

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

Faisal Shahzad



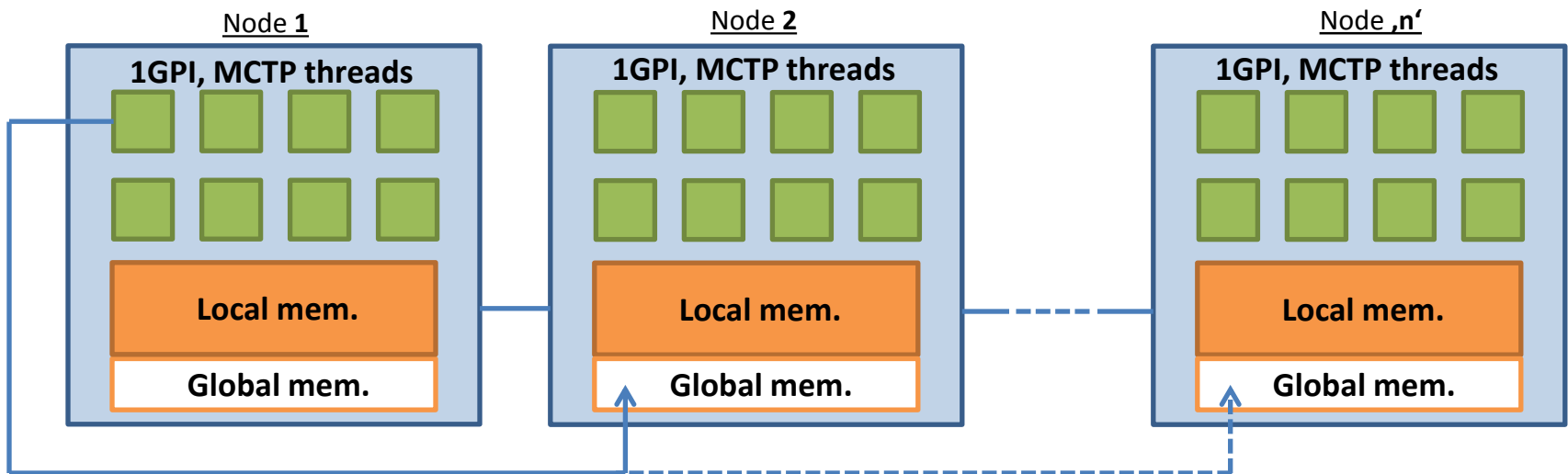
- **Motivation**
- **GPI Introduction**
- **Experimental framework**
- **SpMVM with GPI, results**
- **LBM with GPI, results**
- **Conclusion**



- **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 talk:* performance comparison between GPI and MPI variants of the program**



- **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





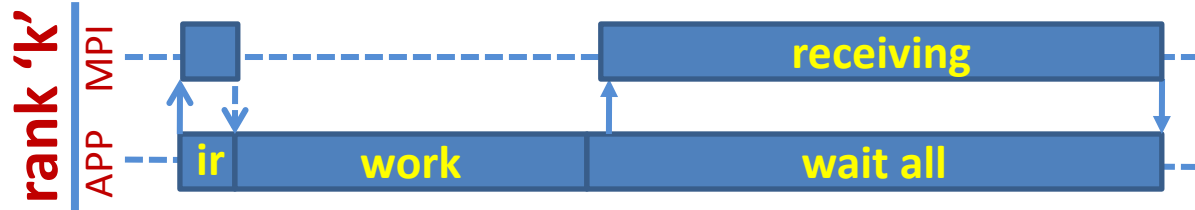
- **Applications:**
 - A prototype CFD solver based on a Lattice Boltzmann Method (LBM)
 - Sparse Matrix Vector Multiplication (SpMVM) algorithm

