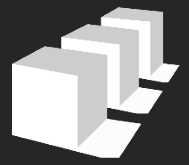
Universidad Zaragoza

1542



Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Instituto Universitario de Investigación en Ingeniería de Aragón Universidad Zaragoza

Advanced GPU programming

Mario Morales Hernández (mmorales@unizar.es)

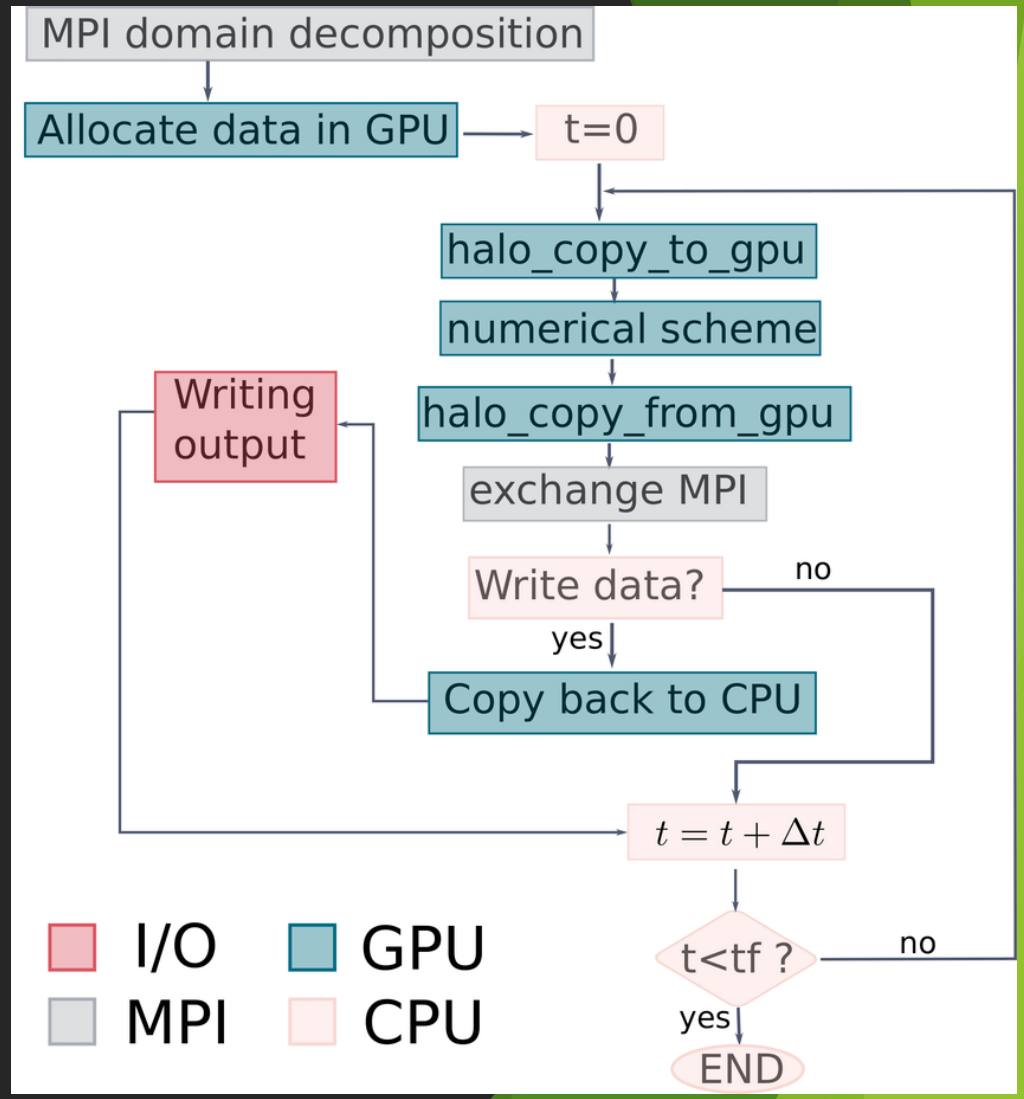
March19 2025

Outline

- Introduction
- Multi-GPU
  - MPI and communication
  - Domain decomposition
  - Scaling
  - Load balancing
- Profiling

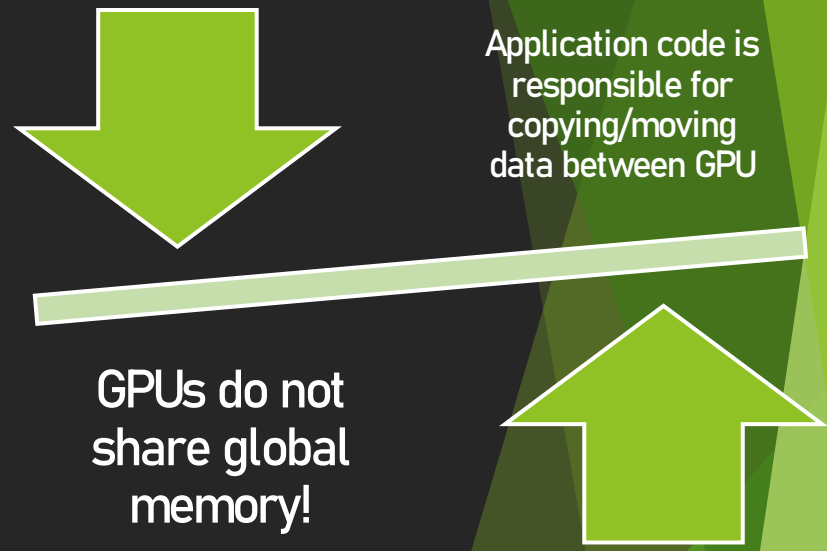
Introduction to Multi-GPU Programming

- Benefits of using multiple GPUs:
  - Increased computational power
  - Parallel processing for large-scale problems
- Challenges in multi-GPU programming:
  - Inter-GPU communication overhead
  - Load balancing and synchronization
  - Debugging and profiling complexities



MPI and communication

- **A problem of memory**
- **MPI** (Message Passing Interface) as a solution:
  - A standard for communication between distributed systems
  - Data transfer between GPUs using PCIe or NVLink
  - Synchronization: ensuring all GPUs work in harmony
  - Data sharing: efficiently distributing data across GPUs

