PGAS implementation of Sparse Matrix Vector Multiplication and Lattice Boltzmann Method using GPI

Faisal Shahzad, **Markus Wittman, Moritz Kreutzer, Thomas Zeiser, Georg Hager and Gerhard Wellein**

Erlangen Regional Computing Center

University of Erlangen-Nuremberg, Erlangen, Germany

Outline

- Motivation
- GPI Introduction
- Experimental framework
- SpMVM with GPI, results
- LBM with GPI, results
- Conclusion

Motivation & Introduction

- **Can communication be more efficient?**

- **Can communication be made truly asynchronous?**

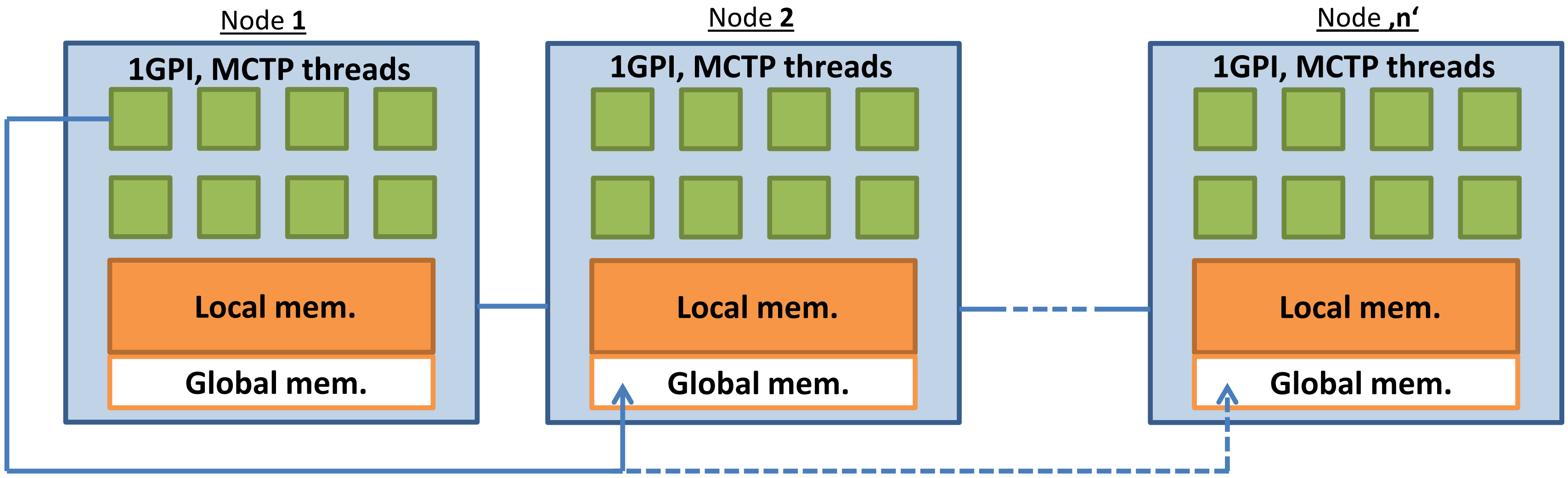
- **GPI: Global address space Programming Interface**
  - A PGAS model API developed by Fraunhofer ITWM, Kaiserslautern, Germany
  - Why GPI (our motivation)
    - GPI targets to incorporate fault tolerant behavior
  - Fault tolerance: Node failures do not crash the whole GPI application

- ***Focus in this paper:* performance comparison between GPI and MPI varients of the program**

GPI Introduction (II)

- **Two memory parts**
  - Local: only local to the GPI process (and its threads)
  - Global: Available to other processes for reading and writing.

- **Hybrid approach (like MPI/OpenMP)**
  - Each Node/NUMA domain can have only one GPI process
  - MCTP/OpenMP threads within node/NUMA domain



