UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

MASTER ICT

MI1.07 –SOFTWARE DEVELOPMENT PROJECT

Huy-Duc LE
University of Science and Technology of Hanoi
Hanoi, Vietnam
lehuyduc3@gmail.com

Vinh-Nam HUYNH
University of Science and Technology of Hanoi
Hanoi, Vietnam
protossnamjune2nd@gmail.com

Duc-Quyen NGUYEN
University of Science and Technology of Hanoi
Hanoi, Vietnam
hakonaryuuji@gmail.com

Vu-Hung NGUYEN
University of Science and Technology of Hanoi
Hanoi, Vietnam
hungnga25197@gmail.com

March 2020

GROUP PROJECT – REPORT

Project REDUCE – Programmer's guide

Table of Contents

1 Preface 2

2 Classes 2

2.1 iFloat 2

3 Program sequence ..... **Error! Bookmark not defined.**

1 Preface

"You shall not crash" – Group REDUCE

One of the most important features that our group focus on is to make sure the program does not crash, no matter what the user does. Technically it might cause the computer to crash (out of memory error), but that's the computer's fault for not having enough ram. Also, if the program freeze (usually during file input/output) and has "not responding problem", just don't click anything, or click "wait for the program to response" because it's still running normally.

Every single input is checked whenever the user does something. Also, we place *if … throw …* statements almost anywhere in our program. This allows us to debug more quickly, as we can identify the problem very quickly.

Also, our program makes heavy use of Qt signal/slot. While it is not fast, it is insignificantly small when compared to the time it takes to run the experiment. So the ease of programming Qt signal/slot provides make it a good option.

The program use simple C++11 with no special feature (aside from the Boost C++). Now let's move into the technical details of the program.

2 Classes

2.1 iFloat

- This is the most basic class in our program. We use boost C++ **cpp_dec_float_40** (called **float40)** for this class.
- It can be used the same way as a **double** or **float**, just with higher precision.
- The reason we need this class instead of just using **float40** is that **float40** use static memory, which might cause the program to crash because Stack memory is only around 1MB. By using a wrapper, we make sure that stack overflow does not happen.
- The reason we choose number **40** (before it was 50 and 30) is because it's more than 2 times the precision of **double** (16 digits vs 40 digits). That means the user can sum a huge amount of number or multiplying a large amount of **double** (which will usually result in +/-∞ value, or 0 because underflow) without any error. For summing, even 30 digits is more than enough.

2.2 Matrix

- This is a standard implementation of the matrix class.
- You can access the i-th element (starting from row 0, column 0) using operator [], or the (i,j) element using operator ().
- You have to initialize the values of the matrix yourself.

2.3 Distribution

- We use Inverse Sampling method. The parameters are number of bins, lower bound, and upper bound.
- Basically, we divide the x-axis into **binNumber** rectangles from lower bound to upper bound.
- By doing this, we can sum/multiply/… two distributions very easily. Because it's just element wise operation (two distributions need to have the same binNumber, lower bound, and upper bound).
- We use 3 types of distribution. The implementation is very simple and can be seen in the code.

2.4 Random generator

- This is a simple class. It generates a random distribution according to the input distribution.

2.5 Array generator / Matrix generator

- Note: for pair classes like this, I will only explain the array class.
- The class can receive/emit signal which allows it to create an array inside a different thread.

2.6 Array experiment / Matrix experiment

- This is the class where calculation take place.
- We can have different types of data type in an experiment, so we use template.
- However, a template class can't inherit can't use QObject (for receive/emit signal), so we make a base class that inherit QObject, makes important function virtual, and implement receive/emit functions inside that base class. Then, ArrayExperiment inherit that class.
- To perform an experiment, the class receives a list of **function pointers.** This allows the user to add new function without changing the experiment class.

2.7 Array Experiment controller / Matrix Experiment controller

- We use this class to connect the MainWindow (UI) class and the Experiment class.
- This class receive the inputs, names of algorithm to test, precision type.
- It converts the list of names into a list of function pointers that point to the correct algorithm.
- Finally, it creates the correct Experiment object and runs the experiment

2.8 UtilityEnum / ReduceAlgorithms

- This is where the algorithms in the experiment are implemented.
- UtilityEnum declares the operators, data types, algorithms, and precision level of the program. If you want to add something new, you need to write it here.
- ReduceAlgorithms is where all the algorithms are implemented.

- Here is where the magic happen: everything in this class is templated. So, the user only needs to program the algorithm and doesn't need to care about the data type. Then, when the ExperimentController class receives the list of algorithms, it will convert it into a list of function. So the user doesn't need to touch other classes at all.
- If the user want to add a new algorithm, you need to:

    + UtilityEnum.h: update AlgoName, AlgoNameList, algo2string and string2algo.

    + ReduceAlgorithms.h: update algo2functor(), forMatrix(), and write the new algorithm using the same template as other algorithms. Which means:

```
template <typename dtype>
inline iFloat newAlgo(const vector<dtype> &inputs, Op op) {
        ...
        Return iFloat(res);
}
```

    + And done! The UI will be updated with the new algorithm. And ExperimentController class use function pointer so it can use the new algorithm without any update.

- Note: there is one downside to this method. The algorithm must be compatible with both matrix class and numeric class, even if it does not work with matrix (forMatrix() = false). That means any function or operator that is used on inputs[i] must work on both numeric class and Matrix<anyType> class. In the fast2sum section, you can see an example (**anyAbs** function).

    + The solution to this problem is very simple. Just make the functions you need to have template, and specialize it for matrix with an empty function. If for some reasons it's not possible, then you must change/add functions to the matrix class.

- If you pass an algorithm that has forMatrix() = false to the MatrixExperimentController object, it will ignore that algorithm. So there's nothing to worry about.

2.9 MainWindow

- This is the UI class. It is the main thread.
- It contains all the Action Listener (buttons). Also, this class call new threads to perform actions that takes a long time.
- It will be described in the sequence section below.