GPU-accelerated 3D MOC in Apollo3

SERMA/LLPR

Cheickh Diop
Director

Santandrea Simone
Supervisor

Gammicchia Andrea
Co-Supervisor

Garrido Ignacio
Student

1. Introduction

Apollo3 and TDT

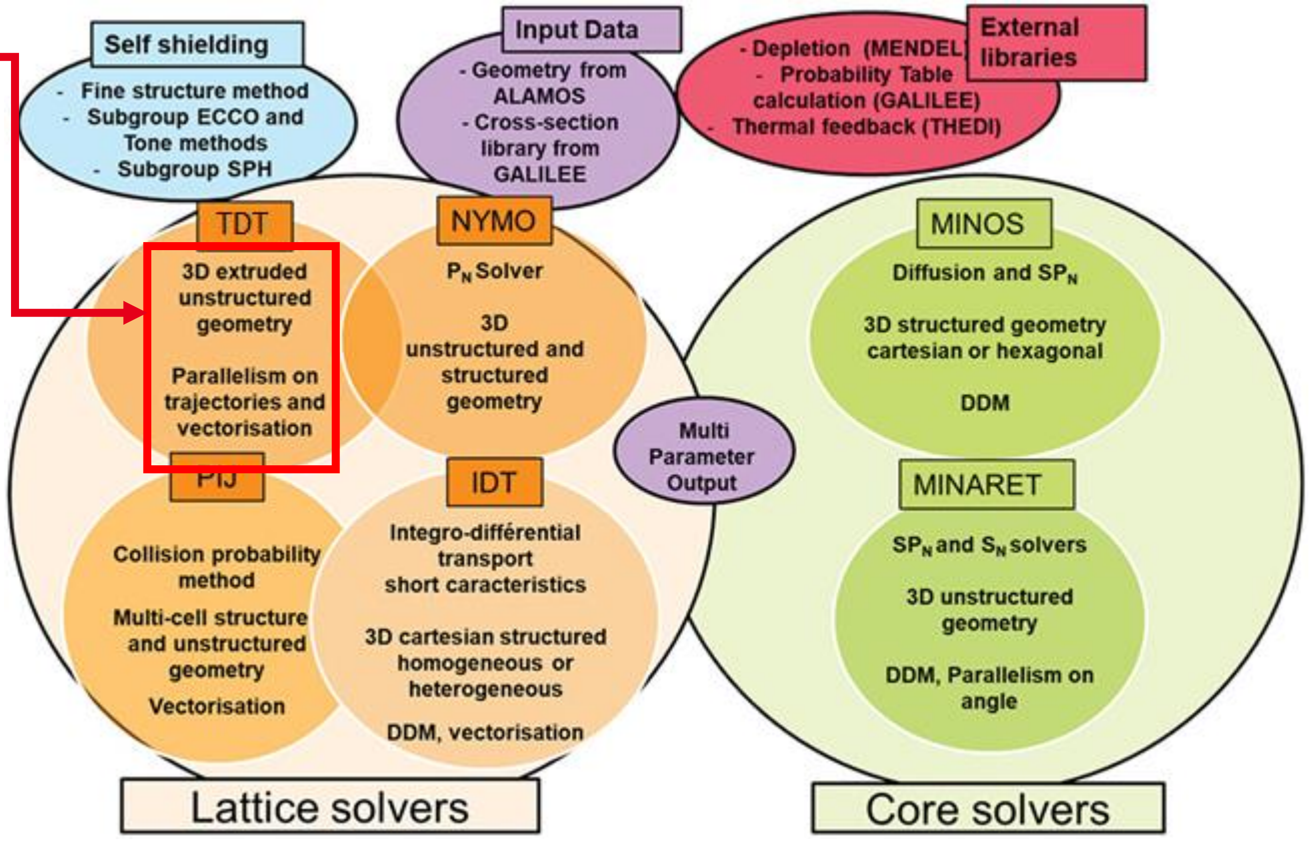
What is Apollo3?

- Deterministic code developed by the CEA with support from EDF and Framatome.
- Designed to solve reactor-physics problems in 2D and 3D, including extruded geometries.

Deterministic Transport by Trajectories

What is the TDT module?

It performs high-fidelity deterministic calculations using the Method of Characteristics (MOC), applied to 3D extruded geometries.


Self shielding
- Fine structure method
- Subgroup ECCO and Tone methods
- Subgroup SPH
Input Data
- Geometry from ALAMOS
- Cross-section library from GALILEE
External libraries
- Depletion (MENDEL)
- Probability Table calculation (GALILEE)
- Thermal feedback (THEDI)
TDT
3D extruded unstructured geometry
Parallelism on trajectories and vectorisation
NYMO
P_N Solver
3D unstructured and structured geometry
PIJ
Collision probability method
Multi-cell structure and unstructured geometry
Vectorisation
IDT
Integro-différential transport short caracteristics
3D cartesian structured homogeneous or heterogeneous
DDM, vectorisation
Multi Parameter Output
MINOS
Diffusion and SP_N
3D structured geometry cartesian or hexagonal
DDM
MINARET
SP_N and S_N solvers
3D unstructured geometry
DDM, Parallelism on angle
Lattice solvers
Core solvers

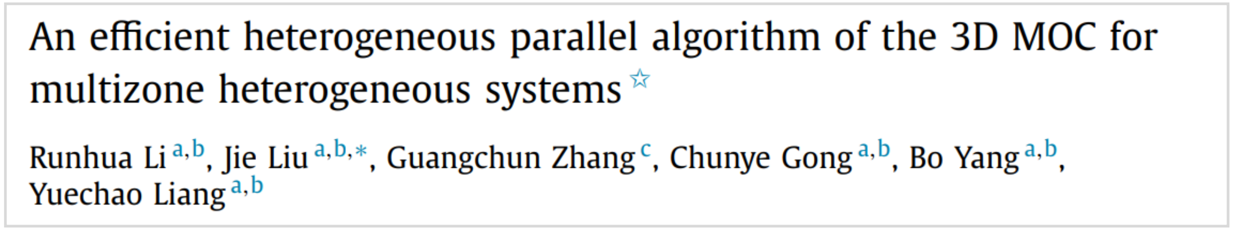
HPC and CUDA Fortran

Why HPC?

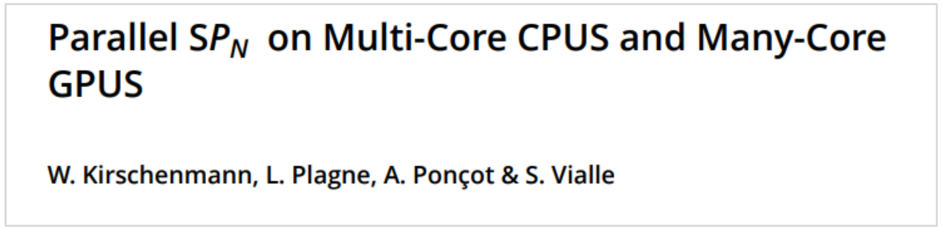
•3-D MOC involves millions of independent trajectories—perfect for massive parallel computing.
•GPUs provide an architecture well suited to exploit this parallelism.
•The goal is to drastically shorten simulation time without any loss of accuracy.

Why GPU and CUDA Fortran?

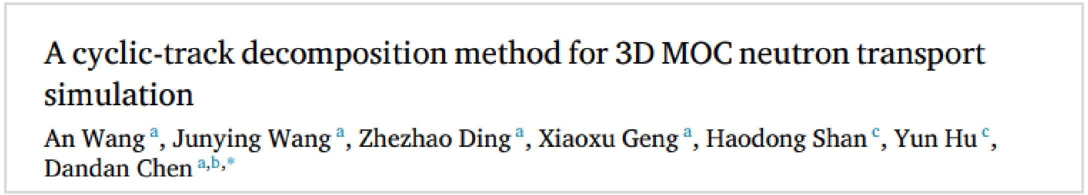
•MOC $\rightarrow$ millions of independent trajectories—ideal for GPUs.
•CUDA Fortran lets GPUs be integrated with the existing codebase.
•Objective: move the entire iterative loop onto the GPU and eliminate unnecessary data transfers.


An efficient heterogeneous parallel algorithm of the 3D MOC for multizone heterogeneous systems ☆

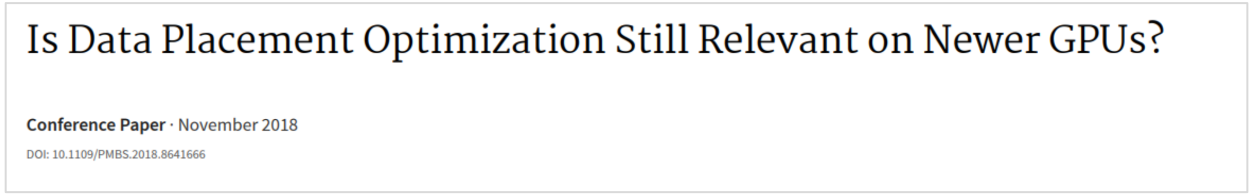
Runhua Li [a,b], Jie Liu [a,b,*], Guangchun Zhang [c], Chunye Gong [a,b], Bo Yang [a,b], Yuechao Liang [a,b]


Parallel SP_N on Multi-Core CPUS and Many-Core GPUS

W. Kirschenmann, L. Plagne, A. Ponçot & S. Vialle


A cyclic-track decomposition method for 3D MOC neutron transport simulation

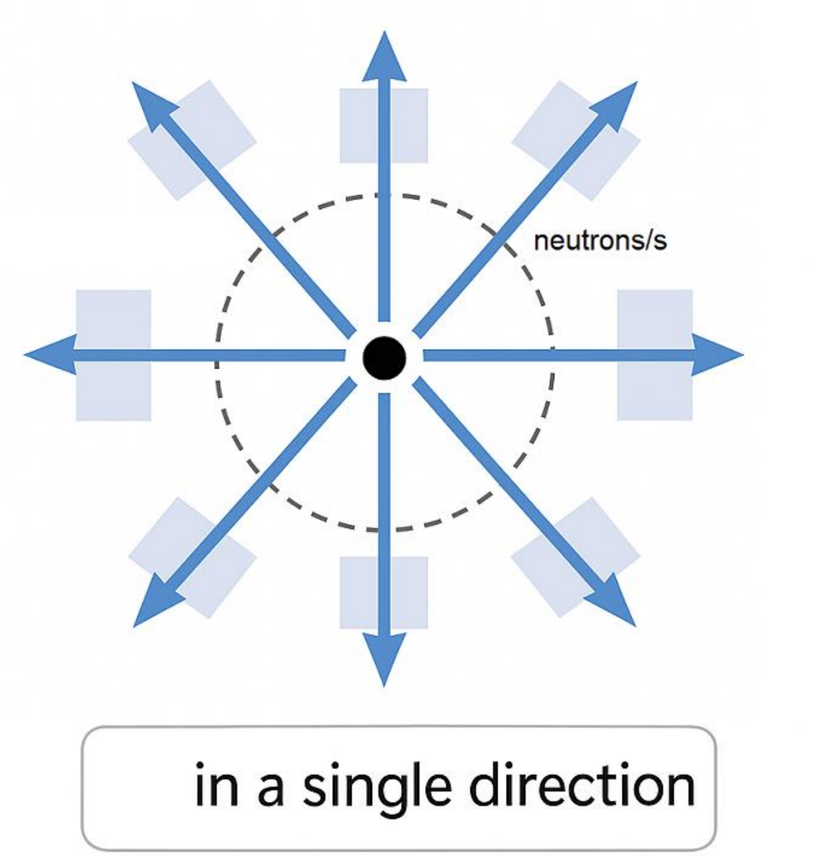
An Wang [a], Junying Wang [a], Zhezhao Ding [a], Xiaoxu Geng [a], Haodong Shan [c], Yun Hu [c], Dandan Chen [a,b,*]

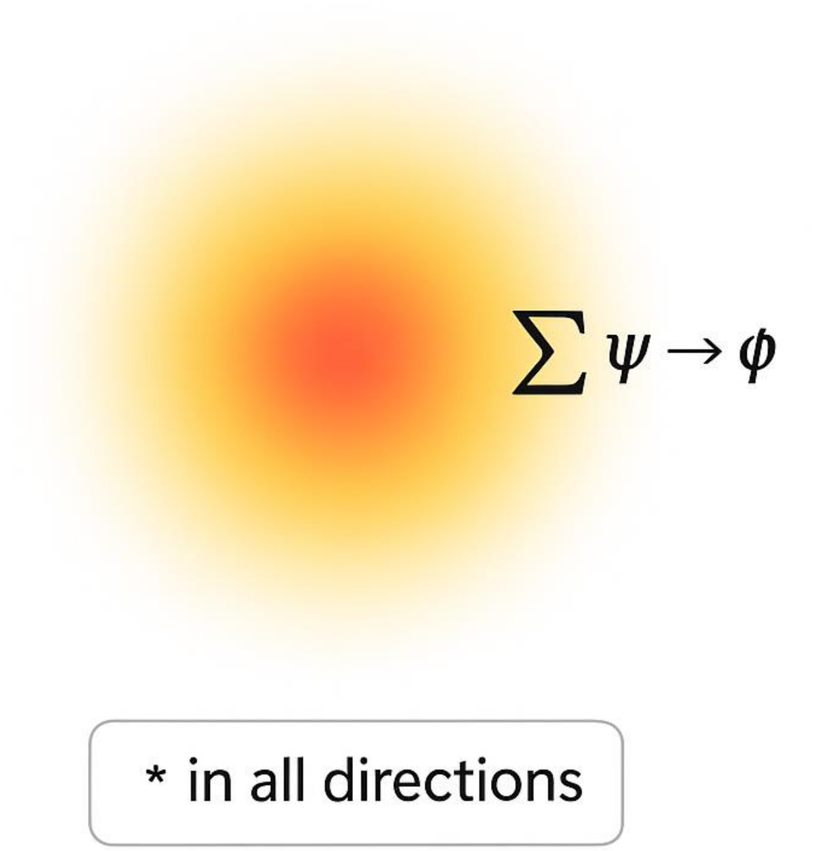

Is Data Placement Optimization Still Relevant on Newer GPUs?

