Centre de Thermique de Lyon
CETHIL UMR 5008

EDF R&D

Lip

The TheLMA Project:
Multi-GPU Implementation of the Lattice Boltzmann Method

Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau,
and Jean-Jacques Roux

CCGSC – September 8, 2010

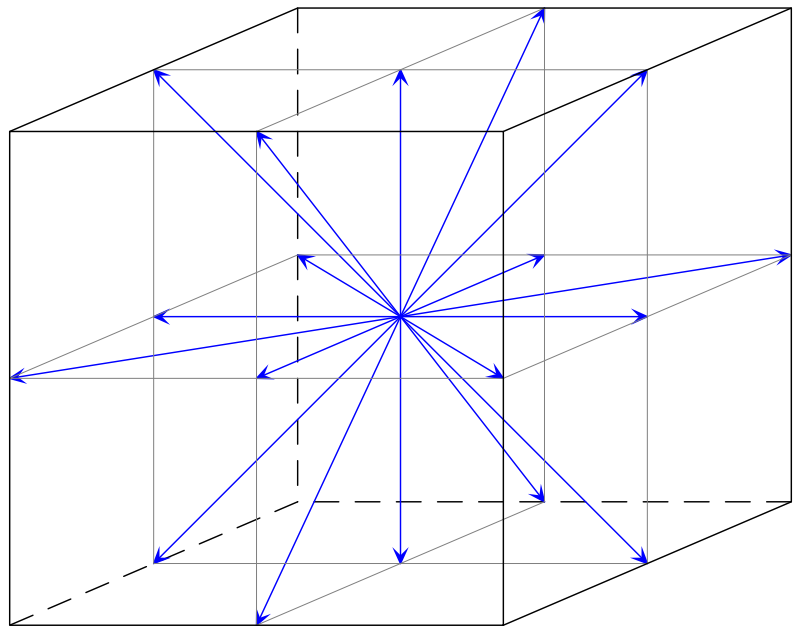
Motivations

- Due to computational cost, fluid dynamics is often neglected in building physics. Modelling of energy efficient buildings requires to take these effects into account.
- The lattice Boltzmann method (LBM) is an innovative approach in CFD. Besides other advantages, parallel implementations of the LBM are rather straightforward.
- GPUs, e.g. CUDA capable hardware, provide an inexpensive and efficient way to perform parallel computations.
- In real life computations, the device memory of a single GPU is too small to store a whole lattice, hence multi-GPU implementations are necessary.

I – Lattice Boltzmann Method

Lattice Boltzmann Method

Mass transfer is performed in discrete time and space using a finite set of velocities, as the D3Q19 stencil:



Lattice Boltzmann Equation

The fluid is represented by a discrete distribution f_i associated to the velocities $\mathbf{e}_i$, and obeying to the following equation:

$$f_i(\mathbf{x} + \delta t\, \mathbf{e}_i,\ t + \delta t) - f_i(\mathbf{x},\ t) = \Omega_i(f(\mathbf{x},\ t))$$

where Ω_i is a collision operator.

The macroscopic quantities are given by:

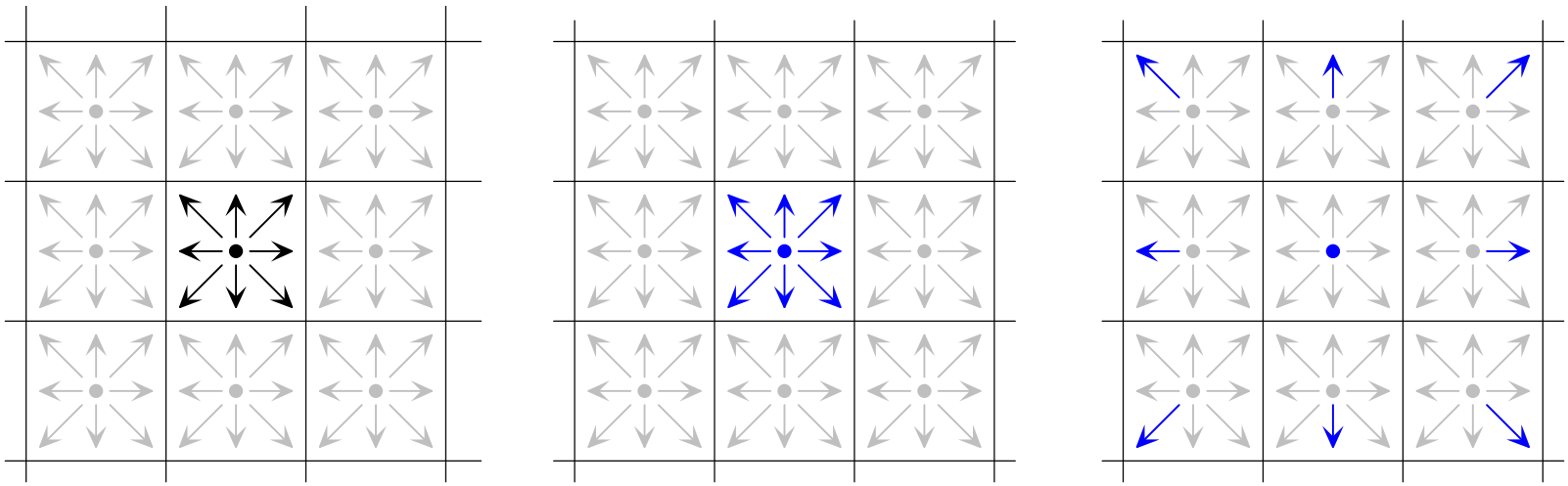
$$\rho = \sum_i f_i \qquad\qquad \mathbf{u} = \frac{1}{\rho} \sum_i f_i \mathbf{e}_i$$

