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.

© 2018 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

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
2352-7110/© 2018 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

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.

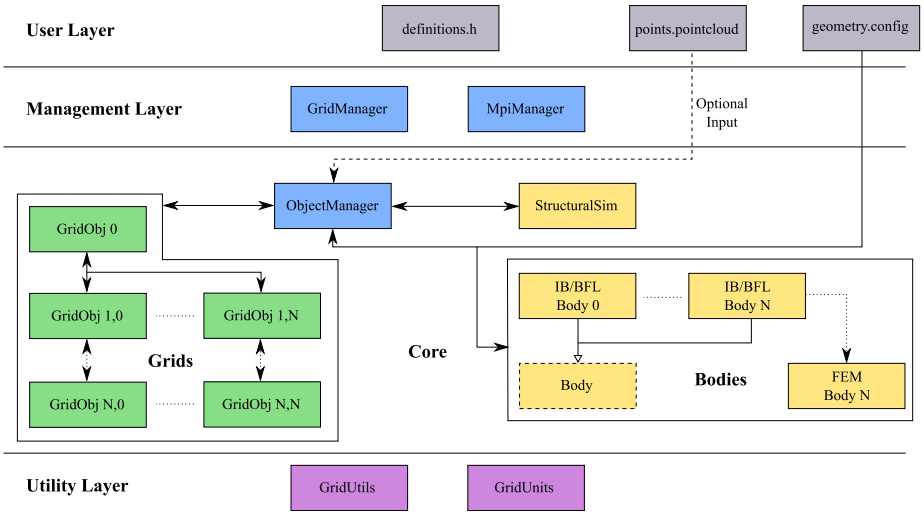
2.3. Fluid–structure coupling

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.

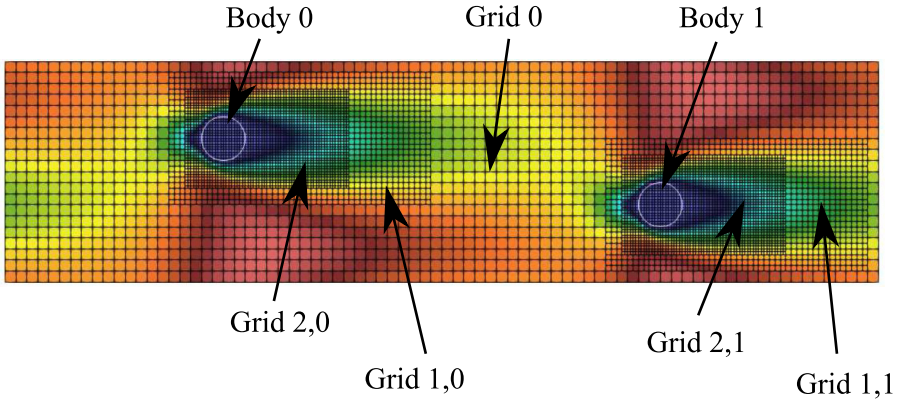
2.4. Software functionalities

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



(a) LUMA architecture showing segregation of classes by function and the system of abstraction which allows modelling to be visible but supporting framework to be hidden from a novice developer. Flow of data is indicated by the solid arrows. There is a single layer of inheritance between the `Body` class and its derived children. `GridObj` instances can own one another to allow the construction of a hierarchy.



(b) Illustration of how multiple `Body` instances and multiple `GridObj` instances can be used to build complex problems. The grid instances are labelled based on their level in the hierarchy as well as their branch (region of refinement)

Fig. 1. The architecture of the LUMA software (top) and a visual representation of a typical problem using multiple bodies and grids and how these map onto objects in the software environment (bottom). (For interpretation of the references to colour in this figure, the reader is referred to the web version of this article.)

restarting a simulation as well as the writing of datasets for later analysis. Although HDF5 files may be read directly into scientific visualisation applications such as Matlab, and TecPlot, a complementary post-processor, custom-built using the Visualization Tool Kit (VTK) API [21], produces time-slice datasets aimed at enabling the more advanced features of the Paraview visualisation tool. The integration of these features makes LUMA a powerful, capable, integrated package for engineering flow simulation.