- **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

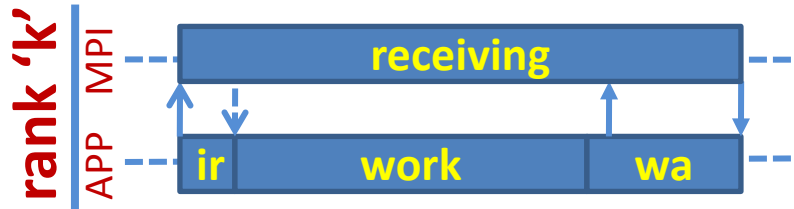
- **Cluster:**
 - LiMa (Erlangen) : **500** nodes (Dual socket Intel Xeon 5650 “Westmere”), **QDR Infiniband**, Lustre based PFS Bandwidth ~ 3GB/s



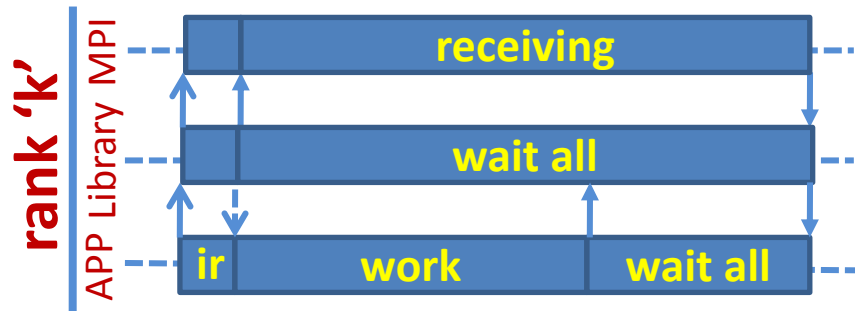
- Motivation -> MPI's non-blocking calls are not necessarily asynchronous



non-blocking MPI Irecv,
without asynchronous progress



Ideal non-blocking MPI Irecv,
with asynchronous progress



relevant calls are intercepted through PMPI interface

progress thread:
dedicated thread driving the progress



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

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

Where \mathbf{A} is an $n \times n$ 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. 0^{th} process : 2,3,4,5,7
- Resultant vector can be seen as summation of two components
 1. local-part
 2. remote-part

1		2		3	4			0
5	6		7	8			9	1
	10		11				12	2
		13	14	15				3
				16	17	18		4
19	20		21	22	23			5
		24				25		6
26	27	28		29	30			7



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

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

Where \mathbf{A} is an $n \times n$ 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.

1		2		3	4			0
5	6		7	8			9	1
	10		11				12	2
		13	14	15				3
				16	17	18		4
19	20		21	22	23			5
		24				25		6
26	27	28		29	30			7



```
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_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();
}
```