Algorithmic Aspect

The LBM breaks up in two elementary steps, i.e. collision and propagation:

$$\tilde{f}_i(\mathbf{x},\ t) = f_i(\mathbf{x},\ t) + \Omega_i(f(\mathbf{x},\ t))$$

$$f_i(\mathbf{x} + \delta t\, \mathbf{e}_i,\ t + \delta t) = \tilde{f}_i(\mathbf{x},\ t)$$



Algorithm

for each time step t **do**
  for each lattice node **x** **do**
    read velocity distribution $f_i(\mathbf{x}, t)$
    if node **x** is on boundaries **then**
      apply boundary conditions
    end if
    compute updated distribution $\tilde{f}_i(\mathbf{x}, t)$
    propagate to neighbouring nodes $\mathbf{x} + \delta t \mathbf{e}_i$
  end for
end for

II – Implementing the LBM on the GPU

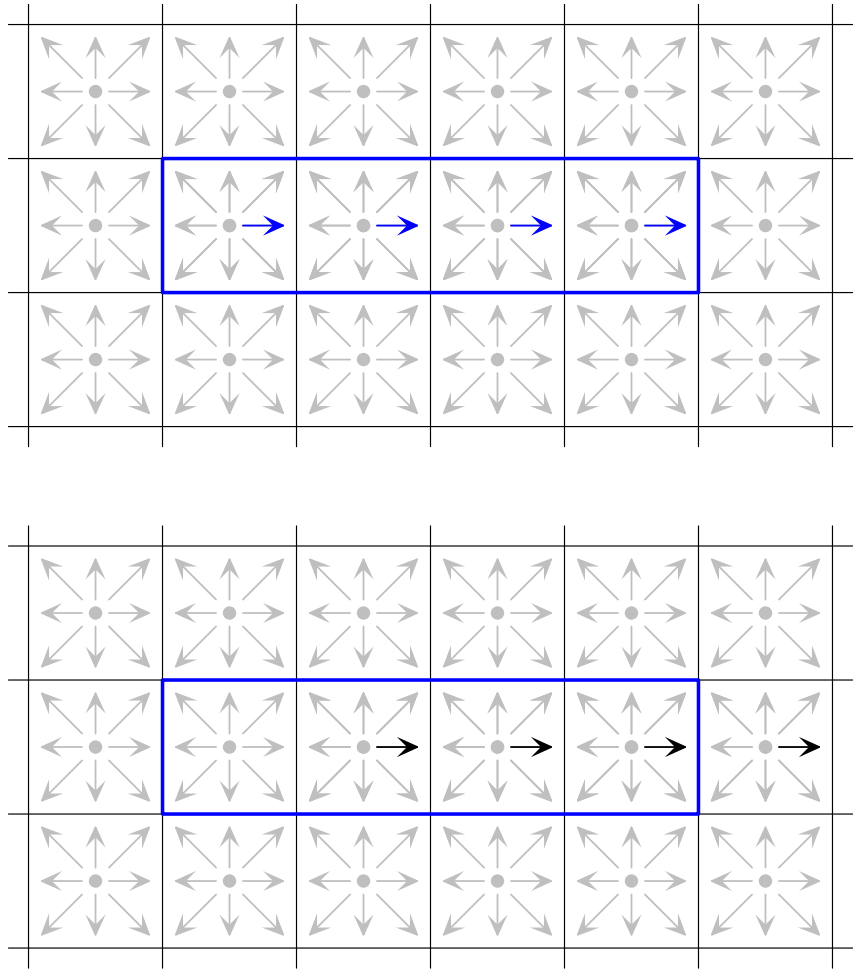
Single-GPU Implementation of the LBM

- Assign one node per thread to take advantage of massive parallelism.
- Store the velocity distribution in global memory using a structure of array (SoA) like layout to enable coalescing.
- Launch one kernel for each time step to ensure global synchronisation.

