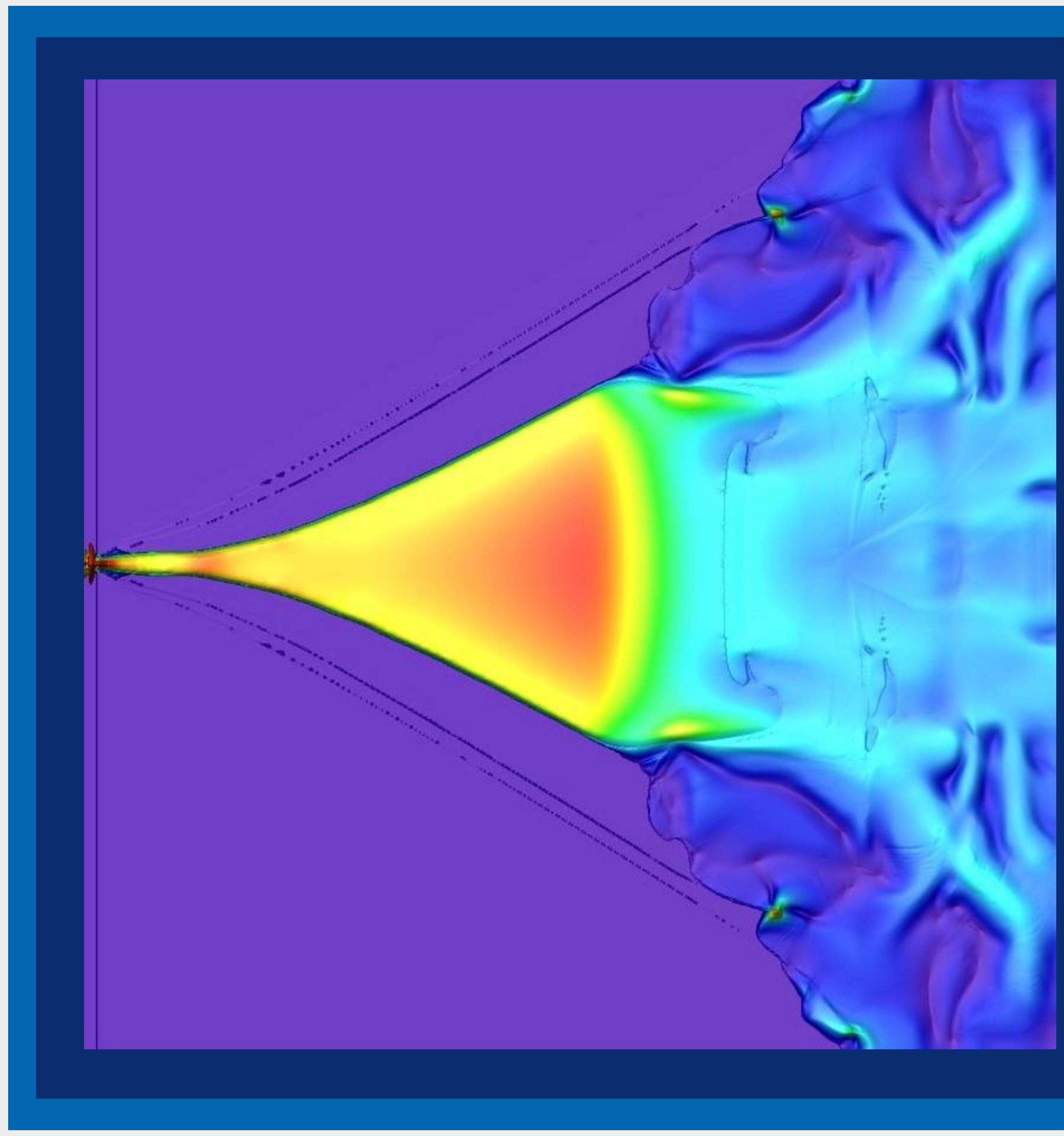
# Software Development Case Study: The Acceleration of a Distributed Application using GPUs

Martin Kuhnel (Chevron)
Alex Loddoch, Tao Sun



*An example flow output from a CompStrat simulation*

## ABSTRACT

We present a practical approach for the acceleration of an industrial and scientific application using graphics processing units (GPUs). Our original application is a computational stratigraphy codebase that couples fluid flow and sediment deposition submodels. The application uses domain decomposition and a halo exchange to split the workload among multiple workers in a distributed system. Our methodology abstracts and conserves the host data structures while re-writing computational elements in the GPU programming language CUDA. Utilizing high performance GPU machines in the Azure cloud, we show a minimum 90x speedup compared to a high-end CPU based cluster. In this poster, we give a brief description of the original algorithm, followed by a discussion of required software changes and additions. Although this case study focuses on a specific example, we hope this approach inspires similar efforts in other applications.