Experimental Setup

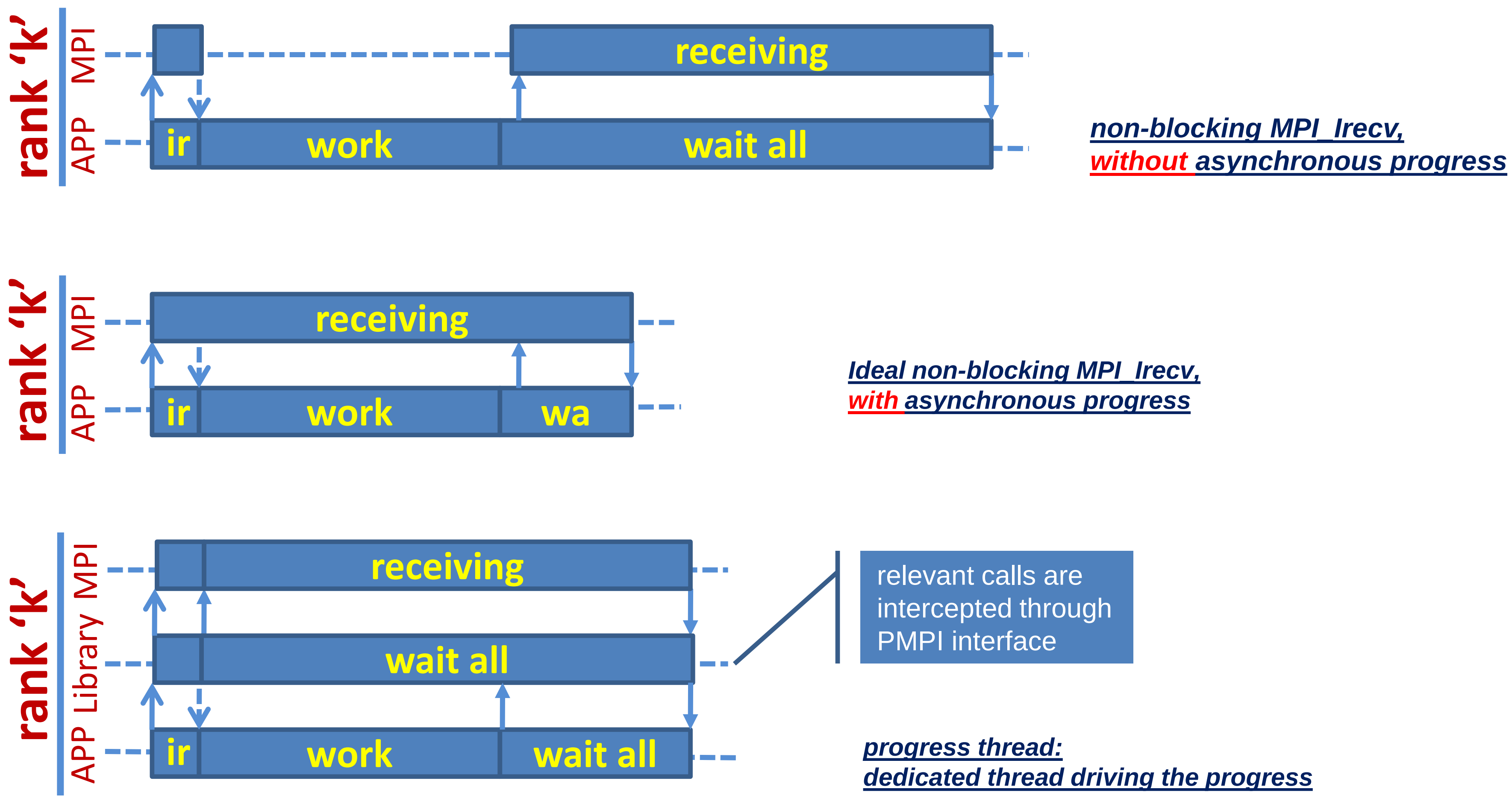
- **Applications:**
  - A prototype CFD solver based on a Lattice Boltzmann Method (LBM)
  - Sparse Matrix Vector Multiplication(SpMVM) algorithm

- **Approaches compared:**
  - Blocking MPI communication
  - Non-blocking MPI communication
  - Non-blocking MPI communication with explicit non-blocking communication support (APSM library; *details next slide*)
  - Synchronous GPI communication
  - Asynchronous GPI communication

- **Cluster:**
  - LiMa (Erlangen) : **500** nodes (Dual socket Intel Xeon 5650 “Westmere”), **QDR Infiniband,** Lustre based PFS Bandwidth ~ 3GB/s