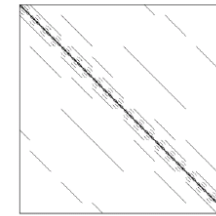
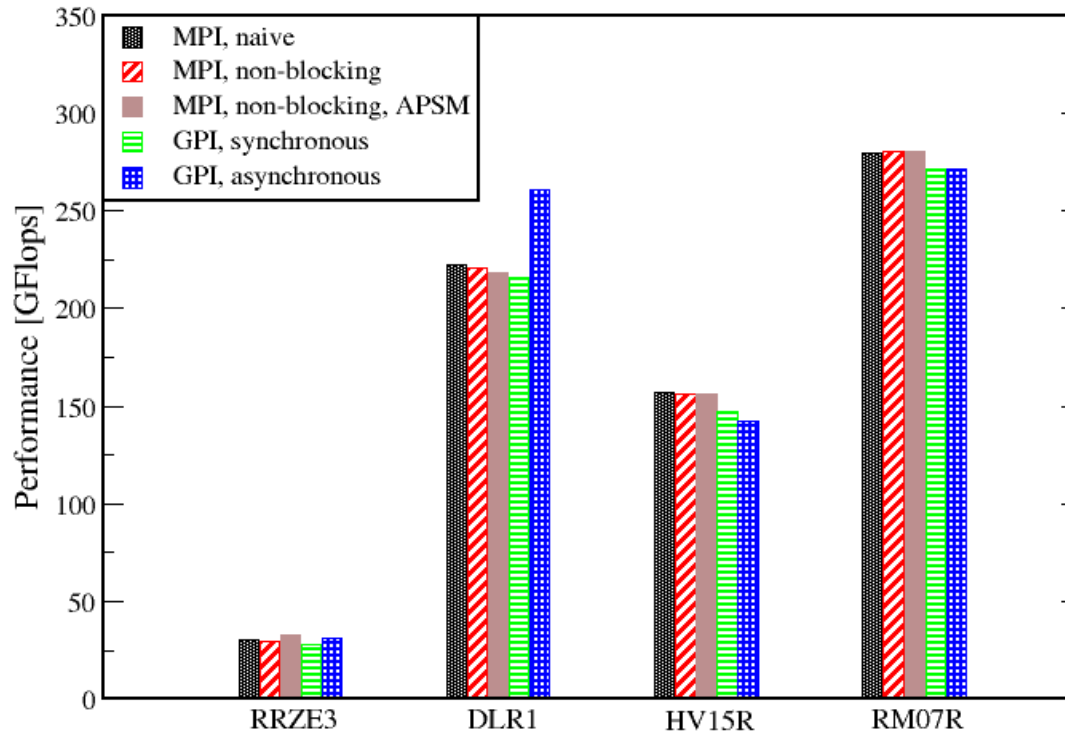
The SpMVM function with asynchronous GPI communication

- 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

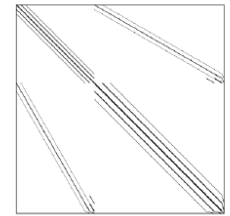
SpMVM benchmark results (I)



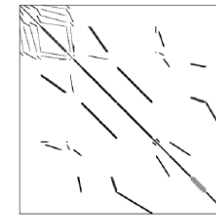
Matrix	Dimension	Avg. NNZ per row	size in MB
RRZE3	$6.2 \cdot 10^6$	19	1530
DLR1	$2.8 \cdot 10^5$	144	642
HV15R	$2 \cdot 10^6$	140	4545
RM07R	$3.8 \cdot 10^5$	98	602



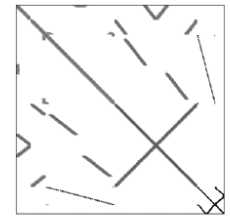
(a) Matrix RRZE3



(b) Matrix DLR1



(c) Matrix HV15R



(d) Matrix RM07R

Num. of nodes: 32

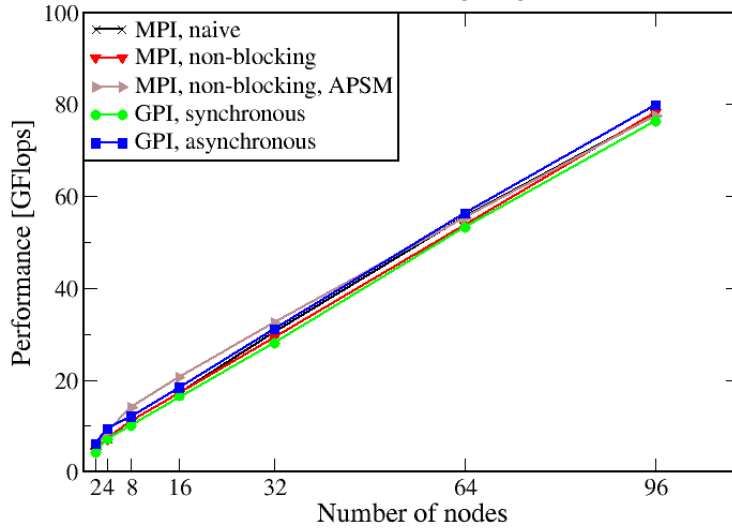
The performance depends on the structure of the matrix.