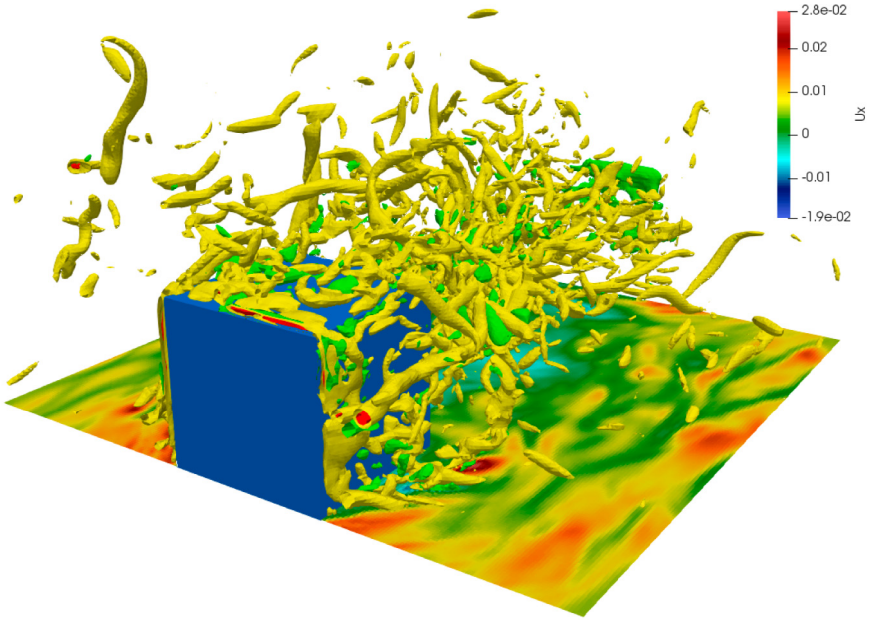
The key features of LUMA are listed in Table 1 with many of these capabilities demonstrated in the test cases contained within the online documentation.

2.5. Software architecture

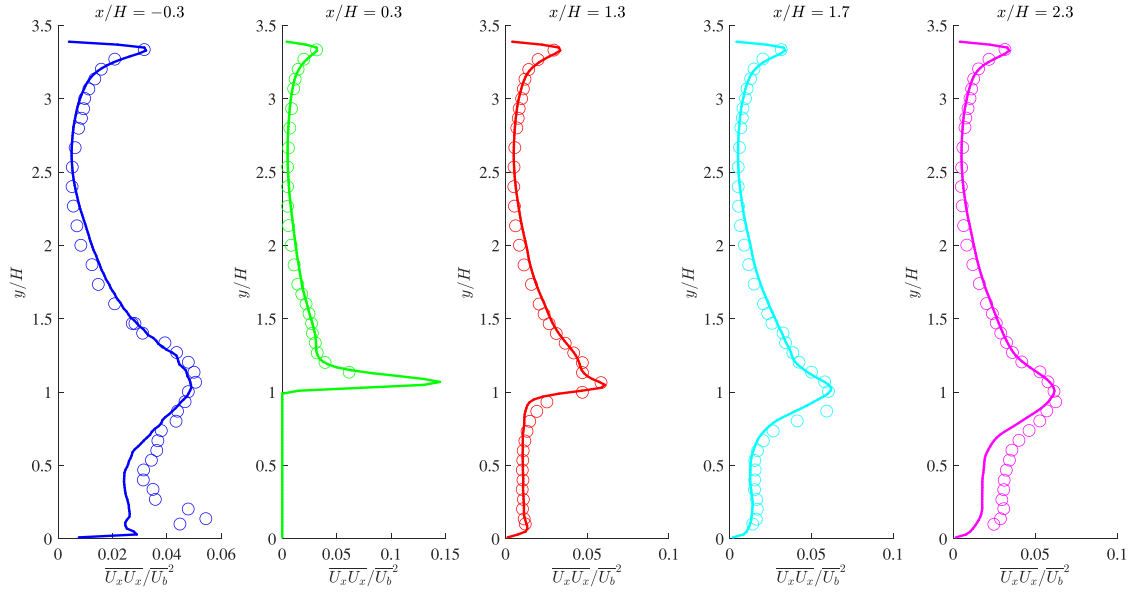
As the principal ethos of the LUMA project is simplicity and accessibility, the software architecture reflects this throughout. The class hierarchy is minimalistic and shallow. Object orientated design is used to package capabilities rather than to over-generalise objects in the hierarchy. The resulting software is then represented completely by the small number of classes depicted in Fig. 1(a). The colour-coding of the diagram indicates how capabilities are packaged by these classes.