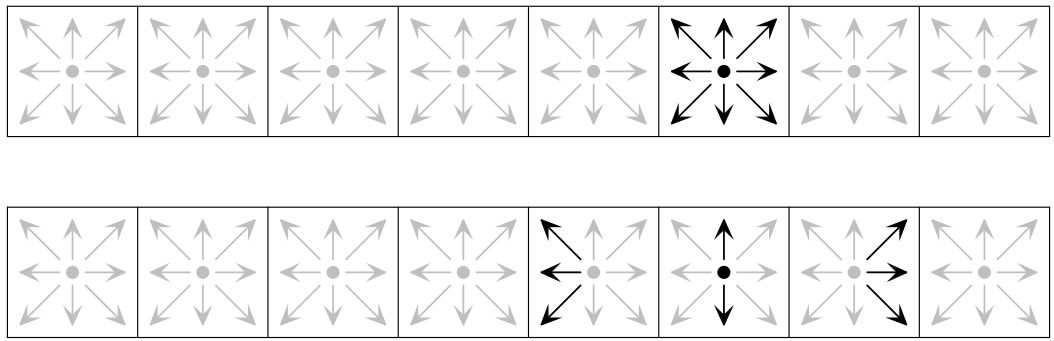
The limiting factor for single-GPU implementation is the global memory maximum throughput.

Misaligned Memory Transactions

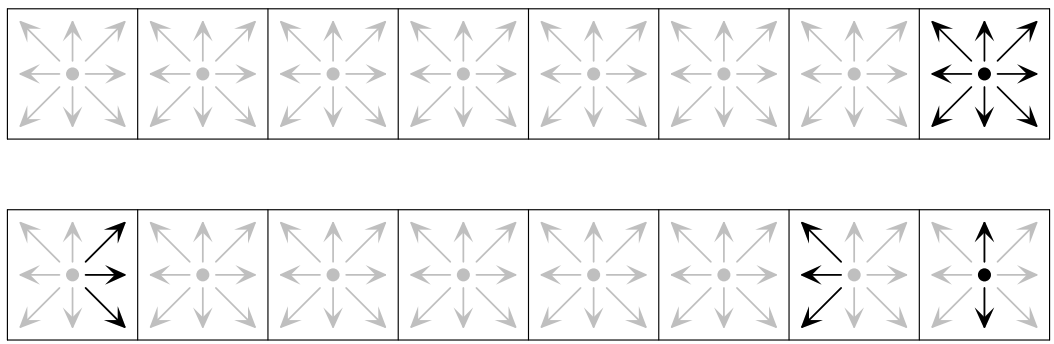
The major issue in optimising single-GPU implementation is to reduce the impact of misaligned memory transactions.



