Accelerating lattice Boltzmann method with C++ standard language parallel algorithm

Ziheng Yuan
Graduate School of Engineering, The University of Tokyo

Takashi Shimokawabe
Information Technology Center, The University of Tokyo

1. Background

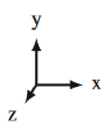
- The lattice Boltzmann method is a fluid simulation algorithm that is easily parallelizable and widely used in industry field.
- C++ standard language parallel algorithm (stdpar) provides programmers with parallel algorithms within the standard library, reducing the complexity of developing parallel programs. NVIDIA C++ compiler is one of the compilers that support C++ stdpar. This experiment is conducted on an NVIDIA GPU.
- In practical simulations, it is often necessary to use multiple GPUs for computation. Currently, C++ stdpar does not provide functions for inter-GPU communication. How to utilize existing libraries, such as MPI or CUDA, to implement multi-GPU computing while retaining the advantages of C++ is the main focus of this presentation

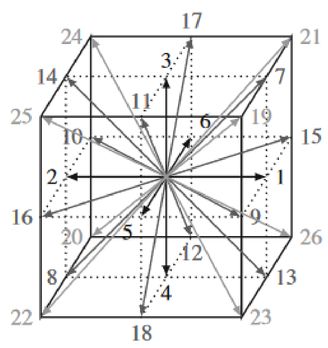
2. Lattice Boltzmann method

Lattice Boltzmann method (LBM) is one of the computation methods for incompressible viscous fluids. This method calculates the collisions and translations of the particles using velocity distribution function.

$$\begin{cases} f_i^{eq} = E_{i\rho}\left[1 + 3e_i u + \frac{9}{2}(e_i u)^2 - \frac{3}{2}u^2\right] & 1 \\ f_i(x + e_i\Delta x, t + \Delta t) = f_i(x,t) + \Omega_i(x,t) & 2 \\ \rho = \sum_{i=1}^{N} f_i \qquad u = \frac{1}{\rho}\sum_{i=1}^{N} e_i f_i & 3 \end{cases}$$

step 1 : calculate equilibrium distribution
step 2 : calculate collision and streaming
step 3 : calculate density and velocity





This experiment uses IB-LBM(Immersed Boundary Lattice Boltzmann Method). In order to improve accuracy, the cumulant model is used in the calculation of collision step.

3. C++ Standard parallel algorithm

Firstly provided by C++ 17 standard.
Parallelizing the algorithms in C++ standard library through introducing concept named execution policy.
Three different execution policies:

- std::execution::par
- std::execution::par_unseq
- std::execution::seq

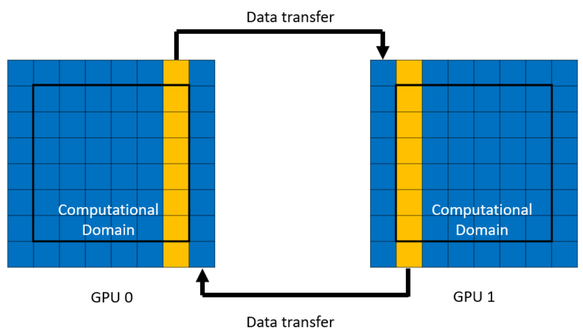
Inside the execution policies, std::par and std::par_unseq will provide multiple threads for the algorithm, which will provide parallel execution.
Frequently used algorithms support execution policy:

- std::transform()
- std::reduce()
- std::for_each_n()

C++ standard language parallel algorithm is supported by the NVIDIA C++ compiler, allows to compile and run the code on GPUs.

4. Multiple GPU computing

When computing with multiple GPUs, the computational domain needs to be divided into suitable small blocks, each stored in the memory of different GPUs, and exchange edge data with neighboring GPUs. Many factors constrain the speed of multi-GPU computing, including the communication overhead between GPUs and synchronization overhead, among others. Compared to exchanging data via the CPU, using NVLink to directly exchange data between GPUs is more efficient.

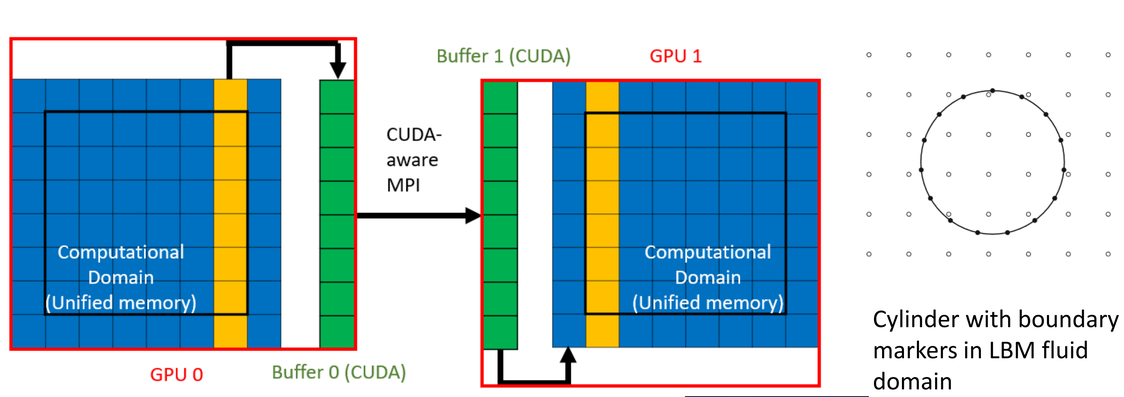


However, C++ stdpar cannot directly utilize NVLink for communication. Therefore, it requires declare buffers in GPU memory through CUDA to transfer the data needing communication into these buffers, before utilizing NVLink for communication.