Conference Paper · November 2018

DOI: 10.1109/PMBS.2018.8641666

MOC *– Theory*





Angular neutron flux: The angular neutron flux is the rate at which neutrons cross a unit area per unit time and per unit solid angle in a given direction.

Scalar neutron flux: The scalar neutron flux is the total rate at which neutrons cross a unit area per unit time, equal to the integral of the angular flux over all directions.

MOC – *Theory*

The transport equation is **solved along each trajectory**. The **integral form of the transport equation** along a trajectory *t* is obtained as in (1).

The results are averaged within each region to **update the flux $\overline{\psi_r}$**, completing the iterative scheme. The neutron balance equation in a homogeneous region *r* is expressed as in (2).

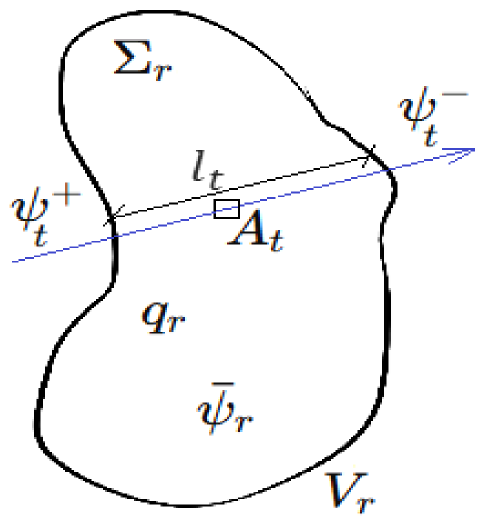
(1). $$\psi^+ = \psi^- e^{-\tau} + \frac{q_r}{\Sigma_r}(1 - e^{-\tau})$$

(2). $$\bar{\psi}_r = \frac{q_r}{\Sigma_r} - \frac{1}{V_r \Sigma_r} \sum_{t \in \text{trajectories}} A_t(\psi^+ - \psi^-)_t$$

- $\bar{\psi}_r$ → **Region-averaged angular flux** in region r (neutron density per unit area, direction, and energy).
- q_r → **Total source term** in region r, including external sources and in-scattering contributions.
- Σ_r → **Macroscopic total cross-section** in region r, representing the probability of neutron interactions.
- V_r → **Volume of region** r.
- t → A specific **trajectory** crossing region r.
- A_t → **Cross-sectional area** associated with trajectory t.
- ψ^+ → **Outgoing angular flux** at the end of the trajectory segment within region r.
- ψ^- → **Incoming angular flux** at the beginning of the trajectory segment within region r.
- τ → **Optical thickness** of the trajectory segment, defined as:

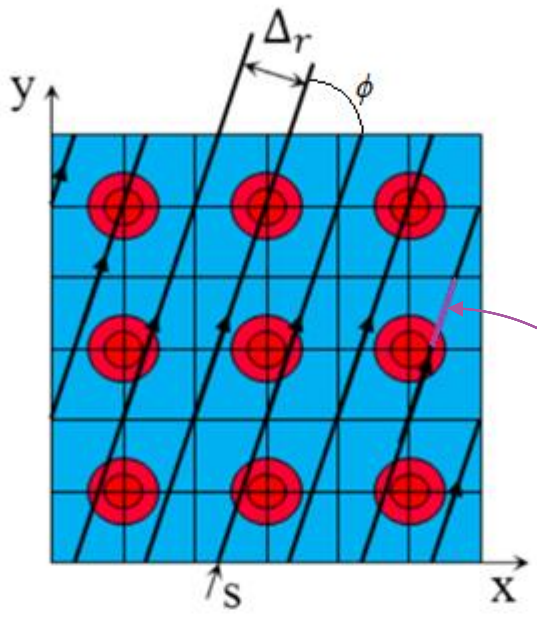
$$\tau = \Sigma_r l_t$$

- l_t → **Chord length**, the distance traveled by a neutron within region r along the trajectory.
- $e^{-\tau}$ → **Attenuation factor**, representing neutron absorption along the trajectory.
- $\frac{q_r}{\Sigma_r}(1 - e^{-\tau})$ → **Neutron source contribution** along the trajectory.



2. Algorithm

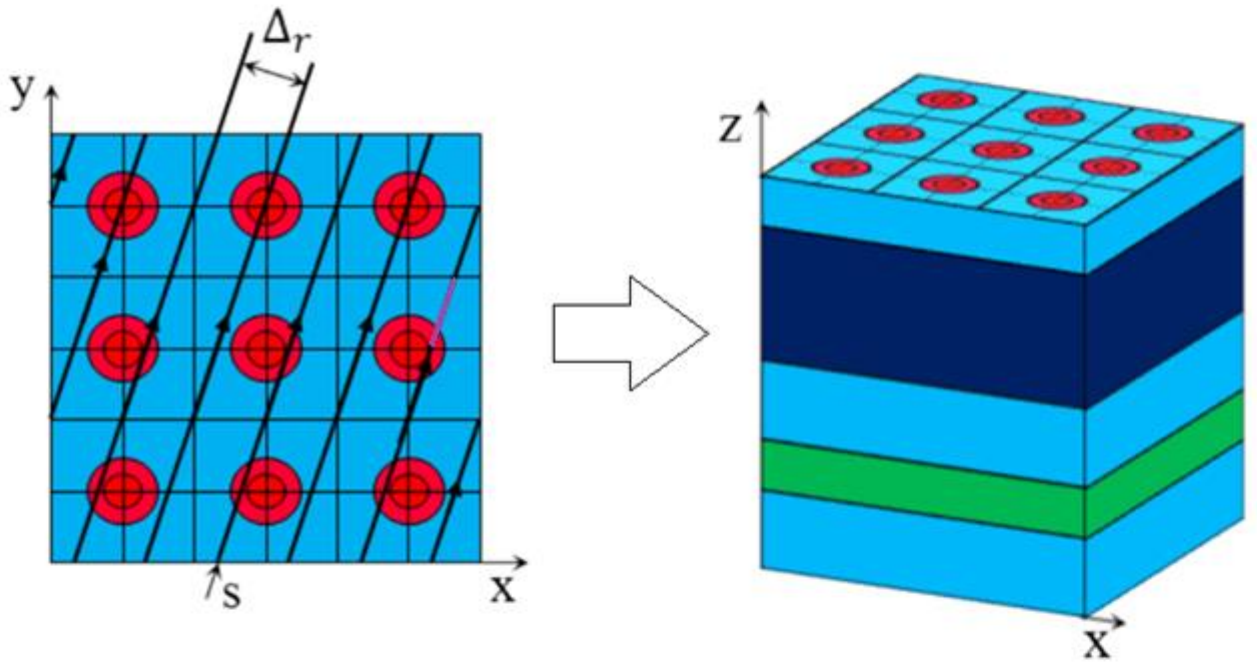
MOC – *Implementation*



Trajectory Construction in 2D:
A set of straight-line paths (**trajectories**) is defined across a 2D unstructured cross-section of the geometry. These lines represent neutron travel directions.

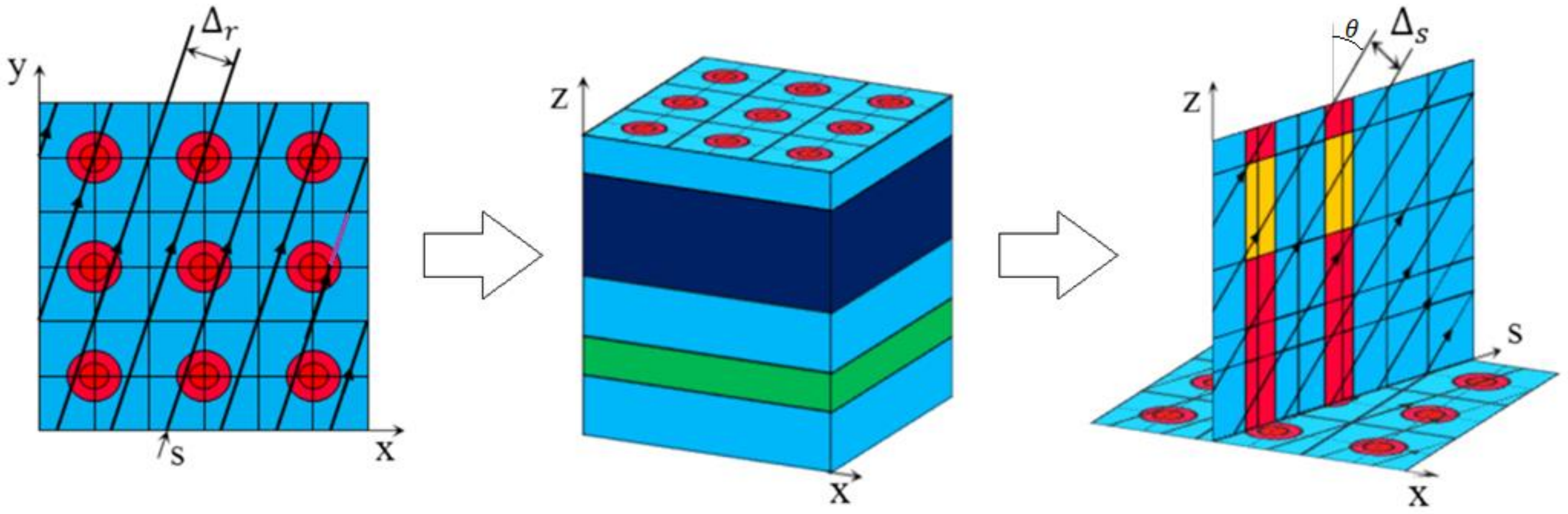
Chords definition:
A **chord** is the segment of a trajectory within a material region. Its **length** determines neutron interactions, essential for solving the **transport equation**.

MOC *– Implementation*



Extension to 3D:
The mesh and 2D characteristics are **extruded** along the **axial direction**, generating a **unstructured 2D-3D extruded geometry**.

MOC *– Implementation*

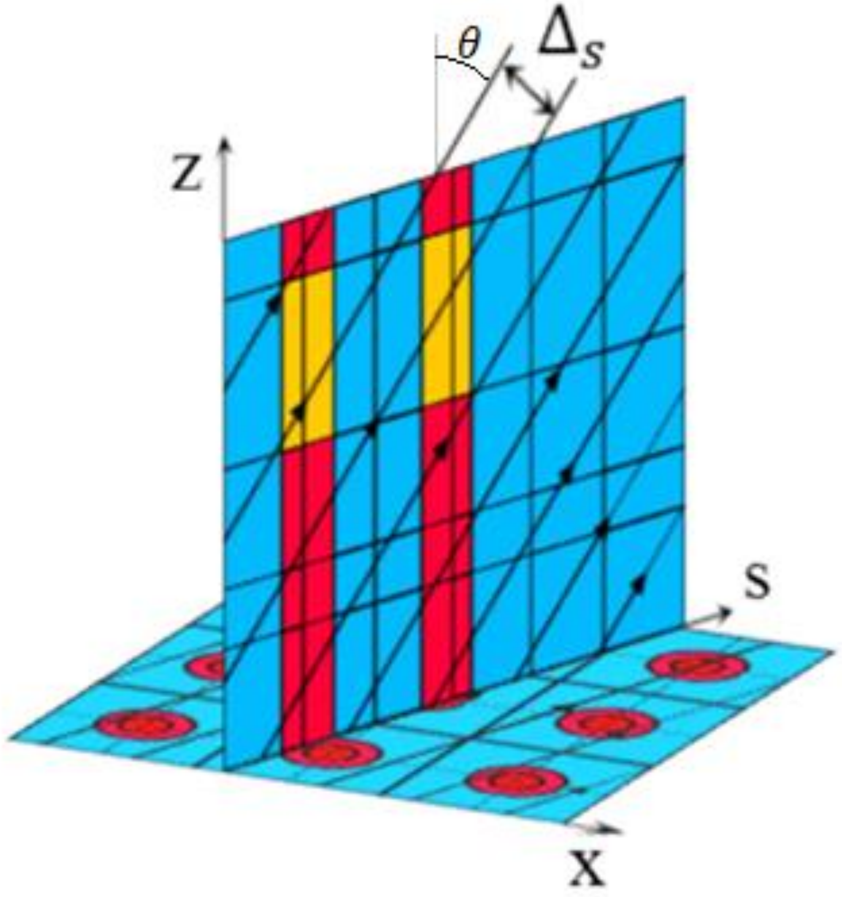


3D Solution: This extrusion forms **SZ planes** with a **cartesian mesh**, where characteristics are drawn using the same criteria as in 2D, effectively reducing the **3D problem into a set of 2D transport calculations**.