Shared Memory Approach

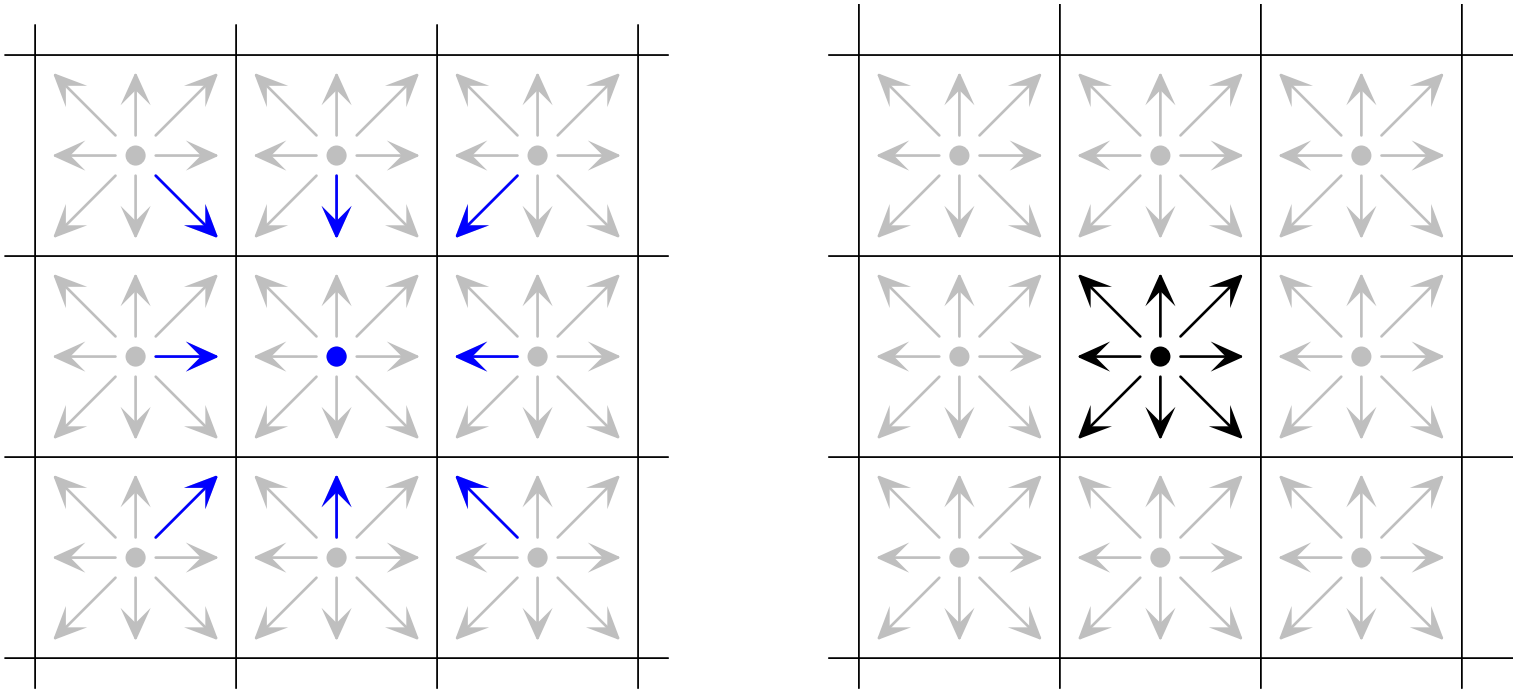


Normal case



Block boundaries

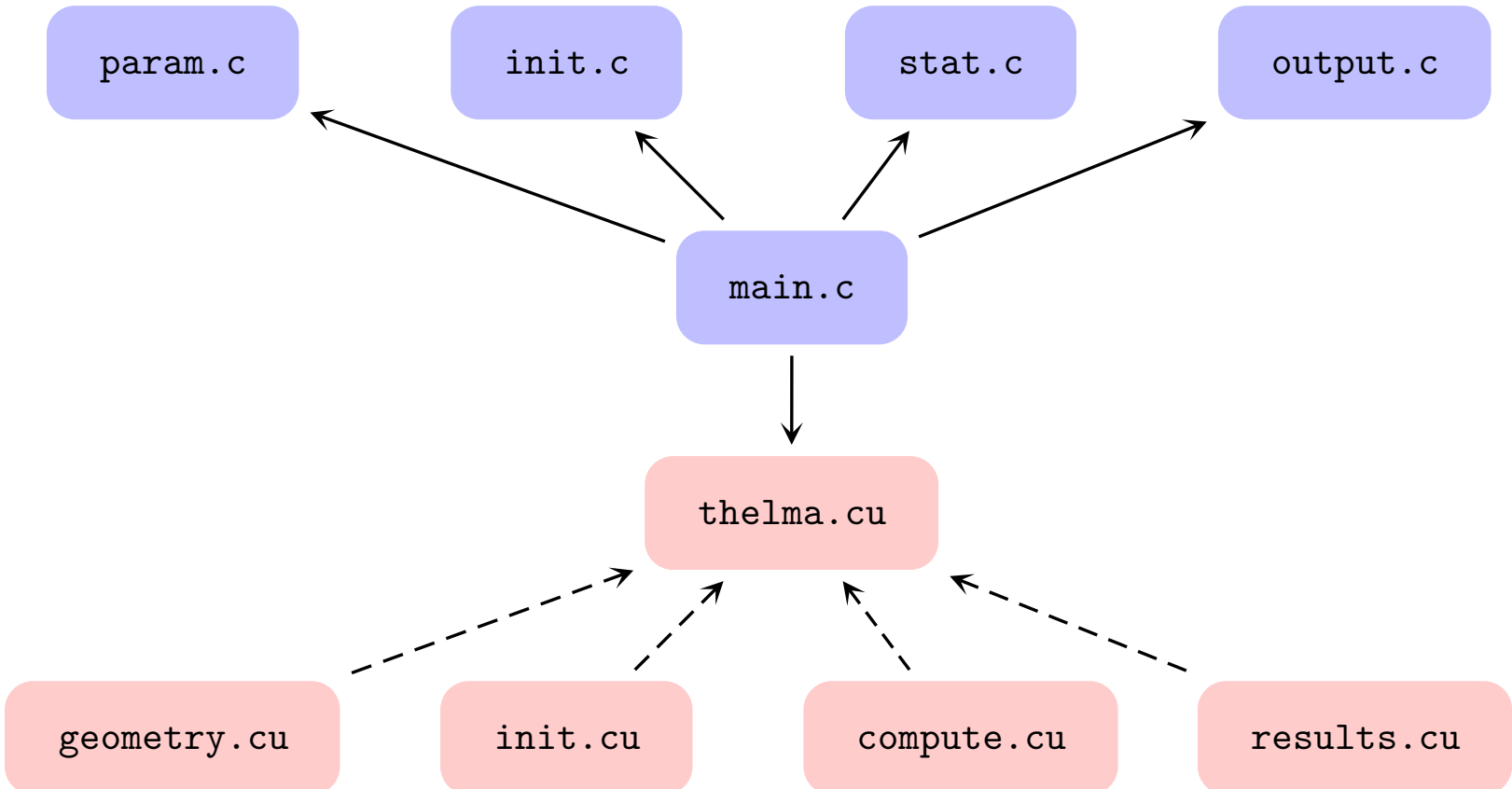
In-Place Propagation



