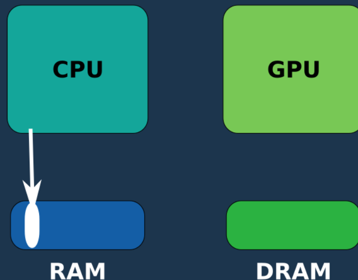


Memory Management

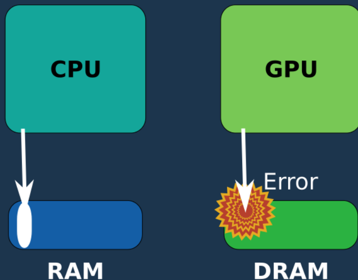
CUDA

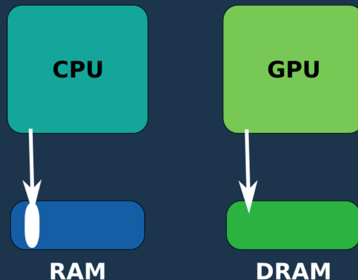


nvc++/nvfortran
Unified Memory



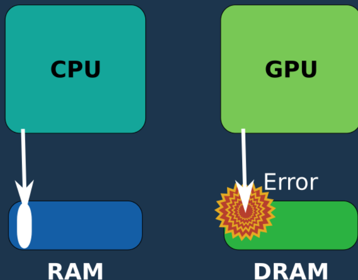
Memory Management



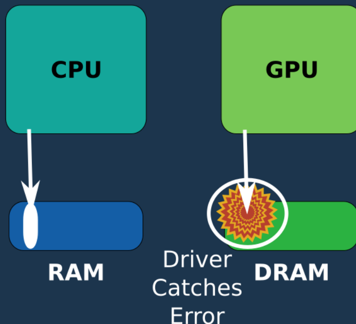


Memory Management

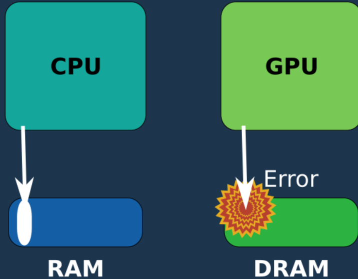
CUDA

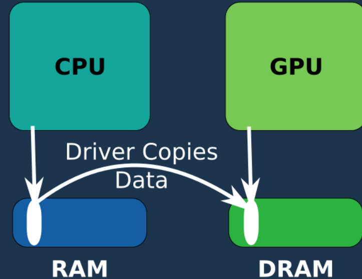


nvc++/nvfortran
Unified Memory

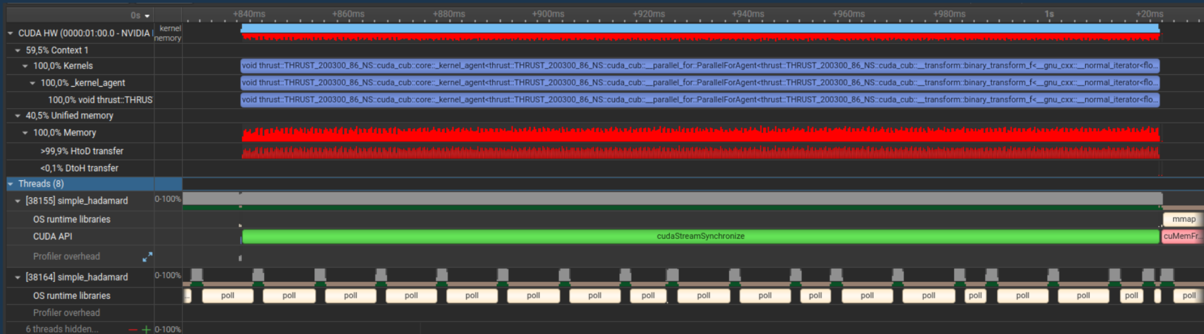


Memory Management





NSight : Hadamard Product



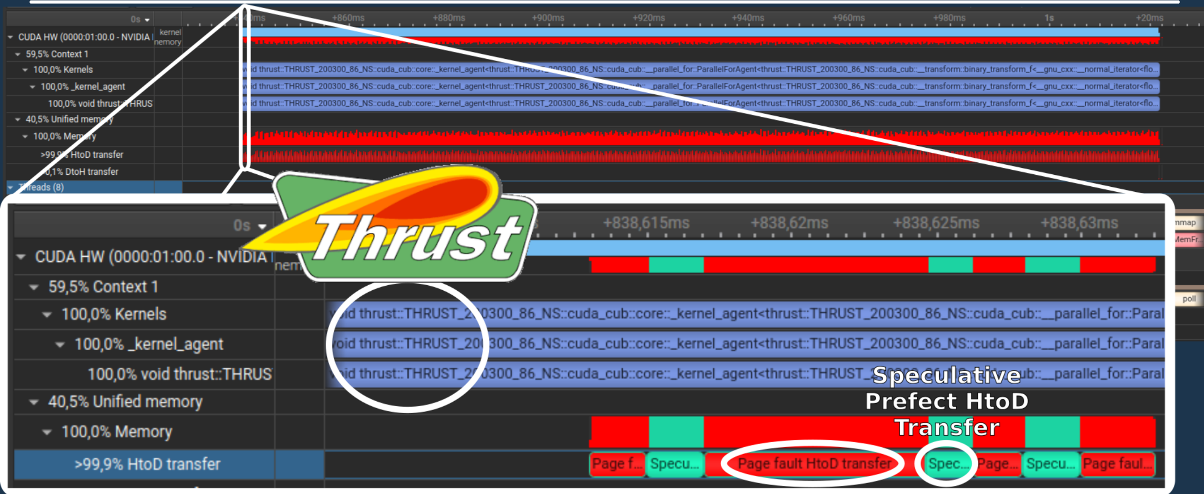
NSight : Hadamard Product

LAPP
Laboratoire d'Annecy de Physique des Particules

NSight : Hadamard Product

NSight : Hadamard Product

NSight : Hadamard Product



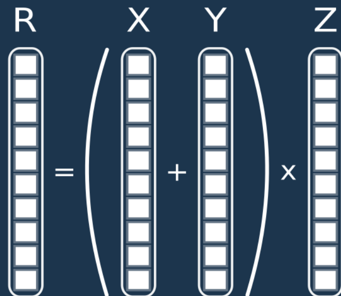
std::transform : triadic

std::transform : triadic

Triadic : **z = x + y**

std::transform : triadic

Triadic : **z = x + y**

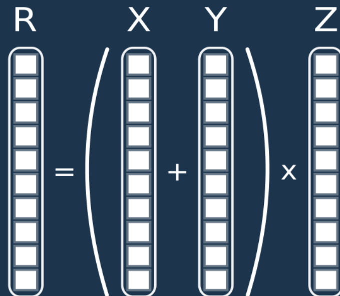


std::transform : triadic

Triadic : **z = x + y**

