Module Guide for Lattice Boltzmann Solvers

Peter Michalski

December 9, 2019

1 Revision History

Date	Version	Notes
Nov. 25, 2019	1.0	Initial Document

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
CA	Commonality Analysis
DAG	Directed Acyclic Graph
LBM	Lattice Boltzmann Method
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
Lattice Boltzmann Solvers	Program Solves LBM Problems using Libraries
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1985). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1985), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Commonality Analysis (CA) (refer to Michalski), the Module Guide (MG) is developed (Parnas et al., 1985). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the CA. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The control algorithm for Lattice Boltzmann Solvers.

AC3: The algorithm for reading input data.

AC4: The algorithm for checking input data parameters.

AC5: The control algorithm for the LBM libraries.

AC6: The algorithm for streaming particles.

AC7: The algorithm for collision between particles.

AC8: The format of the problem models.

AC9: The format of the lattice models.

AC10: The format of the boundary conditions.

AC11: The output image rendering algorithm.

AC12: The format of non-primitive data structures.

AC13: The input types of non-primitive data structures.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware Hiding Module

M2: System Control Module

M3: Input Reading Module

M4: Input Checking Module

M5: LBM Control Module

M6: Streaming Module

M7: Collision Module

M8: Problem Module

M9: Lattice Module

M10: Boundary Module

M11: Image Rendering Module

M12: Data Structure Module

M13: Input Types Module

Level 1	Level 2
Hardware-Hiding Module	M1 Hardware Hiding Module
Behaviour-Hiding Module	M2 System Control Module
	M3 Input Reading Module
	M4 Input Checking Module
	M5 LBM Control Module
	M6 Streaming Module
	M7 Collision Module
	M8 Problem Module
	M9 Lattice Module
	M10 Boundary Module
Software Decision Module	M11 Image Rendering Module
	M12 Data Structure Module
	M13 Input Types Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the CA. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1985). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Lattice Boltzmann Solvers* means the module will be implemented by the Lattice Boltzmann Solvers software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the Commonality Analysis (CA) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the CA.

Implemented By: –

7.2.1 System Control Module (M2)

Secrets: The algorithm to control Lattice Boltzmann Solvers.

Services: Controls the execution of Lattice Boltzmann Solvers, including calling submodules.

Implemented By: Lattice Boltzmann Solvers

7.2.2 Input Reading Module (M3)

Secrets: The algorithm to read the input data.

Services: Reads input data and formats it.

Implemented By: Lattice Boltzmann Solvers

7.2.3 Input Checking Module (M4)

Secrets: The algorithm to check that input values fall within allowable parameters.

Services: Verifies that the input data falls within allowable parameters.

Implemented By: Lattice Boltzmann Solvers

7.2.4 LBM Control Module (M5)

Secrets: The algorithm to control how the LBM library will be executed.

Services: Controls how the chosen LBM library will solve the problem.

Implemented By: Lattice Boltzmann Solvers

7.2.5 Streaming Module (M6)

Secrets: The algorithm to calculate the streaming of particles.

Services: Calculates the streaming of fluid particles along lattice links.

Implemented By: Lattice Boltzmann Solvers

7.2.6 Collision Module (M7)

Secrets: The algorithm to calculate the collision of particles.

Services: Calculates the collision of fluid particles as they travel along lattice links.

Implemented By: Lattice Boltzmann Solvers

7.2.7 Problem Module (M8)

Secrets: The structure of LBM input parameters.

Services: Converts the problem data into an input data structure for the LBM library.

Implemented By: Lattice Boltzmann Solvers

7.2.8 Lattice Module (M9)

Secrets: The structure of the lattice model.

Services: Gathers data for the selected lattice model into a data structure for M8.

Implemented By: Lattice Boltzmann Solvers

7.2.9 Boundary Module (M10)

Secrets: The structure of the model boundary.

Services: Converts data for the model boundary into a data structure for M8.

Implemented By: Lattice Boltzmann Solvers

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the CA.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Image Rendering Module (M11)

Secrets: The algorithm to convert the LBM algorithm output into an image.

Services: Converts the information from the LBM algorithm into an image.