There is a clear segregation between the fluid solver components (green) and structural solver components (yellow) and the software management framework supporting them. This structure allows engineers to edit just a single class in which all related models are stored without the need for detailed knowledge of anything else. The fundamental flow physics objects of the simulation are instances of the `GridObj` class. Objects of this class may be organised hierarchically to provide an intuitive structure representative of the physical domain space. These objects contain all data and methods required to perform LBM on their respective grids. Specifically, they contain 1D arrays of physical parameters at each node in the grid such as density and velocity, as well as methods for executing the different parts of the fluids algorithm on that grid. Data is stored as a structure of arrays to facilitate efficient indexing during loops. A snippet from the class header is provided for reference in the Appendix (Listing 1). Note that multiple `GridObj` and `Body` instances are permitted which facilitates simulations which require embedded grid refinement and multiple bodies as illustrated in the example in Fig. 1(b).

For run-time efficiency, most parameters such as decomposition dimensions, feature switches and fundamental physical definitions which characterise a particular problem or deployment



(a) Q-criterion illustration of vorticity in the fully-developed flow. Stream-wise velocity projected onto the floor of the channel.



(b) Stream-wise Reynolds stress $\overline{uu}/\overline{U_b}^2$ at different stream-wise locations at the lateral location $z/H = 0$ compared with experimental data.

Fig. 2. Results from using LUMA to simulate the experimental case of Meinders and Hanjalić [25].

are defined as pre-processor macros in the special header file `definitions.h`. The LUMA core must, hence, be recompiled if these compile-time options are changed. However, this process is handled automatically by the supplied makefiles to maintain ease of use. Body details are specified in the tab-separated `geometry.config` file, which allows for an arbitrary number of bodies in the same flow as well as dynamic body removal without the need to recompile. Output files are written for each grid in the simulation with snap shots appended within the same file. Typical data-flow during execution of LUMA is illustrated by the arrows in Fig. 1(a).

The `MpiManager`, `ObjectManager` and `GridManager` are classes of singleton design and act as self-contained elements with a clear remit. Each class manages many-core decomposition and communication, object processing and FSI facilitation, and flow solver with grid refinement respectively. Finally, the `GridUtils` and `GridUnits` classes are organisational classes of static utility methods including frequently-used vector and matrix operations, logging capabilities and unit conversions.

As the software is designed to run in parallel on multiple processors, structural objects are owned by a single processor in the available cluster which is responsible for performing the structural update. This allows multiple objects to be included efficiently with all structural calculations able to take place concurrently. Synchronisation is then enforced by the `ObjectManager` before the LBM continues.

Performance tests are performed regularly as part of the software development process. Strong and weak parallel efficiency test results are available in the online documentation.

