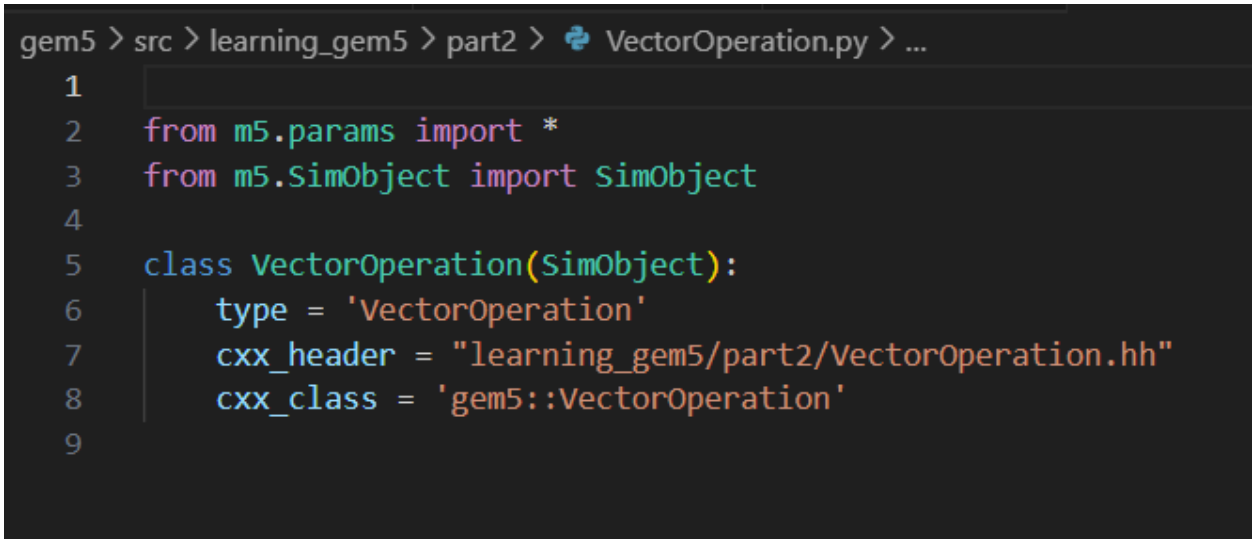
ASSIGHNMENT 2

Rajat Jaiswal 2021184

Creating VectorOperation.py



To begin, we create a file named "VectorOperation.py" and add the provided code to it. In the initial part of the code, we include necessary import statements from the "src/learninggem5/part2" module. Following that, we define a class named "VectorOperation," which takes "SimObject" as its argument. We set the type of this class as "VectorOperation" and declare it in the "cxx_header" section.

Creating hh File:

gem5 > src > learning_gem5 > part2 > VectorOperation.hh > ...

```
1
2   #ifndef _LEARNING_GEM5_VECTOROPERATION_HH_
3   #define _LEARNING_GEM5_VECTOROPERATION_HH_
4
5   #include <string>
6
7   #include "params/VectorOperation.hh"
8   #include "sim/sim_object.hh"
9
10  namespace gem5
11  {
12
13  class VectorOperation : public SimObject
14  {
15    private:
16      void VectorSubtraction();
17      EventFunctionWrapper event;
18      void VectorCrossProduct();
19      EventFunctionWrapper event1;
20      void VectorNormalize();
21      EventFunctionWrapper event2;
22
23    public:
24      VectorOperation(const VectorOperationParams &p);
25      void startup();
26  };
27
28  } //namespace gem5
29
30  #endif
```

Next, we create a file named "VectorOperation.hh" in the same folder. In the gem5 coding convention, it's essential to wrap all header files with #ifndef/#endif directives. In this file, you can observe that we begin with #ifndef and conclude with #endif. Within this structure, we define the SimObject within the "gem5" namespace, adhering to convention. To establish the inheritance of the "VectorOperation" class as a C++ SimObject, we indicate that "VectorOperation" is indeed a SimObject. Moving forward, we proceed to define the types for the two vectors and their sizes, specifically a 3x1 configuration as specified in the question. Furthermore, we declare the functions within the class, specifying the types of parameters they will accept. To maintain clarity and conformity, we also create EventWrappers to declare events for the functions we have defined.

