Project Notes

Peter Michalski

1 List of LBM solutions

1. aromanro/LatticeBoltzmann
   https://github.com/aromanro/LatticeBoltzmann
   Found from list at https://github.com/topics/lattice-boltzmann
   A 2D Lattice Boltzmann program

2. Atruszkowska/LBM_MATLAB
   https://github.com/atruszkowska/LBM_MATLAB
   Found from list at https://github.com/topics/lattice-boltzmann
   MPI-style parallelized Shan and Chen LBM with multiscale modeling extension

3. ch4-project
   https://github.com/ecalzavarini/ch4-project
   Found using Google search.
   Eulerian-Lagrangian fluid dynamics platform A general purpose Lattice-Boltzmann code for fluid-dynamics simulations. It includes : fluid dynamics (with several volume forcing terms for Channel flow, Homogeneous Isotropic Turbulence, buoyancy) temperature dynamics (advection, diffusion , sink/source or reaction terms) phase change (enthalpy formulation for solid/liquid systems) scalar transport (same functionalities as temperature) lagrangian dynamics (tracers, heavy/light and active point-like particles; non-shperical Jeffery rotation, gyrotaxis) large eddy simulation (Smagorinsky, Shear Improved Samgorinsky with Kalman Filter)

4. CUDA-LBM-simulator
   https://github.com/henryfriedlander/CUDA-LBM-simulator
   Found from list at https://github.com/topics/lattice-boltzmann
   This is a Lattice-Boltzmann simulation using CUDA GPU graphics optimization.

5. CudneLB (TCLB)
   https://github.com/CFD-GO/TCLB
   Found from list at https://github.com/topics/lattice-boltzmann
   CudneLB is a MPI+CUDA or MPI+CPU high-performance CFD simulation code, based on Lattice Boltzmann Method.

6. DL_Meso
   https://www.scd.stfc.ac.uk/Pages/DL_MESO.aspx
   Found using Google search.
   DL_MESO is a general purpose mesoscale simulation package developed by Michael Seaton for CCP5 and UKCOMES under a grant provided by EPSRC. It is written in Fortran 2003 and C++ and supports both Lattice Boltzmann Equation (LBE) and Dissipative Particle Dynamics (DPD) methods. It is supplied with its own Java-based Graphical User Interface (GUI) and is capable of both serial and parallel execution.

7. ESPResSo
   http://espressomd.org/html/doc/index.html
   Found from list at https://github.com/topics/lattice-boltzmann
   ESPResSo is a simulation package designed to perform Molecular Dynamics (MD) and Monte Carlo (MC) simulations. It is meant to be a universal tool for simulations of a variety of soft matter systems. It features a broad range of interaction potentials which opens up possibilities for performing simulations using models with different levels of coarse-graining. It also includes modern and efficient algorithms for treatment of Electrostatics (P3M, MMM-type algorithms, constant potential simulations, dielectric interfaces,), hydrodynamic interactions (DPD, Lattice-Boltzmann), and magnetic interactions, only to name a few. It is designed to exploit the capabilities of parallel computational environments. The program is being continuously extended to keep the pace with current developments both in the algorithms and software.

8. ESPResSo++
   http://www.espresso-pp.de/
   Found from list at https://github.com/topics/lattice-boltzmann
   ESPResSo++ is a software package for the scientific simulation and analysis of coarse-grained atomistic or bead-spring models as they are used in soft matter research.
   ESPResSo++ has a modern C++ core and flexible Python user interface. ESPResSo and ESPResSo++ have common roots however their development is independent and they are different software packages.
   ESPResSo++ is free, open-source software published under the GNU General Public License (GPL).

9. firesim
   https://github.com/kynan/firesim
   Found using Google search.
   Lattice Boltzmann Method (LBM) fluid solver driving a particle engine for the simulation and real-time visualization of fire

10. fvLBM
    https://github.com/zhulianhua/fvLBM
    Found using Google search.
    finite volume lattice Boltzmann method

11. HemeLB
    https://github.com/UCL/hemelb
    Found using Google search.
    A software pipeline that simulates the blood flow through a stent (or other flow diverting device) inserted in a patients brain.

12. JavaCFD
    https://github.com/SihaoHuang/JavaCFD
    Found from list at https://github.com/topics/lattice-boltzmann

Computational fluid dynamics software written in Java using the Lattice-Boltzmann method. Allows custom-defined, arbitrary geometries in 2D incompressible flow field.