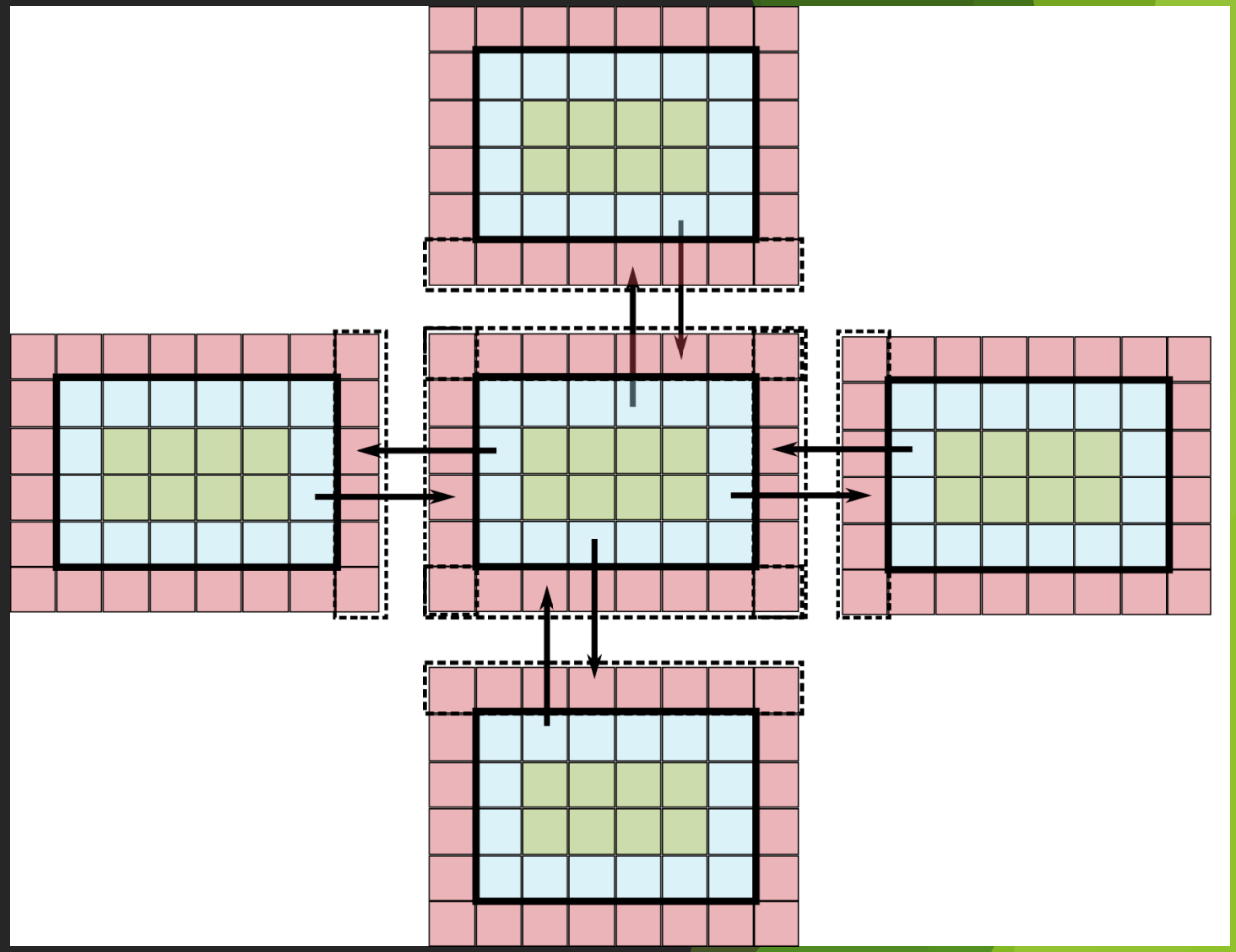


Domain decomposition

- Dividing a computational domain into smaller subdomains
- Each GPU processes a subdomain
- Types of decomposition:

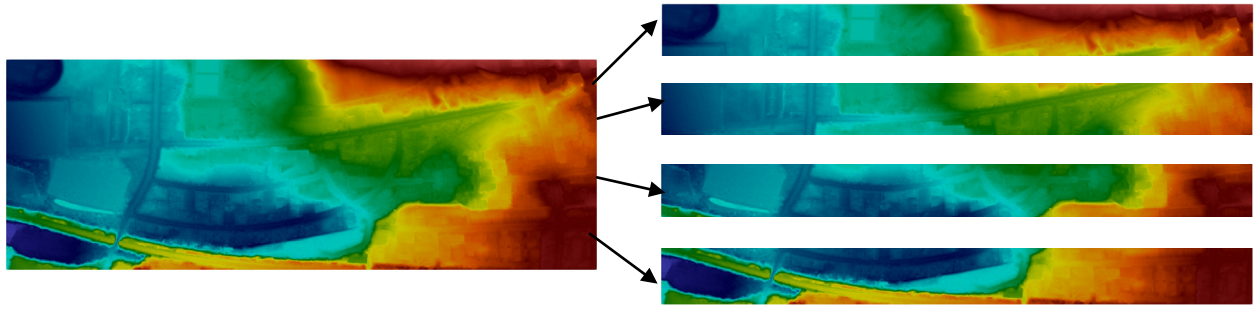
  - Spatial decomposition: Splitting the domain into spatial regions: can be 1D, 2D or 3D

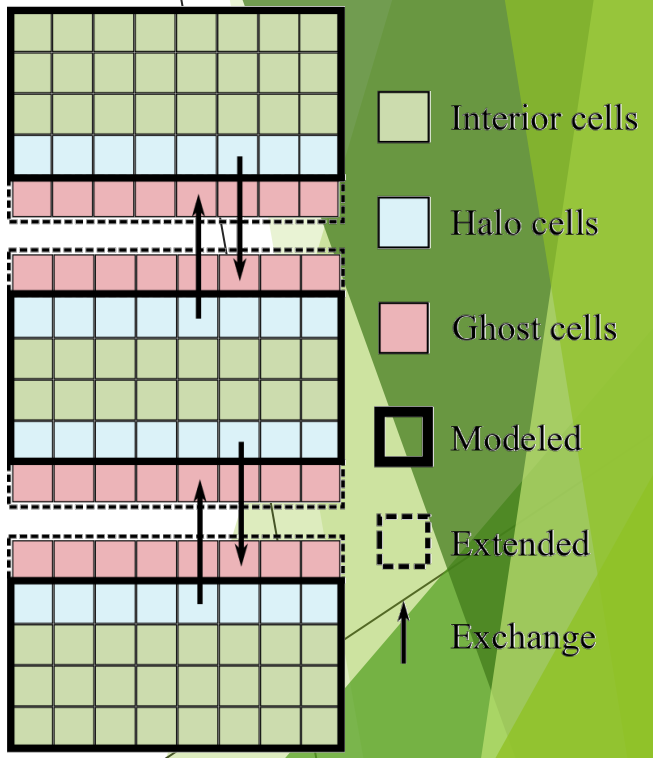
  - Functional decomposition: Assigning different tasks to GPUs



Domain decomposition and halo exchange: an example

- 1D row-wise domain decomposition
- The information has to be exchanged every time step (4 point stencil)
- Needed overlapped ghost/halo cells
  - Extra boundary cells used to share data between neighboring subdomains in parallel computing.
  - Ensure continuity and consistency in computations across subdomain boundaries.
- Non-blocking MPI communication using MPI_Isend and MPI_Irecv