## 1 STARTING POINT

- Our goal is to accelerate a proprietary geology modelling application termed CompStrat [1], which models the flow of sediment-carrying water through a basin.
- The application uses a finite volume spatial discretization scheme, computing the flow and sediment fluxes for each cell.
- The flux calculations can be done in parallel if information about neighbors is kept up to date which is done via MPI.

## 2 GPU ACCELERATION APPROACH

- Requires porting the computation to CUDA kernels [2].
- A full rewrite can potentially generate the most performant code but is not viable from a time-to-production standpoint.
- Individual kernel swaps suffer in performance due to excessive copies between the host and device.
- Our final approach is "in-place" and preserves the original code structure, while being GPU-native for the computational heavy simulation. See **Figure 1**.

## 3 DOMAIN MAPPING

- To maintain compatibility with the original data structures, the domain is abstracted on the GPU using index maps.
- The domain on the GPU is simply represented in directly indexed arrays of data that are mapped via indices to their type, neighbors, and to their location back on the host. See **Figure 2**.

## 4 MEMORY BUFFER

- GPU cards have limited memory that cannot contain the entire growing sediment grid.
- We introduce a fixed-size GPU buffer containing the top layers in our sediment columns which resynchronizes only when needed.

## 5 HALO EXCHANGE

- We assign one MPI rank to one available GPU, using the PCI bus topology to make optimal selections.
- We can use the MPI structure of original CPU code by copying the data in the halo back to the host.
- Depending on the hardware available, the halo exchange can be accelerated with the use of NVLink, which we use via the NCCL library to perform GPU-to-GPU transfers [3]. See **Figure 3**.

## 6 RESULTS

- The results for five internal benchmarks are shown in **Figure 4**.
- Simulations originally taking multiple weeks on upwards of 40 nodes can now be run over the weekend on a single GPU node.

## 7 FUTURE WORK

- Since memory is no longer distributed over many nodes, we will implement a new memory management system.
- We will also apply this approach to other similar applications.

**References**

[1] Lisa Goggin Maisha Amaru, Tao Sun and Ashley Harris. 2017. Integration of computational stratigraphy models and seismic data for subsurface characterization. The Leading Edge 36, 11 (November 2017). https://doi.org/10.1190/tle36110947a1.1

[2] NVIDIA 2023. CUDA Toolkit Documentation 12.2 Update 1. NVIDIA. https://docs.nvidia.com/cuda/.

[3] NVIDIA [n. d.]. NVIDIA Collective Communication Library (NCCL) Documentation. NVIDIA. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs.

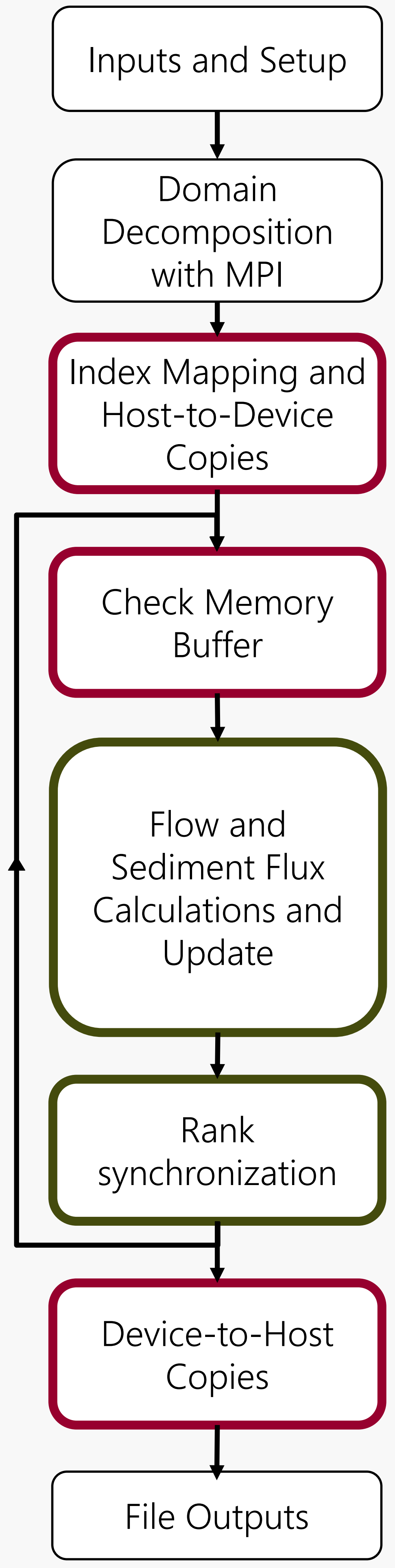

Inputs and Setup
Domain Decomposition with MPI
Index Mapping and Host-to-Device Copies
Check Memory Buffer
Flow and Sediment Flux Calculations and Update
Rank synchronization
Device-to-Host Copies
File Outputs