Image: Daniele Sciannandrone

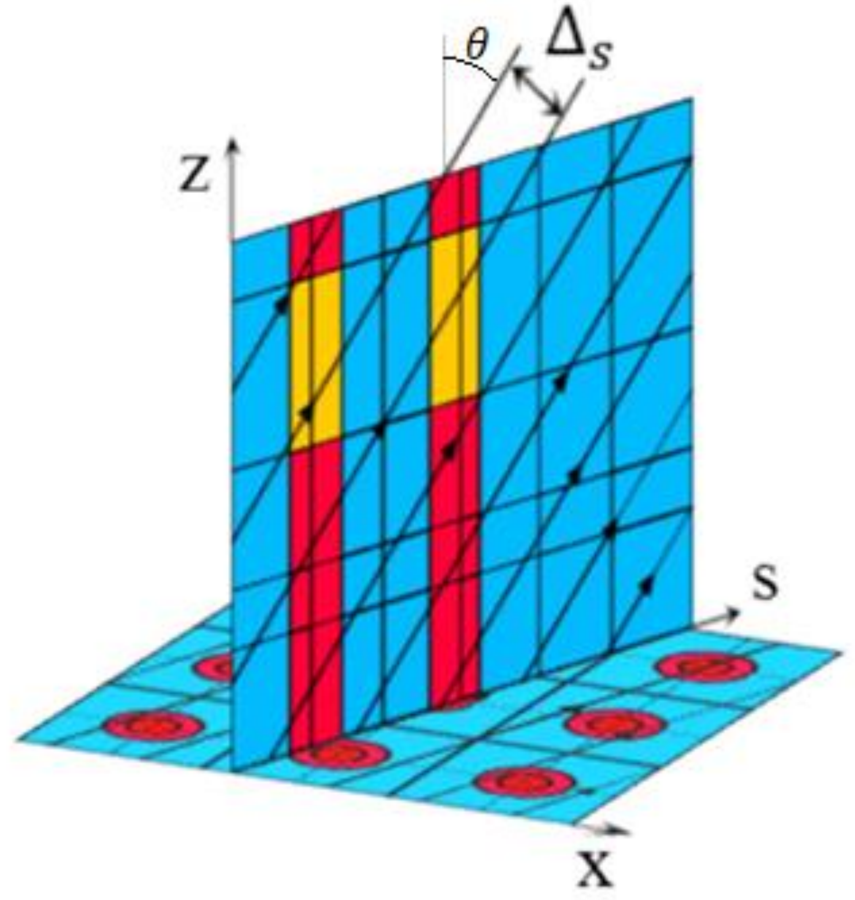
GPU Strategy *– parallelization over 3D trajectories*

- Previous strategy (OpenMP):
  - parallelism across 2D trajectories + angles θ
    → limited scalability
- New CUDA strategy:
  - 1 GPU thread ≡ 1 3D trajectory
    → fine-grained, massive parallelism
  - Blocks of 1024 threads: all trajectories with the same 2D origin and θ processed together
- Grouping 3D trajs from the same 2D traj:
  - Reduces warp divergence (geometrically similar trajectories)
- Efficiency depends on the integration step (3D-trajs/sz-plane)



CPU/GPU Strategy – *Kernel structure*

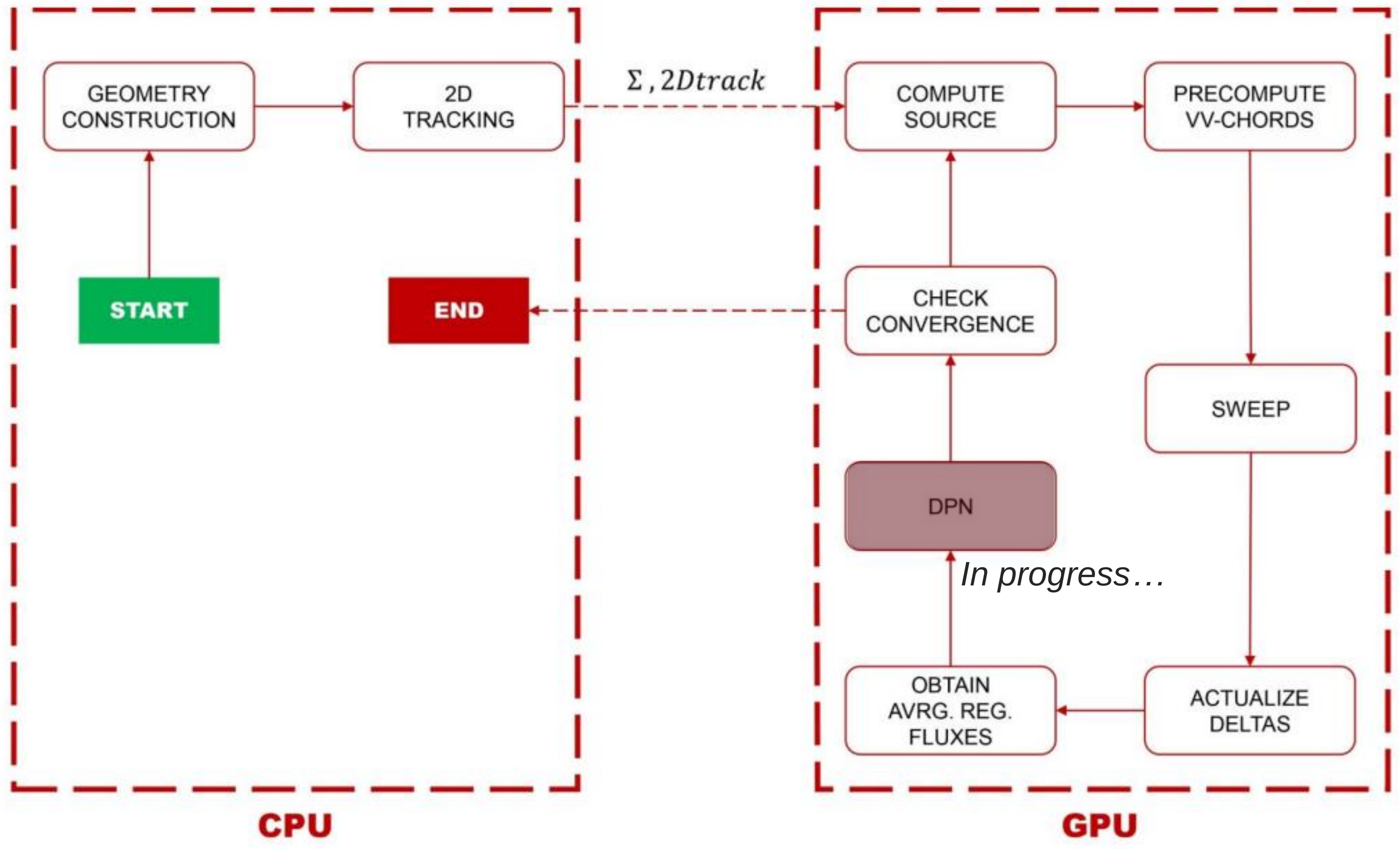
```
ROUTINE SWEEP KERNEL(traj2D, teta)
    IdTraj = Idthread
    CALL LOAD_TRJDATA_PRIVATE
    FOR STEPS_FORWARD
        CALL GET_COEFFICIENTS
        CALL SWEEP_CHORD
        CALL ACTUALIZE_TRJ_DATA
    ENDFOR
    // EQUIVALENT FOR BACKWARDS
END ROUTINE
```



CPU–GPU Workflow

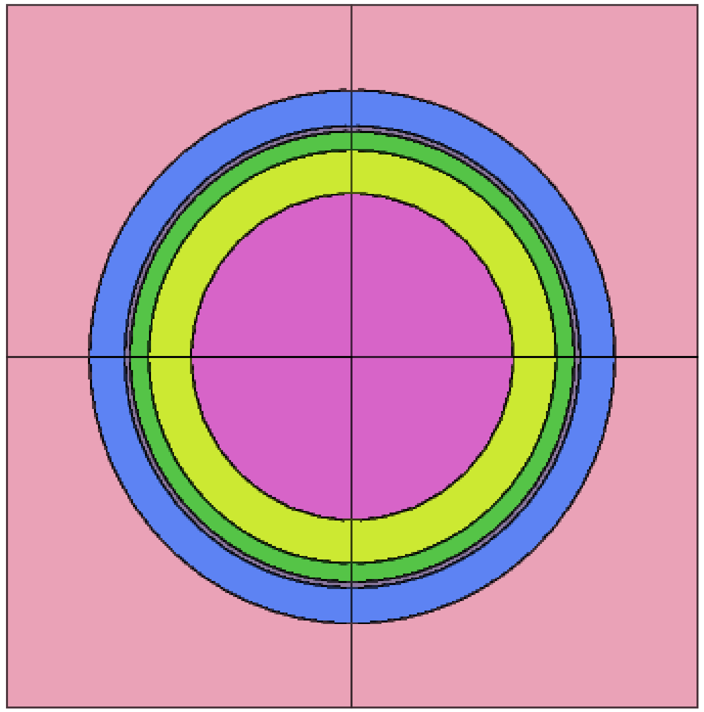
The whole inner iteration now runs entirely inside the GPU. **MAIN GOAL**:

Eliminate CPU↔GPU data shuttling, cutting latency and unlocking the GPU's full parallel power.



3. Results

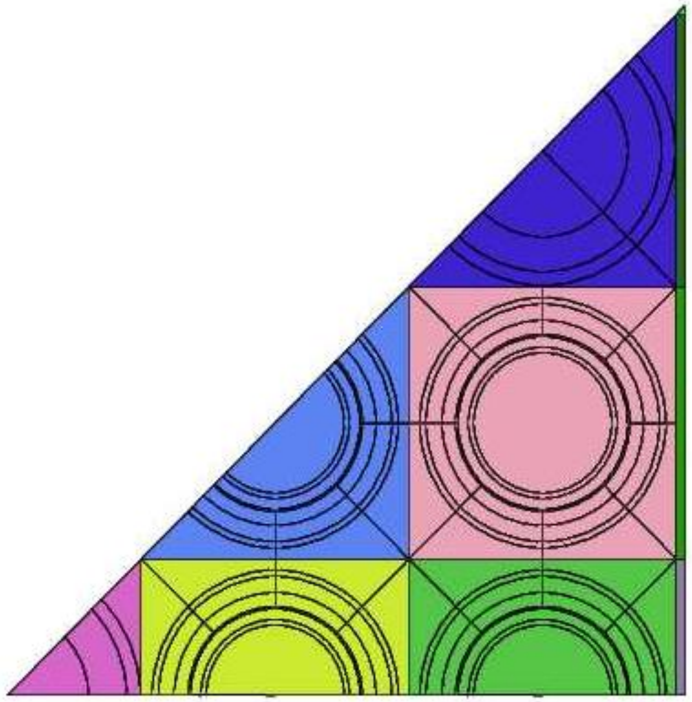
Test Cases



CASE CELL 3D

(2 planes Z)

Used for constant flux representation

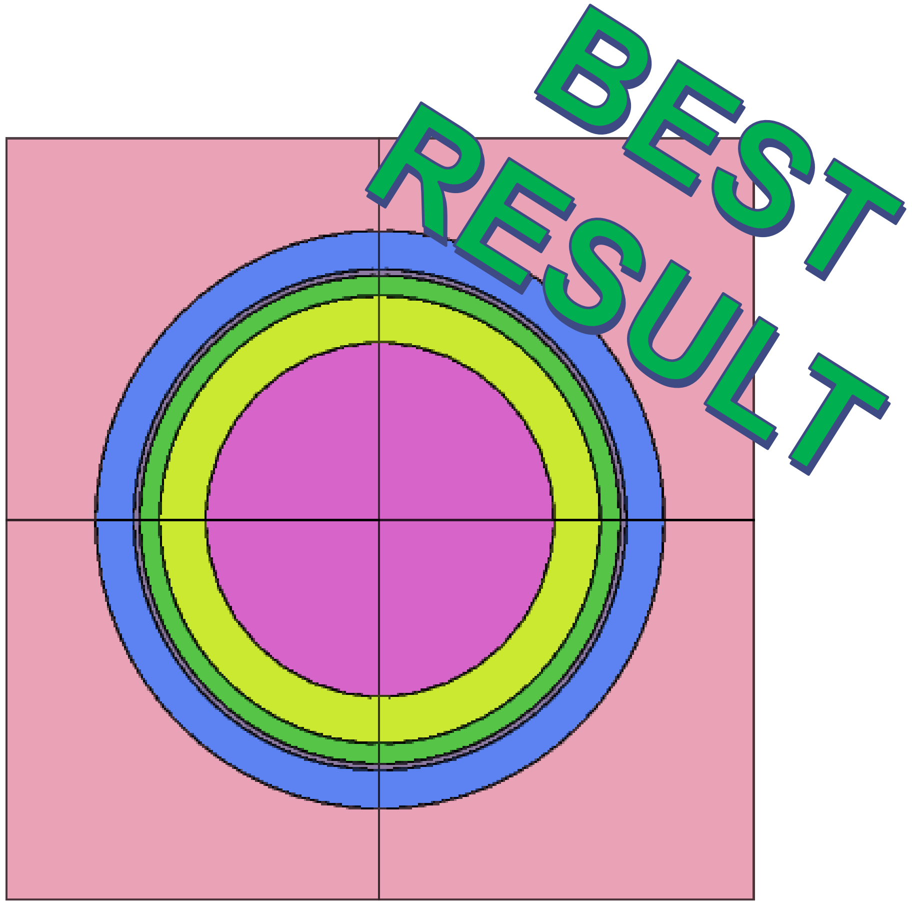


CASE 5x5 3D

(9 planes Z)

Used for polynomial flux representation

Before ...



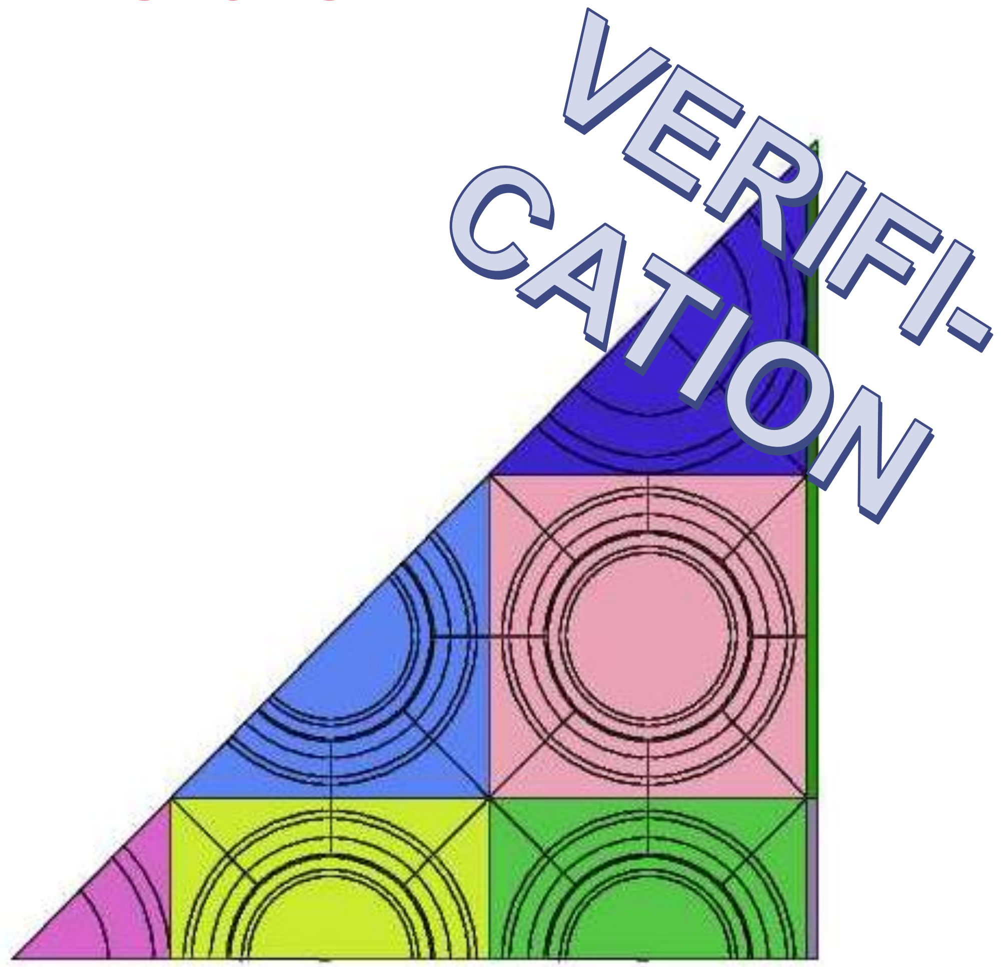
- **CASE CELL:**
  - **Constant flux representation**
  - **Some of the new optimizations not considered**

- CPU:
  - Time: 15329.22 seconds

- GPU:
  - Time: 859.35 seconds

x17,84

Before ...