SpMVM benchmark results (I)



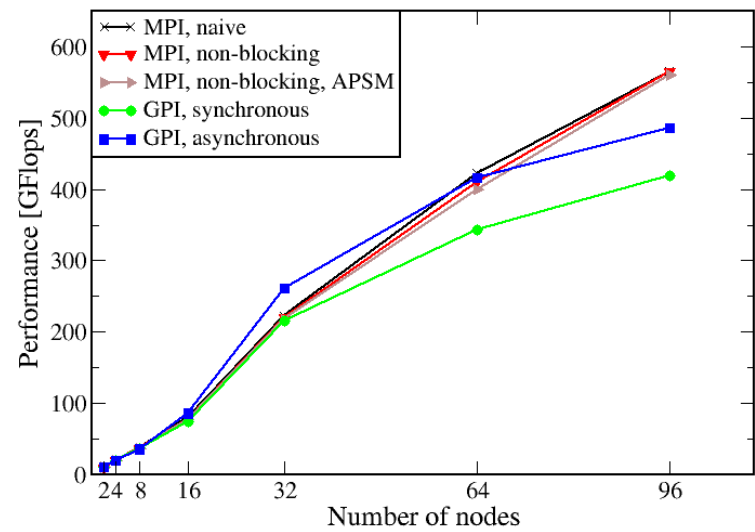
SpMVM Performance: MPI vs. GPI

Matrix: RRZE3, strong scaling



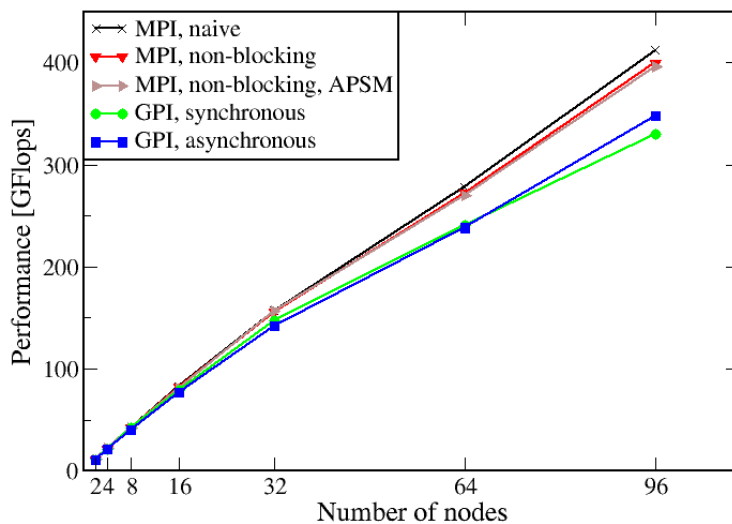
SpMVM performance: MPI vs. GPI

Matrix: DLRI, strong scaling