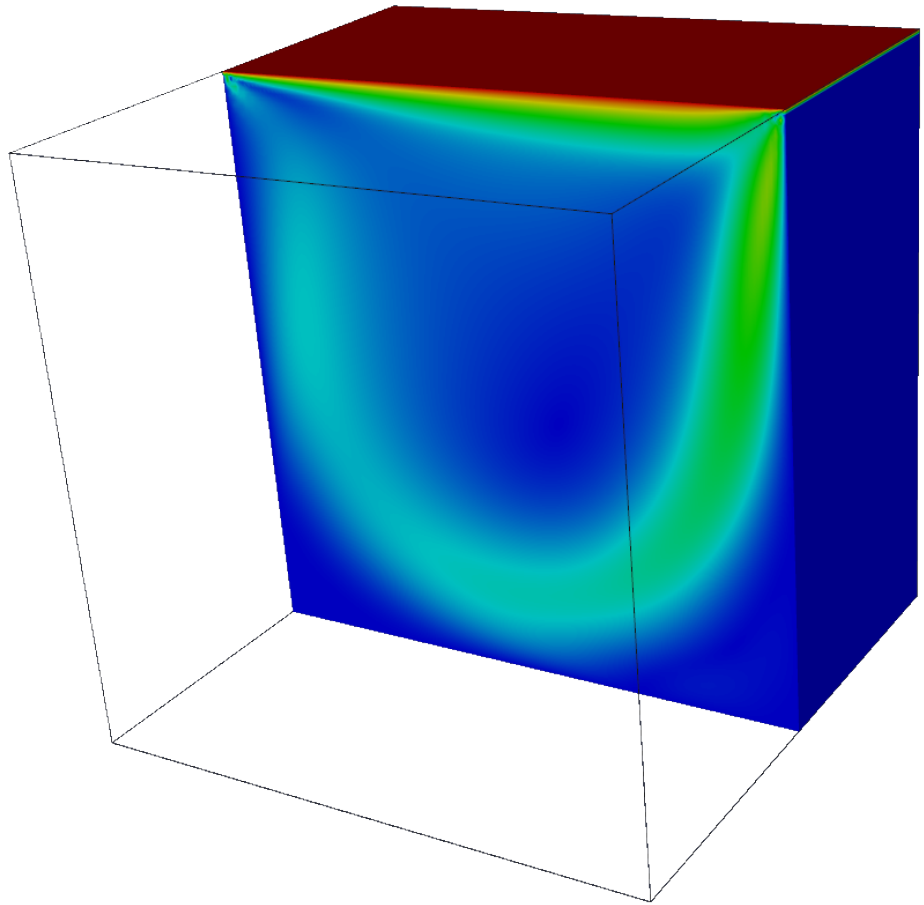
Misaligned reads being far less expensive than misaligned writes, an alternative to the shared memory approach is to use in-place propagation.