Classic C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```



std::transform : triadic

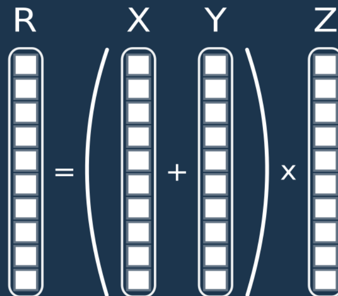
Triadic : **z = x + y**

Classic C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

C++ 17 / 20

```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

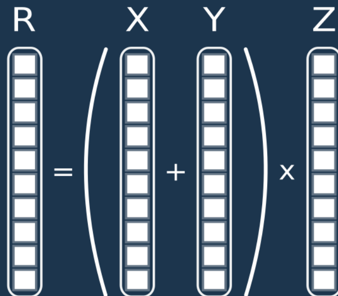


std::transform : triadic

Triadic : **z = x + y**

Classic C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
»	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```



C++ 17 / 20

vecY, **vecZ** have to be **std::vector**

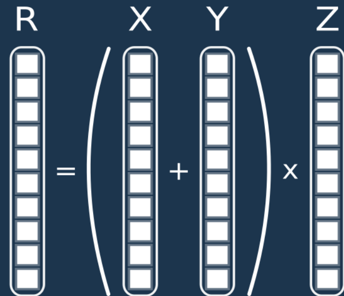
```
std::transform(std::execution::par_unseq,
»	std::begin(vecIndex), std::end(vecIndex),
»	std::begin(vecX), std::begin(vecRes),
»	[=](int i, float x){
»	»	return (x + vecY[i]) * vecZ[i];
»	});
```

std::transform : triadic

Triadic : **z = x + y**

Classic C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```



C++ 17 / 20

vecY, **vecZ** have to be **std::vector**