- **CASE 5X5:**
  - **Polynomial flux representation**
  - **Verification to check convergence**

- CPU:

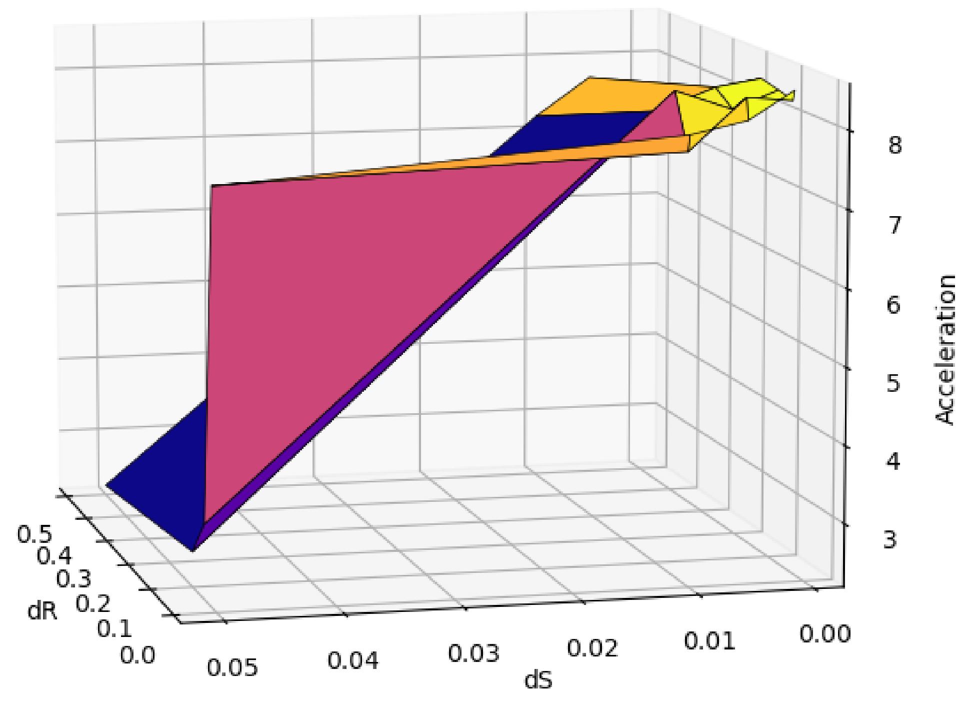
Time:	4294.87 seconds
Iterations	32473
Eigenvalue:	6,155393E-01

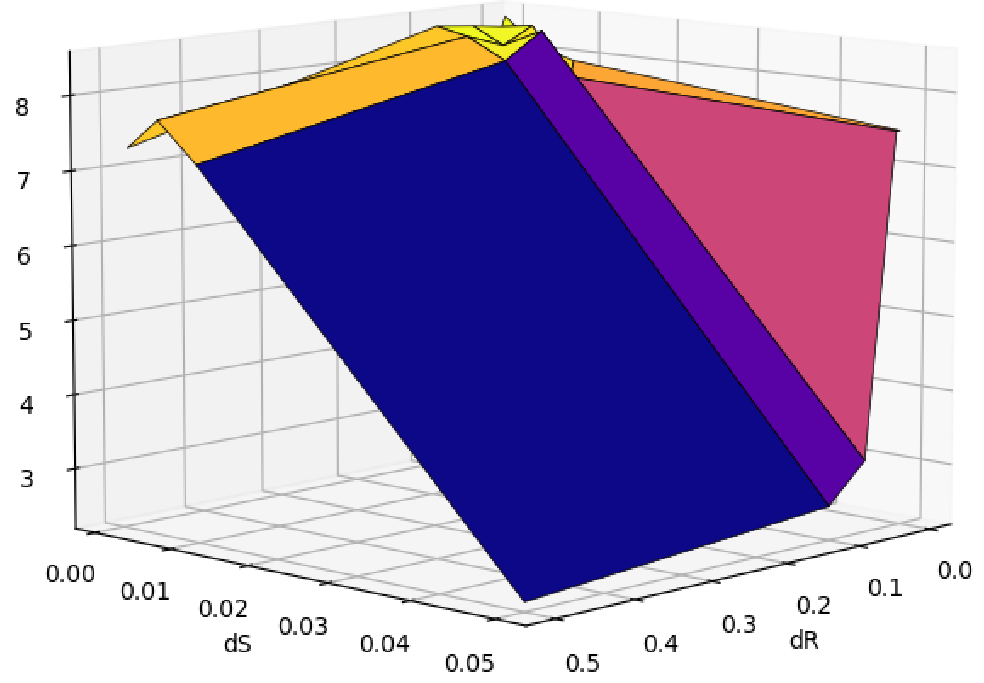
- GPU:

Time:	1857.21 seconds
Iterations	32653
Eigenvalue:	6,155379E-01

Physics preserved: GPU reproduces CPU results to < 0.3 pcm and 1e-4 flux error.

Results “Polynomial flux” *– Case 5x5*

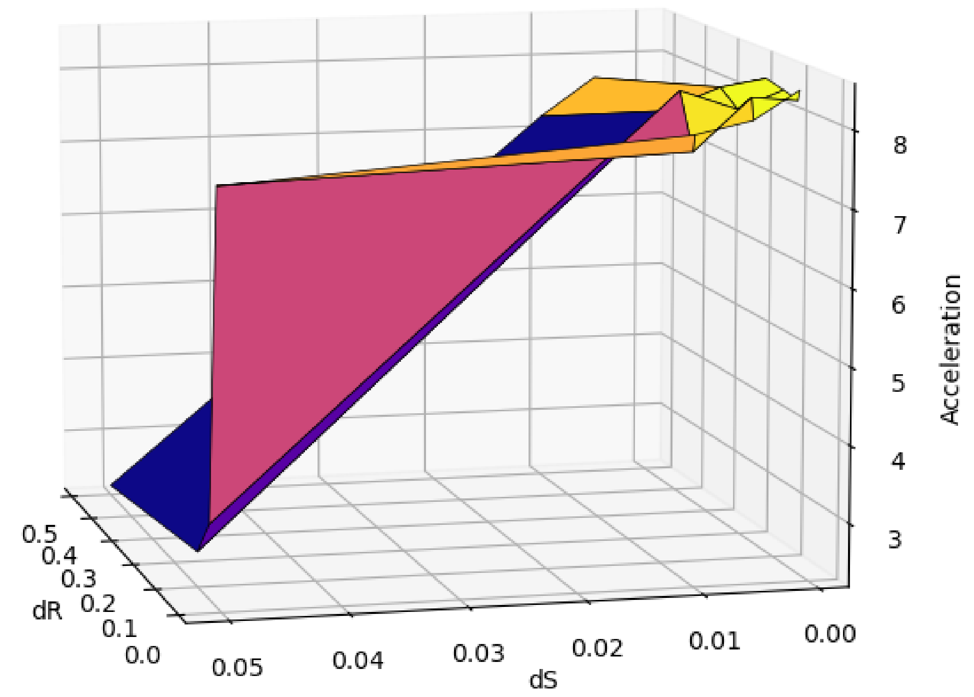


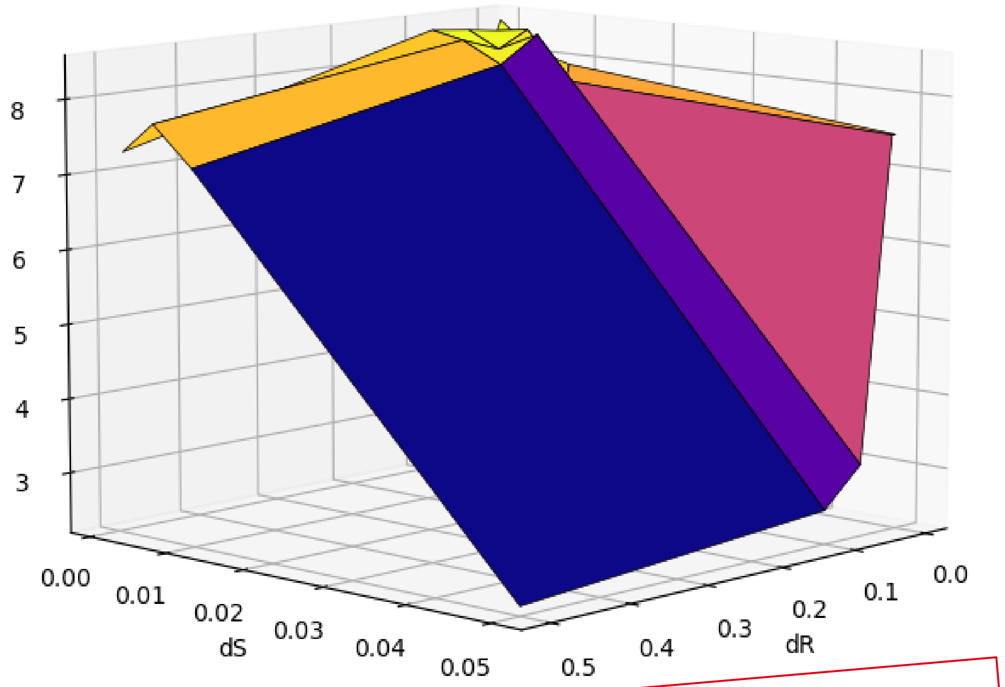


Results are shown for only one complete sweep

The GPU performs the sweeping >8 times faster

Results “Polynomial flux” *– Case 5x5*





13 times faster

ping >6 times faster

4. Conclusion

Conclusion

- **3D track-level parallelization works.**
  By assigning **one CUDA thread per 3-D traj.** and grouping tracks that share the same 2-D source point plus polar angle θ, warp divergence is reduced and the massive parallelism inherent in the MOC is fully exploited. The entire inner loop now runs on the GPU, reducing CPU ↔ GPU transfers.

- **Very significant reductions in compute time with no loss of accuracy.**
  - **CELL case (constant flux):** 18 ×
  - **5 × 5 case (polynomial flux):** 8,5 ×

  In both cases, **λ / k-eff remains equal.**

- **The GPU memory hierarchy is critical.**
  Proper use of shared memory/L1 versus global memory, together with minimizing atomic operations, accounts for much of the observed speed-up.

- **HPC feasibility demonstrated for Apollo3-TDT in 3-D.**
  With a single GPU, performance matches or surpasses the CPU OpenMP setup, paving the way for multi-GPU scaling and full-core simulations in reasonable runtimes.

Future work…

- PROFILING AND OPTIMIZING VIA PROFILER.
- PRECOMPUTE LONG TRAJECTORIES.

  Some recent papers show the possibility of precomputing and storing some 3D trajs to gain around 1,25 x .