**Figure 1: An "In-Place" Approach**

New Sections
Sections Rewritten in CUDA

With our added index mapping and memory buffer, we conserve the input and output, domain decomposition, and CPU data structures. Meanwhile, we use the GPU for the compute-intensive flow and sediment flux calculations with minimal communication to the host.

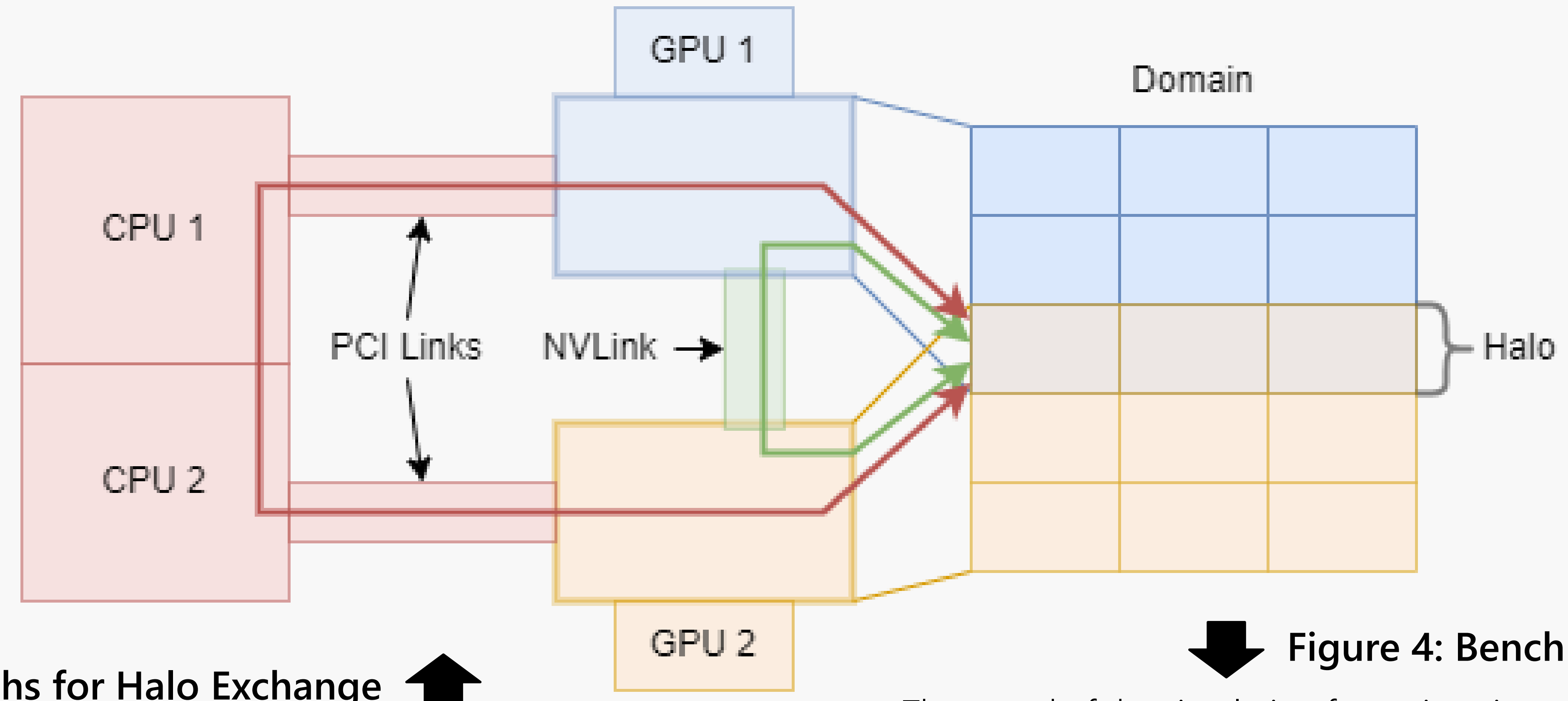


**Figure 3: Multiple Paths for Halo Exchange**

Each GPU computes a portion of the domain with an overlapping halo with its neighbors. The data transfer of the halo can either be done through the PCI links (red arrow) or through the faster and more efficient NVLink (green arrow).

