ECE 511/ CSE 511: Computer Architecture

Simulation Assignment-2

- Aditya Saini, 2018125, B.Tech ECE

Introduction

- The objective of this assignment was to get familiar with the workflow behind the creation of Gem5 SimObjects. The required SimObject needed to perform matrix inversion while providing the functionality of displaying the input and output matrices through the use of DEBUG flags.
- As we were given freedom on using any input matrix, I've taken a 3x3 matrix = [[1,2,1],[2,1,3],[4,5,1]] as my input.
- The entire basis of this assignment was the Gem5 Simobject tutorial, available at http://learning.gem5.org/book/part2/index.html

Contents

- Basic Overview of the Directory Structure of Assignment
- Using the Matrix Inversion SimObject
    - **MATRIX Debug Flag**
    - **RESULT Debug Flag**
- Understanding the Codebase
    - **MatrixObject.py :** Wrapper class for MatrixObject defined in matrix_object.cc
    - **matrix_object.hh :** Header file for matrix_object.cc
    - **matrix_object.cc :** IMplementation of MatrixObject
    - **SConscript :** Build file for scons
    - **run_matrix.py :** Configuration file which calls SimObject

Basic Overview of the Directory Structure

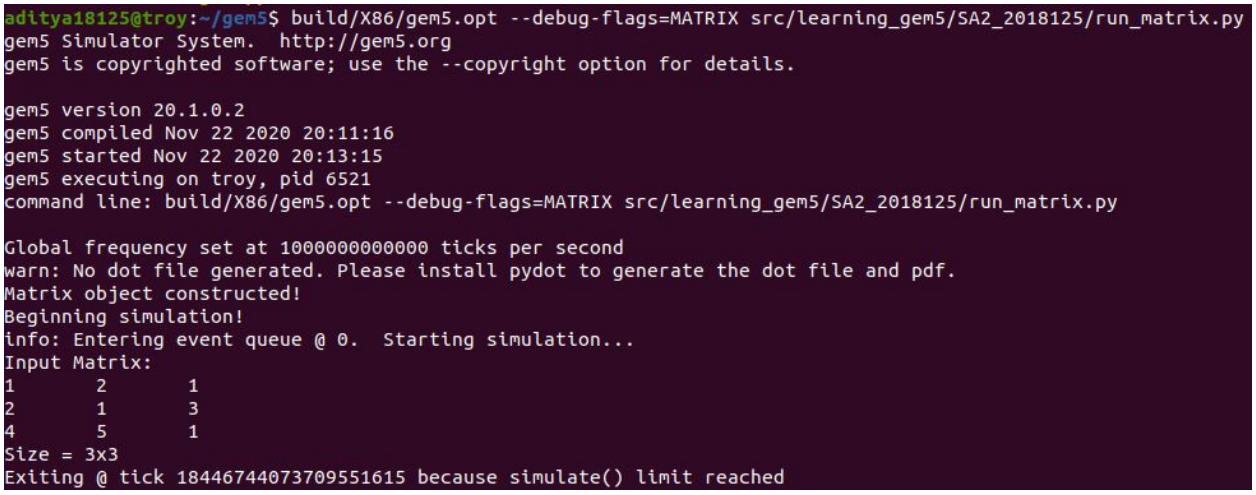
- The project was implemented in the /gem5/src/learning_gem5/SA2_2018125.
    - This was done in accordance with the one followed by the Gem5 tutorial
- So, the relative paths mentioned in the code are with respect to the mentioned directory.

Using the Matrix Inversion SimObject: MATRIX Debug Flag

To see the **input matrix and its size,** we can deploy the MATRIX debug flag by entering the following command

```
build/X86/gem5.opt --debug-flags=MATRIX src/learning_gem5/SA2_2018125/run_matrix.py
```

which outputs

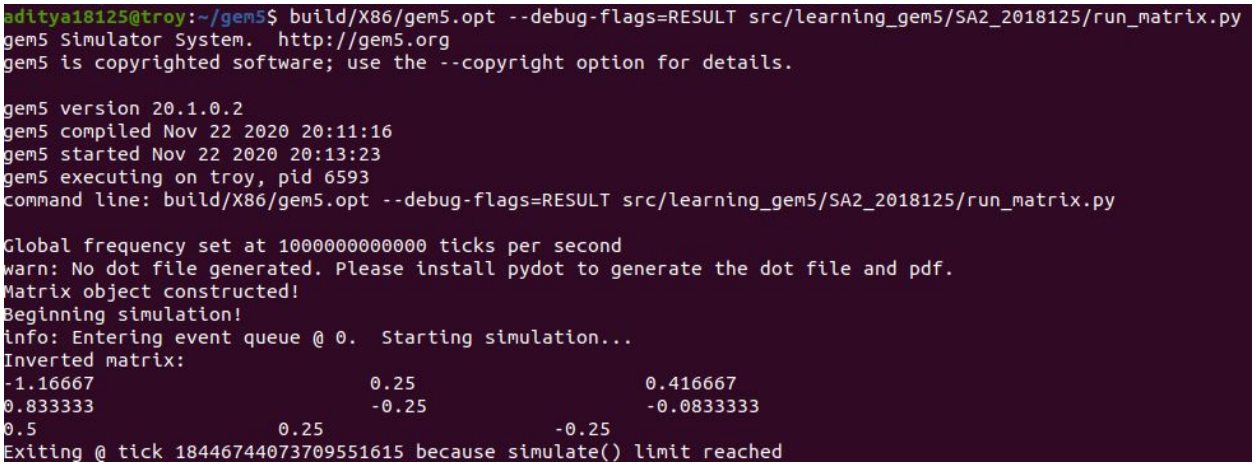


Using the Matrix Inversion SimObject: RESULT Debug Flag

To see the output/inverted, we can deploy the RESULT debug flag by entering the following command

```
build/X86/gem5.opt --debug-flags=RESULT src/learning_gem5/SA2_2018125/run_matrix.py
```

which outputs



Understanding the Codebase: MatrixObject.py

```
from m5.params import *
from m5.SimObject import SimObject

class MatrixObject(SimObject):
    type = 'MatrixObject'
    cxx_header = "learning_gem5/SA2_2018125/matrix_object.hh"
```

- The basic function of this file is to wrap the MatrixObject(written in C++; matrix_object.cc) in a Pythonic way so that we can pass it onto the config file which is written in Python(run_matrix.py).
- The cxx_header links to the header files corresponding to the MatrixObject.

