SoftwareX 7 (2018) 88–94

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

LUMA: A many-core, Fluid–Structure Interaction solver based on the Lattice-Boltzmann Method

Adrian R.G. Harwood *, Joseph O'Connor, Jonathan Sanchez Muñoz, Marta Camps Santasmasas, Alistair J. Revell

School of Mechanical, Aerospace and Civil Engineering, The University of Manchester, Sackville Street, M1 3BB, United Kingdom

ARTICLE INFO

Article history:
Received 11 January 2018
Received in revised form 13 February 2018
Accepted 21 February 2018

Keywords:
Lattice-Boltzmann Method
Finite-Element Method
Flow simulation
Fluid–structure interaction

ABSTRACT

The Lattice-Boltzmann Method at the University of Manchester (LUMA) project was commissioned to build a collaborative research environment in which researchers of all abilities can study fluid–structure interaction (FSI) problems in engineering applications from aerodynamics to medicine. It is built on the principles of accessibility, simplicity and flexibility. The LUMA software at the core of the project is a capable FSI solver with turbulence modelling and many-core scalability as well as a wealth of input/output and pre- and post-processing facilities. The software has been validated and several major releases benchmarked on supercomputing facilities internationally. The software architecture is modular and arranged logically using a minimal amount of object-orientation to maintain a simple and accessible software.

Code metadata

Current code version	v1.7.3
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-18-00007
Legal Code License	Apache License 2.0
Code versioning system used	Git
Software code languages, tools, and services used	C++, MATLAB, Python, MPI, OpenMP
Compilation requirements, operating environments & dependencies	Windows/Linux/Mac OS, C/C++ Compiler, MPI, HDF5, LAPACK, VTK
Link to developer documentation/manual	https://github.com/aharwood2/LUMA/wiki
Support email for questions	adrian.harwood@manchester.ac.uk

1. Motivation and significance

Computational Fluid Dynamics (CFD) is the science of simulating the physical behaviour of fluids using computers. It is an essential tool for design, analysis and validation. A suitable set of discretised transport equations governing the physics of the fluid are solved at discrete time-steps to produce a time-varying spatial field of physical quantities such as velocity, density and pressure. Similarly, the structural mechanics of deformable bodies can be modelled using Newton's laws of motion and solved over time, according to the level of detail required.

Fluid–Structure Interaction (FSI) is the coupled analysis of CFD with structural mechanics. Existing software implementations of these solvers vary in complexity, accuracy and speed depending on the modelling strategies chosen. Fluid dynamics solvers are generally based on either Eulerian approaches such as the Finite Volume Method (FVM) [1–4], or Lagrangian methods such as Smoothed Particle Hydrodynamics [5]. The Finite Element Method (FEM) is used widely in engineering for structural modelling [6,7]. The development of LUMA continues to be motivated by the wide range of FSI problems for which simulation is essential. In particular, the modelling of flexible filaments are an area of significant importance in the fields of aerodynamic drag reduction [8], flow control [9] and sensing [10].

In research environments, modelling and simulation software should be a capable platform for the development of new features, while also retaining simplicity in order to facilitate modification and debugging. This accelerates the uptake of the software by students with a range of programming experience and reduces barriers for meaningful contribution in, what can be, short project time frames. Full-featured, open-source engineering software [3,11] makes thorough use of object-orientated features for maximum code reuse and flexibility. However, this software can be difficult to customise for the novice user and can be an obstacle to research. LUMA has been developed for willing contributors with less experience of object-oriented programming (OOP) languages by using a logical, but simplified set of OOP features in class design.

* Corresponding author.
E-mail address: adrian.harwood@manchester.ac.uk (A.R.G. Harwood).

https://doi.org/10.1016/j.softx.2018.02.004

Table 1
Table of features available since LUMA v1.7.

Features since LUMA v1.7	
Git Version Control	Doxygen Documentation
Complete Wiki with Validation Cases	
LBM Pull Kernel	Immersed Boundary Method [22]
Embedded Grid Refinement [13]	MPI Many-Core Parallelisation
Load-Balancing Decomposition	Body Force Calculation [23]
Bounce-Back No-Slip BCs	Interpolated Bounce-Back BCs [24]
Forced Equilibrium Velocity BCs	Regularised Velocity BCs
Regularised Pressure BCs	Symmetry BCs (Specular Reflection)
Periodic BCs	
Body Forcing	Smagorinksy Turbulence Model
SRT/BGK Collision Model	KBC Collision Model
Time-Averaging Facility	Restart Facility
Point Cloud Reader	HDF to VTU Post-Processor
HDF5 Output	Library of Point Cloud Input Files

Inheritance is kept to a single level with functions not related to physics abstracted away into manager classes. Methods and fields are intuitively named and coding standards are imposed to promote clarity over elegance.