3. Illustrative examples

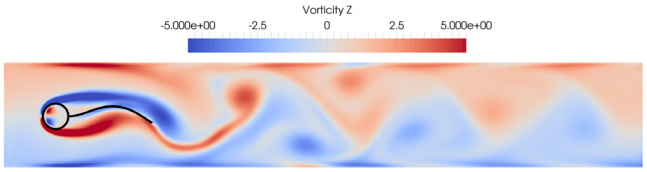
The following example cases demonstrate the key capabilities of LUMA: turbulent flow simulation and fluid–structure interaction. They may be reproduced by users following the details provided in the 'Validation' section of the online documentation.

3.1. Wall-mounted cubes in a turbulent channel

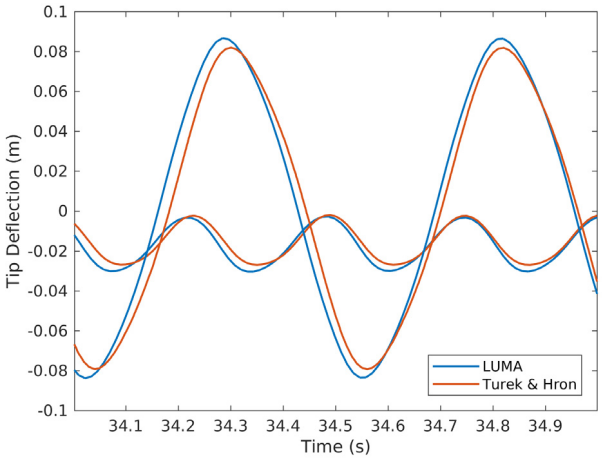
This test case is based in the wind tunnel experiment by Meinders and Hanjalić [25] and models the flow around one of the central cubes in a 25 × 10 array of cubes inside a channel. The flow Reynolds number $Re = 3850$ is based on the height of the cube H and the bulk velocity at the inlet of the channel U_b. Fig. 2(b) shows the stream-wise Reynolds stress $\overline{uu}/\overline{U_b}^2$ over five vertical lines at different positions relative to the front face of the cube. This example uses the Smagorinsky turbulence model [26] with $c_{smag} = 0.3$, the bounce-back rigid-wall boundary condition and the point cloud reader and geometry input file are used to place the cube. The post-processing is achieved using the post-processing tools supplied with LUMA and data visualised using Matlab and Paraview.

3.2. Dynamics of a flexible filament at low Reynolds number

The configuration for this 2D FSI case consists of a rigid cylinder with an attached flexible plate embedded slightly off-centre within a channel flow [27]. A parabolic velocity profile is specified at the inlet which is ramped up over the first two seconds. No-slip conditions are used at the channel walls, and the outlet boundary condition may be any of the velocity or pressure boundary conditions available in LUMA. At the intermediate Reynolds number case ($Re = 100$), the plate exhibits steady periodic flapping. Fig. 3(b) shows good agreement between LUMA and the benchmark tip displacement histories at this Reynolds number.



(a) Illustration of the vortex street behind the cylinder–filament combination. Colours represent vorticity.



(b) Comparison of tip displacement histories against benchmark data.

Fig. 3. Results from using LUMA to simulate the benchmark by Turek and Hron [27]. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4. Impact

The current version of the LUMA software is the culmination of a long-term project to provide an accessible training and development platform for lattice-Boltzmann researchers. It has been deployed and benchmarked on the UK National Supercomputer ARCHER and continues to be used on this and other HPC facilities by The University of Manchester and our UK, EU and Chinese collaborators.

LUMA is currently serving as a test bed for contributing new features for the development of a new long-term software flagship project funded by the UK Engineering and Physical Sciences Research Council for the UK Consortium on Mesoscale Engineering Sciences (UKCOMES), an organisation involving over 10 research institutions across the UK.

The accessibility of the LUMA source files as well as their extensive commenting, auto-generating documentation and suite of support tools and scripts, form a capable, coherent platform for training as well as supporting publication-standard numerical studies. The complete LUMA package has enabled researchers at The University of Manchester to use the software and contribute to its development immediately with little formal support.

