CA Assignment 2

MatInv.py

We created a python class which inherits the “SimObject” class. The parent class “SimObject” has all the gem5 objects in it, therefore we have to import it into our code to get its functionality in our code. We can change its functionality according to our need simply by tweaking the parameters in the python file. The type of our class will be the name of our code file. Also, we have to define the location of our “cxx_header” file. The header file will contain all the necessary data members and member functions of our class. This python file is basically needed to run the C++ file.

mat_inv.hh

Now, everything is written in C++ as this file basically contains all the C++ object header files which we are going to use. Using **“#ifndef”** and **“#endif”**, gem5 will automatically include all the header files in the <needed file>, and the <needed file’s location>. Their main aim is to prevent C++ header files from being imported multiple times. If we are defining the header file for the first time, it will be created (**“#define”**) and imported. But if the header file already exists, then **“#ifndef”** will make sure that there are no multiple copies of the header file, it will simply import that previous header into the file. These are the **preprocessors directives**. These lines are executed first before even the processor compiles the code.

#include "params/MatInv.hh" : This file is automatically created by gem5 in build/X86/. . . location during the time of rebuilding gem5.opt
#include "sim/sim_object.hh" : We imported this file to get all the functionality of “SimObject” in our code.

Here we define a C++ class, as our main class is of “SimObject” type. Therefore this class must also inherit the C++ “SimObject” class. We defined some public and private members functions and data members of our C++ class. We only declared the data function here. The function definition is done in the **“.cc” file.** Function declaration is done simply by writing the function’s name, its return type and its parameters or arguments.

Data members like “N” and “A” are the matrix size and the matrix itself respectively. Users are supposed to define them on their own. Member functions like “adjoint”, “getCofactor”, “determinant”, and “inverse” are declared here. These functions help in calculating the inverse of the matrix.

We also wrote the function declaration of the event-driven helper function. This helps us in writing our own function or functionality using “SimObject”. Like in this case matrix multiplication. Event-driven functions are explained in the next section in detail.
MatInv(MatInvParams *p): This is the constructor function for the “SimObject” class which takes an “Object” as parameter. This “Object” parameter is automatically generated by the gem5’s build system at the time of rebuilding (scons/build/X86).

mat_inv.cc

Firstly, we imported all the necessary header files including **“DEBUG” flags.** Then we created a constructor and gave the default values for initialization. After this we wrote the main code in the “processEvent()” function and other member functions of the class are also defined in order to calculate the inverse of the matrix. All the other matrix inverse helper functions are completely defined in this code file.

<u>Below are the event-driven helper functions:</u>

- **processEvent():** This is the first function which gets executed in this file (after the constructor function). We defined our main logic for matrix multiplication in this function because this function gets executed every time the “event” fires (processEvent()). Or in other words, the code snippet written in this function gets executed according to the user’s need. Sort of an event-driven programming style.
- **EventFunctionWrapper:** It is an instance of “event”. It allows the user to write any function and it will be executed.
- **void startup():** In order to execute the event, we need to specify the time or schedule the event. We make use of the “Schedule” function which takes event and execution time as input. We schedule the event in this startup function. We can only schedule internal events and it does not get executed until the “simulate()” function gets executed in our Python configuration script.

We use the “DEBUG” flags to replace the traditional **“stdout”** of C++ for printing. The flags here are basically your “SimObject”. You can print the value or state of various variables in the code using these “DEBUG” flags. These are very helpful in debugging the code. We have to specify the flags in our code and also enable/disable these flags according to our need while writing in the command line.

“DTRACE” function returns a boolean value true or false if the flag is enabled or not respectively. Users have to specify whether to enable or not the flag in the **command line**. So we put an **“if condition”** in our code to check if this flag is enabled or not and accordingly print the required things like size of the matrix, original matrix and the inverse of the matrix.

At last we have to implement the **“create()”** function which is inherited from the **“SimObjectParams”** Python class. This “.hh” file is automatically created by gem5 at the time of rebuild. The file’s location is **“build/X86/params/<NAME>.hh”**. This create() is very important to be implemented for your “SimObject”.

run_matinv.py

This file is needed in order to simulate our results. The “root” is a data member of the m5 class, which is handed over the control to our python file. We imported the m5 objects. Then an instance of our main class is created, which inturn compiles the C++ file of the matrix multiplication using SimObjects. It also shows the total ticks in our clock cycle. This file basically runs or simulates our results of the code which we have written in C++ and python or both.

SConscript

In order to reflect all the changes made in the python and C++ files, we need to compile everything together. This “Scons” file includes all the other files which are needed to be compiled at the time of rebuild. Every directory/sub-directory contains a “SConscrit”. In our case, this file contains the main “SimObject” python class file, “.cc” file and the “DEBUG” flag statements.

Reference

- Code for Matrix Inverse: https://www.geeksforgeeks.org/adjoint-inverse-matrix/