Understanding the Codebase: matrix_object.hh

```
#ifndef __MATRIX_OBJECT__
#define __MATRIX_OBJECT__

#include "params/MatrixObject.hh"
#include "sim/sim_object.hh"

class MatrixObject : public SimObject
{
  private:
    //Defining the input matrix
    int input_matrix[3][3] ={{1,2,1},{2,1,3},{4,5,1}};

    //Defining the matrix size
    int matrix_size=3;

    //All the functionalities will be called in the mainEvent
    void mainEvent();

    //Declaring the inverse function
    double** inverse();

    //The EventWrapper wraps around mainEvent() and helps in scheduling it
    EventFunctionWrapper event;
  public:
    //Constructor of the MatrixObject
    MatrixObject(MatrixObjectParams *p);
    void startup(); // Helps in scheduling mainEvent()
};

#endif
```

- This file forms the header file for the matrix_object.cc file. It has all the function declarations and variable instantiations which are required for the MatrixObject to function.
- Please refer the comments for information on individual methods and variables.

Understanding the Codebase: matrix_object.cc

```
#include "learning_gem5/SA2_2018125/matrix_object.hh"
#include <iostream>
#include "base/trace.hh"
#include "debug/MATRIX.hh"
#include "debug/RESULT.hh"
#include <string>

//Implementing the constructor
MatrixObject::MatrixObject(MatrixObjectParams *params) :
      SimObject(params), event([this]{mainEvent();}, name())
{
    std::cout << "Matrix object constructed!" << std::endl;
}

//Scheduling the event so that it happens at the 100th tick
void
MatrixObject::startup()
{

    schedule(event, 100);
}

//Defining the function which returns the inverse of input_matrix
double**
MatrixObject::inverse(){
   //The code is useable for 3x3 matrices only
   //Inverse(Matrix A)=adjoint(A)/determinant(A)

   //Finding determinant
   double determinant = 0;
   for(int row=0;row<this->matrix_size;row++){
      determinant += this->input_matrix[0][row] *
(this->input_matrix[1][(row+1)%this->matrix_size]*this->input_matrix[2][(row+2)%
this->matrix_size]-this->input_matrix[1][(row+2)%this->matrix_size]*this->input_
matrix[2][(row+1)%this->matrix_size]);
   }
```

```
   //Finding result by dividing adjoint by determinant

   double** ans=new double*[this->matrix_size];
   for(int i = 0; i<this->matrix_size;i++){
      ans[i]=new double[this->matrix_size];
      for(int j=0;j<this->matrix_size;j++){
            double  adj  =
(this->input_matrix[(j+1)%3][(i+1)%3]*this->input_matrix[(j+2)%3][(i+2)%3]) -
(this->input_matrix[(j+1)%3][(i+2)%3]*this->input_matrix[(j+2)%3][(i+1)%3]);
            ans[i][j] = adj/determinant;
      }
   }

   return ans;
}




//Implementing the mainEvent
void
MatrixObject::mainEvent()
{ //Checking the debug flags

  //If MATRIX flag is on print the input matrix and matrix_size
   if(DTRACE(MATRIX)){
         std::cout<<"Input Matrix:\n";
         for(int i =0;i<this->matrix_size;i++){
            for(int j=0;j<this->matrix_size;j++){
                  std::cout<<this->input_matrix[i][j]<<"\t";
            }
            std::cout<<"\n";
         }
         std::cout<<"Size = "<< this->matrix_size << "x" << this->matrix_size <<
"\n";

   }
```

```
   //If RESULT flag is on, print the resultant inverted matrix
         else if(DTRACE(RESULT)){
         double** inverted_ans = inverse();
         std::cout<<"Inverted matrix:"<<"\n";
         for(int i = 0; i<this->matrix_size;i++){
                  for(int j=0;j<this->matrix_size;j++){
                  std::cout<< inverted_ans[i][j] <<"\t\t\t";
                  }
            std::cout<<"\n";
         }
   }

}

MatrixObject*
MatrixObjectParams::create()
{
    return new MatrixObject(this);
}
```

- This is the main MatrixObject fil. It contains
    - inverse(): Returns a 2d array containing the inverse of the input_matrix defined in matrix_object.hh
    - mainEvent(): Checks whichever DebugFlag is turned on, and calls the appropriate functions
    - startup(): Schedules mainEvent() at 100th tick

Understanding the Codebase: run_matrix.py

```
import m5
from m5.objects import *

root = Root(full_system = False)
root.matrix = MatrixObject()

m5.instantiate()

print("Beginning simulation!")
exit_event = m5.simulate()
print('Exiting @ tick {} because {}'.format(m5.curTick(),
exit_event.getCause()))
```

- This is the main config file. It calls the MatrixObject and starts the simulation.

Understanding the Codebase: SConscript

```
Import('*')

SimObject('MatrixObject.py')
Source('matrix_object.cc')

DebugFlag('MATRIX')
DebugFlag('RESULT')
```

- This files is used by scons to build the appropriate binaries for running the simulation.