# PYLBMFLOW: A FULLY PYTHON-ENABLED LARGE-SCALE HIGH-PERFORMANCE 3D LATTICE BOLTZMANN MULTI-PHASE FLOW SOLVER

**CHUANFU XU, YONGGANG CHE AND ZHENGHUA WANG**

National University of Defense Technology (NUDT)
College of Computer
410073 Changsha, P.R. China
e-mail: {xuchuanfu,ygche,zhhwang}@nudt.edu.cn

**Key words:** Python, Parallel Computing, Lattice Boltzmann Method, 3D Multi-Phase Flow.

**Summary.** Python is becoming one of the most popular programming languages in areas such as data science and artificial intelligence. Although Python provides fundamental supports for scientific and engineering computation since its emergence, in HPC applications, Python is typically used as a gluing language orchestrating performance-sensitive calculation kernels written in FORTRAN or C/C++. In this paper, we present a fully Python-enabled large-scale high-performance 3D Lattice Boltzmann multi-phase flow solver PyLBMFlow, and demonstrate the difference of implementation and optimization of Python-based parallel numerical codes with traditional FORTRAN or C/C++ codes. With a range of Python-specific optimizations, we dramatically improve the efficiency of Python numerical kernels by about 100X for a serial run. Furthermore, we present a 3D decomposition method and implement a hybrid MPI+OpenMP parallelization using mpi4py and Cython. Tests for 3D multi-phase problem simulating drop impact with gravity effect using D3Q19 Lattice Boltzmann discretization and Shan-Chen BGK single relaxation time collision model are presented, achieving a weak parallel efficiency of above 90% in going from 64 to 1024 compute nodes.

## 1 INTRODUCTION

As an alternative to classical Computational Fluid Dynamics (CFD) simulations solving Navier-Stokes equations and turbulent models, Lattice Boltzmann Methods (LBM) regard fluids as Newtonian fluids from a microscopic perspective, divide flow field into small lattices (mass points), and simulate fluid evolution dynamics through collision models (lattices collision and streaming)[1]. Currently, LBM has been increasingly used for real-world flow problems with complex geometries and various boundary conditions. Large-scale LBM simulations with increasing resolution and extending temporal range require massive high performance computing resources. In areas of computational science and engineering like CFD, large-scale highly efficient numerical codes on parallel supercomputers usually are written in FORTRAN or C/C++. Unlike those compiled languages, Python is a dynamically interpreted object-oriented language. Due to rich third-party libraries and high development productivity, Python is becoming one of the most popular programming languages in areas such as data science and artificial intelligence. Python also provides fundamental supports for

computational science and engineering with various libraries and tools such as NumPy and SciPy since its emergence. Meanwhile Python offers APIs for popular parallel programming models. For example, mpi4py is a Python library for message passing interface; PyCUDA an PyOpenCL are the corresponding Python libraries for heterogeneous programming modes CUDA and OpenCL. Researchers can use libraries like Cython and numba to optimize the perormance of Python codes, without increasing codes complexity.

Some recent efforts have been delivered to enabe Python-based large-scale high-performance CFD applications. For example, Mikael Mortensen et al[2] implemented direct numerical simulations with Python using several thousands of CPU cores on the Shaheen supercomputer. Peter Vincent et al[3] present a high-order unstructured CFD solver PyFR based on the flux reconstruction method. Their Python implementation achievs a sustained performance of 13.7PFLOP/s on the Titan supercomputer and was selected into the finalist for the 2016 ACM Gorden Bell prize. The implementation and optimization of Python-based parallel numerical codes are quite different with traditional FORTRAN or C/C++ codes. In this paper, we present a fully Python-enabled large-scale high-performance 3D Lattice Boltzmann multi-phase flow solver PyLBMFlow. We design LBM flow data structures and computational kernels with Python NumPy multi-dimensional arrays and universal functions. With a range of Python-specific optimizations and reconstruction of boundary conditions, we dramatically improve the efficiency of Python numerical kernels by about 100X for a serial run. Furthermore, we present a 3D decomposition method and implement a hybrid MPI+OpenMP parallelization using mpi4py and Cython. Tests for 3D multi-phase (liquid and gases) problem (about 17 Billion lattices) simulating drop impact with gravity effect using D3Q19 Lattice Boltzmann discretization and Shan-Chen BGK single relaxation time collision model are presented, achieving a weak parallel efficiency of above 90% in going from 64 to 1024 compute nodes.

## 2 RESULTS AND DISCUSSIONS

The numerical methods and solution procedure of PyLBMFlow can be found in our previous papers[4]. It was originally written in C. Table 1 shows various versions of Python implementation and optimization. In *v0*, we use intrinsic Python list to express multi-dimensional arrays. In *v1*, we use NumPy.narray to replace Python list and thus the universal functions (ufunc) operating multi-dimensional arrays are applied in calculation kernels of *v2*. In *v3*, we change the data layout of the particle distribution function *fn* (a four dimensional array with 19 components for D3Q19 model) from AoS(Array of Structure) to SoA(Structure of Array) to enhance the cache efficiency of memory access. In *v4*, we use Numy.item() and Numy.itemset() to directly access array elements and avoid the transformation of array elements to Python objects. *v5* uses NumPy.where() instead of Python boolean arrays to determine boundary conditions. In *v6*, we redesign the bounceback boundary mechanism. In *v7*, we use Cython to rewrite calculation kernels and finally in v8 runtime compilation optimization is enabled using numba.