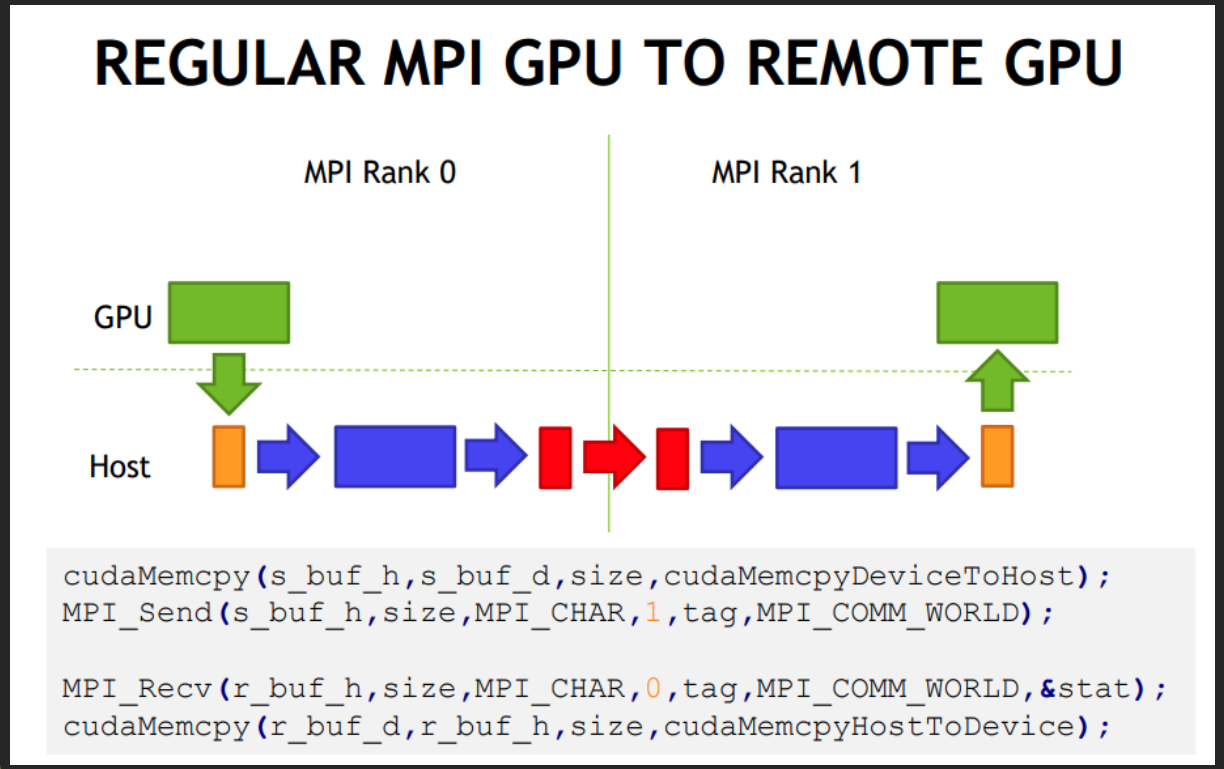




CUDA-Aware MPI

- Data is allocated in the device (GPU) but regular MPI calls require pointers to host (CPU) memory.
- Regular solution: memcpy (host/device) to send and receive
- Downside: this might decrease the performance

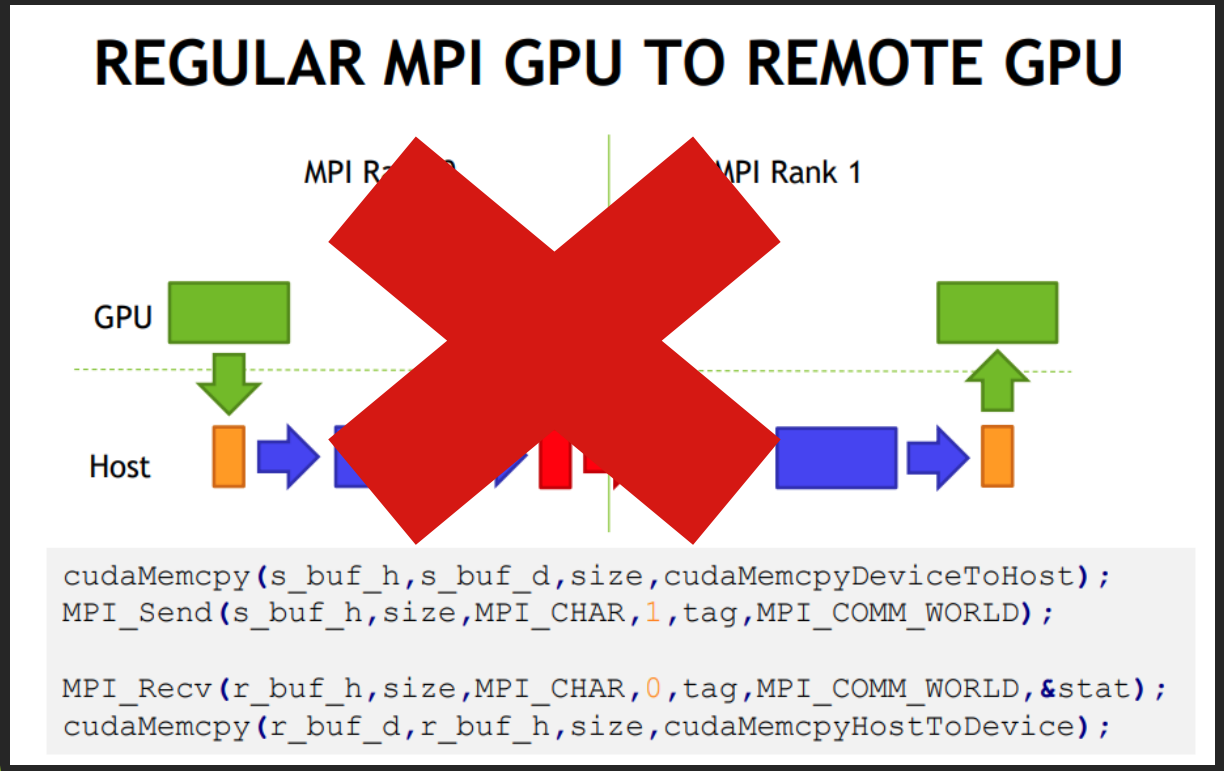
Images from Nvidia

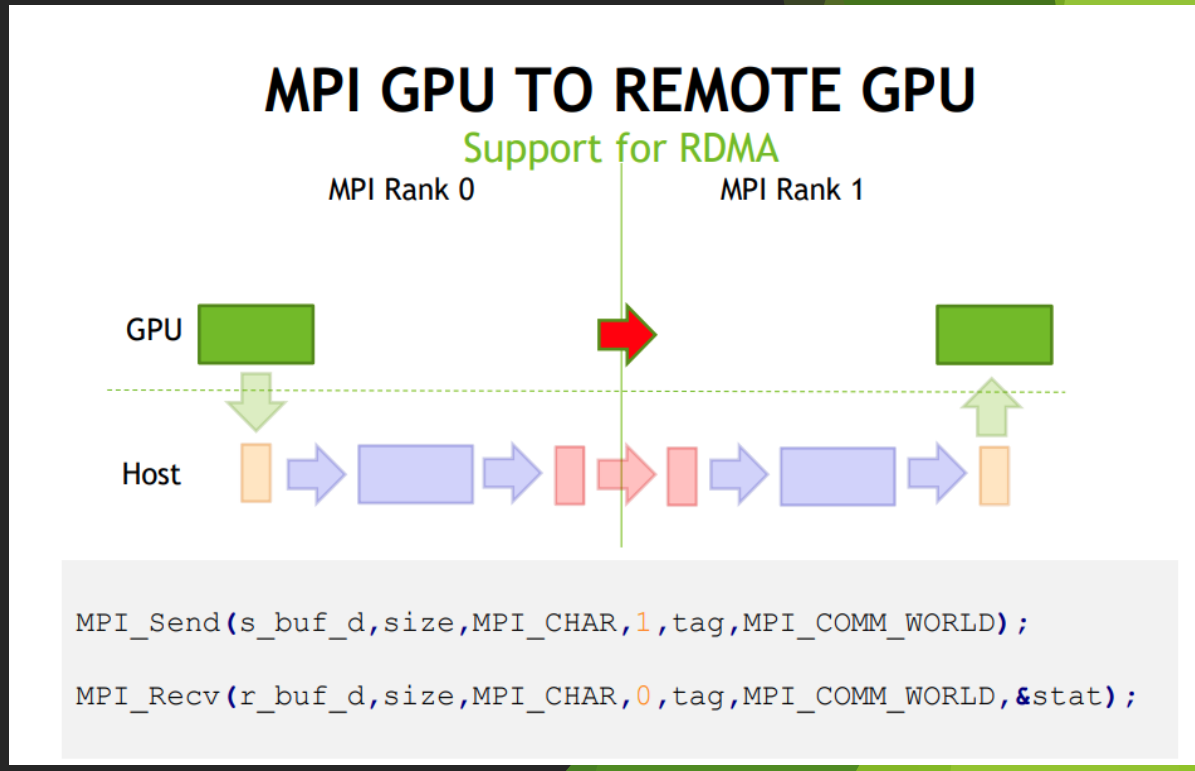


CUDA-Aware MPI

- Use CUDA-Aware MPI to avoid the copy to the CPU
- GPU buffers are passed directly to MPI calls (send and receive)