13. JFlowSim
    https://github.com/ChristianFJanssen/jflowsim
    Found from list at https://github.com/topics/lattice-boltzmann
    jFlowSim is an interactive, thread-parallel Lattice Boltzmann solver in two dimensions.

14. laboetie
    https://github.com/maxlevesque/laboetie
    Found from list at https://github.com/topics/lattice-boltzmann
    laboetie is a computational fluid dynamics code for chemical applications. It uses the Lattice-Boltzmann algorithm.

15. LatBo.jl
    https://github.com/UCL/LatBo.jl
    Found using Google search.
    Lattice-Boltzmann implementation in Julia

16. lettuce
    https://github.com/Olllom/lettuce
    Found using Google search.
    GPU-acclerated Lattice Boltzmann in Python

17. LB2D_Prime
    http://faculty.fiu.edu/ sukopm/LBnD_Prime/LBnD_Prime.html
    Found using Google search.
    LB2D_Prime is a lattice Boltzmann (LB) code capable of simulating single and multi-phase flows and solute/heat transport in geometrically complex domains.

18. LB3D
    http://ccs.chem.ucl.ac.uk/lb3d
    Found using Google search.
    A parallel implementation of the Lattice-Boltzmann method for simulation of interacting amphiphilic fluids. LB3D provides functionality to simulate three-dimensional simple, binary oil/water and ternary oil/water/amphiphile fluids using the Shan-Chen model for binary fluid interactions.

19. LBDEMcoupling-public
    https://github.com/ParticulateFlow/LBDEMcoupling-public
    Found using Google search.
    Coupling between the Lattice-Boltzmann code Palabos and the DEM code LIGGGHTS

20. LBSim
    https://github.com/noirb/lbsim

Found using Google search.
A small and simple Lattice-Boltzmann Method fluid simulator supporting complex boundaries.

21. Limbes
    https://code.google.com/archive/p/limbes/
    Found using Google search.
    Open source (GPL) code in 2D based on Gauss-Hermite quadrature, parallel (openmp), fortran 90. LIMBES is the recursive acronym for LIMBES Is May be a Boltzmann Equation Solver. Version 1.0 solves numerically by a Lattice Boltzmann like method the BGK-Boltzmann equation for gas in two dimensions.

22. listLBM
    https://github.com/sorush-khajepor/listLBM
    Found from list at https://github.com/topics/lattice-boltzmann
    ListLBM is a sparse lattice Boltzmann solver for multiphase flow in porous media

23. LUMA
    https://github.com/ElsevierSoftwareX/SOFTX-D-18-00007
    Found from list at https://github.com/topics/lattice-boltzmann
    LUMA: A many-core, FluidStructure Interaction solver based on the Lattice-Boltzmann Method

24. MP-LABS
    https://github.com/carlosrosales/mplabs
    Found using Google search.
    MP-LABS is a suite of numerical simulation tools for multiphase flows based on the free energy Lattice Boltzmann Method (LBM). The code allows for the simulation of quasi-incompressible two-phase flows, and uses multiphase models that allow for large density ratios. MP-LABS provides implementations that use periodic boundary conditions, but it is written in a way that allows for easy inclusion of different boundary conditions. The output from MP-LABS is in plain ASCII and VTK format, and can be analyzed using other Open Source tools such as Gnuplot and Paraview.

    The objective of the MP-LABS project is to provide a core set of routines that are well documented, highly portable, and have proven to perform well in a variety of systems. The source code is written in Fortran 90 and MPI and uses separate subroutines for most tasks in order to make modifications easier.

25. Openlb
    https://www.openlb.net/
    Found from list at https://github.com/topics/lattice-boltzmann
    The OpenLB project provides a C++ package for the implementation of lattice Boltzmann methods that is general enough to address a vast range

of tansport problems, e.g. in computational fluid dynamics. The source code is publicly available and constructed in a well readable, modular way.

26. Palabos
    https://palabos.unige.ch/
    Found using Google search.
    The Palabos library is a framework for general-purpose computational fluid dynamics (CFD), with a kernel based on the lattice Boltzmann (LB) method. It is used both as a research and an engineering tool: its programming interface is straightforward and makes it possible to set up fluid flow simulations with relative ease, or, if you are knowledgeable of the lattice Boltzmann method, to extend the library with your own models. Palabos stands for Parallel Lattice Boltzmann Solver. The librarys native programming interface in written in C++.