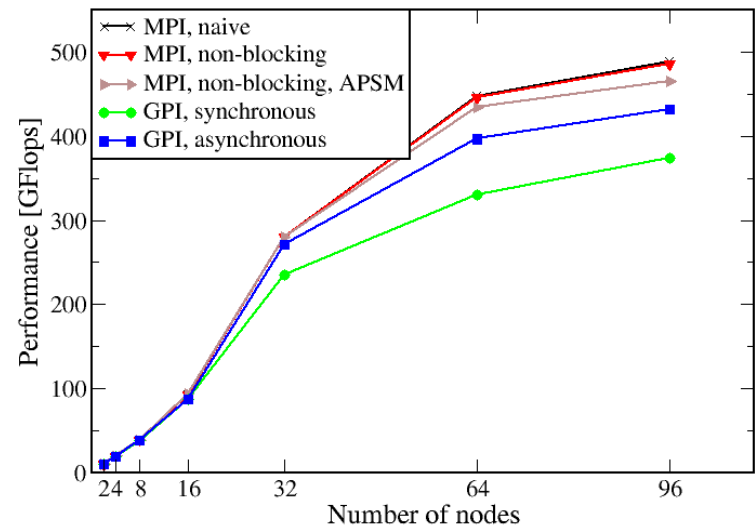
SpMVM performance: MPI vs. GPI

Matrix: HV15R, strong scaling



SpMVM performance: MPI vs. GPI

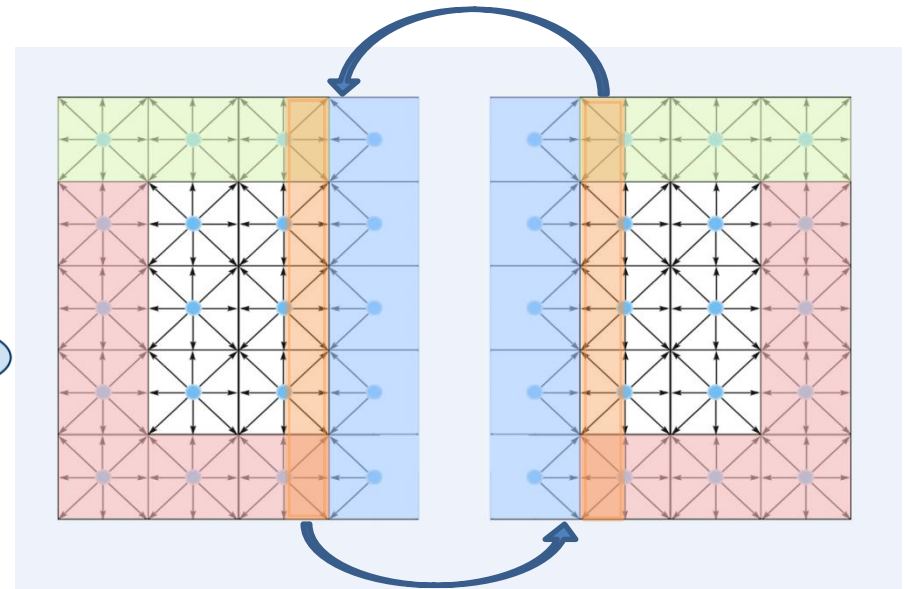
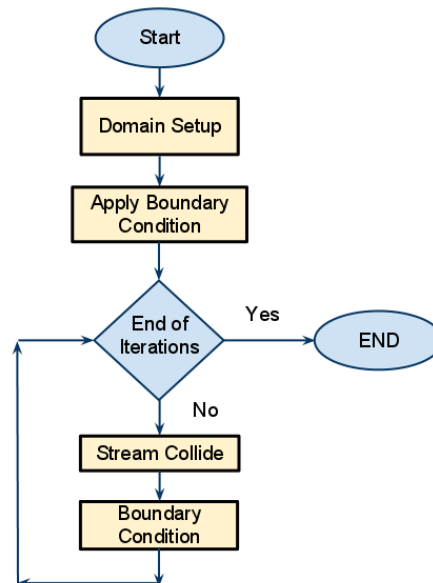
Matrix: RM07R, strong scaling