- DPN_ACCELLERATION on GPU – *high expectations!*
- WORK ON SCHEDULING
- MPI + CUDA Fortran ? → MPS (multiples CPU threads)

  "MPS is a runtime service designed to let multiple MPI processes using CUDA to run concurrently on a single or multiples GPUs"

Merci

CEA SACLAY

91191 Gif-sur-Yvette Cedex

France

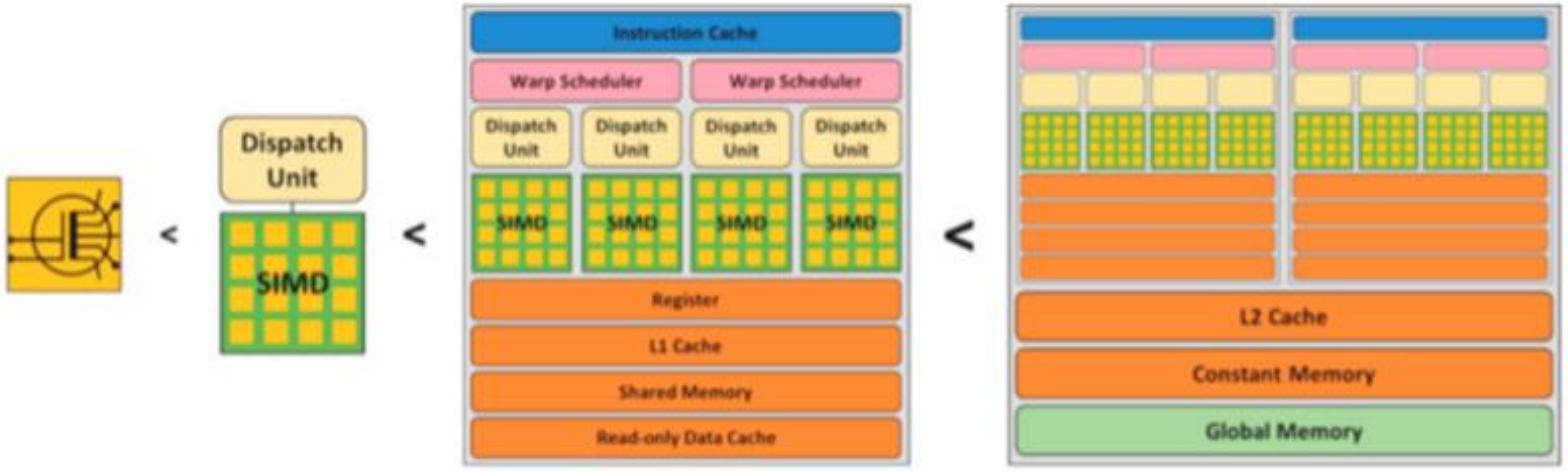
ignacio-martin.garrido@cea.fr

X. EXTRAS

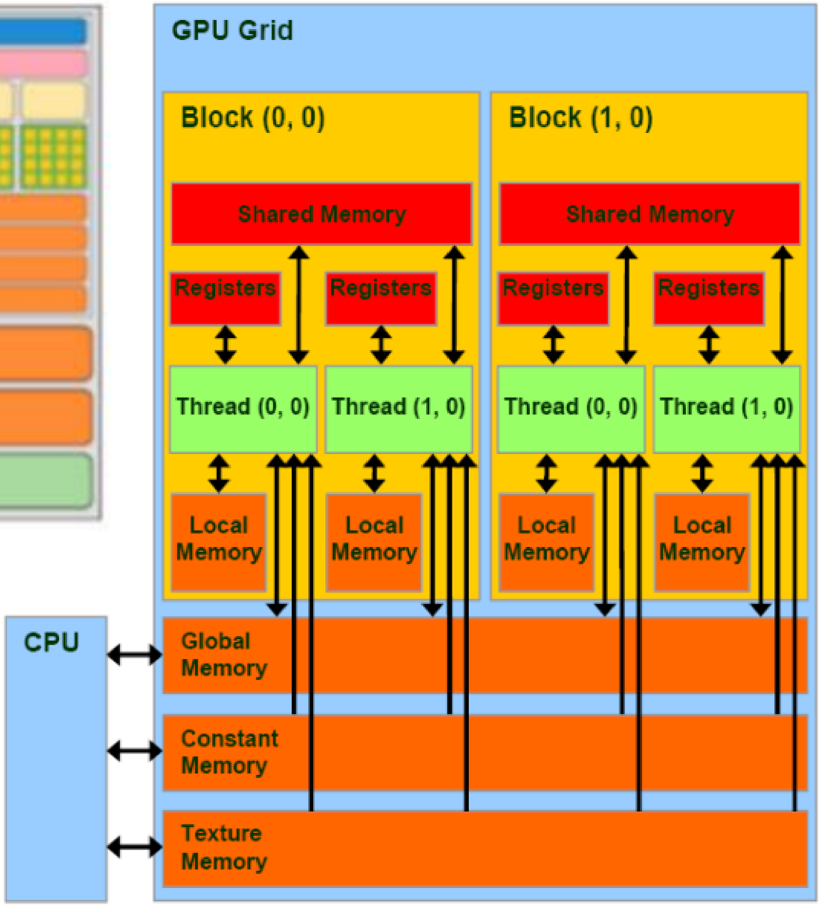
GPU Microarchitecture

The GPU microarchitecture is a hierarchical structure with global, shared, and CUDA core scales, each providing different resources for parallel computation.

HARDWARE

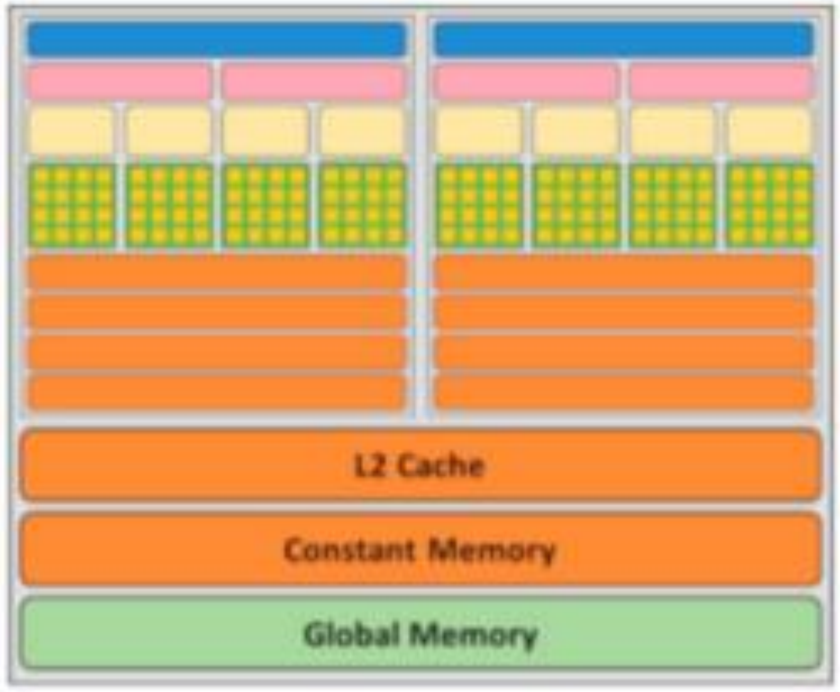


SOFTWARE



Global Scale

Global Scale refers to the GPU as a single unit of processing. All resources mentioned at this level are accessible to every core within the GPU.



(d)

Global Memory:
- **Size:** Several gigabytes
- **Max Transfer Speed:** Approximately 200 GB/s

Constant Memory:
- **Size:** Typically 64 KB
- **Max Transfer Speed:** Approximately 2000 GB/s

L2 Cache:
- **Size:** Varies between 200 KB and 6000 KB
- **Max Transfer Speed:** Approximately 2000 GB/s

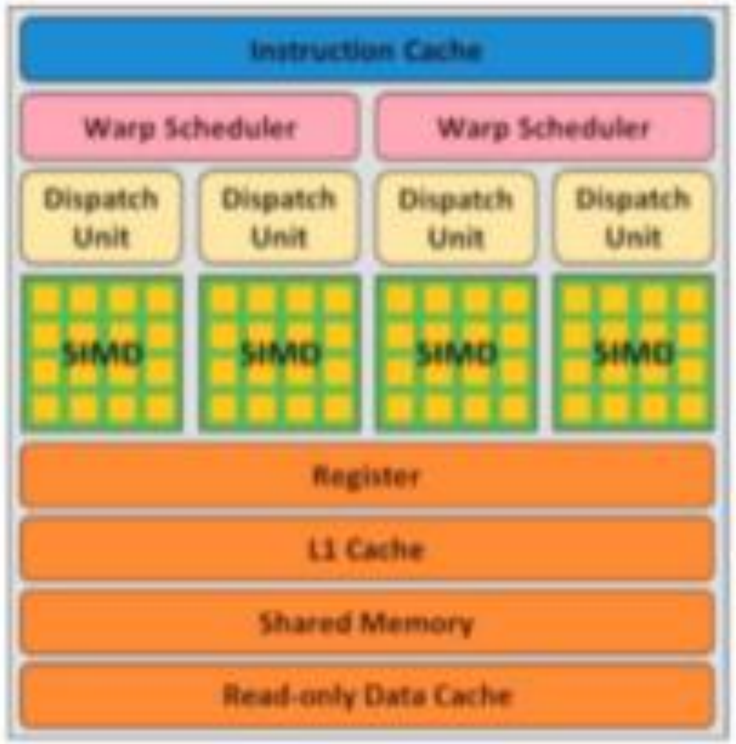
Texture Memory:
- **Size:** Typically 48 KB
- **Max Transfer Speed:** Approximately 3000 GB/s

Control Units:
- Registers
- Memory Controllers
- Output Units
- Other essential components

Streaming Multiprocessors (SMs):
- Independent units that can execute different tasks concurrently.
- Each SM leads to the next scale: the **SM Scale**.

Shared Scale

Shared Scale refers to a Streaming Multiprocessor (SM). All resources mentioned at this level are accessible to every core within the SM.



(c)

Shared Memory: *Programmable* Can be allocated and managed by the programmer.
- **Size:** Typically 96 KB (shared with L1 cache)**Max Transfer Speed:** Approximately 1 TB/s to 2 TB/s
- **Accessibility:** Accessible by all threads within a block.

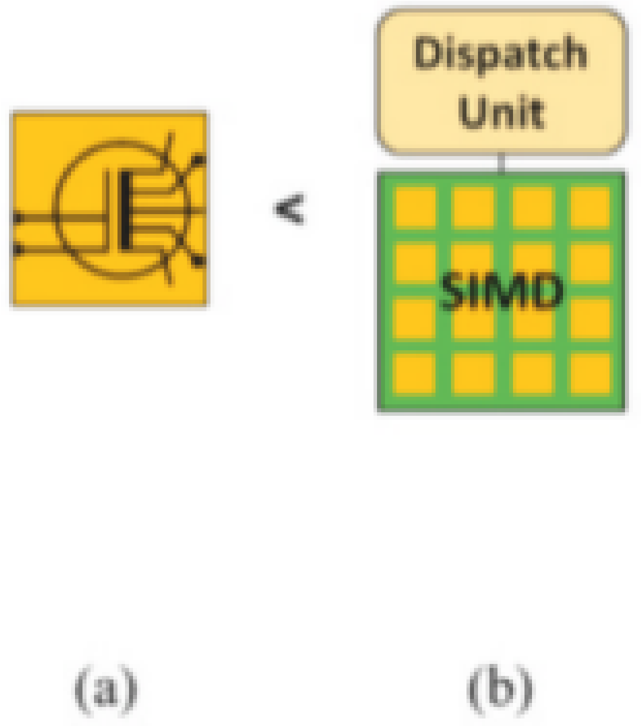
L1 Cache: *Non-Programmable* Automatically managed by the hardware.
- **Size:** Typically 96 KB (shared with shared memory)
- **Max Transfer Speed:** Approximately 1.5 TB/s to 3 TB/s

Control Units
- Registers
- Memory Controllers
- Scheduler Units
- Other essential components

Warps
- **Definition:** A warp is a cluster of 32 threads that must execute the same instructions simultaneously.
- **Number:** Varies depending on the architecture, but 32 warps per SM is common.

CUDA Core Scale

CUDA Core Scale refers to a single CUDA core. Each core runs one thread, and the resources mentioned at this level are accessible only by that core.



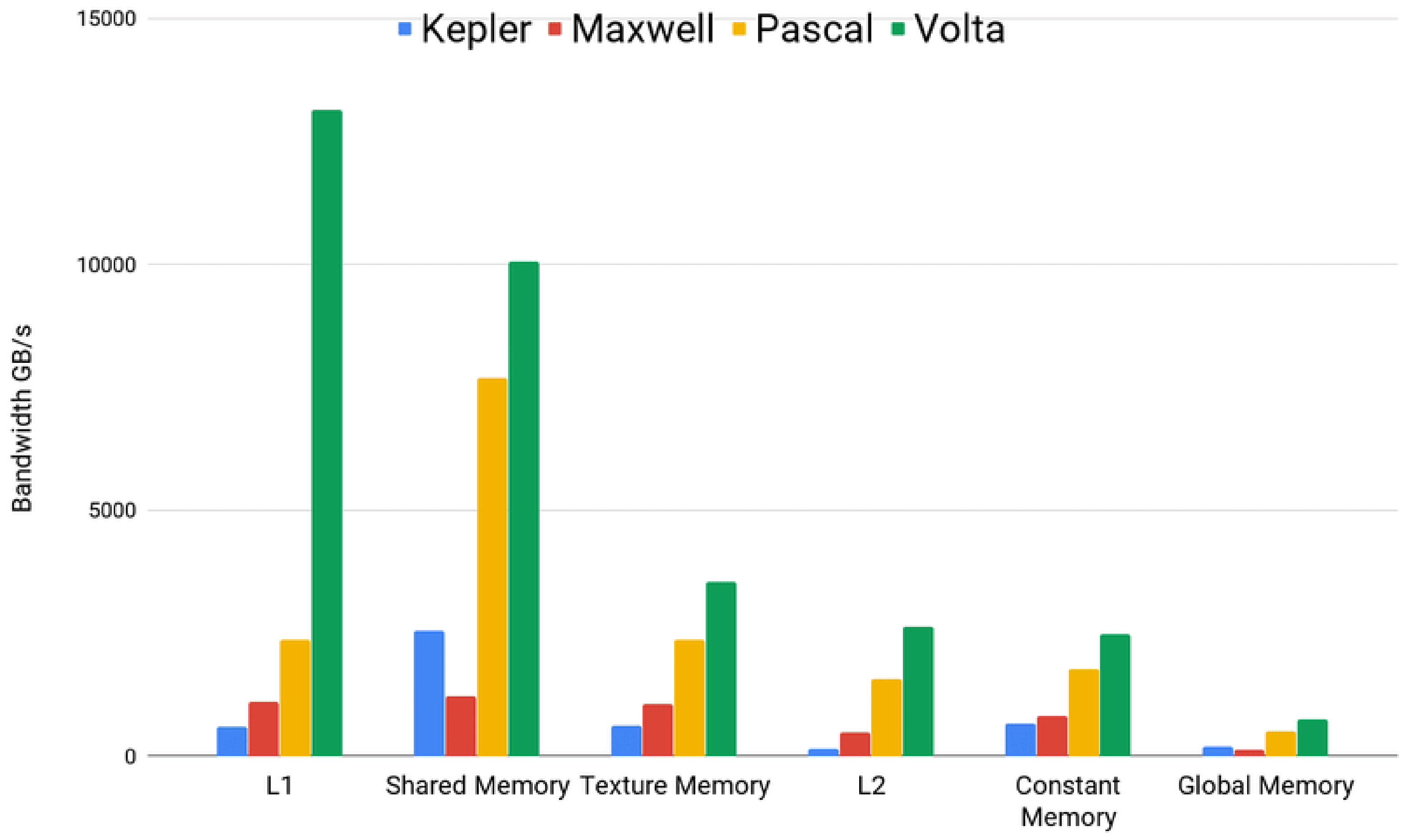
Local Memory: *Non-Programmable* Automatically managed by the hardware.
- **Function:** Serves as the core's cache.
- **Overflow:** If local memory is overloaded, it uses reserved space in global memory.