5. Implementation

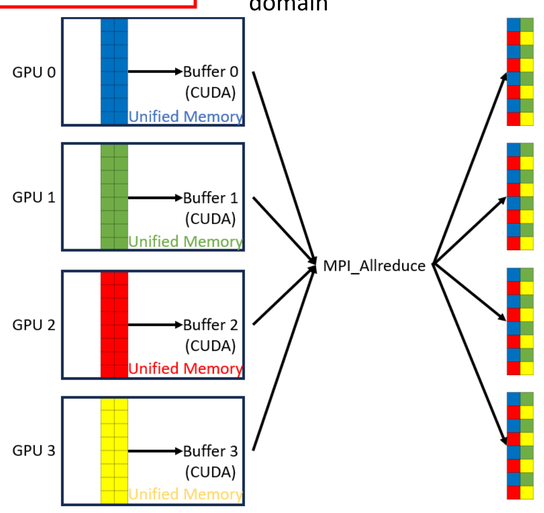
5.1 Naïve method
Transfer boundary data stored in unified memory between GPUs directly
Data must pass through CPU

5.2 Applying exchange buffer
Avoid transfer unified memory data directly
Using cudaMalloc function to allocate buffer space on GPU device



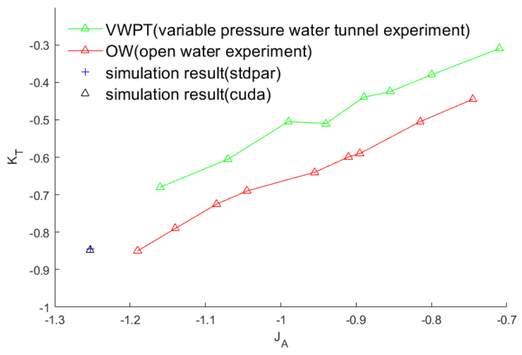
Cylinder with boundary markers in LBM fluid domain

5.3 Applying Allreduce buffer
The IB-LBM method requires the computation of body force by summing up the forces on the boundary markers of the object.
MPI_Allreduce() is applied to calculate the body force when boundary points are located on multiple GPUs. CUDA buffer can increase the efficiency of reduce operation.



6. Experiment

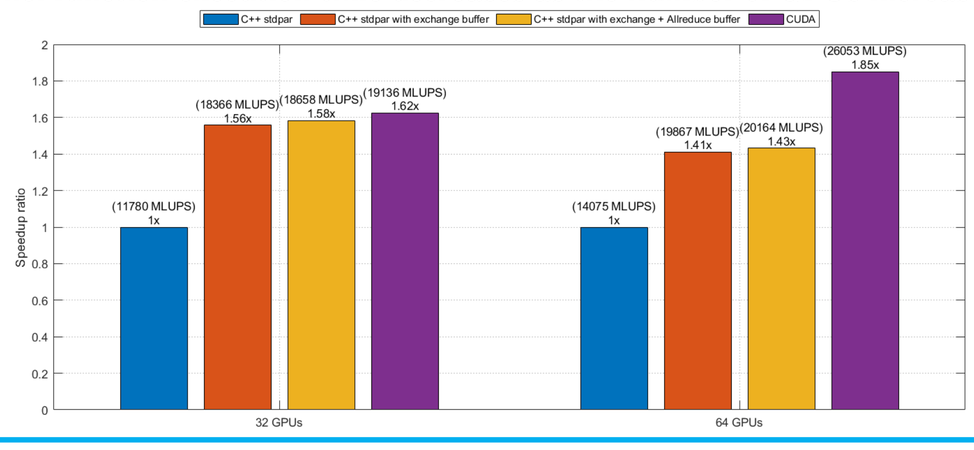
Benchmark: propeller simulation
Size: 700*700*2560
Platform: Wisteria Aquarius UTokyo
Computation node:
CPUs: Intel Xeon Platinum 8360Y *2
GPUs: NVIDIA A100 *8
Performance Comparation unit: MLUPS (Mega Lattice Update Per Second)



6.1 Simulation result:
By comparing the relationship between advance ratio J_A and mean thrust coefficient K_T, it can be observed that the simulation results are meaningful. The result from stdpar and CUDA are almost identical.

6.2 Performance:
The results are as shown in the figure below; after adding the exchange buffer, the performance of the program has improved compared to the naïve C++ stdpar method. Allreduce buffer could increase the performance slightly more.
The reason for the performance gap between A and B:

- The kernel function execution time of stdpar is always slightly longer than that of CUDA, with a difference of about 2-5 seconds between them.
- Kernel function execution time decreases as the number of GPUs increases..



7. Conclusion

In this poster, we discussed the method of applying C++ stdpar to multi GPU computation. And verified the performance of this method in 32 and 64 GPUs cases.
Exchange buffer and Allreduce buffer can increase the performance of the code.