- 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





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

LBM iteration loop with synchronous communication

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



```
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

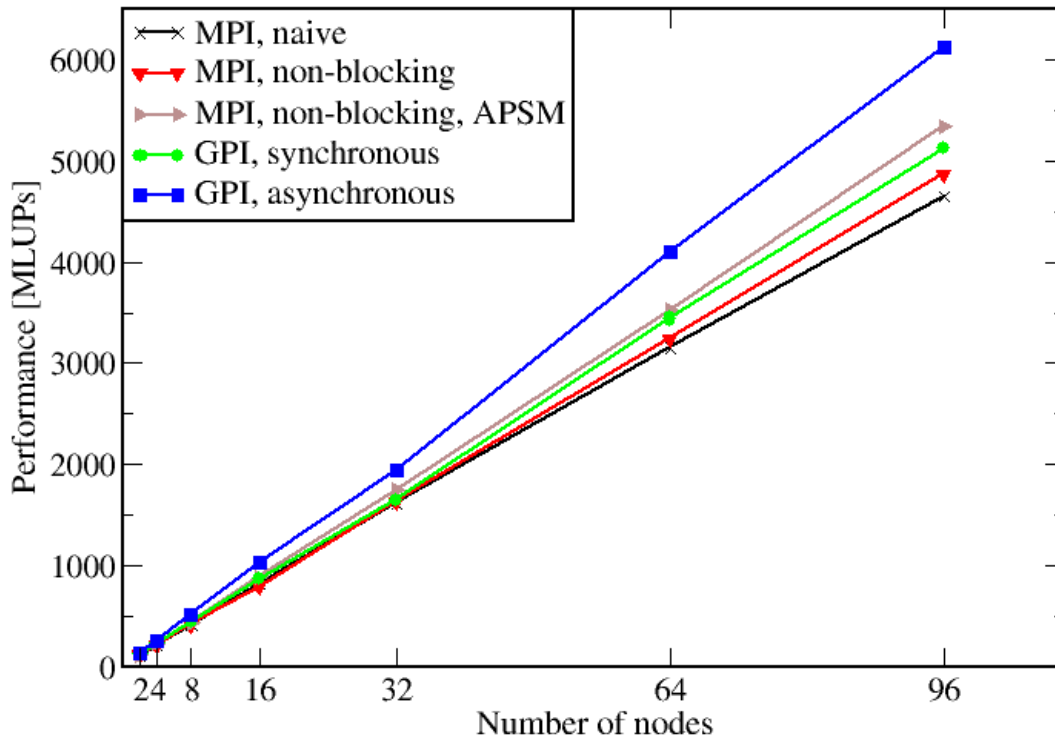
- LBM iteration loop with asynchronous communication with relaxed sync.
- Flags („boundary_ready“) are used to for inter-process synchronization to avoid the need of a global synchronization.
- Communication is overlapped with computational step on the inner cells.



GPI: Weak scaling Cluster: LiMa

LBM Benchmark: MPI vs. GPI

weak scaling, domain size = 2100x2100x12 cells per node

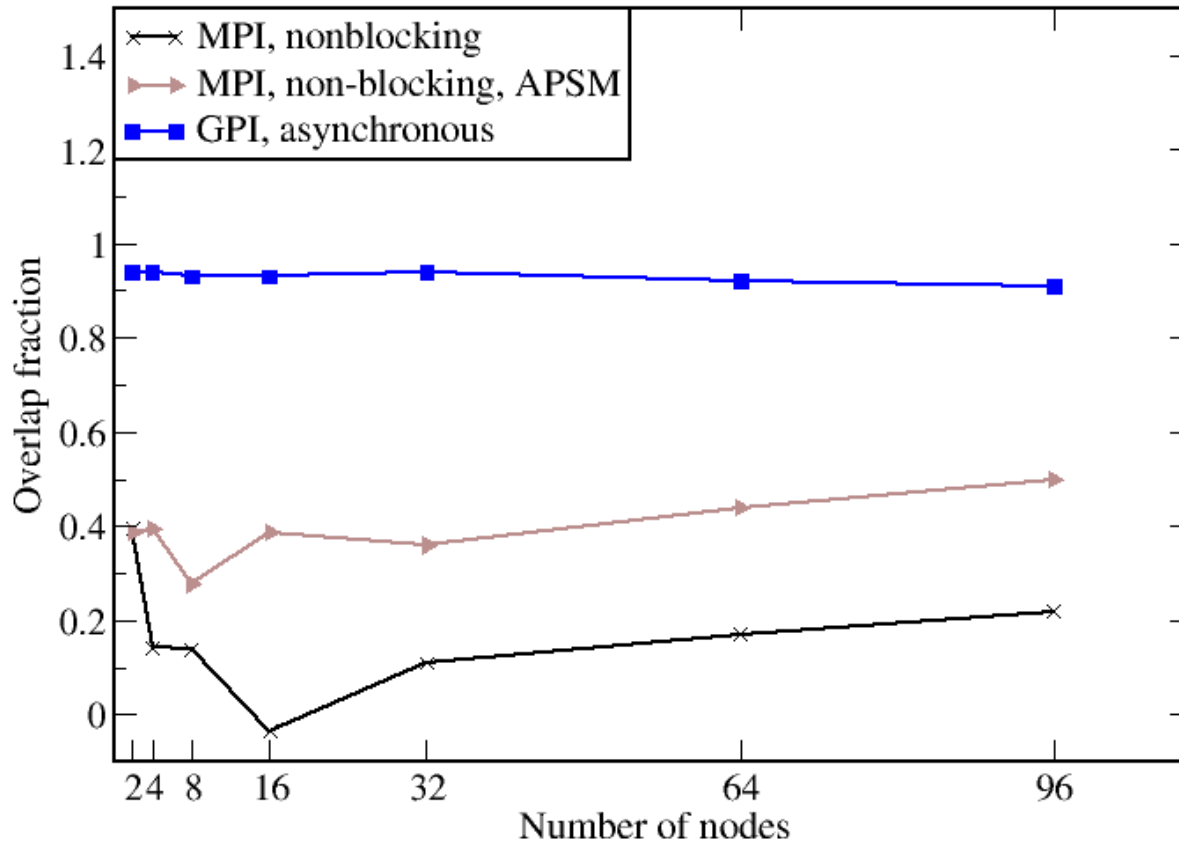


Weak scaling: Problem size (domain) increases with the same factor as number of nodes.

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



LBM Overlap Fraction: MPI vs. GPI



Communication overlap fraction

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