III – Multi-GPU Implementation

The TheLMA Framework

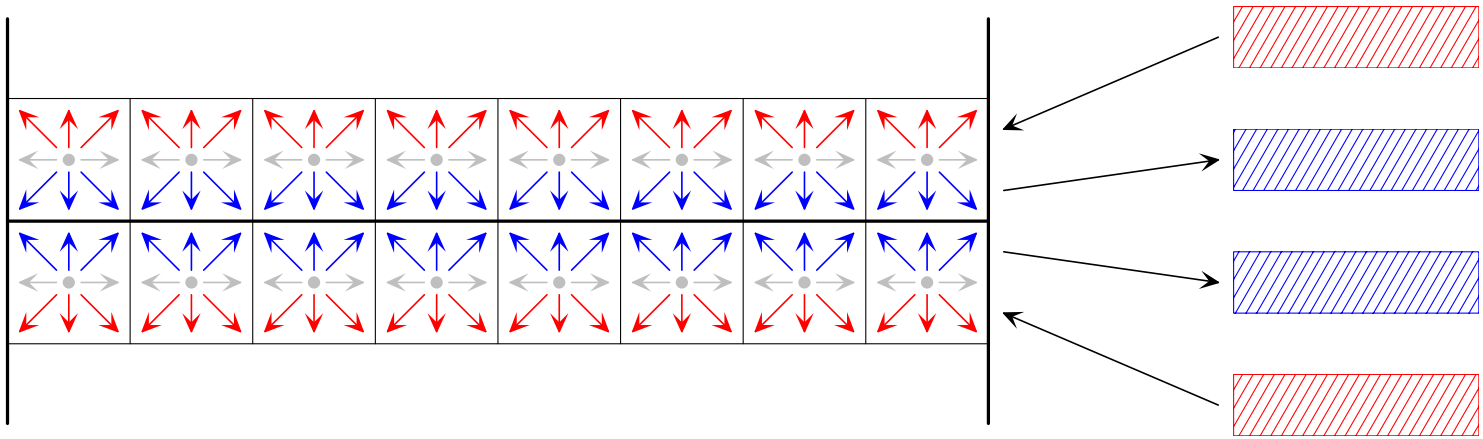


Lid-Driven Cubic Cavity



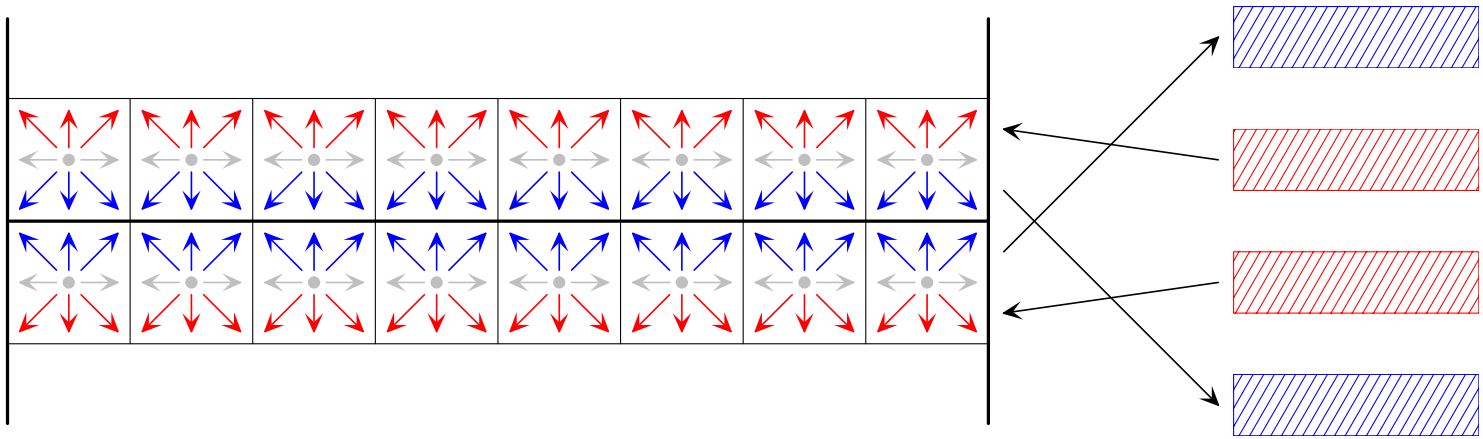
To validate our code, we implemented the lid-driven cubic cavity test case. The lattice is split in cuboid sub-domains along the major dimension.

Inter-GPU Communication Scheme



Communication between GPUs is performed using page-locked CPU memory and zero-copy memory transactions.