Asynchronous Progress Support for MPI (APSM) Library

- **Motivation ->MPI's non-blocking calls are not necessarily asynchronous**



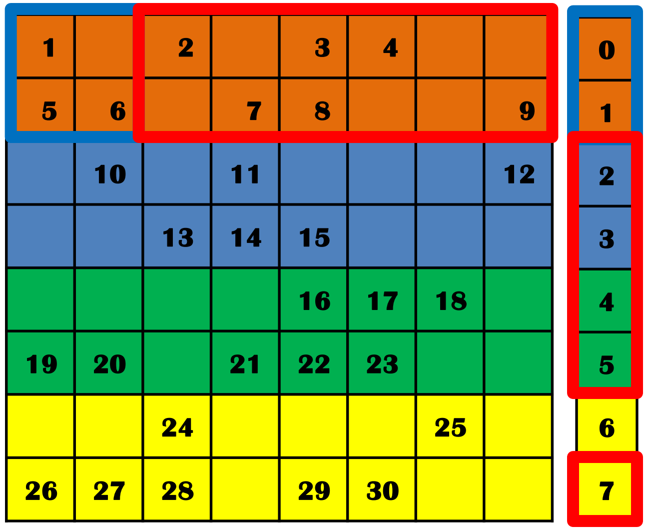
SpMVM with GPI (I)

$$\vec{y} = \mathbf{A}\vec{x}$$

$$y_i = \sum_j (A)_{i,j} \cdot x_j .$$

Where **A** is an nxn matrix and x,y are n dimensional vectors.

- Each process has RHS corresponding to its matrix rows.
- Each process requires remote RHS values e.g. 0th process : 2,3,4,5,7
- Resultant vector can be seen as summation of two components
  1. local-part
  2. remote-part



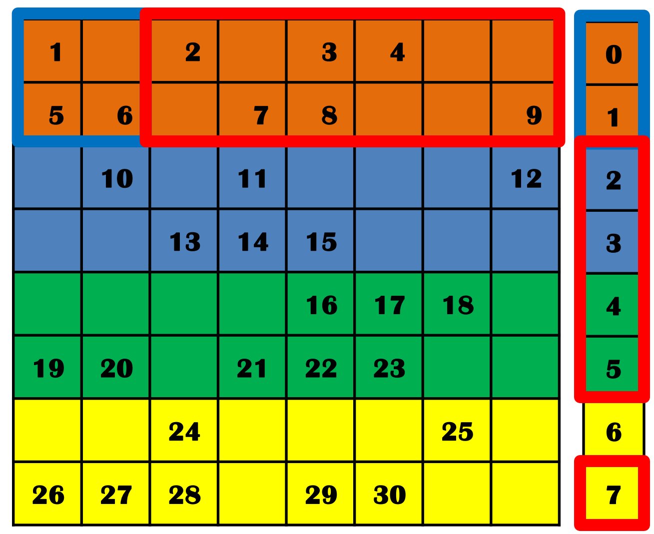
SpMVM with GPI (II)

$$\vec{y} = \mathbf{A}\vec{x}$$

$$y_i = \sum_j (A)_{i,j} \cdot x_j.$$

Where **A** is an nxn matrix and x,y are n dimensional vectors.

- Before each iteration of SpMVM, the required RHS values need to be fetched from remote processes.
- The communication can be hidden behind local part of the SpMVM computation.



SpMVM with GPI (III): Synchronous communication case

```
SpMVM_sync(mat,rhs,res)
{
        rhs->communicate_remote_RHS_blocking();
        spMVM(mat, rhs, res);
}
```

The SpMVM function with synchronous MPI communication

```
SpMVM_sync(mat,rhs,res)
{
        rhs->communicate_remote_RHS_one_sided();
        rhs->wait();
        sync();
        spMVM(mat, rhs, res);
        sync();
}
```

The SpMVM function with synchronous GPI communication

- Each process gathers (read/write) all its required RHS elements before the result vector is computed.
- Local and remote parts are calculated together.
- Synchronization is necessary for one-sided communication (GPI).

SpMVM with GPI (IV): Asynchronous communication case

```
SpMVM_async(mat,rhs,res)
{
        rhs->communicate_remote_RHS_non-blocking()
            ;
        spMVM_local(mat, rhs, res);
        rhs->wait();
        spMVM_remote(mat, rhs, res);
}
```