LUMA is suitable for the simulation of complex fluid–structure interaction and turbulent flow problems. These are some of the most challenging applications of modelling simulation in engineering which require capable software to underpin their investigation. The extended capability of LUMA such as embedded grid refinement and the independent parallelisation of fluid and structural solvers across a many-core architecture are critical enabling technologies for the pursuit of applications of flow control and bio-inspired engineering solutions.

Prior to the creation of LUMA, the number of open-source solvers with this level of capability were limited, typically lacking appropriate modelling, and difficult for a novice user to modify. The LUMA initiative and the accompanying software platform has made a significant difference to internal work flows and encouraged a more collaborative atmosphere for researchers at all levels. The support framework for LUMA is currently under further development to allow coupling to other CFD solvers. In this way, the LBM core can be coupled to traditional FVM/FEM solvers. This novel, hybrid approach to computation allows for more efficient load balancing and power efficient, accelerated calculations.

Finally, a fork of the LUMA LBM core has been used for 'interactive', mobile CFD simulations [28,29]. This is an emerging use mode of simulation and in [29], a GPU-accelerated LUMA core is used to conduct the simulations, which will be included in a future release of LUMA. The interactive fork of LUMA benefits from all the support structure of the trunk code with no modification which illustrates the modularity of the original software.

5. Conclusions

This article has introduced LUMA, a 3D many-core fluid–structure interaction solver with embedded grid refinement. The modular architecture, thorough commenting and clear coding standards have produced an accessible yet capable solver fit for future applications of modelling and simulation in engineering. The software is in use already by UK and international researchers and is the central tool for a number of UK EPSRC-funded projects. Development on the software is active with new features in the pipeline. Future plans include the merging of the GPU-accelerated fork of the LBM core to facilitate deployment on heterogeneous systems. The software is also being adapted to allow coupling with other solvers to enable the use of hybrid methodologies in CFD analysis.

Acknowledgements

This work was supported by Engineering and Physical Science Research Council Impact Accelerator Account (Grant No. EP/K503782/1) and the UK Consortium on Mesoscale Engineering Sciences (Grant No. EP/L00030X/1).

Appendix. Code sample

The fundamental class of the software is the `GridObj` class which is declared as follows.

```
class GridObj
{

// Cstor & Dstor etc... //

/************** Member Data **************/

public :

// Cell typing
IVector<eType> LatTyp;  ///< Flattened 3D array of site labels

// Grid Scalars
double dh;              ///< Dimensionless lattice spacing
int region_number;      ///< Region number
int level;              ///< Level in embedded grid hierarchy
double dt;              ///< Dimensionless time step size
double dm;              ///< Dimensionless mass reference
int t;                  ///< Number of completed iterations
double nu;              ///< Kinematic viscosity (in lattice units)
double omega;           ///< Relaxation frequency
double gravity;         ///< Gravity force
double uref;            ///< Reference velocity

// Local grid sizes
int N_lim;              ///< Local size of grid in X-direction
int M_lim;              ///< Local size of grid in Y-direction
int K_lim;              ///< Local size of grid in Z-direction

private :

/// 1D array of sub-grid pointers (size = L_NUM_REGIONS)
std::vector<GridObj*> subGrid;

// 3D or 4D arrays of nodal properties
IVector<double> f;              ///< Distribution functions
IVector<double> feq;            ///< Equilibrium distribution functions
IVector<double> fNew;           ///< Copy of distribution functions
IVector<double> u;              ///< Macroscopic velocity components
IVector<double> u_n;            ///< Velocity at start of time step (IBM)
IVector<double> force_xyz;      ///< Macroscopic body force components
IVector<double> force_i;        ///< Mesoscopic body force components
IVector<double> rho;            ///< Macroscopic density


/************** Member Methods **************/

public :

// Initialisation functions
void LBM_initGrid();                            // Grid initialiser
void LBM_initSubGrid(GridObj& pGrid);           // Initialise sub-grid
void LBM_initGridToGridMappings(GridObj& pGrid);// Initialise mappings
void LBM_initBoundLab();                        // Initialise labels for walls
void LBM_initRefinedLab(GridObj& pGrid);        // Initialise labels on refined
void LBM_initVelocity();                        // Initialise the velocity field
void LBM_initRho();                             // Initialise the density field
void LBM_addSubGrid(int RegionNumber);          // Construct a sub-grid

// IO methods
void io_restart(eIOFlag IO_flag);               // Reads/writes restart data
void io_probeOutput();                          // Write values at probes
int io_hdf5(double tval);                       // HDF5 writer

// Master method to perform an LBM iteration using the private methods below
void LBM_multi(int subcycle = 0);


private :

// LBM implementation
void _LBM_stream(int i, int j, int k, int id, eType type_local, int subcycle);                          // Streaming step