Images from Nvidia





Scaling in multi-GPU systems

- The efficiency is usually measured in terms of scaling plots:

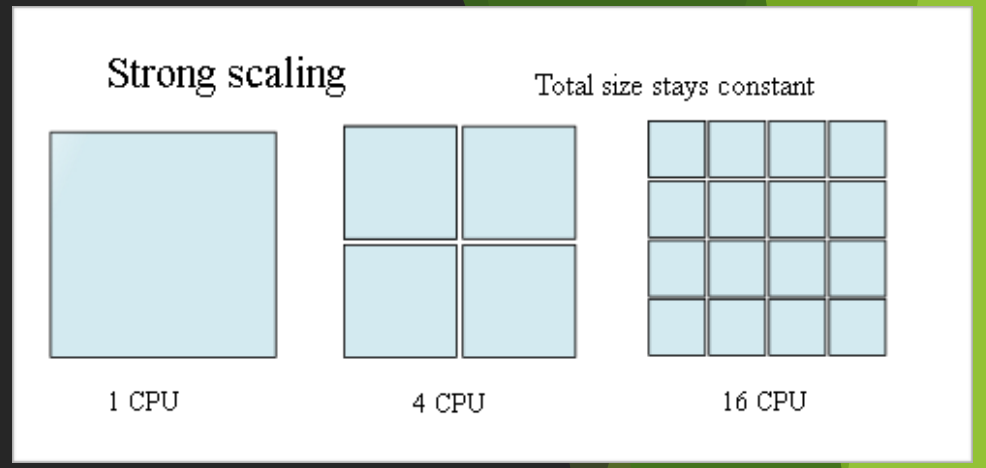
  - **Weak scaling**: measures performance with increasing problem size as GPUs increase. Goal: constant runtime.

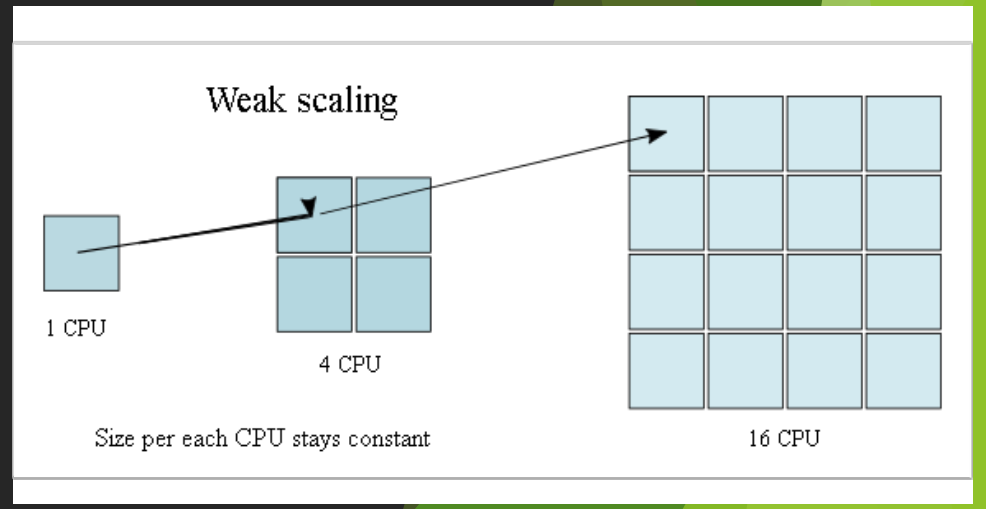
  - **Strong scaling**: measures runtime reduction for a fixed problem size as GPUs increase

- Performance metrics:

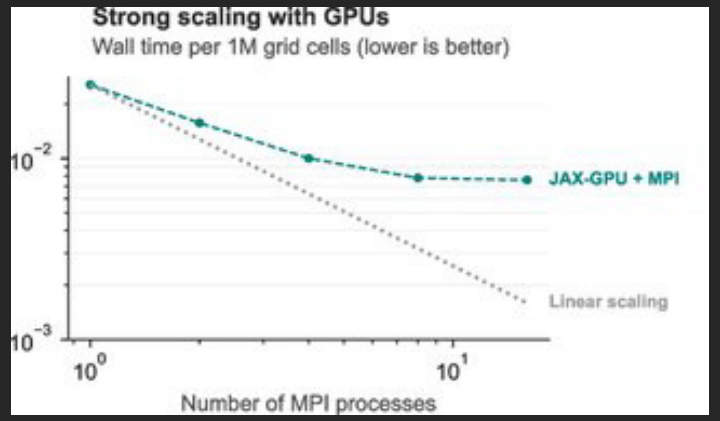
  - **Speedup**: Runtime improvement with multiple GPUs.

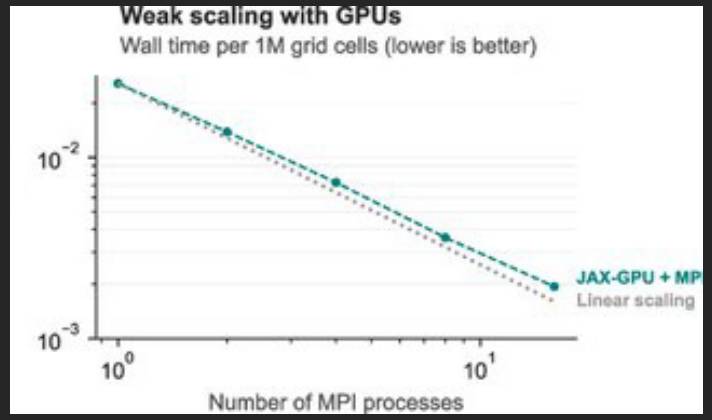
  - **Efficiency**: Speedup normalized by GPU count (utilization).