Creating cc File:

gem5 > src > learning_gem5 > part2 > VectorOperation.cc

```
87
88
89  #include "learning_gem5/part2/VectorOperation.hh"
90  #include <iostream>
91  #include <cmath>
92  #include "base/trace.hh"
93  #include "debug/VECTOR.hh"
94  #include "debug/RESULTSUB.hh"
95  #include "debug/RESULTCROSS.hh"
96  #include "debug/NORMALIZE.hh"
97
98
99  namespace gem5
100 {
101
102 VectorOperation::VectorOperation(const VectorOperationParams &params) :
103     SimObject(params),event([this]{VectorSubtraction();}, name()),event1([this]{VectorCrossProduct();}, name()),event2([this]{VectorNormalize();}, n
104 {
105
106     double arr1[] = { 15, 25, 80};
107     double arr2[] = { 5, 10, 15};
108     DPRINTF(VECTOR, "Input Vector 1 :%f %f %f \n", arr1[0], arr1[1], arr1[2]);
109     DPRINTF(VECTOR, "Input Vector 2 :%f %f %f \n", arr2[0], arr2[1], arr2[2]);
110 }
```

We first include all the header files. We then have all our functions defined in the gem5 namespace. We then have constructor wherein we have hardcoded the vector values .

gem5 > src > learning_gem5 > part2 > VectorOperation.cc

```
111 void
112 VectorOperation::VectorSubtraction()
113 {
114     int finalAns[3];
115     double arr1[] = { 15, 25, 80};
116     double arr2[] = { 5, 10, 15};
117     for(int i=0; i< sizeof(finalAns)/sizeof(int); i++)
118     {
119         finalAns[i] = arr1[i] - arr2[i];
120
121     }
122
123     DPRINTF(RESULTSUB, " Subtraction : %d %d %d \n",finalAns[0],finalAns[1],finalAns[2]);
124 }
125
126 void
127 VectorOperation::VectorCrossProduct(){
128     double arr1[] = {15, 25, 80};
129     double arr2[] = {5, 10, 15};
130     double finalAns[3] = {0.0, 0.0, 0.0}; // Store the cross product as an array of three elements
131
132     finalAns[0] = arr1[1] * arr2[2] - arr1[2] * arr2[1];
133     finalAns[1] = arr1[2] * arr2[0] - arr1[0] * arr2[2];
134     finalAns[2] = arr1[0] * arr2[1] - arr1[1] * arr2[0];
135
136     DPRINTF(RESULTCROSS, "Cross Product: (%lf, %lf, %lf)\n", finalAns[0], finalAns[1], finalAns[2]); // Print the result using DPRINTF
137
138
139 }
140
```

```
void
VectorOperation::VectorNormalize(){
    double final1[3];
    double final2[3];
    double arr1[] = { 15, 25, 80};
    double arr2[] = { 5, 10, 15};
    double t = arr1[0]*arr1[0] + arr1[1]*arr1[1] + arr1[2]*arr1[2];
    double y = arr2[0]*arr2[0] + arr2[1]*arr2[1] + arr2[2]*arr2[2];
    double modA = sqrt(t);
    double modB = sqrt(y);

    for(int i = 0; i<3; i++){
        final1[i] = arr1[i]/modA;
        final2[i] = arr2[i]/modB;

    }

    DPRINTF(NORMALIZE, "NORMALIZE :%f %f %f \n",final1[0],final1[1],final1[2]);
    DPRINTF(NORMALIZE, "NORMALIZE :%f %f %f \n",final2[0],final2[1],final2[2]);
}
void
VectorOperation::startup(){
    schedule(event, 150);
    schedule(event1, 1500);
    schedule(event2,15000);
}
} // namespace gem5
```