ALU (Arithmetic Logic Unit):
- **Purpose:** Performs arithmetic and logical operations.

Control Units
- Other essential components
- Registers
- Flow Control Units

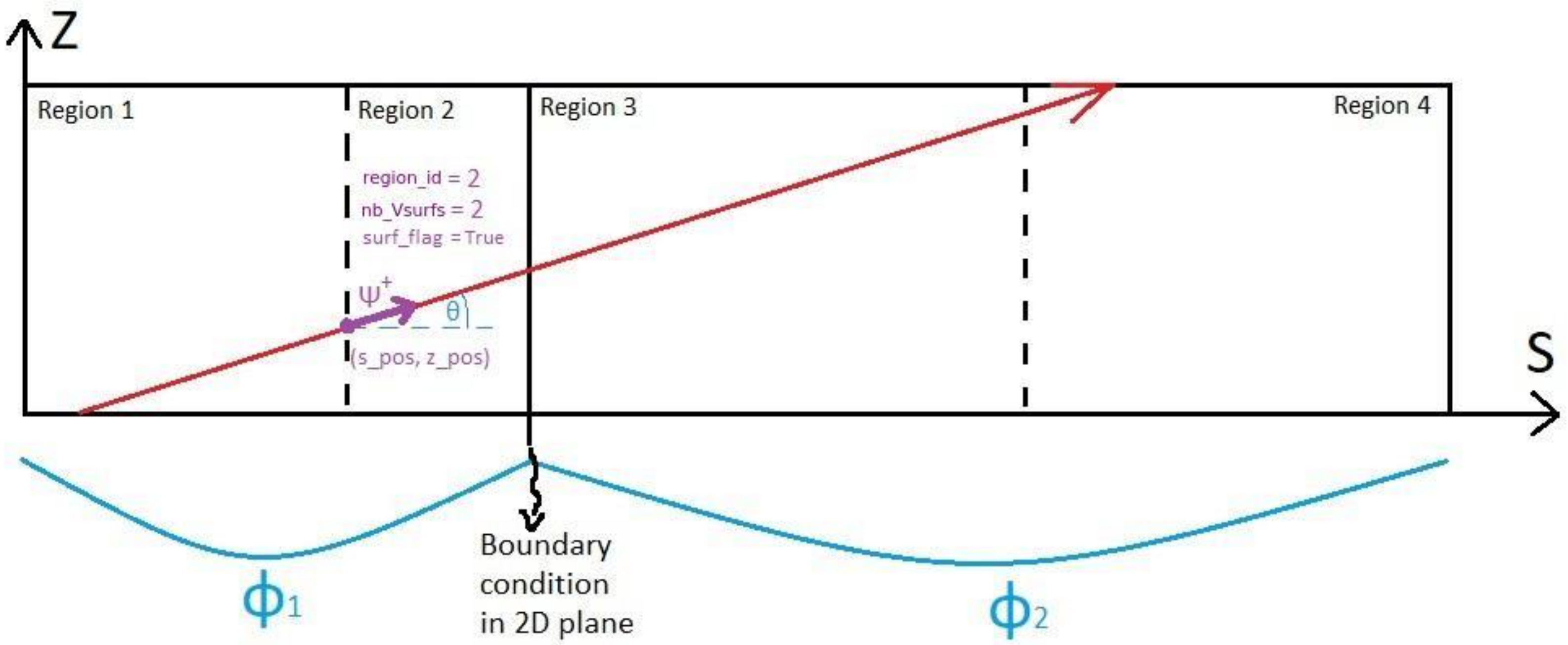
Memory Bandwidth Comparison

Shared memory and L1 cache offer significantly higher bandwidth than global memory. Understanding these differences is crucial for optimization.



TRAJECTORY type

```
TYPE TRAJECTORY
  INTEGER                                      :: nb_vsurfs, region_id, &
                                                  chordnum, sign_for_backw, &
                                                  HSS_pos
  LOGICAL                                      :: surf_flag
  double precision                             :: s_inic
  double precision                             :: flux
END TYPE TRAJECTORY
```



Why GPU and CUDA Fortran?

The MOC (Method of Long Characteristics) method excels in precision and versatility but demands significant computational resources, especially in 3D geometries where the number of unknowns can reach tens of billions.

01

PARALLELISM & SCALABILITY

MOC's independent calculations align well with GPUs' parallel architecture. GPUs can handle larger models and simulations efficiently

02

PERFORMANCE & EFFICIENCY

GPUs offer faster execution times for computationally intensive MOC simulations. GPUs can reduce power consumption while maintaining high performance.

03

SEAMLESS INTEGRATION

Natural extension to standard Fortran. Compatible with existing Fortran compilers and libraries.

04

GPU SPECIFIC FEATURES

Such as device memory, kernels, and synchronization primitives. Access to Tools and techniques for optimizing code.

Atomic Operations

PROS	CONS
Data consistency	Performance overhead
Simplified programming	Limited operations
Widely supported	Bottlenecks
Synchronization	Optimization

Atomic operations provide a mechanism for safely accessing and modifying shared memory variables in a concurrent environment. They guarantee that the operation is performed as a single, indivisible unit, preventing race conditions and ensuring data consistency.

Common Atomic Operations:

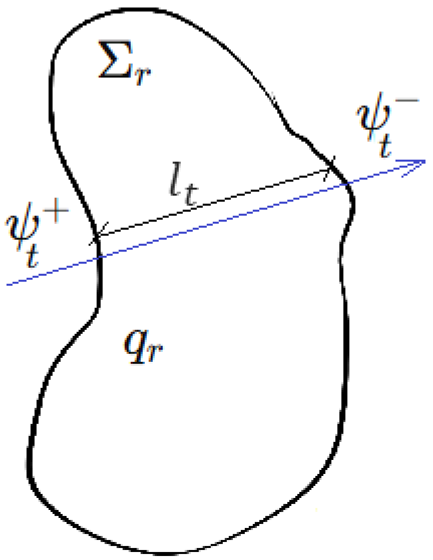
- **atomicAdd**: Adds a value to a shared variable atomically.
- **atomicSub**: Subtracts a value from a shared variable atomically.
- **atomicCAS**: Performs a compare-and-swap oper. on a shared variable.

MOC *– Linear Transport Equation and MOC Solution*

The **Method of Characteristics (MOC)** is a numerical technique used to solve the neutron transport equation by discretizing the geometry into a set of trajectories along which the transport equation is integrated.
The transport equation is **solved along each trajectory**. The **integral form of the transport equation** along a trajectory *t* is:

$$\psi^{+} = \psi^{-} e^{-\tau} + \frac{q_r}{\Sigma_r}\left(1 - e^{-\tau}\right)$$

where $\tau = \Sigma_r l_t$ is the optical length along the trajectory.



- ψ^{+} → **Outgoing angular flux** at the end of the trajectory segment.
- ψ^{-} → **Incoming angular flux** at the beginning of the trajectory segment.
- τ → **Optical thickness** of the trajectory segment, defined as:

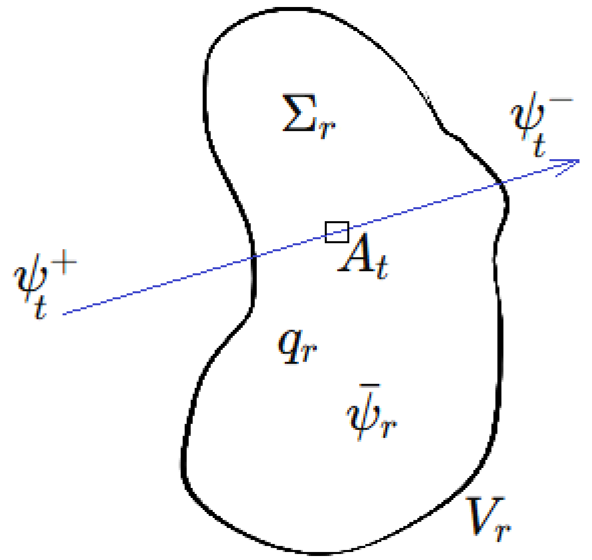
$$\tau = \Sigma_r l_t$$

- l_t → **Chord length**, the distance traveled by a neutron within region r along the trajectory.
- $e^{-\tau}$ → **Attenuation factor**, representing neutron absorption along the trajectory.
- $\frac{q_r}{\Sigma_r}(1 - e^{-\tau})$ → **Neutron source contribution** along the trajectory.

MOC – *Method of Characteristics (MOC) in Neutron Transport*

The results are averaged within each region to **update the flux** $\overline{\psi_r}$, completing the iterative scheme.
The neutron balance equation in a homogeneous region *r* is expressed as:

$$\bar{\psi}_r = \frac{q_r}{\Sigma_r} - \frac{1}{V_r \Sigma_r} \sum_{t \in \text{trajectories}} A_t (\psi^+ - \psi^-)_t$$

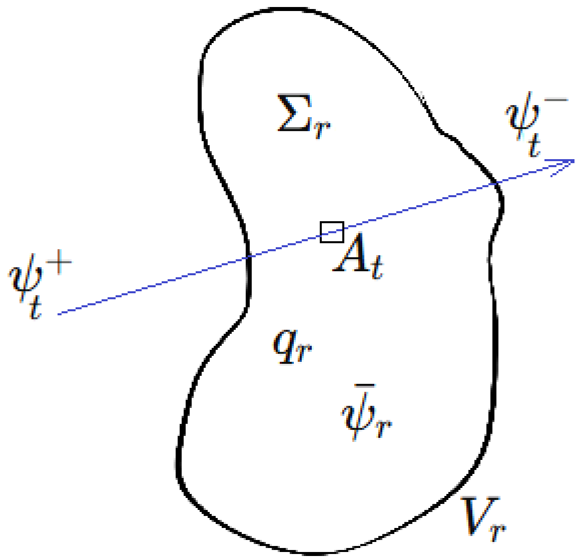


- $\bar{\psi}_r$ → **Region-averaged angular flux** in region r (neutron density per unit area, direction, and energy).
- q_r → **Total source term** in region r, including external sources and in-scattering contributions.
- Σ_r → **Macroscopic total cross-section** in region r, representing the probability of neutron interactions.
- V_r → **Volume of region** r.
- t → A specific **trajectory** crossing region r.
- A_t → **Cross-sectional area** associated with trajectory t.
- ψ^+ → **Outgoing angular flux** at the end of the trajectory segment within region r.
- ψ^- → **Incoming angular flux** at the beginning of the trajectory segment within region r.

MOC – *Method of Characteristics (MOC) in Neutron Transport*

The results are averaged within each region to **update the flux** $\overline{\psi_r}$, completing the iterative scheme.
The neutron balance equation in a homogeneous region *r* is expressed as:

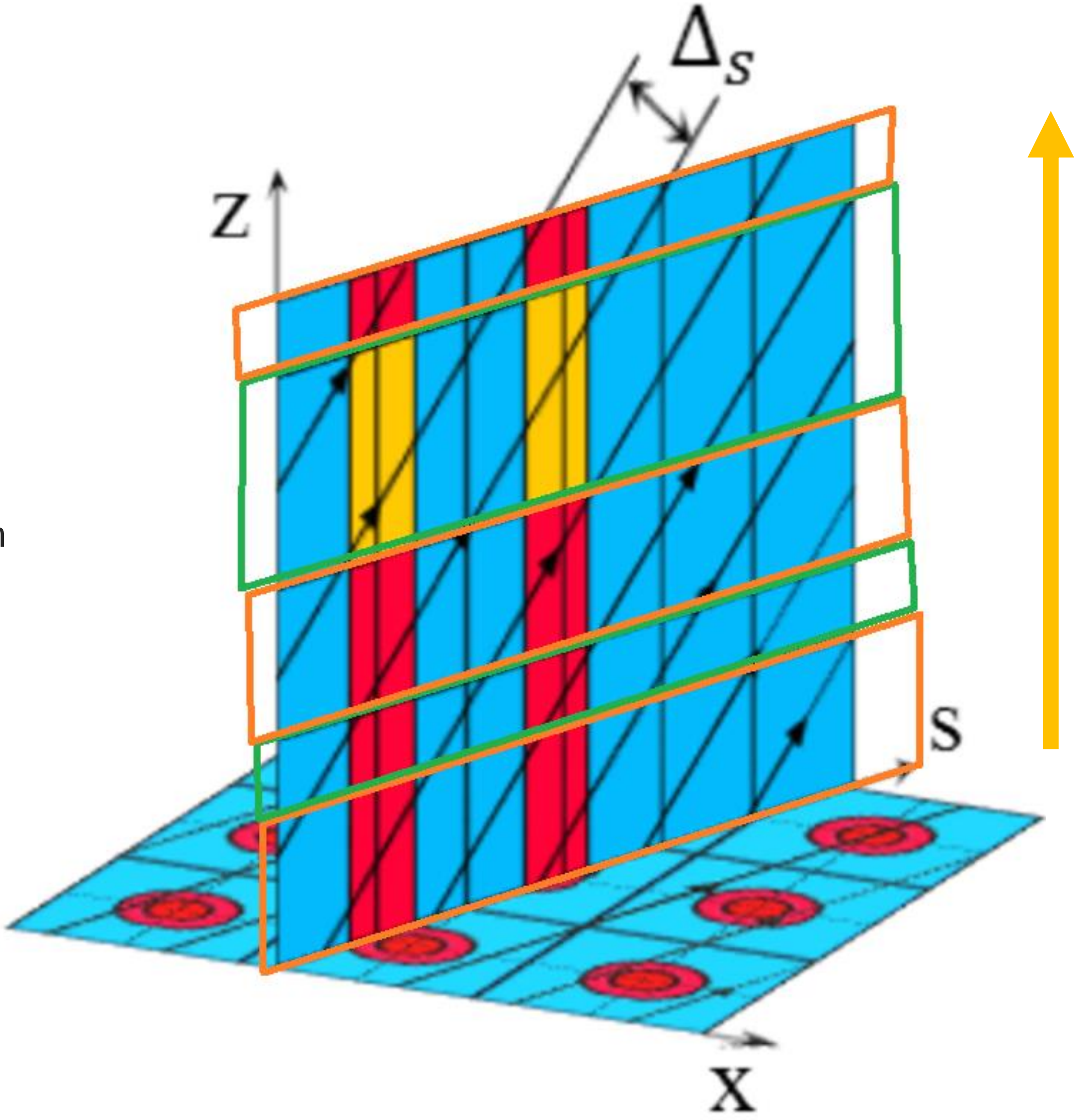
$$\bar{\psi}_r = \frac{q_r}{\Sigma_r} - \frac{1}{V_r \Sigma_r} \boxed{\sum_{t \in \text{trajectories}} A_t(\psi^+ - \psi^-)_t} \quad \textit{delta}$$