The SpMVM function with asynchronous MPI communication

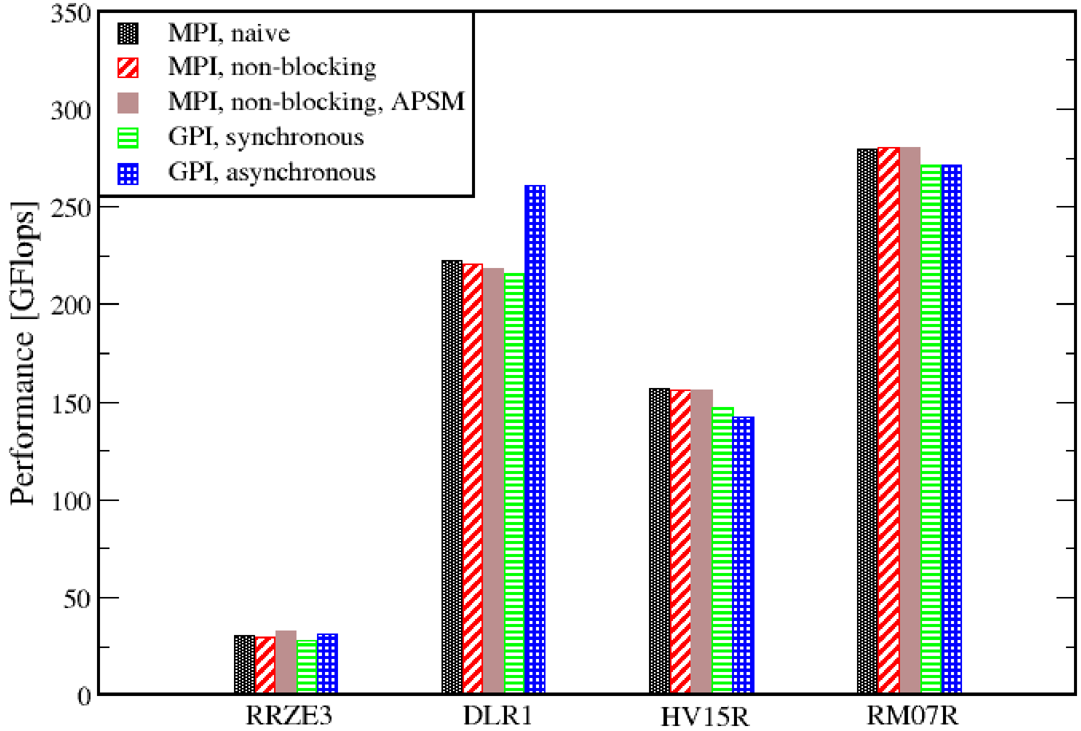
```
SpMVM_async(mat,rhs,res)
{
        rhs->communicate_remote_RHS_one_sided();
        spMVM_local(mat, rhs, res);
        rhs->wait();
        sync();
        spMVM_remote(mat, rhs, res);
        sync();
}
```

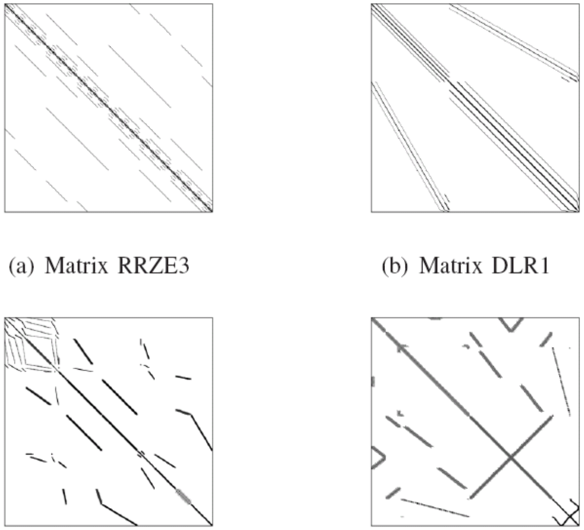
The SpMVM function with asynchronous GPI communication

- Local and remote parts are calculated separately.
- Communication is done asynchronously with the local computation part.
- For one-sided communication (GPI), a barrier is essential before and after communication

SpMVM benchmark results (I)

Matrix	Dimension	Avg. NNZ per row	size in MB
RRZE3	$6.2 \cdot 10^6$	19	1530
DLR1	$2.8 \cdot 10^5$	144	642
HV15R	$2 \cdot 10^6$	140	4545
RM07R	$3.8 \cdot 10^5$	98	602