Now, let's define the core functions within our "VectorOperation" class: Add Function: In this function, we perform vector Subtraction by subtracting corresponding values of both matrices and storing the result in the resultant vector. We use the "DPRINTF" function to call the debug flag, which in this case is named "RESULTSUB." cross Product Function: This function calculates the cross product of two vectors. The output is then displayed as the result. The debug flag for this function is named "RESULTCROSS." Normalization Function: This function accepts two vectors of size 3x1 as parameters and performs vector normalization. For each vector, it first calculates the sum of squared elements and then divides each element by the square root of the sum. Two additional for loops are used to store the results in new vectors and display the result. The debug flag for this function is named "NORMALIZE." These functions encapsulate essential vector operations within the "VectorOperation" class and are equipped with debug flags for troubleshooting and monitoring.

Making run_copy.py file

gem5 > src > learning_gem5 > part2 > run_copy.py

```
import m5
from m5.objects import *

root = Root(full_system = False)

root.hell = VectorOperation()

# instantiate all of the objects we've created above
m5.instantiate()

print("Beginning simulation!")
exit_event = m5.simulate()
print('Exiting @ tick %i because %s' % (m5.curTick(), exit_event.getCause()))
```

Making Sconsript file

gem5 > src > learning_gem5 > part2 > SConscript

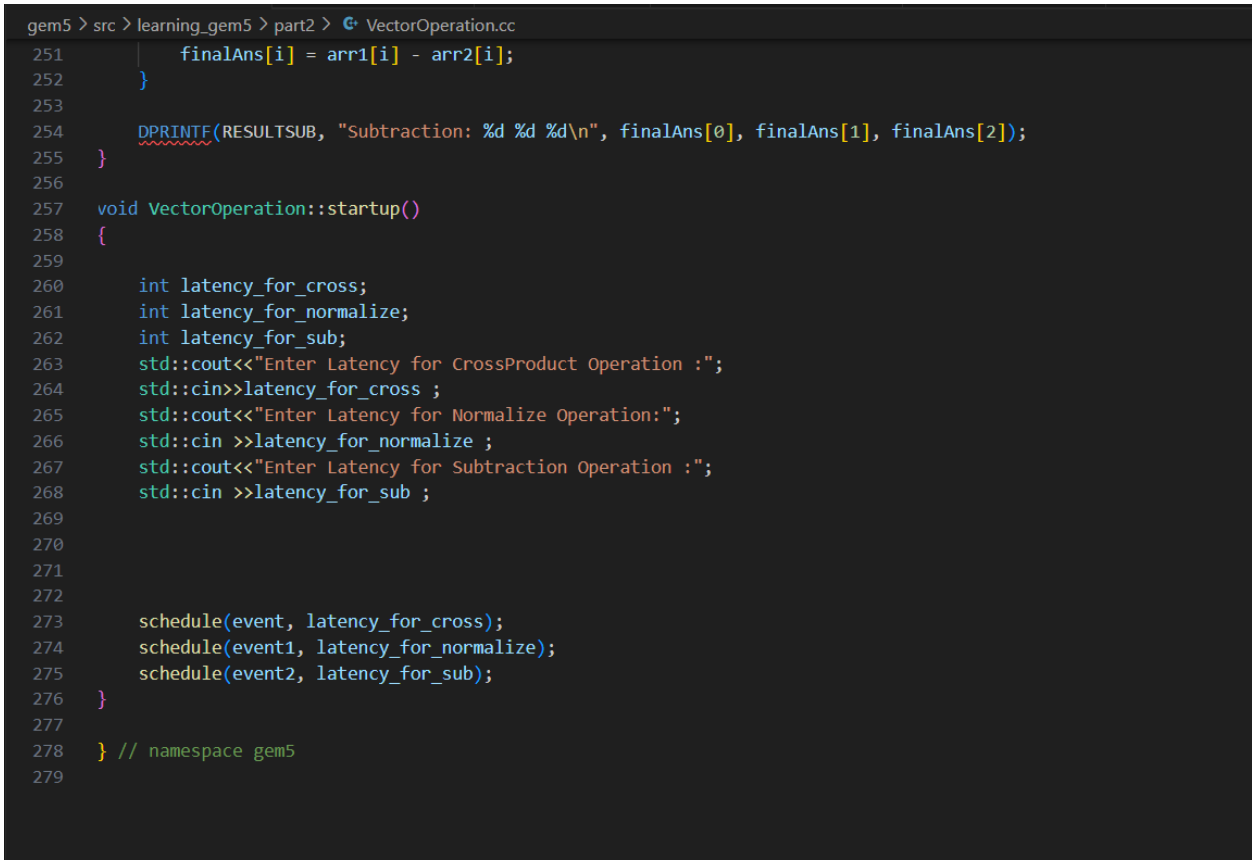
```
1
2   Import('*')
3
4   SimObject('VectorOperation.py', sim_objects=['VectorOperation'])
5   Source('VectorOperation.cc')
6   DebugFlag('VECTOR')
7   DebugFlag('RESULTSUB')
8   DebugFlag('RESULTCROSS')
9   DebugFlag('NORMALIZE')
```