2. Software description

Lattice-Boltzmann at The University of Manchester (LUMA) is an initiative which aims to develop novel, physical modelling for complex engineering simulation, underpinned by a flexible, but developer-friendly, many-core accelerated software framework. Development is collaborative , inclusive and centred on a simple version control process with a regular developer release schedule and continuous validation. At the heart of the initiative are applications in aerodynamics, bio-fluids and flow control.

2.1. Flow solver

The choice of flow solver is crucial for achieving desired levels of accuracy while managing complexity. The Lattice-Boltzmann Method (LBM) [12] is an alternative to traditional methods for simulating flow physics and is characterised by its simplicity. Rather than solving the Navier–Stokes as in the majority of CFD software, LBM represents fluid motion at a smaller scale using the Boltzmann equation

$$\left(\frac{\partial}{\partial t} + \vec{e}\cdot\nabla\right) f = \Omega \tag{1}$$

where f is the probability density distribution associated with a group of particles, Ω a local particle collision operator, and $\vec{e}$ a particle velocity. This equation may be discretised in space and time with spatial discretisation based on a uniform lattice of nodal locations which we refer to as a grid. If the velocity space is similarly discretised, such that groups of particles are only allowed to travel along a set of links between spatial nodes, then $f_i \in f$ where f_i is the probability of finding a particle at a given nodal location with velocity $\vec{e}_i$. The discrete, lattice-Boltzmann equation then reduces to

$$f(\vec{x} + \vec{e}_i\delta t, t + \delta t) = \Omega_i(\vec{x}, t). \tag{2}$$

The solution of Eq. (2) proceeds as a two-step process: (1) computation of the particle distributions under the collision operation Ω followed by (2) the convection of these particles to their immediate neighbours along adjoining lattice links. These two steps are referred to as the 'collision' and 'streaming' operations, respectively, in the remainder of this paper.

Variation in the local grid resolution is essential for computational efficiency; fine-grain calculations are only performed in areas of interest. In LUMA, refinement is implemented by embedding grids of higher resolution within other grids. LUMA offers both manual and automatic methods for defining the location of these grids and supports nesting to allow the construction of a grid hierarchy. A spatial and temporal refinement of factor two is applied across each transition. Time stepping proceeds on each grid at its local temporal scale with synchronisation every two cycles between grid pairs. The algorithm of Rohde et al. [13] is used to communicate populations between adjacent grids due to its simplicity and efficiency.

2.2. Structural solver

The structural solver is based on the Finite Element Method (FEM) and has been specifically designed for high aspect ratio structures undergoing large deformations (e.g. flaps, filaments, cilia) [14]. Co-rotational Euler–Bernoulli beam elements are used to represent the structure while geometric non-linearity due to large deformations is incorporated via a non-linear FEM formulation with Newton–Raphson sub-iterations. Second-order time stepping is achieved via the implicit Newmark time integration scheme. Although flexible objects are modelled exclusively using these types of elements at present, the design of the structural solver does allow other elements to be added to LUMA as required. Developers must define and implement suitable mappings for the communication of displacements/forces between the fluid and the elements chosen.

2.3. Fluid–structure coupling

The immersed boundary method (IBM) is used to enforce the no-slip condition on deformable bodies within LUMA [15,16]. The benefit of this approach is that it allows the fluid and structure to be handled separately on their own independent grids – negating the need for regular re-meshing procedures – while also facilitating large structural deformations, which are otherwise challenging with body-fitted grids. Structures are represented as a collection of surface markers with a sphere of influence. Bi-directional force information is passed between markers and fluid points within this sphere each fluid time step. For efficiency purposes, LUMA is only capable of handling cases where the fluid mesh resolution is finer than the structural mesh. This is achieved via a mapping routine between the IBM forces and the FEM structure. To ensure stability across a wide range of input conditions, a strongly coupled sub-iterative scheme is used for the FSI coupling [17]. During the sub-iteration procedure, a fixed relaxation factor is used to update the structural velocities [18] and convergence is achieved when the difference between consecutive iterations reaches a tolerance value.

2.4. Software functionalities

LUMA is written in C/C++ and designed for an x64 machine running Linux, MacOS or Windows. The software may be compiled to run in serial or parallel depending on requirements and target platform capabilities. Parallel computing capabilities are implemented through decomposition of the problem into load-balanced blocks with added halo cells (ghost cells). Inter-block communication between halo cells uses the Message Passing Interface (MPI) [19] and multi-threading is implemented for each block using OpenMP.

Input/Output (I/O) facilities include surface mesh construction and point cloud reading and pre-processing tools. Point clouds can be generated from depth-sensing cameras or from CAD geometry, the latter using our STL to point cloud conversion tool. Parallel, binary data I/O is implemented using Hierarchical Data Format 5 (HDF5) [20] and allows the reading of initial conditions for