Implemented By: –

7.3.2 Data Structure Module (M12)

Secrets: The format of non-primitive data structures used in Lattice Boltzmann Solvers.

Services: Provides non-primitive data types in Lattice Boltzmann Solvers.

Implemented By: Lattice Boltzmann Solvers

7.3.3 Input Types Module (M13)

Secrets: The input types for the non-primitive data structures used in Lattice Boltzmann Solvers.

Services: Specifies non-primitive data types in Lattice Boltzmann Solvers.

Implemented By: Lattice Boltzmann Solvers

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M1, M2, M3, M12, M13
R2	M2, M9, M12, M13
R3	M2, M4, M12, M13
R4	M2, M8, M9, M10, M12, M13
R5	M2, M8, M9, M12, M13
R6	M2, M5, M6, M7, M12, M13
R7	M1, M2, M11, M12, M13

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M4
AC5	M5
AC6	M6
AC7	M7
AC8	M8
AC9	M9
AC10	M10
AC11	M11
AC12	M12
AC13	M13

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1979) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

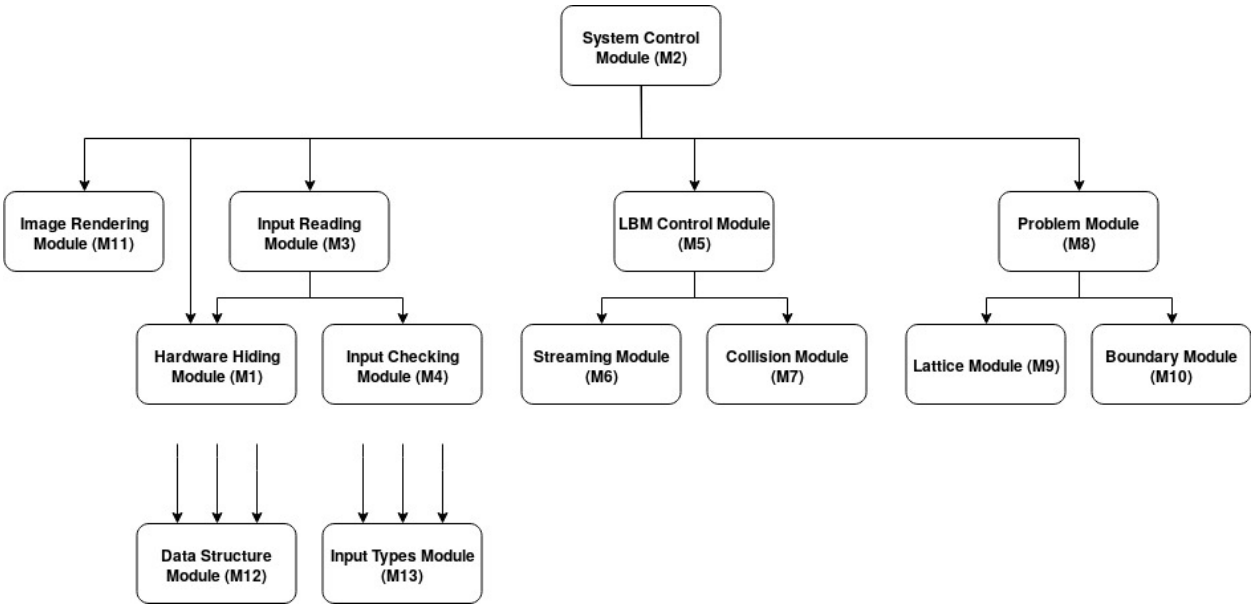

Figure 1: Use hierarchy among modules

References

Peter Michalski. Lattice Boltzmann Solvers - CA. URL https://github.com/peter-michalski/LatticeBoltzmannSolvers/blob/master/docs/SRS/CA.pdf.

David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

David Lorge Parnas. Designing software for ease of extension and contraction. *IEEE transactions on software engineering*, (2):128–138, 1979.

David Lorge Parnas, Paul C Clements, and David M Weiss. The modular structure of complex systems. *IEEE Transactions on software Engineering*, (3):259–266, 1985.