```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

Works well on GPU

std::transform : triadic

Triadic : **z = x + y**

Classic C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

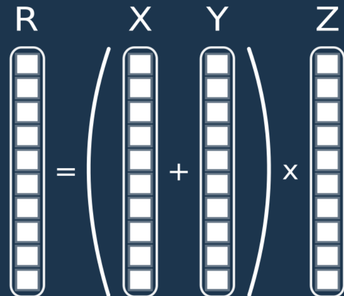
C++ 17 / 20

vecY, **vecZ** have to be **std::vector**

```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

Works well on GPU

Needs extra index table



std::transform : triadic

Triadic : **z = x + y**

Classic C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

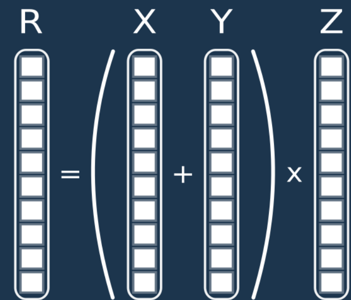
C++ 17 / 20

vecY, **vecZ** have to be **std::vector**

```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

Works well on GPU

Needs extra index table



Close to **CUDA Kernel**

C++23 Way

C++23 Way

Cartesian Product and **iota views**

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

C++23 Way

Cartesian Product and **iota views**

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

Multi-dimensional index

C++23 Way

Cartesian Product and **iota views**

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

No Index Allocation

Multi-dimensional index

C++23 Way

Cartesian Product and **iota views**

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

No Index Allocation

Multi-dimensional index

OK for broadcasting but we can do better

C++23 Way

Cartesian Product and **iota views**

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

No Index Allocation

Multi-dimensional index

Broadcast

OK for broadcasting but we can do better

C++23 Way

Cartesian Product and **iota views**

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

No Index Allocation

Multi-dimensional index

Broadcast

```
auto extentedVecB = std::views::join(std::views::repeat(vecB));
std::transform(std::execution::par, vecA.begin(), vecA.end(),
        std::begin(extentedVecB), vecResult.begin(),
        [=](float a, float b){
                return a * b;
        }
);
```

OK for broadcasting but we can do better

C++23 Way

Cartesian Product and iota views

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

No Index Allocation

Multi-dimensional index

OK for broadcasting but we can do better

Broadcast

```
auto extentedVecB = std::views::join(std::views::repeat(vecB));
std::transform(std::execution::par, vecA.begin(), vecA.end(),
        std::begin(extentedVecB), vecResult.begin(),
        [=](float a, float b){
                return a * b;
        }
);
```

Not vectorized yet

C++23 Way

Cartesian Product and iota views

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

No Index Allocation

Multi-dimensional index

OK for broadcasting but we can do better

Broadcast

```
auto extentedVecB = std::views::join(std::views::repeat(vecB));
std::transform(std::execution::par, vecA.begin(), vecA.end(),
        std::begin(extentedVecB), vecResult.begin(),
        [=](float a, float b){
                return a * b;
        }
);
```

Not vectorized yet

Not parallelized yet

C++23 Way

Cartesian Product and iota views

```
std::mdspan A{input, N, M};
std::mdspan B{output, M, N};
auto v = std::ranges::views::cartesian_product(
        std::ranges::views::iota(0, N),
        std::ranges::views::iota(0, M));

std::for_each(std::execution::par_unseq,
        begin(v), end(v),
        [=] (auto idx) {
                auto [i, j] = idx;
                B[j, i] = A[i, j];
        }
);
```

No Index Allocation

Multi-dimensional index

Broadcast

```
auto extentedVecB = std::views::join(std::views::repeat(vecB));
std::transform(std::execution::par, vecA.begin(), vecA.end(),
        std::begin(extentedVecB), vecResult.begin(),
        [=](float a, float b){
                return a * b;
        }
);
```

Not vectorized yet

Not parallelized yet

OK for broadcasting but we can do better

But only with G++ 13 For now

C++ Evolution

CPU

GPU

C++ Evolution

CPU

GPU

C++ 17

C++ Evolution

CPU

GPU

C++ 20

C++ 17

C++ Evolution

CPU

GPU

C++ 23

C++ 20

C++ 17

C++ Evolution

Conclusion

- Good performances on **GPU**s
  - With **nvcc** (**CUDA**)
  - With **nvc++** (**C++17** / **C++20**)
- Tests with **HPC SDK 24.3**
- Compiler **nvc++** powerful and easy to use with **C++17** and **C++20**
  - **No explicit linking**
  - Automatic **GPU** Targeting or with **CUDA_VISIBLE_DEVICES**
  - **Avoid static allocation**
- Warning about industrial software
  - Will to drive for update
  - Old **GPU**s become **obsolete** :
    - **nvc++** : compute capabilities ≥ 6 (no **K80**)
    - **nvcc** : compute capabilities ≥ 3.5 (no **K80** in **CUDA 12**)
  - Need to save binaries to ensure long usability of **GPU**s