T_{sync} = Runtime with sync. comm.

T_{async} = Runtime with async. comm.

T_{comm} = Communication time



- **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.**



- **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 global synchronization (which limits performance).



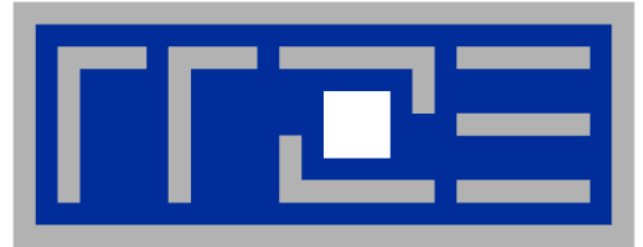
- **LBM:**
 - Synchronization between neighbours done by GASPI notification messages.

- **SpMVM:**
 - Done: GPI -> GPI2.0
 - TODO: Use of notification messages to avoid global synchronization routines.



Thank you!

Questions?



Fault Tolerance with GPI-2

Faisal Shahzad



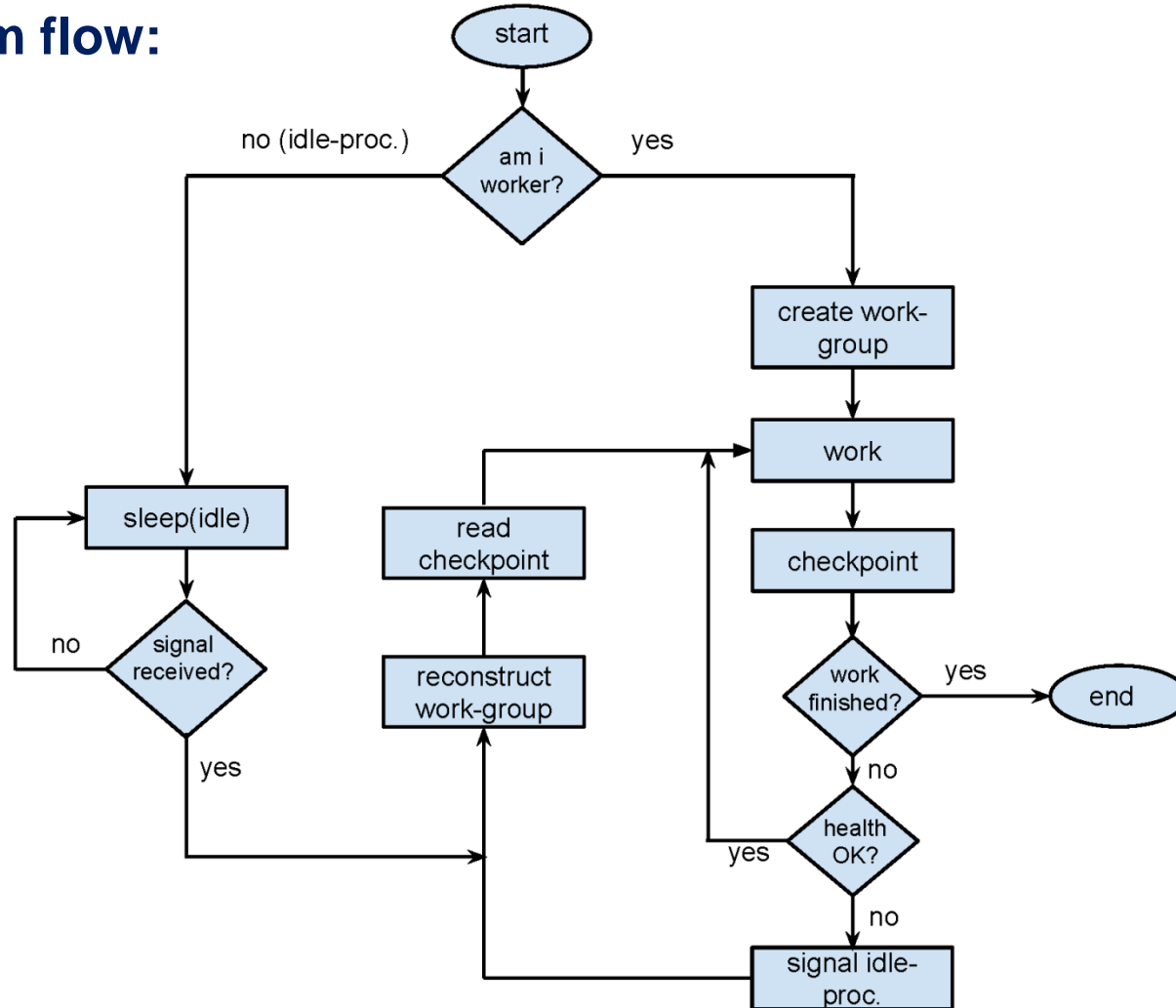
- **Toy FT implementation with LBM**
- **DEMO**
- **Questions and discussion**



- **Idea:**
 - Reserving ,m' idle processes at the start of program setup.
 - Program utilizes ,n-m' processes for work (work-group)
 - In case of a failed process in ,work-group', a process from ,idle-group' is added to ,work-group'.
 - Processes in newly established ,work-group' restart the work from last checkpoint.



■ Program flow:





■ Health check routine

```
void send_msg_to_check_state(gaspi_state_vector_t health_vec, gaspi_rank_t *avoid_list){
    for(int i=0; i<numprocs; ++i){
        if(avoid_list[i]!=1)
        {
            ASSERT(gaspi_write(0, 0, i, 0, 0, sizeof(int), 0, GASPI_BLOCK));
        }
    }
    gaspi_return_t retval;
    retval = gaspi_wait(0, GASPI_BLOCK);
    ASSERT(gaspi_state_vec_get(health_vec));

    for(int i=0; i<numprocs; ++i){
        if(health_vec[i]==1){
            avoid_list[i]=1;
        }
    }
    if(myrank==0)    print_health_vec(health_vec);
}
```



- **Health check: simultaneous call by all processes?**
 - Race condition if not simultaneous
 - Multiple failures at the same time? (1-node, 2-processes failure)
- **Communication with ,gaspi_timeout_t':**
 - Time > 100000... call seems blocking? (at failure, process does not return from gaspi_read)
 - Usage with collective communication ? (at failure, seem to be GASPI_BLOCK currently)
 - How can an optimized ,gaspi_timeout_t' be determined.