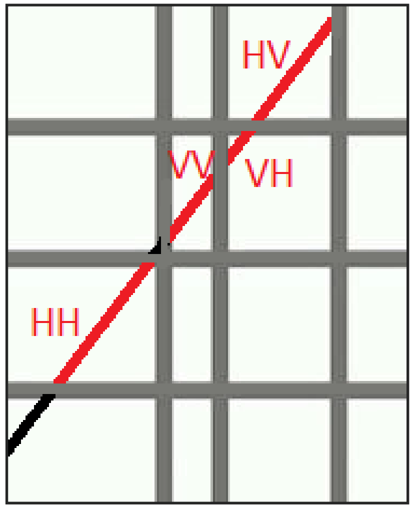
- $\bar{\psi}_r$ → **Region-averaged angular flux** in region r (neutron density per unit area, direction, and energy).
- q_r → **Total source term** in region r, including external sources and in-scattering contributions.
- Σ_r → **Macroscopic total cross-section** in region r, representing the probability of neutron interactions.
- V_r → **Volume of region** r.
- t → A specific **trajectory** crossing region r.
- A_t → **Cross-sectional area** associated with trajectory t.
- ψ^+ → **Outgoing angular flux** at the end of the trajectory segment within region r.
- ψ^- → **Incoming angular flux** at the beginning of the trajectory segment within region r.

GPU Strategy *– parallelization over 3D trajectories*

- Previous strategy (OpenMP):
  - parallelism across 2D trajectories + angles θ
    - → limited scalability
- New CUDA strategy:
  - 1 GPU thread ≡ 1 3D trajectory
    - → fine-grained, massive parallelism
  - Blocks of 1024 threads: all trajectories with the same 2D origin and θ processed together
- Grouping 3D trajs from the same 2D traj:
  - Reduces warp divergence (geometrically similar trajectories)
- Efficiency depends on the integration step (3D trajectories per sz plane)

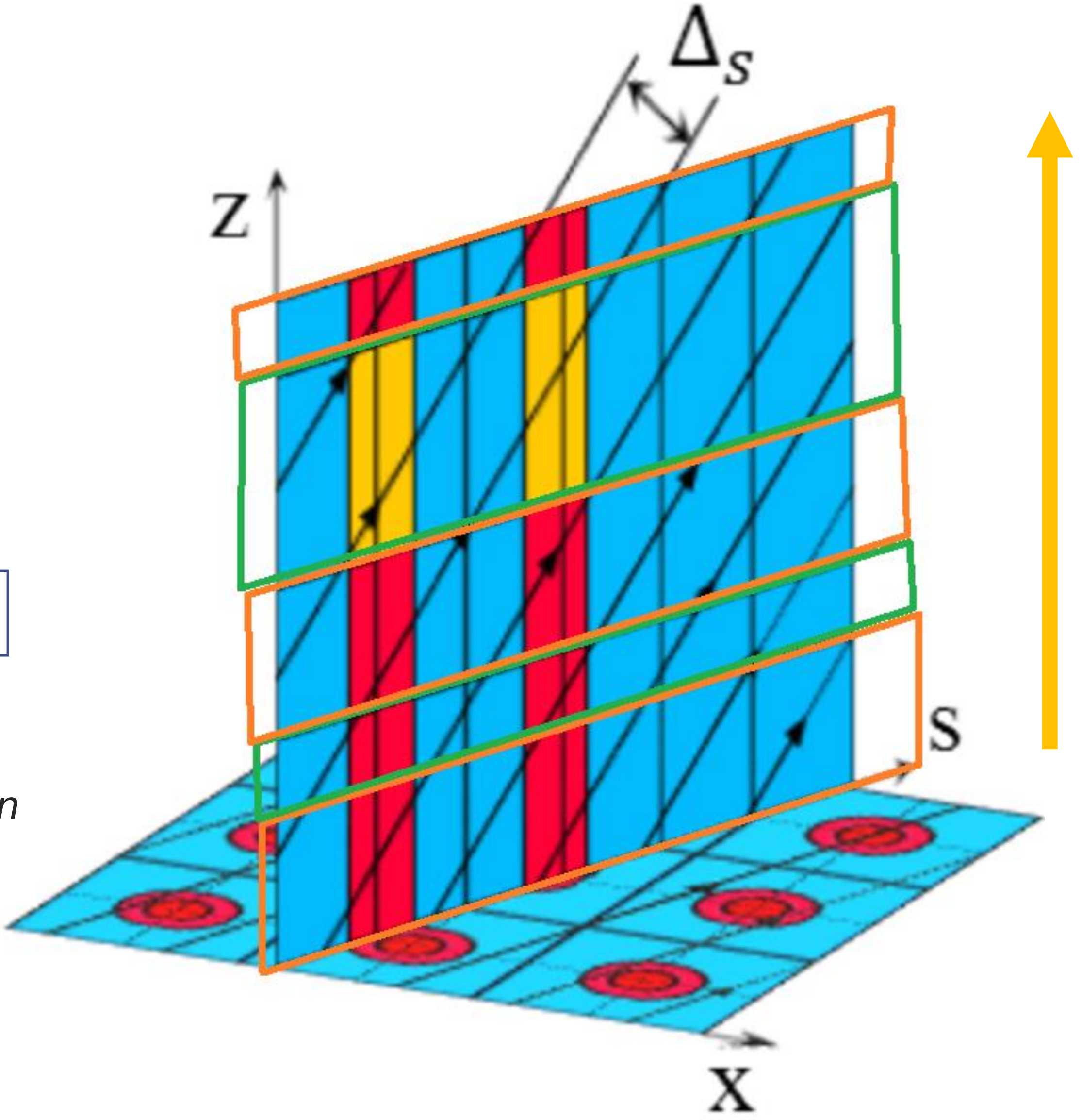


GPU Strategy *– parallelization over 3D trajectories*



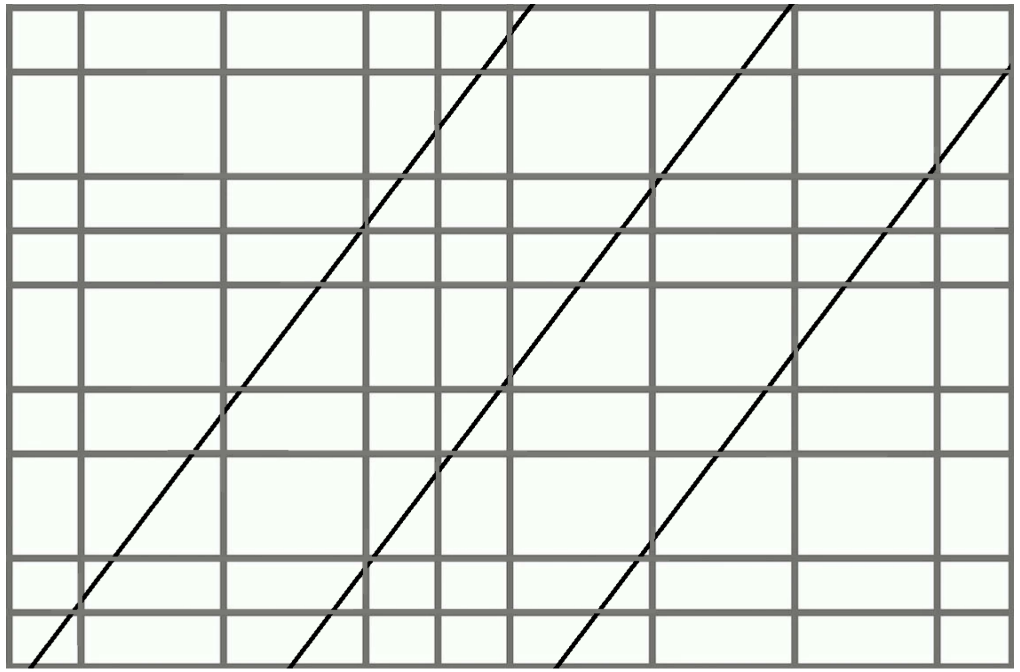
$$\begin{tikzcd}
 & \text{TRAJ. INIT} \arrow[dr] \arrow[ddl] & \\
 & & \text{HV} \arrow[dl] \arrow[d] \\
\text{HH} \arrow[dr] & \text{VH} \arrow[d] & \text{VV} \arrow[l] \arrow[loop right] \\
 & \text{TRAJ. END} &
\end{tikzcd}$$