(a) Matrix RRZE3

(b) Matrix DLR1

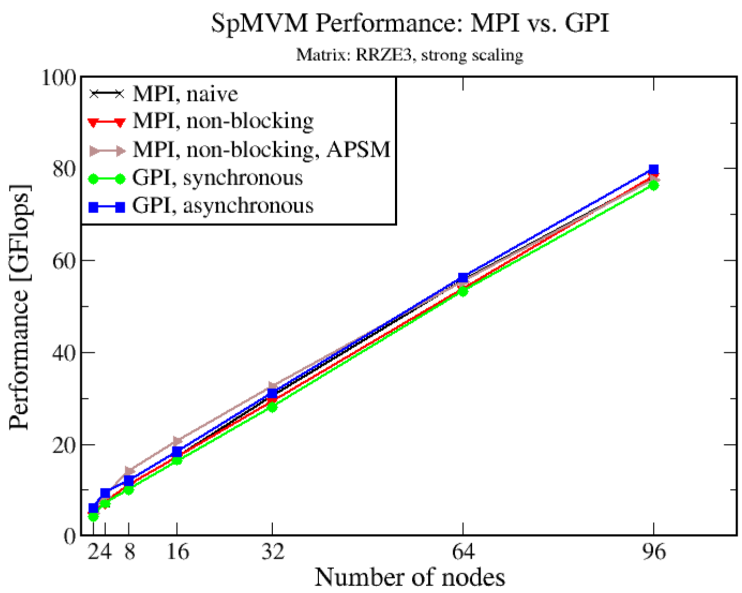
(c) Matrix HV15R

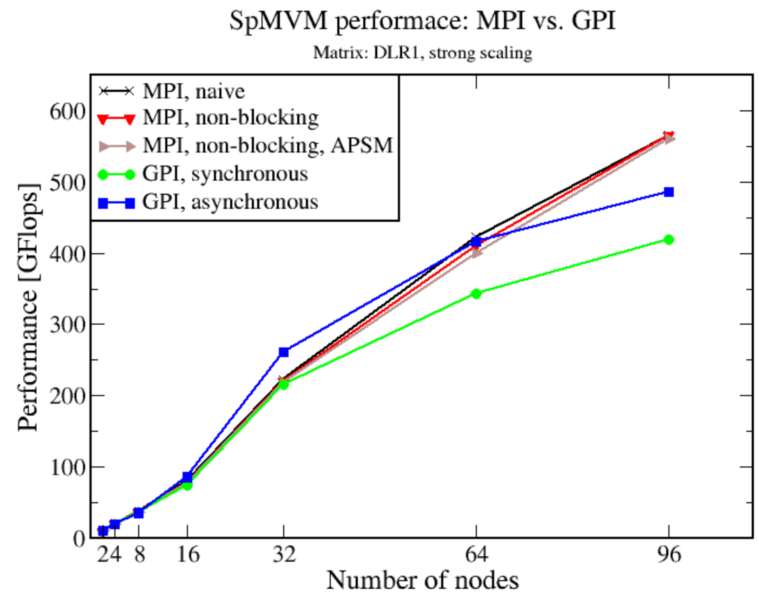
(d) Matrix RM07R

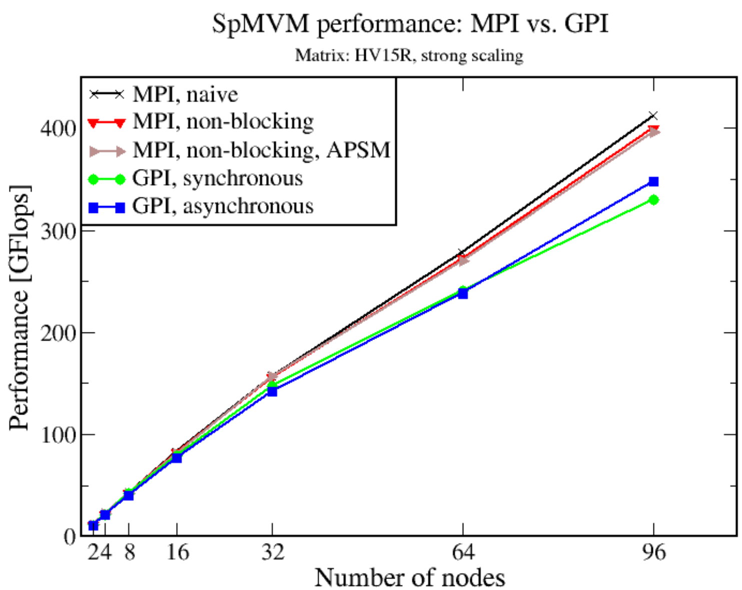
Num. of nodes: 32

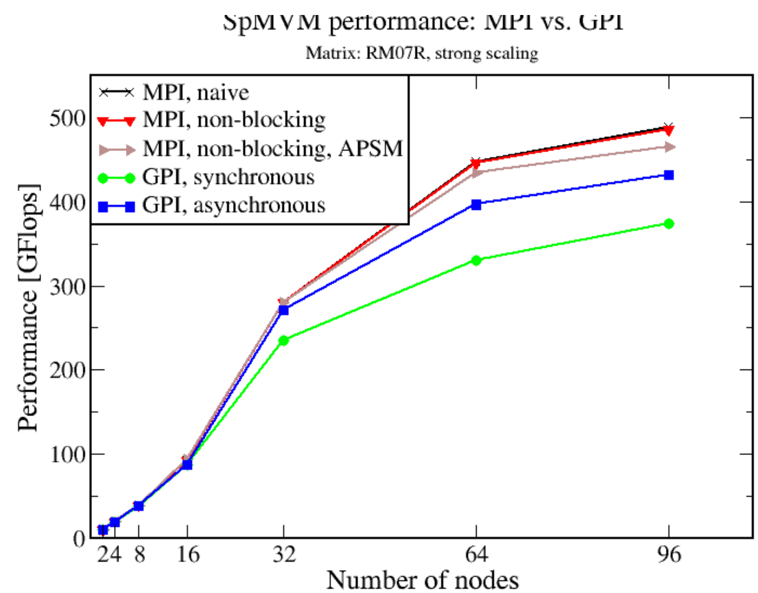
The performance depends on the structure of the matrix.

SpMVM benchmark results (I)





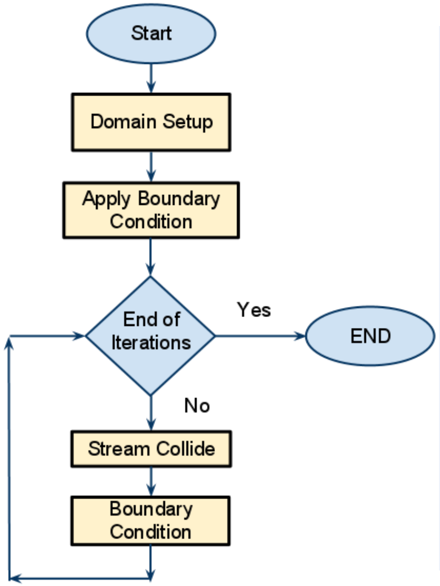


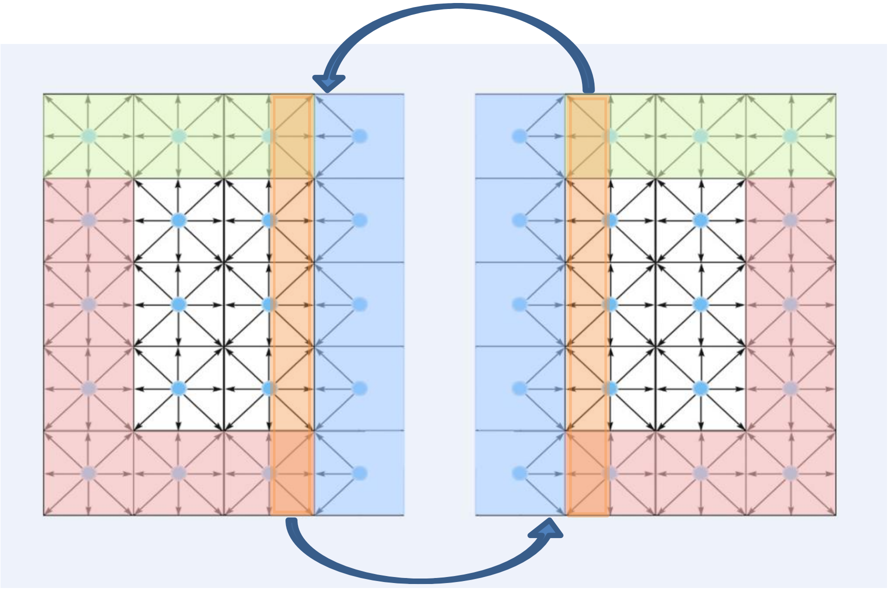


- **A CFD solver based on a simplified kinetic approach derived from the Boltzmann equation**

- **Time and space discretization (D3Q19 Lattice)**
  - Fluid particles are positioned in certain lattice sites
  - May move only in certain, fixed directions
  - The probability of fluid particles to move in certain direction (distribution function) is calculated in each timestep.

- **Algorithm**





LBM with GPI (II)

```
for(int t=1; t <= timesteps; ++t)
{
        update_cells();
        barrier();
        exchange_ghost_cells();
}
```

LBM iteration loop with synchronous communication

- LBM iteration loop with synchronous communication.
- Barrier is essential before one-sided communication is performed.

```
for(int t=1; t <= timesteps; ++t)
{
        update_boundary_cells();
        exchange_ghost_cells_begin();
        update_inner_cells();
        exchange_ghost_cells_end();
}
```

LBM iteration loop with asynchronous communication

- LBM iteration loop with asynchronous communication.
- Communication is overlapped with computational step on the inner cells.
- Barrier is essential after one-sided communication.