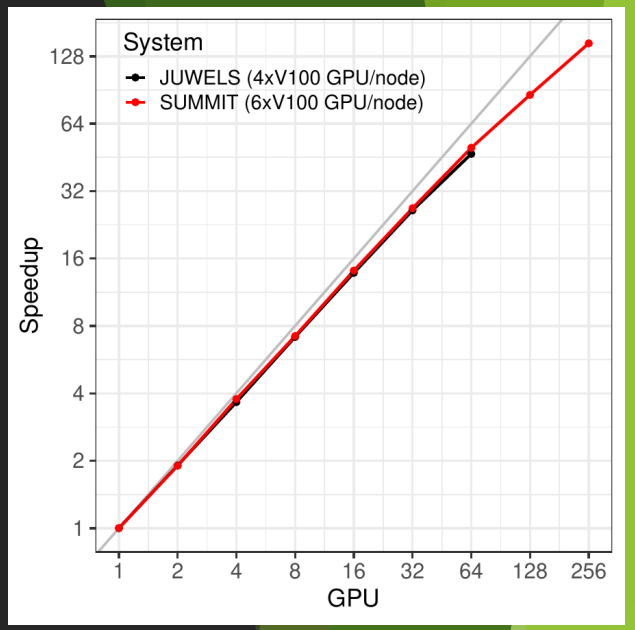


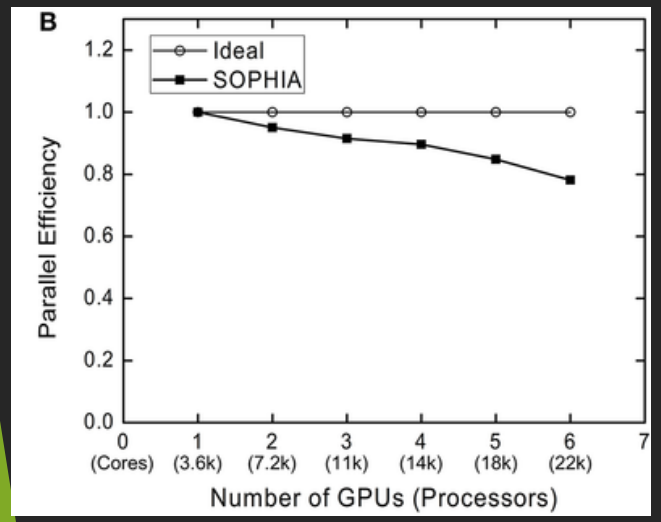


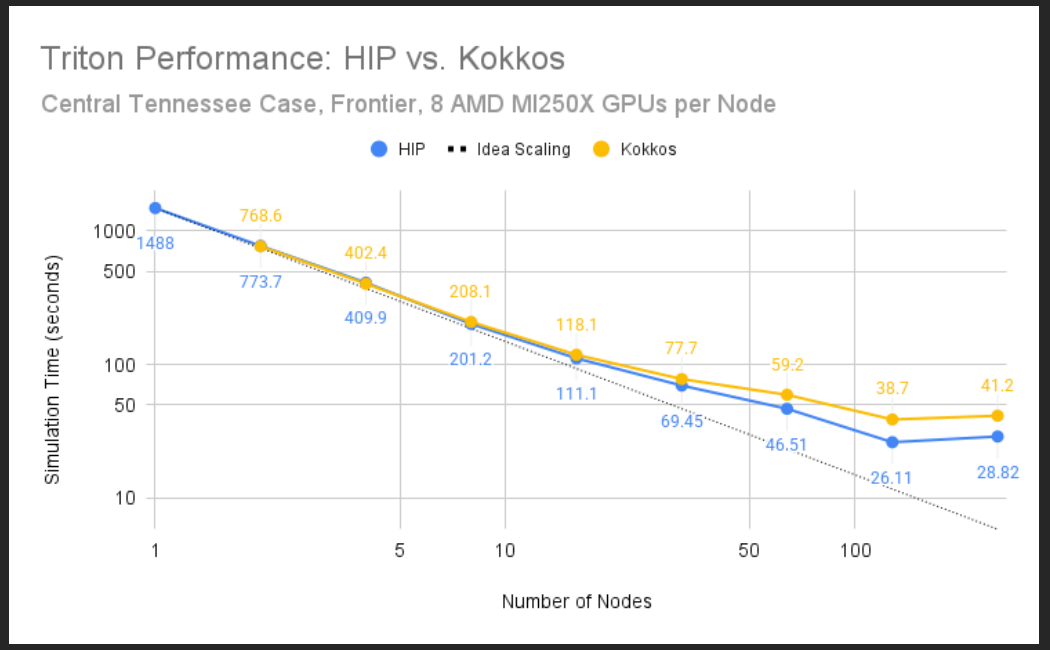
Scaling in multi-GPU systems

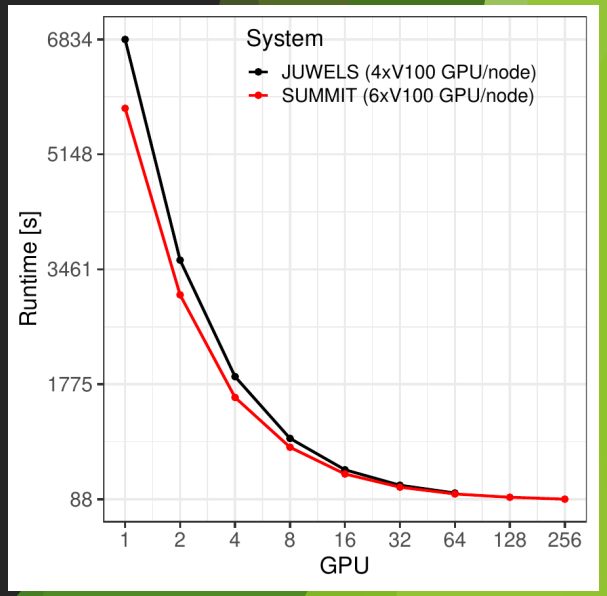












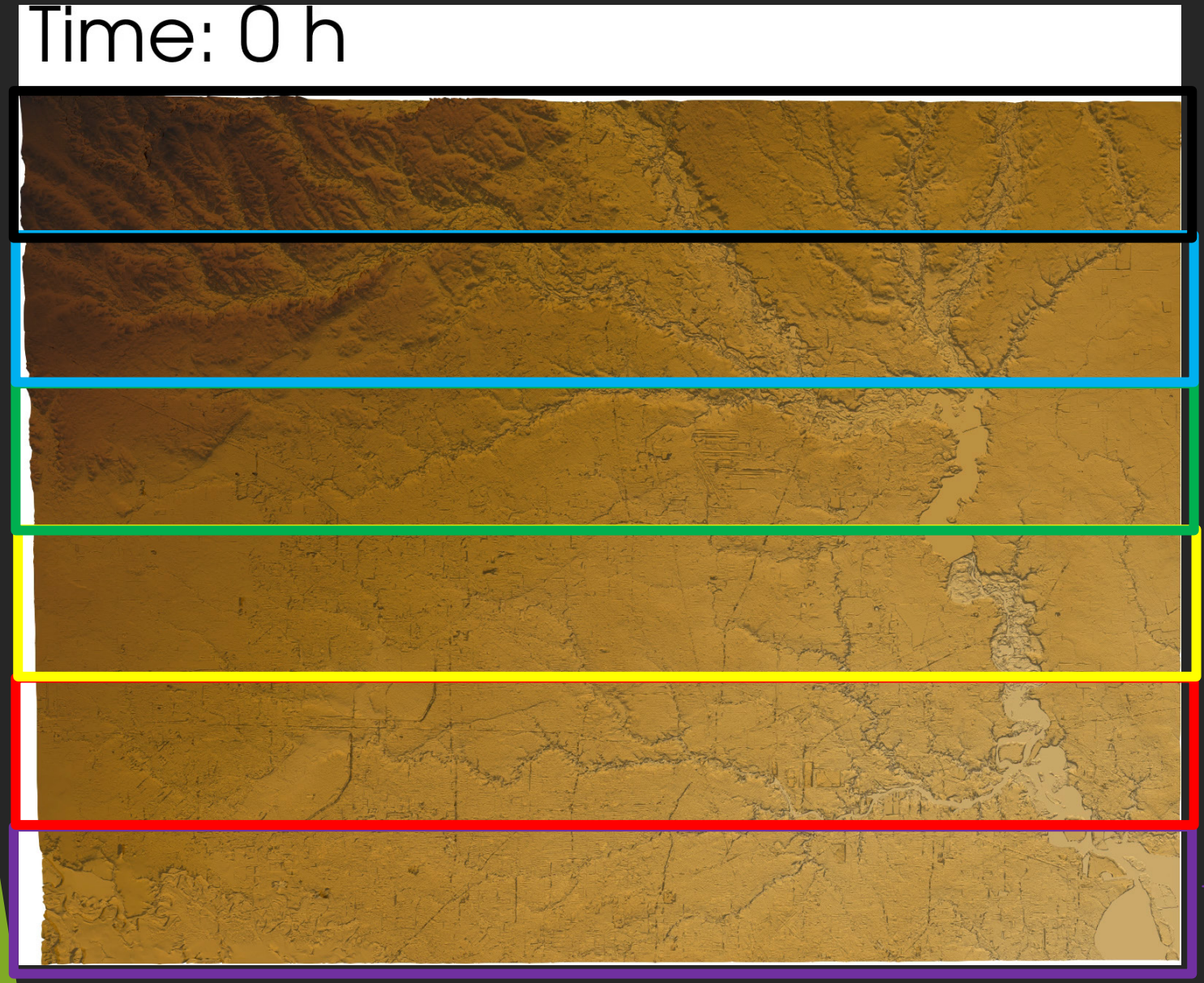
Load balancing

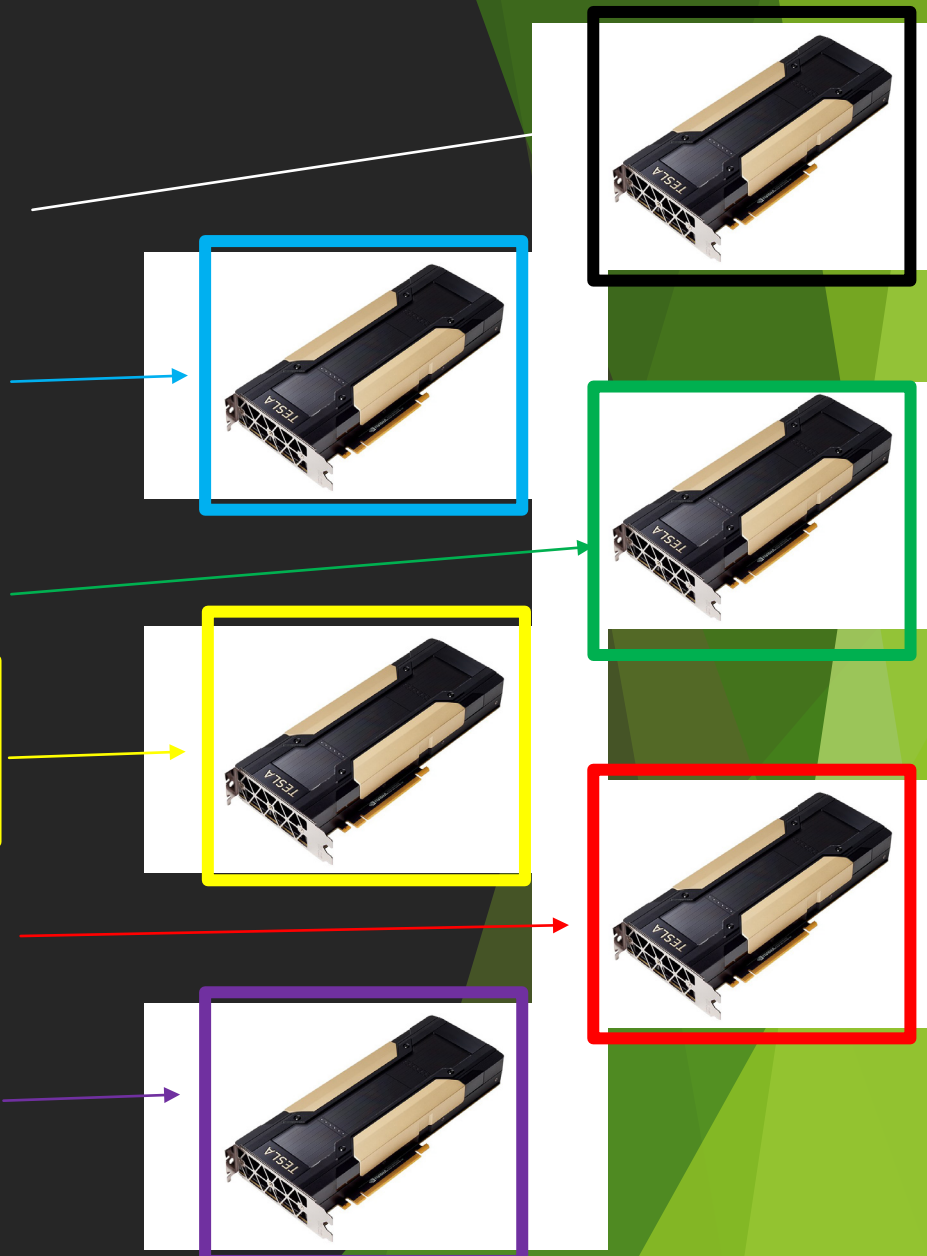
- **Goal**: distribute workload evenly across GPUs to maximize performance and minimize idle time.
- **Challenges**: variations in GPU performance, communication overhead, and workload heterogeneity.