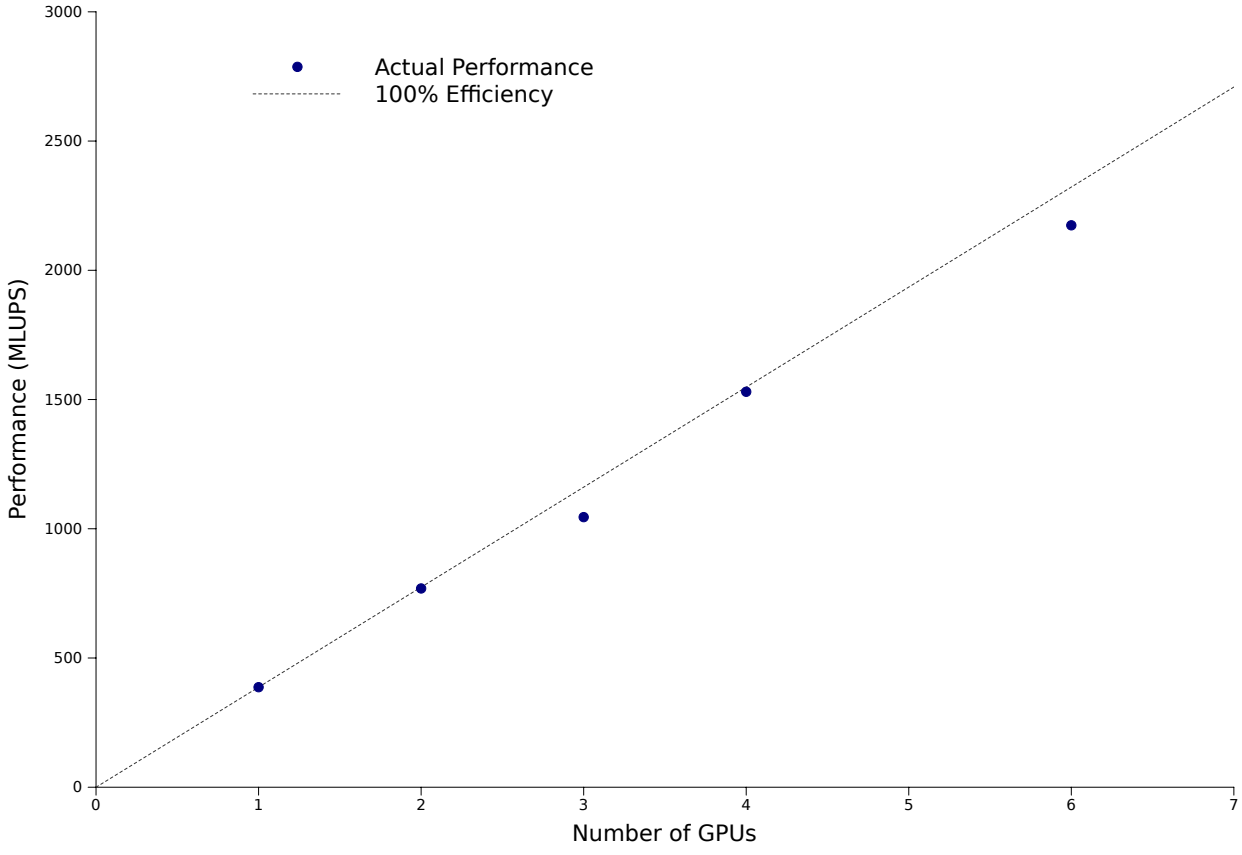
Multi-GPU Communication Scheme



Communication between GPUs is performed using page-locked CPU memory and zero-copy memory transactions.