27. pyLBM
    https://github.com/pylbm/pylbm
    Found from list at https://github.com/topics/lattice-boltzmann
    pylbm is an all-in-one package for numerical simulations using Lattice Boltzmann solvers. This package gives all the tools to describe your lattice Boltzmann scheme in 1D, 2D and 3D problems.

28. Sailfish
    https://github.com/sailfish-team/sailfish
    Found using Google search.
    Lattice Boltzmann (LBM) simulation package for GPUs (CUDA, OpenCL)

29. siramirsaman/LBM
    https://github.com/siramirsaman/LBM
    Found from list at https://github.com/topics/lattice-boltzmann
    Lattice Boltzmann Method Implementation in MATLAB for Curved Boundaries

30. SunlightLB
    http://sunlightlb.sourceforge.net/
    Found using Google search.
    SunlightLB is an open-source 3D lattice Boltzmann code which can be used to solve a variety of hydrodynamics problems, including passive scalar transport problems.

31. Taxila-LBM
    https://github.com/ecoon/Taxila-LBM
    Found using Google search.
    Taxila LBM is a parallel implementation of the Lattice Boltzmann Method for simulation of flow in porous and geometrically complex media.

32. loliverhennigh / Lattice-Boltzmann-fluid-flow-in-Tensorflow
    https://github.com/loliverhennigh/Lattice-Boltzmann-fluid-flow-in-Tensorflow

Found using Google search.
A Lattice Boltzmann fluid flow simulation written in Tensorflow.

33. turbulent_lbm_multigpu
https://github.com/arashb/turbulent_lbm_multigpu
Found using Google search.
Lattice Boltzmann simulation of turbulent fluid flow on GPU Cluster

34. waLBerla
https://www.walberla.net/
Found using Google search.
waLBerla uses the lattice Boltzmann method (LBM), which is an alternative to classical Navier-Stokes solvers for computational fluid dynamics simulations. All of the common LBM collision models are implemented (SRT, TRT, MRT). Additionally, a coupling to the rigid body physics engine pe is available.

35. wlb
https://github.com/weierstrass/wlb
Found using Google search.
A Lattice-Boltzmann code for solving coupled equations in electrohydrodynamics. Three collision operators are implemented for the (incompressible) Navier-Stokes, Nernst-Planck (advection-diffusion) and Poission's equation for electrostatics respectively. Various implementations of Dirichlet/Neumann boundary conditions are also available. The code deals (so far) only with 2D systems. This code is part of a master thesis project carried out at Chalmers University, Gothenburg.

36. Zmhhaha/LBM-Cplusplus-A.A.Mohamad
https://github.com/zmhhaha/LBM-Cplusplus-A.A.Mohamad
Found from list at https://github.com/topics/lattice-boltzmann
The C++ version code of "Lattice Boltzmann Method Fundamentals and Engineering Applications with Computer Codes".

2 Variabilities

Input Variabilities

1. boundary parameters
2. dimension
3. number of velocity directions

Calculation Variabilities

1. computational model
2. decomposition technique
3. coefficient weights
4. input check
5. encoding of output
6. exception check

Other Variabilities

1. language
2. license

3 Quality Measurement

3.1 Robustness

According to our definition, we must place the system in a state not assumed or anticipated in its requirements specification. Some ideas are providing invalid input, using the software on different hardware or OS software than required, or not providing (or deleting) additional libraries or directories. Such changes should of course be analyzed within reason - a different OS version as opposed to a fundamentally different OS, for example.

Provide invalid input and observe behaviour: Does the system crash - is it recoverable? Does the system prevent you from providing invalid input? Does the system return an error message?

Using software on different hardware/OS software than required (would need to make a reasonable list of variance): Ask similar questions as above.

Altering/removing/not providing external libraries and/or directories: Ask similar questions as above.

Furthermore, in regard to these questions, is software that crashes easily but provides an error more robust than a program that rarely crash but does not provide an error message?