```

```
75 void _LBM_coalesce(int i, int j, int k, int id, int v);                    // Get data from sub-grid
76
77 void _LBM_explode(int src_x, int src_y, int src_z, int id, int v, );              // Give data to sub-grid
78
79 void _LBM_collide(int id);                    // Collision step
80 void _LBM_macro(int i, int j, int k, int id, eType type_local);             // Update velocity and density
81
82 void _LBM_forceGrid(int id);                  // Apply body forces
83 void _LBM_resetForces();                      // Reset body forces to initial
84 double _LBM_equilibrium(int id, int v);       // Compute feq value
85
86 // Extra models
87 void _LBM_kbcCollide(int id);                 // KBC version of collision
88 double _LBM_smag(int id, double omega);       // Apply Smagorinsky model
89
90 // Boundary conditions
91 bool _LBM_applyBFL(int id, int src_id, int v, int i, int j, int k, int src_x, int src_y, int src_z);    // Apply BFL BC
92
93
94 bool _LBM_applySpecReflect(int i, int j, int k, int id, int v);                    // Apply Specular Reflect BC
95
96 void _LBM_regularised(int i, int j, int k, int id, eType type, int subcycle);                          // Apply Regularised BC
97
98
99 };
```

Listing 1: Fundamental design of the GridObj class with constructors/destructors and some short utility methods removed for brevity.

References

[1] Zaghi S. OFF, Open source Finite volume Fluid dynamics code: A free, high-order solver based on parallel, modular, object-oriented Fortran API. Comput Phys Comm 2014;185(7):2151–94.

[2] Mortensen M, Valen-Sendstad K. Oasis: A high-level/high-performance open source Navier-Stokes solver. Comput Phys Comm 2015;188(Suppl. C):177–88.

[3] OpenFOAM: The open source CFD toolbox. http://www.openfoam.com. [Accessed 7 December 2017].

[4] Fluids - CFD Simulation Software | ANSYS. http://www.ansys.com/en-gb/products/fluids. [Accessed 7 December 2017].

[5] Gomez-Gesteira M, Rogers B, Crespo A, Dalrymple R, Narayanaswamy M, Dominguez J. SPHysics - development of a free-surface fluid solver - Part 1: Theory and formulations. Comput Geosci 2012;48(Suppl. C):289–99.

[6] Abaqus Unified FEA. https://www.3ds.com/products-services/simulia/products/abaqus/. [Accessed 7 December 2017].

[7] Structure FEA Analysis | ANSYS. http://www.ansys.com/en-gb/products/structures. [Accessed 7 December 2017].

[8] Favier J, Li C, Kamps L, Revell A, O'Connor J, Brücker C. The PELskin project – part I: Fluid-structure interaction for a row of flexible flaps: A reference study in oscillating channel flow. Meccanica 2017;52(8):1767–80.

[9] Wu J, Qiu YL, Shu C, Zhao N. Flow control of a circular cylinder by using an attached flexible filament. Phys Fluids 2014;26(10):103601.

