Project Notes

Peter Michalski

1 List of LBM solutions

1. aromanro/LatticeBoltzmann
2. Atruszkowska/LBM_MATLAB
3. ch4-project
4. CPE490_Lattice_Boltzmann_Project
5. CUDA-LBM-simulator
6. CudneLB (TCLB)
7. DL_Meso
8. ESPResSo
9. ESPResSo++
10. firesim
11. fvLBM
12. HemeLB
13. JavaCFD
14. JFlowSim
15. laboetie
16. LatBo.jl
17. lettuce
18. LB2D_Prime
19. LB3D
20. LBDEMcoupling-public
21. LBSim
22. Limbes
23. listLBM
24. LUMA
25. MP-LABS
26. MUPHY
27. Openlb

28. Palabos
29. PyLBM
30. Sailfish
31. siramirsaman/LBM
32. SunlightLB
33. Taxila LBM
34. Tensorflow
35. turbulent_lbm_multigpu
36. waLBerla
37. wlb
38. Zmhhaha/LBM-Cplusplus-A.A.Mohamad

2 Quality Measurement

2.1 Robustness

Does not crash and provides (descriptive) error upon an invalid input.

Is software that crashes but provides an error more robust than a program that does not crash but does not provide an error?

Mean time to recovery from failures?

DOI: 10.1007/978-3-642-29032-9_16:

The goal of robustness testing is to activate those faults (typically design or programming faults) or vulnerabilities in the system that result in incorrect operation, i.e., robustness failure, affecting the resilience of the system. Robustness testing mostly concentrates on the internal design faults activated through the system interface. 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

2.2 Performance

How many concurrent loads (this will also require hardware parallelism)

How many application instances can be run?

CPU usage (monitor this)

Average Response Time (compared to other solutions)

Error rate (exceptions)

MTBF

2.3 Maintainability

How much time it takes to add a new function (requirement)

Metrics for Assessing a Software System's Maintainability (Oman, Hagemeister)

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

2.4 Reusability

Measuring Software Reusability (Poulin)

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)

2.5 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)

Designing a Measurement Method for the Portability Non-functional Requirement (Talib)

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

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