We can additionally measure mean time to recovery from failures

DOI: 10.1007/978-3-642-29032-9_16:
The robustness failures are typically classified according to the CRASH criteria [540]: Catastrophic (the whole system crashes or reboots), Restart (the application has to be restarted), Abort (the application terminates abnormally), Silent (invalid operation is performed without error signal), and Hindering (incorrect error code is returnednote that returning a proper error code is considered as robust operation). The measure of robustness can be given as the ratio of test cases that exposes robustness faults, or, from the systems point of view, as the number of robustness faults exposed by a given test suite.

1. Injecting Random input
2. Using invalid input
3. Invalid inputs for each function in interface (valid and invalid inputs are type specific)
4. applying mutation techniques

3.2 Performance

1. How many concurrent loads (this may also require hardware parallelism) - Try to run the multiple tests in one instance of the application (pass/fail)?
2. How many application instances can be run? - Try to open and run multiple instances of the application (yes/no)?
3. CPU usage (find a tool to monitor the applications use of CPU), calculate average over the run of tests. Compare CPU usage between solutions - keep in mind the solutions vary significantly.
4. Memory usage - similar to CPU usage
5. Average Response Time (compared to other solutions) - run a pre determined set of test
6. Error rate (exceptions) - would need to automate tests
7. MTBF - would need to automate tests

3.3 Maintainability

How much time it takes to add a new function (requirement)
How many open and closed issues on git
How quickly are issues closed
Update frequency

Metrics for Assessing a Software System's Maintainability (Oman, Hagemeister):
Software system metrics:
1. server maturity attributes (age since release, size, stability, maintenance intensity, defect intensity, reliability, reuse, subjective product appraisals)
size = thousands of non commented source statements (KNCSS)
stability = 1 - change factor
defect intensity = defects reported / KNCSS –check full equation
reuse = the percentage of lines of code reused from other systems

2. source code (control structure, information structure, typography and naming and commenting) - each of these is broken into system and component subcategories

system control structure: modularity, complexity, consistency, nesting, control coupling, encapsulation, module reuse, control flow consistency

component control structure: complexity, use of structured constructs, use of unconditional branching, nesting, span of control structures, cohesion

system information structure: global data types, global data structures, system coupling, data flow consistency, data type consistency, nesting, I/O complexity, I/O integrity

component information structure: local data types, local data structures, data coupling, initialization integrity, span of data

system typography, naming and commenting: overall program formatting, overall program commenting, module separation, naming, symbols and case

component typography, naming and commenting: statement formatting, vertical spacing, horizontal spacing, intramodule commenting

3. supporting documentation (abstraction, physical attributes)

supporting documentation abstraction: descriptiveness appraisals, completeness appraisals, correctness appraisals

supporting documentation physical attributes: readability appraisals, modifiability appraisals

We can then sum up weighted attributes of each dimension, and then further sum up weighted dimensions

Software Metrics for Predicting Maintainability (Frappier, Matwin, Mili):

Documentation criterion (metrics): readability, traceability (NR, UR), coupling (M-MC, HK-IF, R-IF, KPL-IF, IF4, CA-DC), cohesion (M-MS, HK-IF, R-IF, KPL-IF, IF4, CA-DC), size

Source code criterion (metrics): traceability(NR, UR), control structure(v(g), knots, RLC), independence, readability(Vcd, Vcs, Ls), size (LOC, E, RLC), doc accuracy (DAR), consistency (SCC)

UR - Unreferenced Requirements - The number of original requirements not referenced by a lower document in the documentation hierarchy (Formula: UR = number of R1 not referenced by a R2, where..)

NR - Non-referenced requirements - The number of items not referencing an original requirement.(NR = number of R2 not referencing any R1, where..)

Module Coupling (M-MC) - A measure of the strength of the relationships between modules. (Formula: Myers defines six levels of coupling which are, in order of decreasing strength: coupling, common, external, control, stamp, data)

Module Strength (M-MS) - A measure of how strongly related are the elements within a module.(Formula: Module strength is a subjective metric. Myers defines seven levels of strength which are, in order of increasing strength: coincidental, logical, classical, procedural, communicational, informational, functional)

Information Flow (HK-IF) -A measure of the control flow and data flow between modules. (Formula: IFm = (fan-inm*fan-outm)2*int-compm)

Integrated Information Flow of Rombach (R-IF) - A measure of intermodule and intramodule complexity based on information flow.