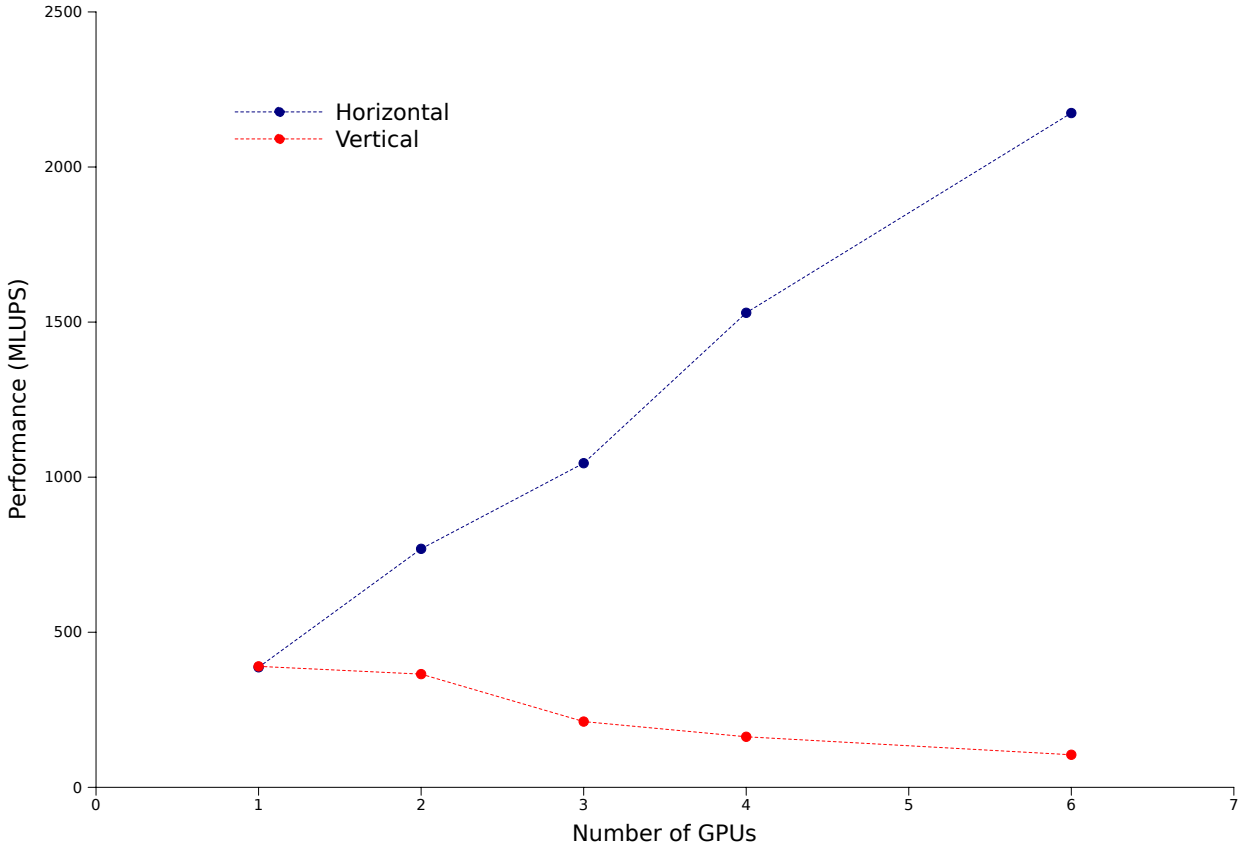
IV – Performance Study

Performance and Scalability



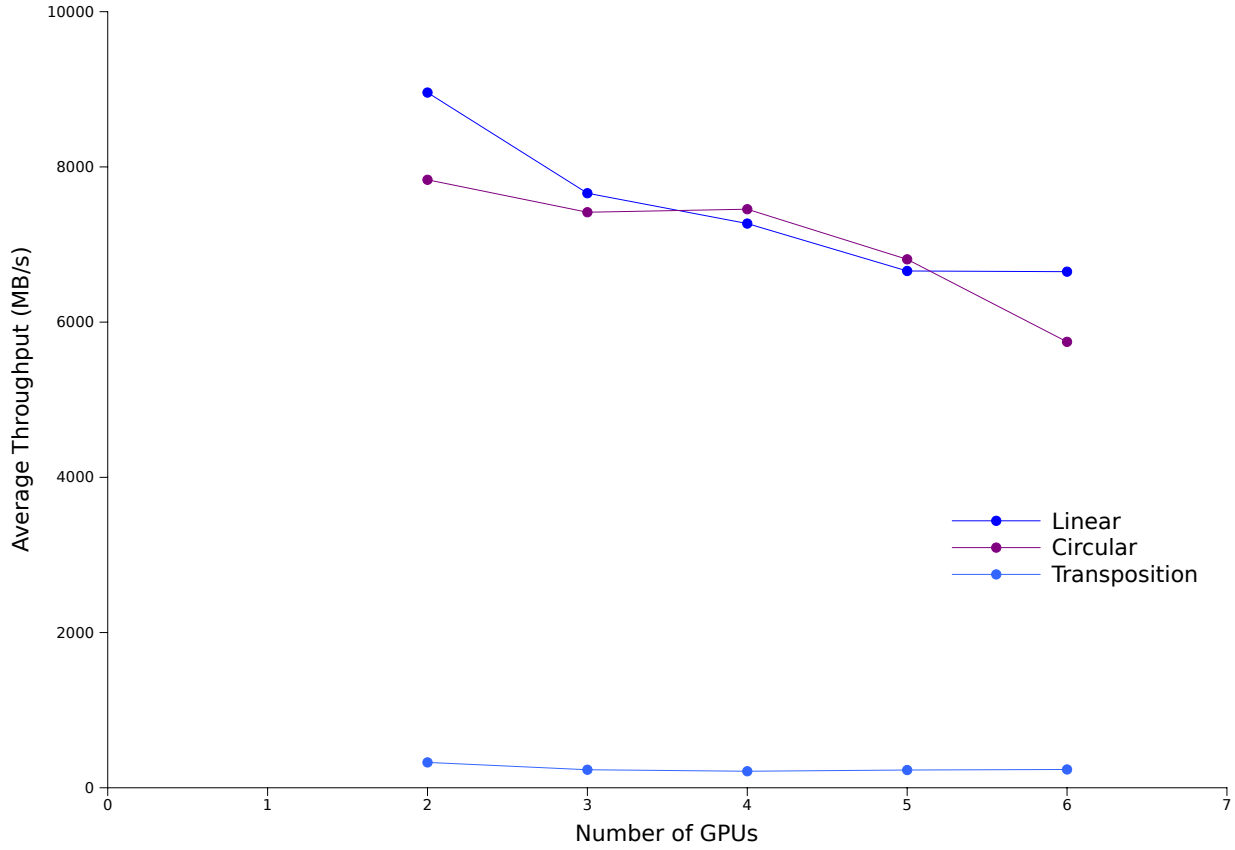
We measured up to 2.17 GLUPS (94% efficiency) using 6 Tesla C1060 on a 192^3 lattice in single precision.

Splitting along the Minor Dimension



Splitting the lattice along the minor dimension leads to sub-domains composed of non contiguous memory cells, which has dramatic effects on performance.

Inter-GPU Throughput



To measure inter-GPU maximum sustained throughput, we used a stripped-off version of our program.

Maximum Sustained Throughput vs Required Throughput

GPUs	Maximum (MB/s)	Kernel duration (μs)	Required (MB/s)
2	8958	9145	322
3	7661	6096	968
4	7270	4572	1935
6	6650	3048	4838

The required throughput is always less than the maximum available. It should be mentioned that the devices are able of efficient communication/computation overlapping.

Summary and Future Work

In this contribution:

- We present an implementation of a multi-GPU LBM solver.
- Our implementation shows good performance and scalability.
- We study effective throughput of inter-GPU communications.

Future work will include:

- Multi-GPU thermal LBM solver.
- Extension of the TheLMA framework.
- MPI based implementation.

Thank you for listening!