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 variants 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

---

**Motivation**

- MPI’s non-blocking calls are not necessarily asynchronous.

**Diagram:**

- Rank ‘k’:
  - APP: receiving
  - MPI: ir, work, wait all

- Ideal non-blocking MPI_Irecv, with asynchronous progress

- Relevant calls are intercepted through PMPI interface

- Progress thread: dedicated thread driving the progress
SpMVM with GPI (I)

\[ \vec{y} = A \vec{x} \]

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

- 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

Where \( A \) is an nxn matrix and \( x,y \) are n dimensional vectors.
SpMVM with GPI (II)

\[ \vec{y} = A \vec{x} \]

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

- 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.

Where \( A \) is an \( nxn \) matrix and \( x,y \) are \( n \) dimensional vectors.
SpMVM with GPI (III): Synchronous communication case

The SpMVM function with synchronous MPI communication

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

- 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).

The SpMVM function with synchronous GPI communication

```c
SpMVM_sync(mat, rhs, res)
{
    rhs->communicate_remote_RHS_one_sided();
    rhs->wait();
    sync();
    spMVM(mat, rhs, res);
    sync();
}
```
SpMVM with GPI (IV): Asynchronous communication case

The SpMVM function with asynchronous MPI communication

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

- Local and remote parts are calculated separately.
- Communication is done asynchronously with the local computation part.

The SpMVM function with asynchronous GPI communication

```c
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();
}
```

- For one-sided communication (GPI), a barrier is essential before and after communication.
The performance depends on the structure of the matrix.

Num. of nodes: 32

PGAS implementation of SpMVM and LBM using GPI
SpMVM benchmark results (I)

SpMVM Performance: MPI vs. GPI
Matrix: RRZE3, strong scaling

SpMVM performance: MPI vs. GPI
Matrix: HV15R, strong scaling

SpMVM Performance: MPI vs. GPI
Matrix: DLR1, strong scaling

SpMVM Performance: MPI vs. GPI
Matrix: RM07R, strong scaling
LBM with GPI (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
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.
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 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 performs 30% better than naive MPI case.
LBM benchmark results (II)

Communication overlap fraction

\[ \mu = \frac{T_{\text{sync.}} - T_{\text{async.}}}{T_{\text{comm}}} \]

- \( T_{\text{sync.}} \) = Runtime with sync. comm.
- \( T_{\text{async.}} \) = Runtime with async. comm.
- \( T_{\text{comm}} \) = 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 cannot 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 global synchronization (which limits performance).
Thank you!

Questions?