Information Flow by Kitchenham et al (KPL-IF) - A measure of intermodule complexity inspired from Henry and Kafura's information flow metric. Since Kitchenham et al experienced some difficulties in understanding the definition of flows provided by Henry and Kafura, they formulated a new set of definitions.

IF4 - A measure of intermodule complexity based on information flow.

Design Complexity of Card and Agresti (CA-DC) - A measure of intermodule and intramodule complexity of a system based on fan-out, number of modules and input/output variables

Cocomo Inspired Metric (COCO) - A selection of appropriate adjustment factors of the intermediate Cocomo metric.

Cyclomatic Complexity Number (v(G)) - The number of independent basic paths in a program.

Knots Description - The number of crossing lines (unstructured goto statements) in a control flow graph

Relative Logical Complexity (RLC) - The number of binary decisions divided by the number of statements

Comments Volume of Declarations (Vcd) - Total number of characters found in the comments of the declaration section of a module. The declaration section comprises comments before the module heading up to the first executable statement of the module body.

Comments Volume of Structures (Vcs) Description: Total number of characters in the comments found anywhere in the module except in the declaration section. The declaration section comprises comments before the module heading up to the first executable statement of the module body.

Average Length of Variable Names (Ls) Description: Mean number of characters of all variables used in a module. Unused declared variables are not included.

Lines of Code (LOC) Description: The number of lines in the source code excluding blank lines or comment lines.

Software Science Effort (E) Description: An estimation of programming effort based on the number of operators and operands. It is a combination of other Software Science metrics.

Documentation Accuracy Ratio (DAR) Description: A verification of the accuracy of the CEI Spec, RS and SDD with respect to the source code.

Source Code Consistency (SCC) Description: The extent to which the source code contains uniform notation, terminology and symbology within itself.

3.4 Reusability

Measuring Software Reusability (Poulin)

Taxonomy of reusability mctrics 1. Empirical methods - - Module oriented - Complexity based - Size based - Reliability based Component oriented
2.Qualitative methods - -Module oriented - Style guidelines Component oriented - Certification guidelines - Quality guidelines

—

Chidamber and Kemerer object-oriented metrics:

https://www.aivosto.com/project/help/pm-oo-ck.html:

eg weighted methods per class, number of children, coupling per class

Software reusability metrics estimation: Algorithms, models and optimization techniques (Padhy)

Cyclomatic complexity: independent paths through source code

Software Reuse and Reusability Metrics and Models (Frakes, Terry)

A new reusability metric for object-oriented software (Barnard)

A metrics suite for measuring reusability of software components (Washizaki)

Reusability Index: A Measure for Assessing Software Assets Reusability (Ampatzoglou)

3.5 Portability

compare effort to port to effort to redevelop

determine if the system can be ported: ram, processor, resolution, OS, browser

Issues in the Specification and Measurement of Software Portability (Mooney):
The term portability refers to the ability of a software unit to be ported (to a given environment). A program is portable if and to the degree that the cost of porting is less than the cost of redevelopment. A software unit would be perfectly portable if it could be ported at zero cost, but this is never possible in practice. Instead, a software unit may be characterized by its degree ofportability, which is a function of the porting and development costs, with respect to a specific target environment.

Some of these issues have been examined by the author [Mooney 931. This work has proposed as a metric the degree of portability of a software unit with respect to a target environment, defined as DPfsu) = 1 - (Cport(su,q) I Crd&req.e2)).
This metric relates portability to a ratio between the cost of porting (which depends on the properties of the existing software unit and on the target environment), and the cost of redevelopment (which depends on the requirements and the target environment). A series of experiments is underway to refine and validate this metric and to determine how to measure or estimate Cpoyf and Crdev. One study by Sheets [94] suggests that the metric can be badly skewed by secondary elements such as inadequate documentation for the existing software.

Designing a Measurement Method for the Portability Non-functional Requirement (Talib):
EffortNew is the total effort needed in working hours units to develop the software functionalities on a new environment

---

https://www.softwaretestinghelp.com/what-is-portability-testing/:

ISO 9126 breaks down portability testing: installability, compatability, adapatability, and replaceability.

check Installability, Adaptability, Replaceability, Compatibility or Coexistence

Installability: validate OS reqs, memory and RAM reqs, clear installation and uninstallation procedures, additional prerequisites

Adapatability: hardware and software dependency, language dependency and communication system