- **Strategies**:
  - Dynamic workload distribution.
  - Overlap computation and communication.
  - Optimize data partitioning and transfer.
- Efficient load balancing ensures high utilization and scalability in multi-GPU systems.

Example

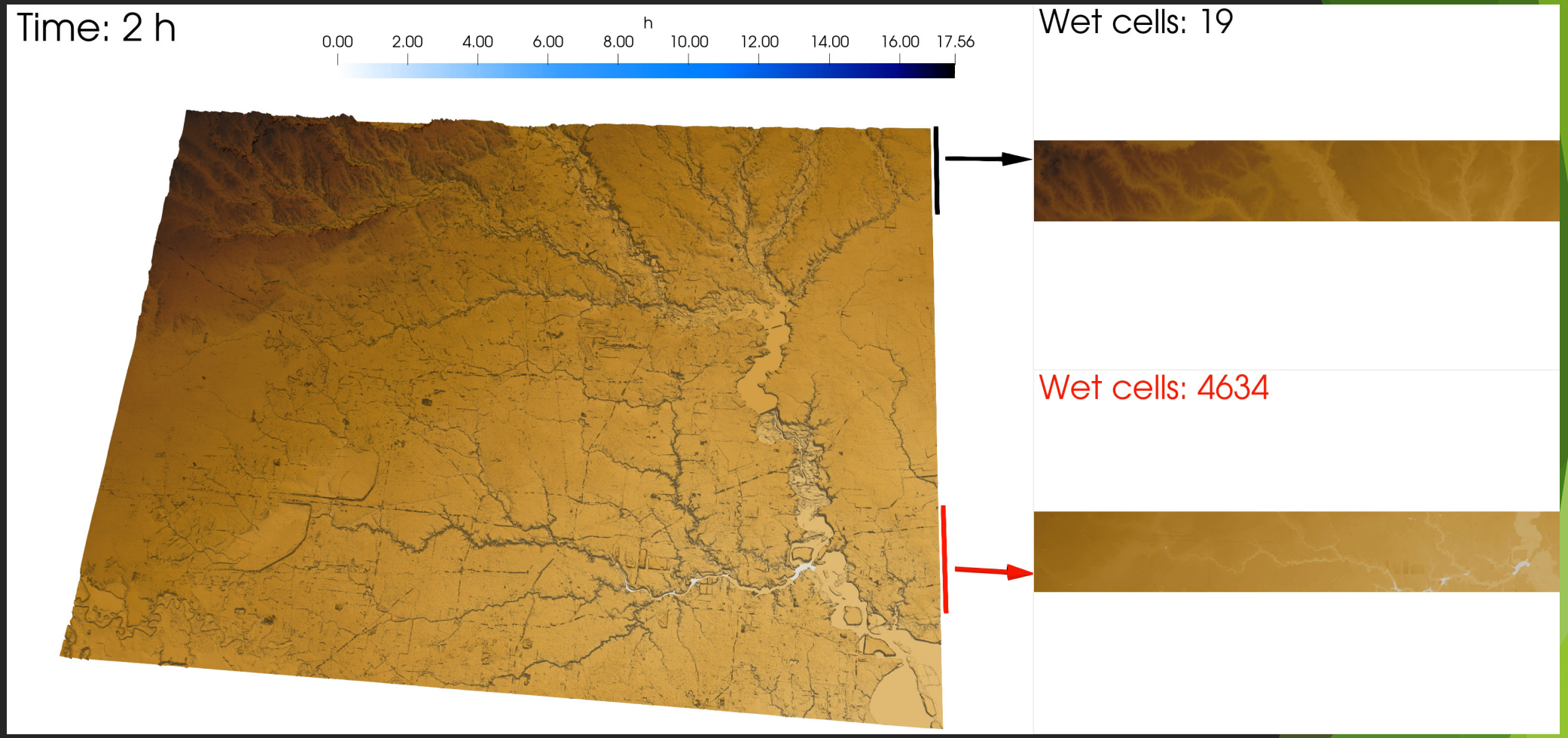
- Flooding problems: low % of wet cells
- Exclude dry cells with and “if statement”
- Downside: uneven distribution of MPI work. Subdomains might finish their computations before and have to wait for the exchange