**Figure 2: GPU Domain Layout**

The GPU works in an abstracted domain with directly indexed arrays. For example, an active cell is represented as an index to its location in larger arrays containing all cell data. Similarly, special types of cells and neighbors are mapped to their data and location in other spaces.

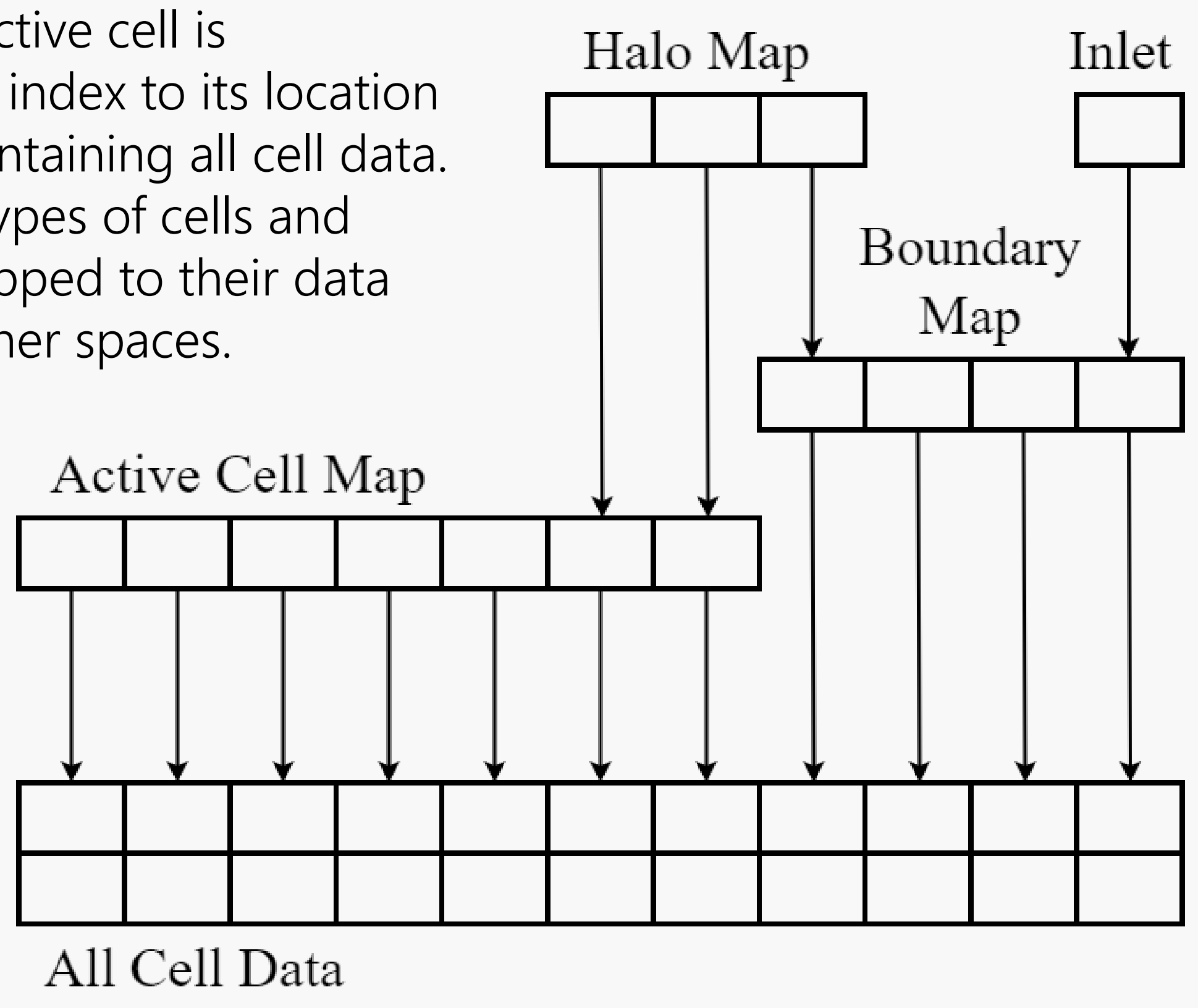
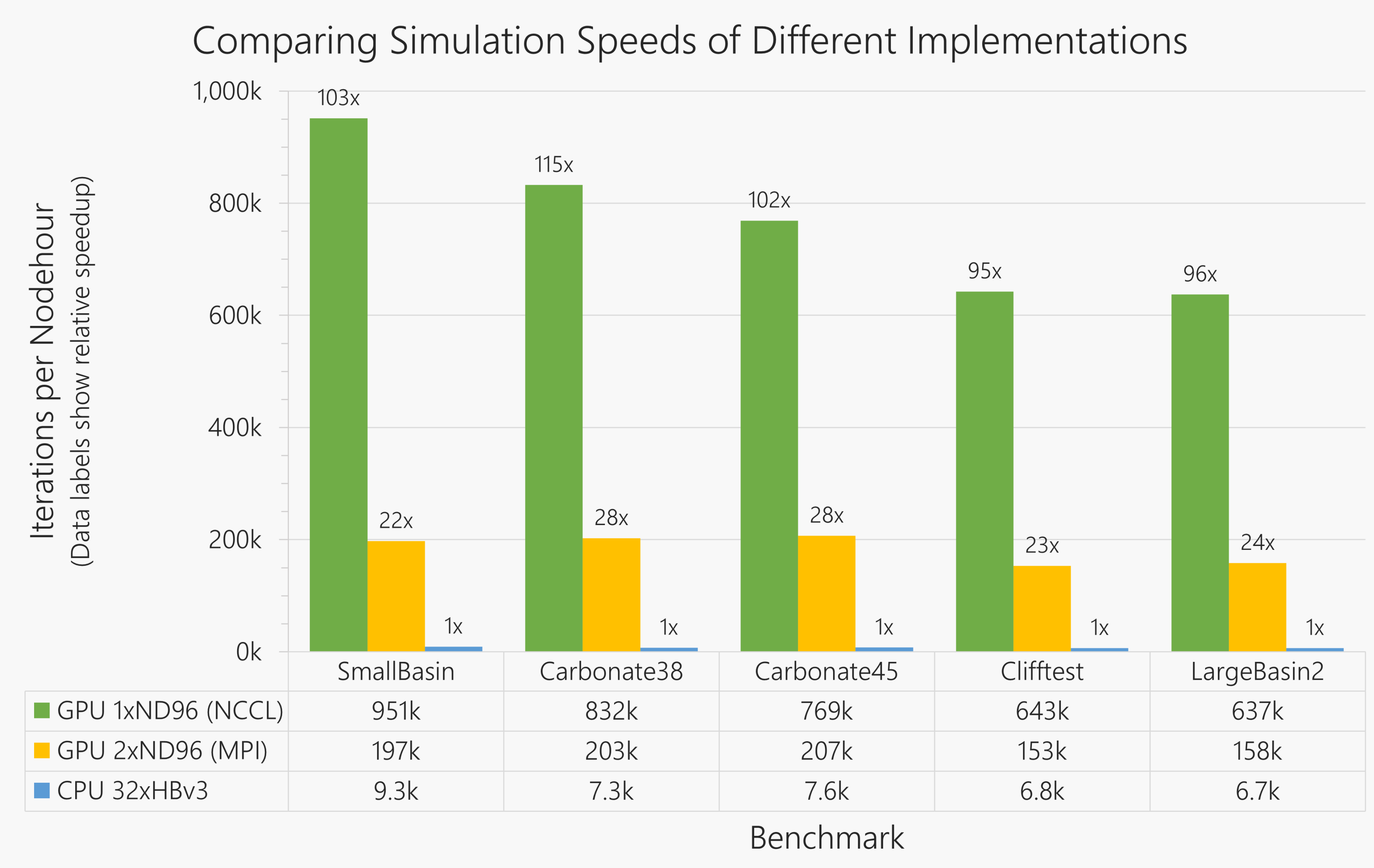


**Figure 4: Benchmark Results**

The speed of the simulation for various internal benchmarks is given in iterations per hour on a given SKU. The CPU code ran on 32 Azure HBv3 nodes with 120 AMD CPU cores each. The GPU codes ran on the Azure ND96 SKUs with 8 A100 GPUs. The new GPU code with NCCL runs approximately 100 times faster than the original.



| | SmallBasin | Carbonate38 | Carbonate45 | Clifftest | LargeBasin2 |
|---|---|---|---|---|---|
| 🟩 GPU 1xND96 (NCCL) | 951k | 832k | 769k | 643k | 637k |
| 🟨 GPU 2xND96 (MPI) | 197k | 203k | 207k | 153k | 158k |
| 🟦 CPU 32xHBv3 | 9.3k | 7.3k | 7.6k | 6.8k | 6.7k |