Down-opened case + reflection

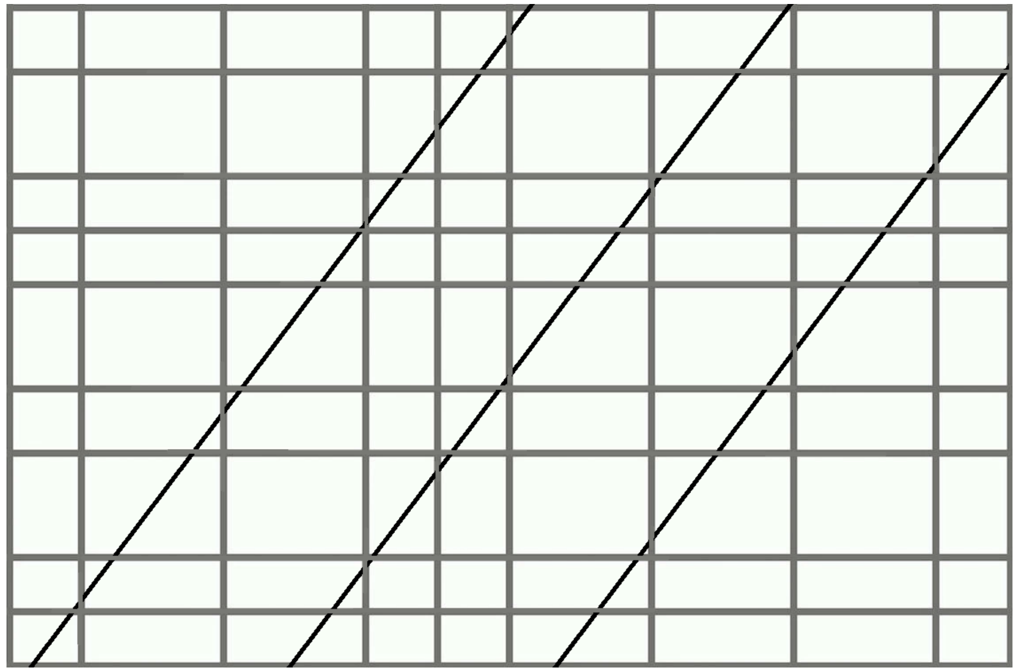


Strategies comparison

CPU

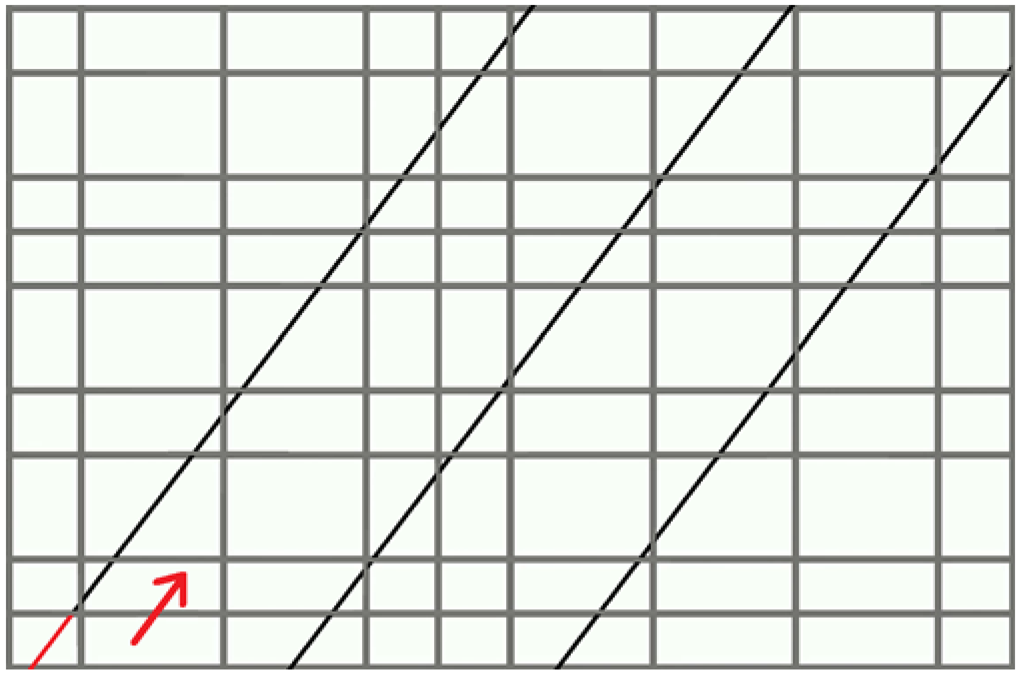


GPU

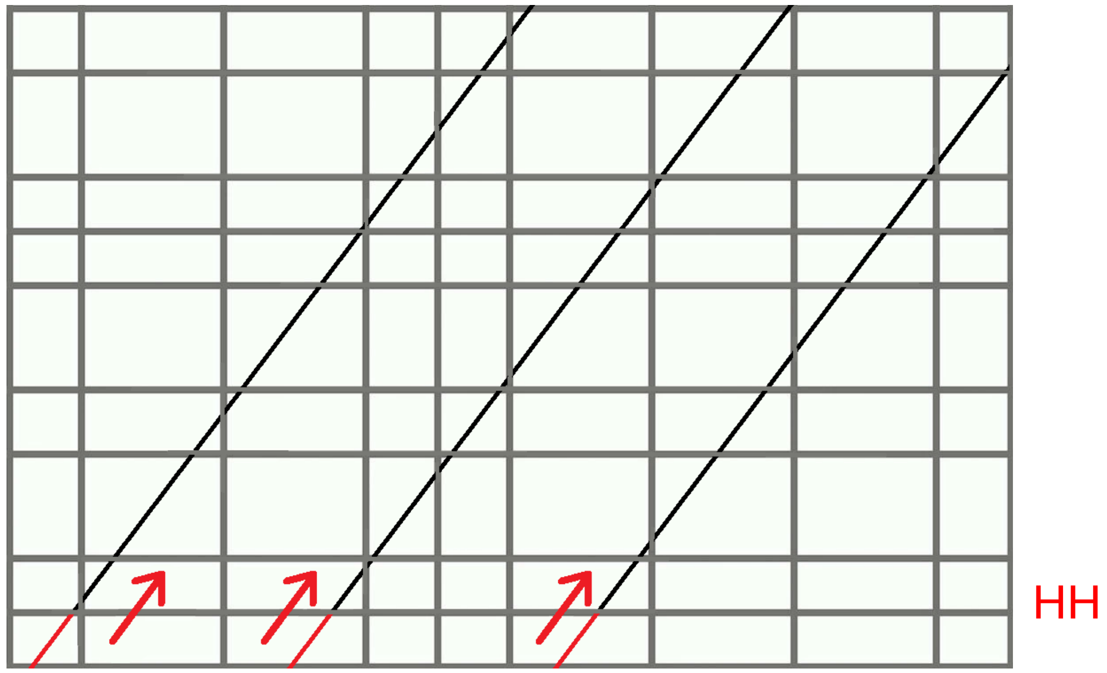


Strategies comparison

CPU

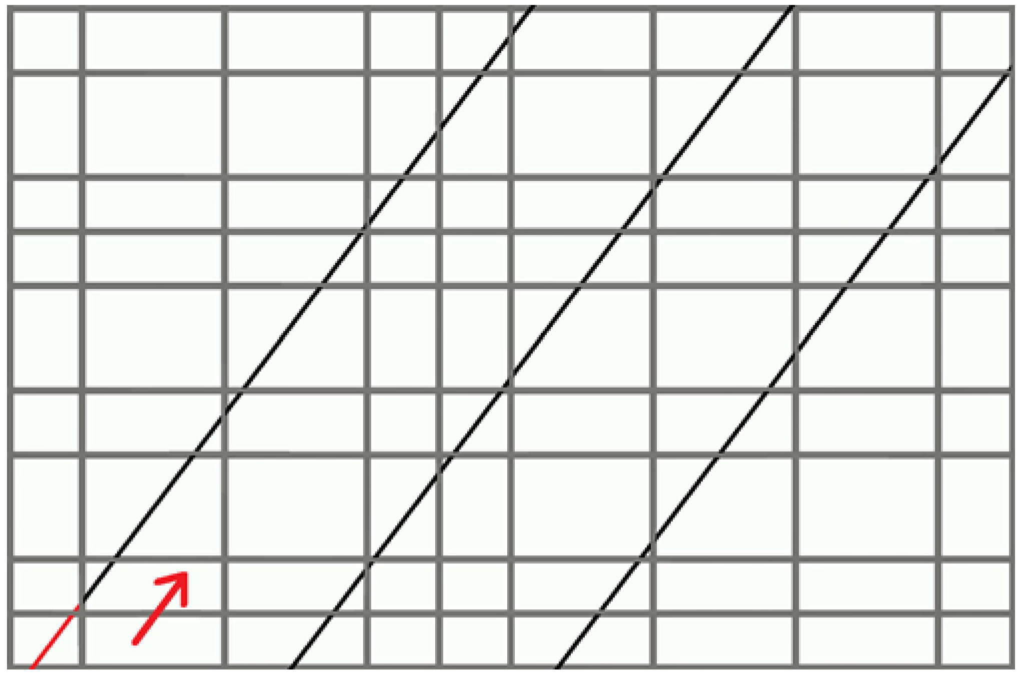


GPU

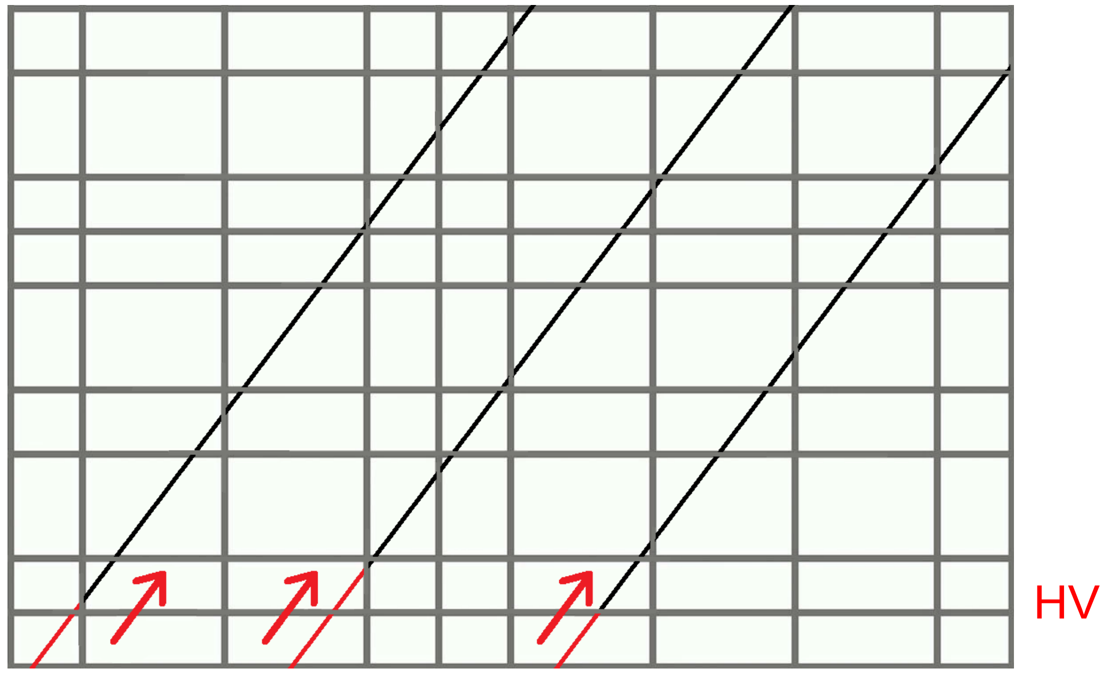


Strategies comparison

CPU

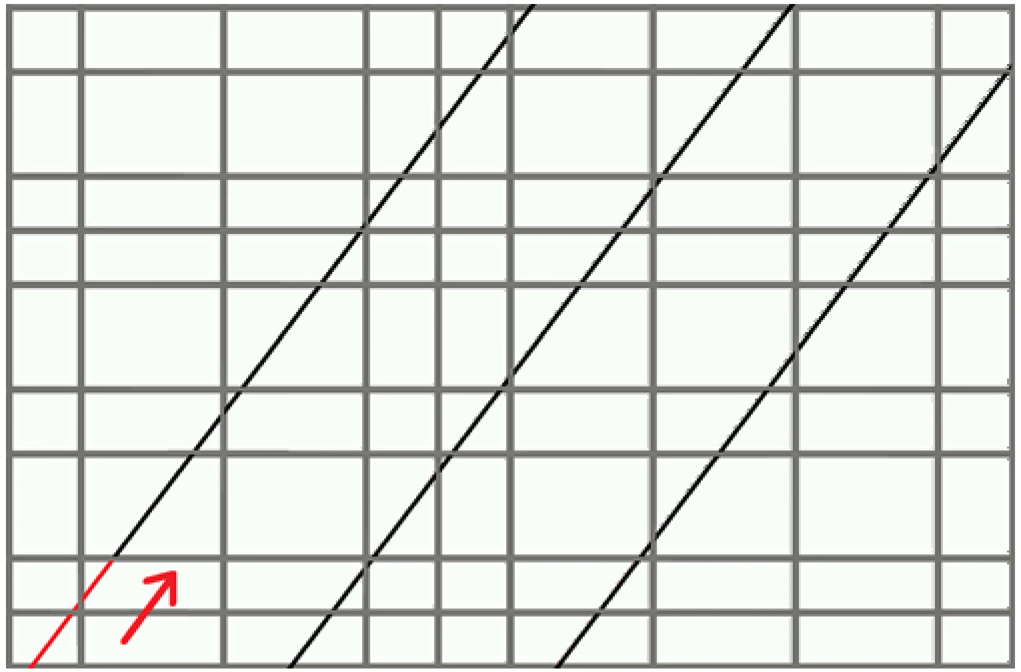


GPU

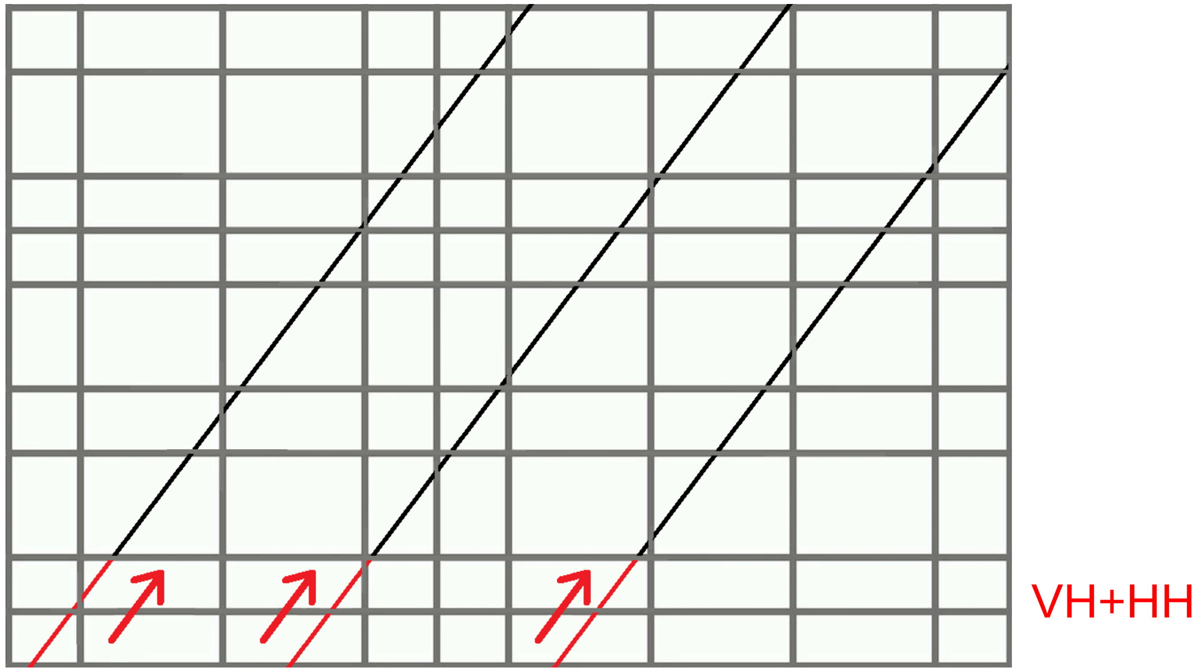


Strategies comparison

CPU

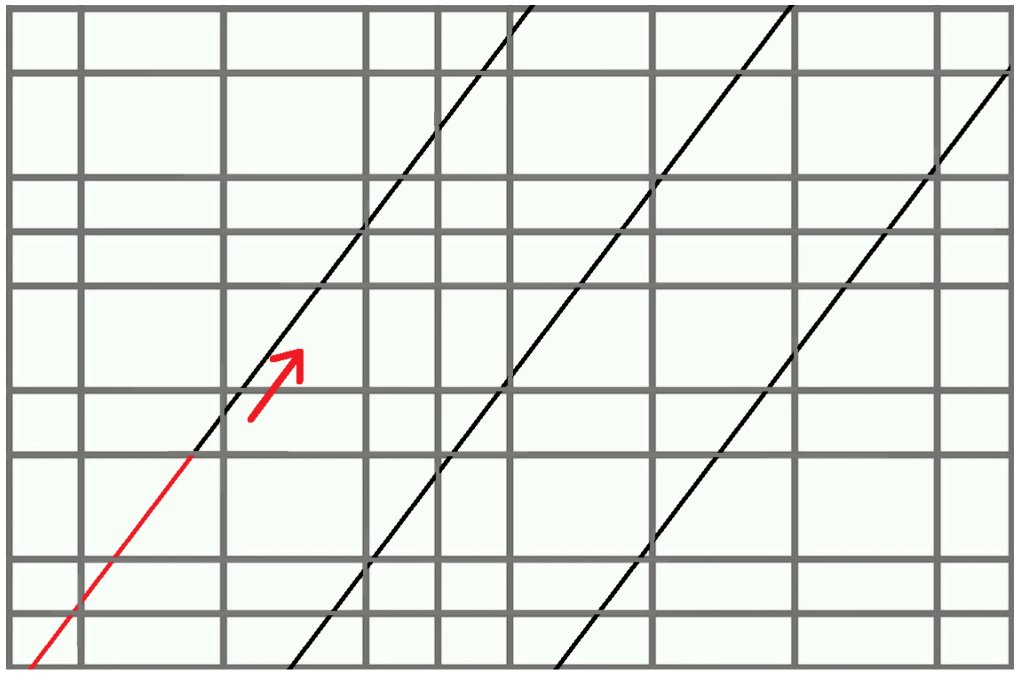


GPU



Strategies comparison

CPU



GPU