[10] O'Connor J, Revell A, Mandal P, Day P. Application of a lattice Boltzmann-immersed boundary method for fluid-filament dynamics and flow sensing. J Biomech 2016;49(11):2143–51.

[11] Palabos –CFD, Complex, Physics. http://www.palabos.org. [Accessed 7 December 2017].

[12] Chen S, Doolen GD. Lattice Boltzmann method for fluid flows. Annu Rev Fluid Mech 1998;30(1):329–64.

[13] Rohde M, Kandhai D, Derksen JJ, van den Akker HEA. A generic, mass conservative local grid refinement technique for lattice-Boltzmann schemes. Internat J Numer Methods Fluids 2006;51:439–68.

[14] Rosis AD, Falcucci G, Ubertini S, Ubertini F. A coupled lattice Boltzmann-finite element approach for two-dimensional fluidstructure interaction. Comput & Fluids 2013;86(Suppl. C):558–68. http://dx.doi.org/10.1016/j.compfluid.2013.08.004.

[15] Favier J, Revell A, Pinelli A. A lattice Boltzmann-immersed boundary method to simulate the fluid interaction with moving and slender flexible objects. J Comput Phys 2014;261(Suppl. C):145–61. http://dx.doi.org/10.1016/j.jcp.2013.12.052.

[16] Li Z, Favier J, D'Ortona U, Poncet S. An immersed boundary-lattice Boltzmann method for single- and multi-component fluid flows. J Comput Phys 2016;304(Suppl. C):424–40. http://dx.doi.org/10.1016/j.jcp.2015.10.026.

[17] Küttler U, Wall WA. Fixed-point fluid-structure interaction solvers with dynamic relaxation. Comput Mech 2008;43(1):61–72. http://dx.doi.org/10.1007/s00466-008-0255-5.

[18] Tian F-B, Dai H, Luo H, Doyle JF, Rousseau B. Fluidstructure interaction involving large deformations: 3D simulations and applications to biological systems. J Comput Phys 2014;258(Suppl. C):451–69. http://dx.doi.org/10.1016/j.jcp.2013.10.047.

[19] Message passing forum, MPI: A Message-Passing Interface standard, Tech. rep., University of Tennessee, 1994, Knoxville, TN, USA.

[20] HDF5 –The HDF Group. https://www.hdfgroup.org/solutions/hdf5/. [Accessed 7 December 2017].

[21] Schroeder W, Martin K, Lorensen B. The visualization toolkit. 4th ed. Kitware; 2006.

[22] Peskin CS. The immersed boundary method. Acta Numer 2002;11:479517.

[23] Ladd AJC. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 2. Numerical results. J Fluid Mech 1994;271:311339.

[24] Bouzidi M, Firdaouss M, Lallemand P. Momentum transfer of a Boltzmann-lattice fluid with boundaries. Phys Fluids 2001;13(11):3452–9. http://dx.doi.org/10.1063/1.1399290.

[25] Meinders E, Hanjalić K. Vortex structure and heat transfer in turbulent flow over a wall-mounted matrix of cubes. Int J Heat Fluid Flow 1999;20(3):255–67.

[26] Yu H, Girimaji SS, Luo L-S. DNS and LES of decaying isotropic turbulence with and without frame rotation using lattice Boltzmann method. J Comput Phys 2005;209(2):599–616.

[27] Turek S, Hron J. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. In: Bungartz H-J, Schäfer M, editors. Fluid-structure interaction: modelling, simulation, optimisation. Berlin, Heidelberg: Springer; 2006. p. 371–85.

[28] Harwood ARG, Revell AJ. Parallelisation of an interactive lattice-Boltzmann method on an Android-powered mobile device. Adv Eng Softw 2017;104(1):38–50.

[29] Harwood ARG, Revell AJ. Interactive flow simulation using Tegra-powered mobile devices. Adv Eng Softw 2018;115(Suppl. C):363–73.