LBM with GPI (III)

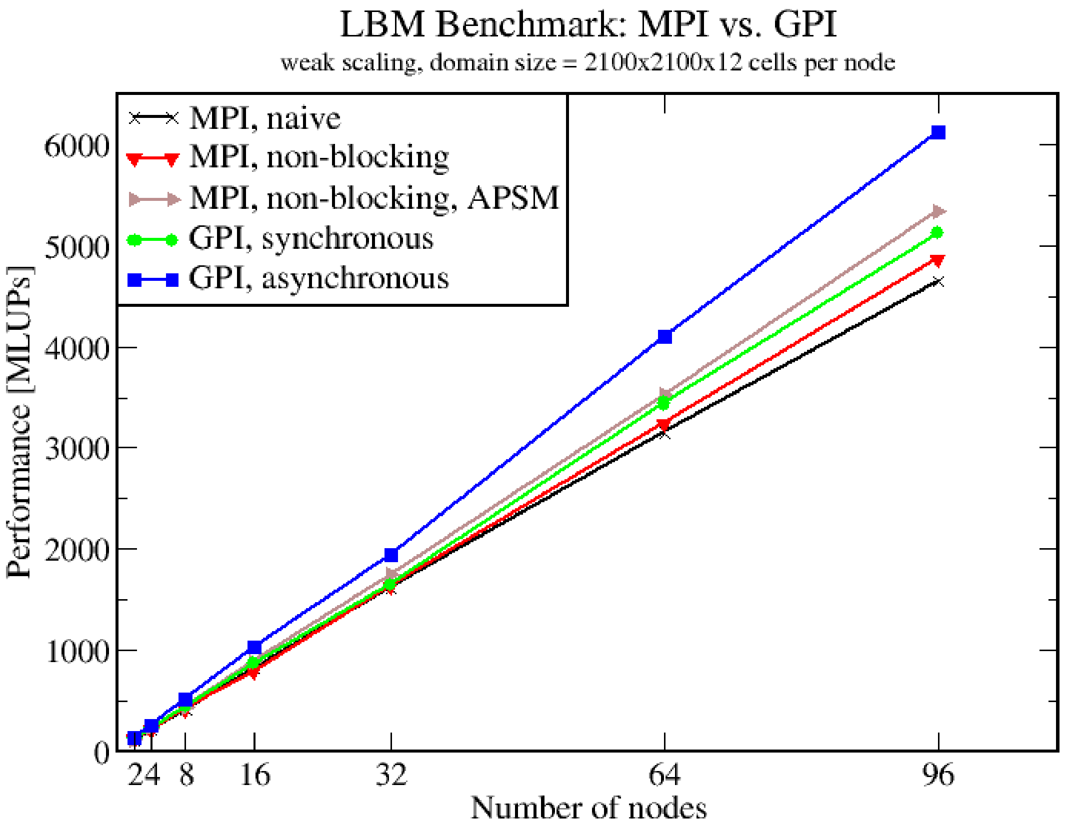
```
for(int t=1; t <= timesteps; ++t)
{
        update_boundary_cells();
        copy_boundary_cells_to_comm_buffer();
        boundary_ready[local_rank][EAST] = 1;
        boundary_ready[local_rank][WEST] = 1;

        wait_for(boundary_ready[remote_rank_east][
            WEST] == 1);
        wait_for(boundary_ready[remote_rank_west][
            EAST] == 1);

        read_remote_boundary_cells();

        boundary_ready[remote_rank_east][WEST]=0;
        boundary_ready[remote_rank_west][EAST]=0;

        update_inner_cells();

        wait_for(boundary_ready[local_rank][EAST]
            == 0);
        wait_for(boundary_ready[local_rank][WEST]
            == 0);
}
```

LBM iteration loop with asynchronous communication and relaxed synchronization

- LBM iteration loop with asynchronous communication with relaxed sync.
- Flags („boundary_ready“) are used to for inter-process synchronization to avoid the need of a global synchronization.
- Communication is overlapped with computational step on the inner cells.