Load balancing



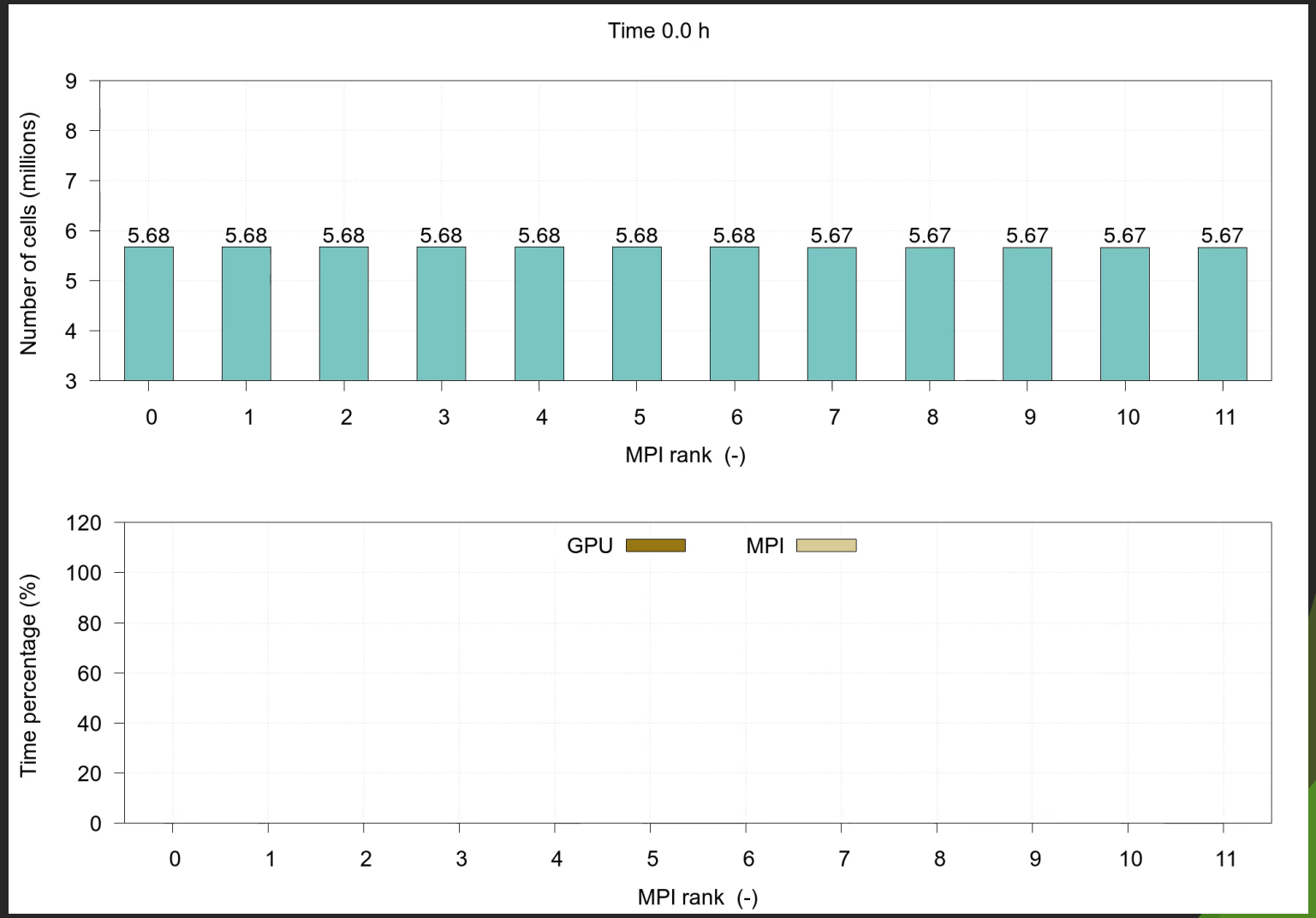


Load balancing



Load balancing

- 12 GPUs. Imbalance between GPU and MPI time



Load balancing

Dynamic domain decomposition

- Measure the MPI time at each iteration for each process

$$t_i$$

- Compute the new partition according to

$$r_i^{k+1} = r_i^k + \lfloor \underbrace{(\frac{t_i}{\bar{t}} - 1)}_{F} \frac{R}{N} f \rfloor$$

- How many times? Let to the user with a factor interval

$$\begin{aligned} F > 0 \quad & \text{if } t_i > \bar{t} \rightarrow \text{partition grows} \\ F < 0 \quad & \text{if } t_i < \bar{t} \rightarrow \text{partition shrinks} \\ F \sim 0 \quad & \text{if } t_i \sim \bar{t} \rightarrow \text{partition holds} \end{aligned}$$