Figure.1 shows the comparison of various versions. The speedup is obtained on a single CPU core of the Tianhe-2 supercomputer[5], and *v0* is used as baseline. Each version has a

notable performance enhancement compared to its previous version except *v1* where we only replace Python list with NumPy.narray and don't use Python ufunc to operate arrays. To sum up, we achieve a speedup of above 77 when comparing the final version *v8* to the original version *v0*. We implemented the OpenMP shared memory multi-threading based on v5. Fig.2 shows that an OpenMP speedup of above 10 is achieved on 24 cores of a Tianhe-2 compute node. Fig.3 presents weak scalability test of hybrid MPI+OpenMP parallelization on the Tianhe-2 supercomputer. We fix the problem size to 256*256*256 for each compute node and achieve a parallel efficiency of about 90% when scaling from 64 nodes to 1024 nodes.

| Versions | Description |
|---|---|
| v0 | implementation using intrinsic Python list |
| v1 | replace Python list with NumPy.narray |
| v2 | replace multi-level loops with NumPy ufunc |
| v3 | data structure reconstruction from AoS to SoA |
| v4 | access array elements using NumPy item() |
| v5 | determine boundary conditions using NumPy where() |
| v6 | redesign the bounceback boundary mechanism |
| v7 | rewrite kernels using Cython |
| v8 | runtime compilation using numba |

Table 1 : Various Python implementation and optimization of PyLBMFlow

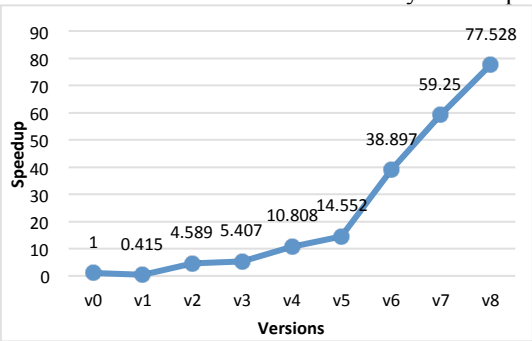


Figure 1: Performance enhancement for various versions

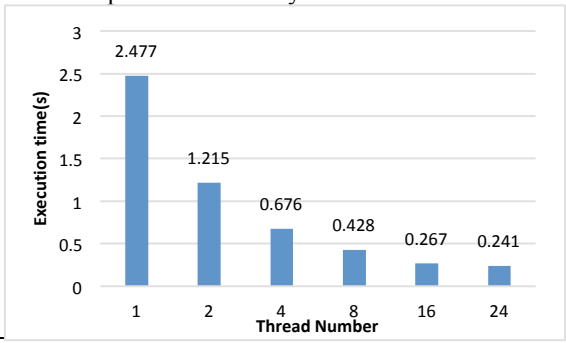


Figure 2: OpenMP speedup

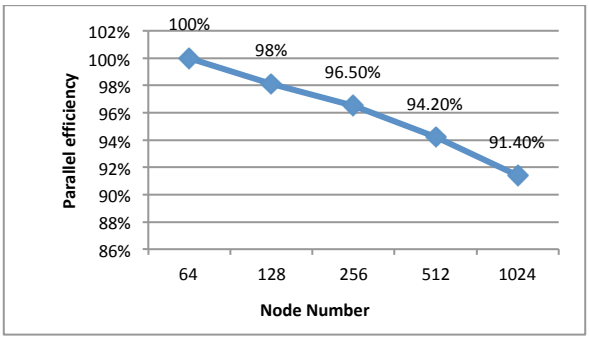


Figure 3: Weak scalability on Tianhe-2

## REFERENCES

[1] S. Succi, R. Benzi, F. Higuera, The lattice Boltzmann equation: A new tool for computational uid-dynamics, Physica D: Nonlinear Phenomena 47 (1) (1991) 219-230.

[2] Mortensen M, Langtangen H P. High performance Python for direct numerical simulations of turbulent flows[J]. Computer Physics Communications, 2016, 203: 53-65.

[3] Peter Vincent, Freddie Witherdeny, Brian Vermeire, Jin Seok Park and Arvind Iyer. Towards Green Aviation with Python at Petascale. SC16 finalist.

[4] D. Li, C. Xu, Y. Wang, Z. Song, M. Xiong, X. Gao, X. Deng, Parallelizing and optimizing large-scale 3D multi-phase ow simulations on the tianhe-2 supercomputer, Concurrency and Computation: Practice and Experience 28 (2015) 1678-169. Doi:10.1002/cpe.3717.

[5] X. Liao, L. Xiao, C. Yang, MilkyWay-2 supercomputer: system and application, Front. Comput. Sci. 8 (3) (2014) 345-356. Doi:10.1007/s11704-014-301-3.