LBM benchmark results (I)

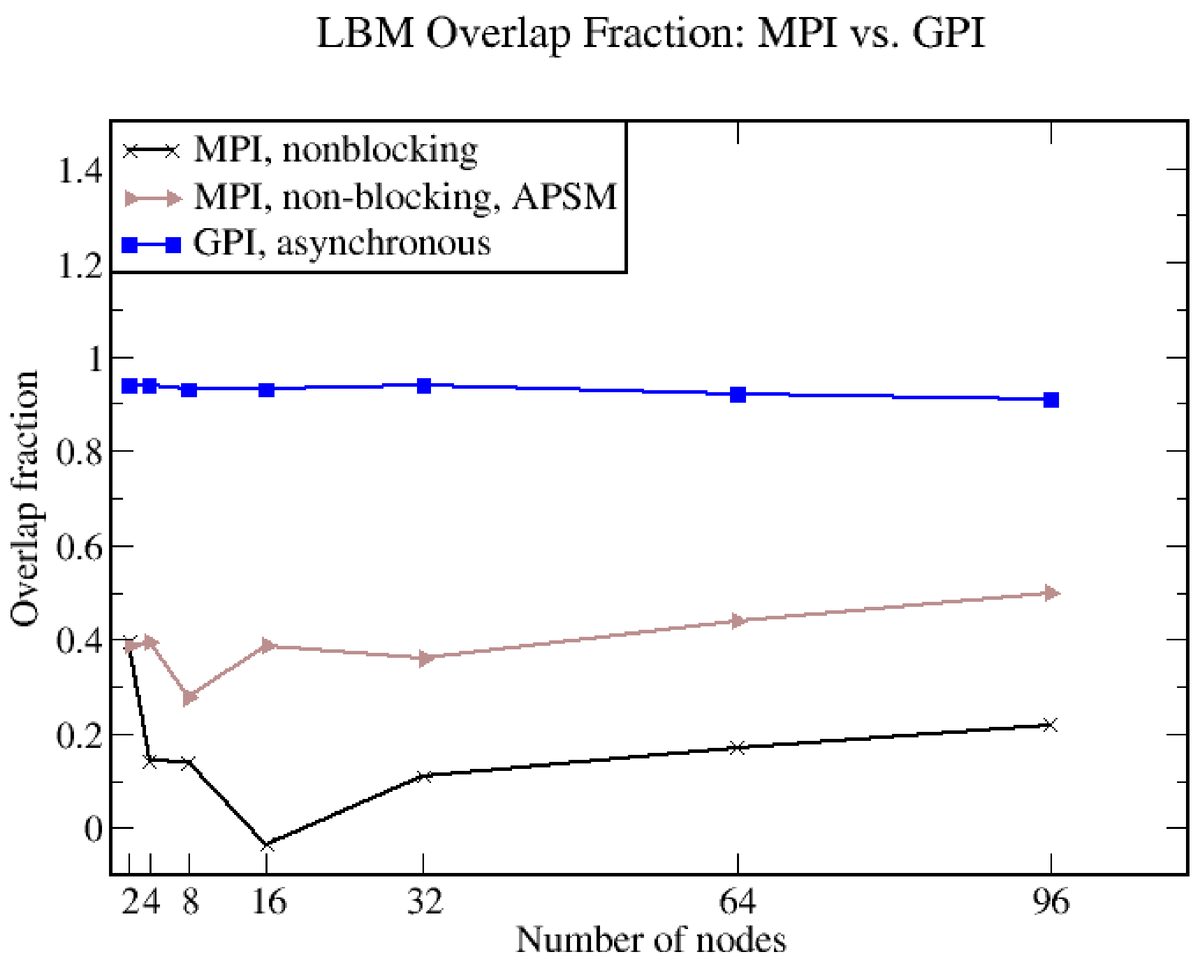
GPI: Weak scaling
Cluster: LiMa



Weak scaling: Problem size (domain) increases with the same factor as number of nodes.

- For 96 nodes, the best async. GPI case performes 30% better than naive MPI case.

LBM benchmark results (II)



<u>*Communication overlap fraction*</u>

$$\mu = \frac{T_{sync.} - T_{async.}}{T_{comm}}$$

T_{sync} = Runtime with sync. comm.

T_{async} = Runtime with async. comm.

T_{comm} = Communication time

Conclusion

- **We compared LBM and SpMVM implementations of GPI with their respective MPI implementations.**
- **Algorithms must be adapted to leverage the PGAS languages.**
- **For LBM lazy synchronization was possible.**
- **For SpMVM a global synchronization can not be avoided.**

Comments on the ease of implementation

- **LBM:**
  - A naive conversion from MPI to GPI code is very simple.
  - Only additional optimizations require more efforts.
  - Overlap between communication & computation.
  - Avoiding global synchronizations.

- **SpMVM:**
  - Setting up the communication structure is comparatively more time consuming.
  - The main communication routines are simple but require gobal synchronization (which limits performance).

Thank you!

Questions?