$$r_i^k = \text{number of rows of partition } i \text{ at iteration } k$$

$$N = \text{partition size (Number of processes)}$$

$$R = \sum_{i=1}^{N} r_i^k = \text{total rows}$$

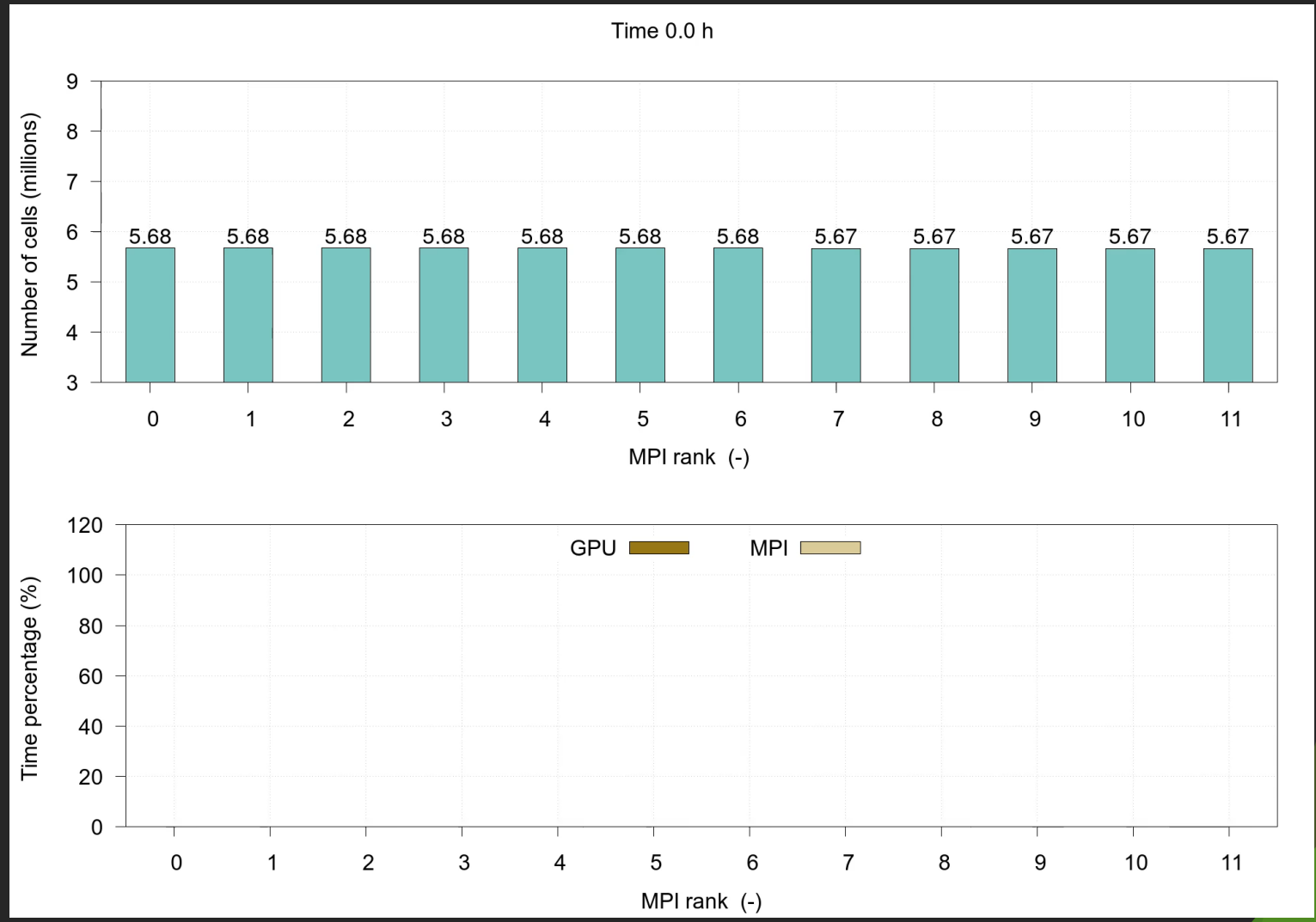
$$t_i = \text{MPI time partition } i$$

$$\bar{t} = \frac{1}{N} \sum_{i=1}^{N} t_i = \text{average MPI time}$$

$$f = 0.1$$

Load balancing

- 12 GPUs. Load balancing



Profiling GPU Applications

Key metrics to monitor

- Memory bandwidth: Measures data transfer rate between memory and GPUs; critical for memory-bound applications.
- Kernel execution time: tracks the runtime of GPU kernels; helps identify slow-performing kernels.
- Occupancy: indicates GPU resource utilization; higher occupancy often leads to better performance.

Profiling tools

- NVIDIA Nsight
- nvprof
- Omnitrace
- Vampir
- Extrae
- ...

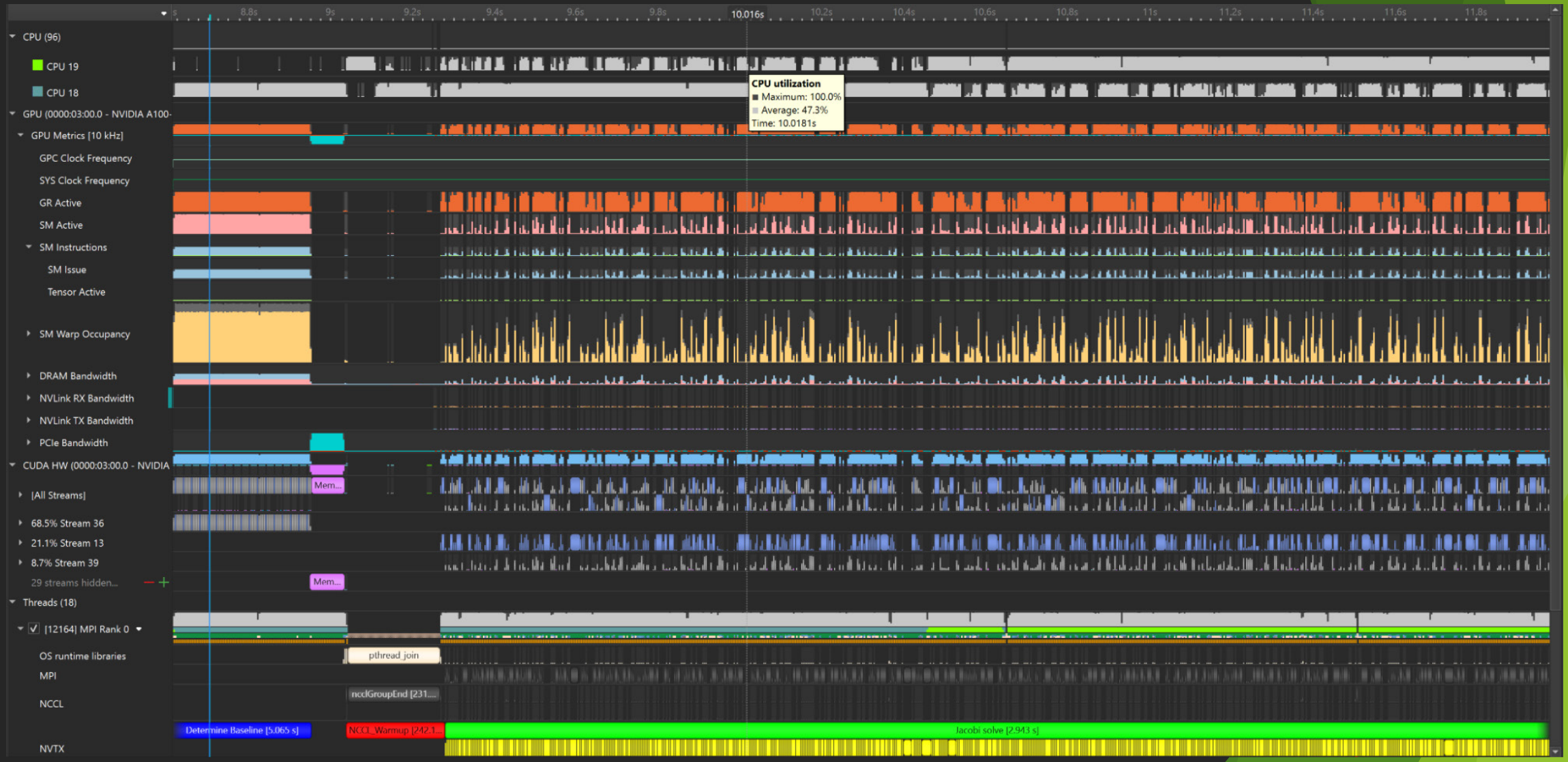
Steps for Profiling

- Identify Bottlenecks
- Optimize Memory Access
- Improve Kernel Performance

Best Practices for Iterative Optimization

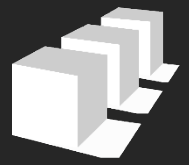
- Profile regularly during development
- Focus on one bottleneck at a time
- Use profiling tools to guide optimization efforts

Profiling GPU Applications



Universidad Zaragoza

1542



Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Instituto Universitario de Investigación en Ingeniería de Aragón Universidad Zaragoza

Advanced GPU programming

Mario Morales Hernández (mmorales@unizar.es)

March19 2025