Creating the above Scons file so that our c++ and python codes are compiled.

Bonus part:

```cpp
#include "learning_gem5/part2/VectorOperation.hh"
#include <iostream>
#include <cmath>
#include "base/trace.hh"
#include "debug/VECTOR.hh"
#include "debug/RESULTSUB.hh"
#include "debug/RESULTCROSS.hh"
#include "debug/NORMALIZE.hh"

// Declare arr1 and arr2 as global variables
double arr1[3];
double arr2[3];

namespace gem5 {

VectorOperation::VectorOperation(const VectorOperationParams &params) :
    SimObject(params),
    event([this]{ VectorCrossProduct(); }, name()),
    event1([this]{ VectorNormalize() ; }, name()),
    event2([this]{ VectorSubtraction(); }, name())
{

    // Initialize arr1 and arr2 with zeros
    for (int i = 0; i < 3; ++i) {
        arr1[i] = 0.0;
        arr2[i] = 0.0;
    }

    // Read user input for arr1 and arr2
    for (int i = 0; i < 3; ++i) {
        std::cout << "Enter a value for arr1[" << i << "]: ";
        std::cin >> arr1[i];
    }

    for (int i = 0; i < 3; ++i) {
        std::cout << "Enter a value for arr2[" << i << "]: ";
        std::cin >> arr2[i];
```

```cpp
void VectorOperation::VectorCrossProduct()
{
    double finalAns[3];
    finalAns[0] = arr1[1] * arr2[2] - arr1[2] * arr2[1];
    finalAns[1] = arr1[2] * arr2[0] - arr1[0] * arr2[2];
    finalAns[2] = arr1[0] * arr2[1] - arr1[1] * arr2[0];

    DPRINTF(RESULTCROSS, "Cross Product: (%f, %f, %f)\n", finalAns[0], finalAns[1], finalAns[2]);
}

void VectorOperation::VectorNormalize()
{
    double modA = sqrt(arr1[0] * arr1[0] + arr1[1] * arr1[1] + arr1[2] * arr1[2]);
    double modB = sqrt(arr2[0] * arr2[0] + arr2[1] * arr2[1] + arr2[2] * arr2[2]);

    double final1[3];
    double final2[3];

    if (modA != 0.0 && modB != 0.0) {
        for (int i = 0; i < 3; i++) {
            final1[i] = arr1[i] / modA;
            final2[i] = arr2[i] / modB;
        }
    } else {
        // Handle the case where modA or modB is zero (division by zero)
        // You can choose an appropriate action here, such as setting final1 and final2 to zero vectors.
    }

    DPRINTF(NORMALIZE, "Normalized First Vector (arr1): %f %f %f\n", final1[0], final1[1], final1[2]);
    DPRINTF(NORMALIZE, "Normalized Second Vector (arr2): %f %f %f\n", final2[0], final2[1], final2[2]);
}

void VectorOperation::VectorSubtraction()
{
    int finalAns[3];
    for (int i = 0; i < 3; i++) {
```

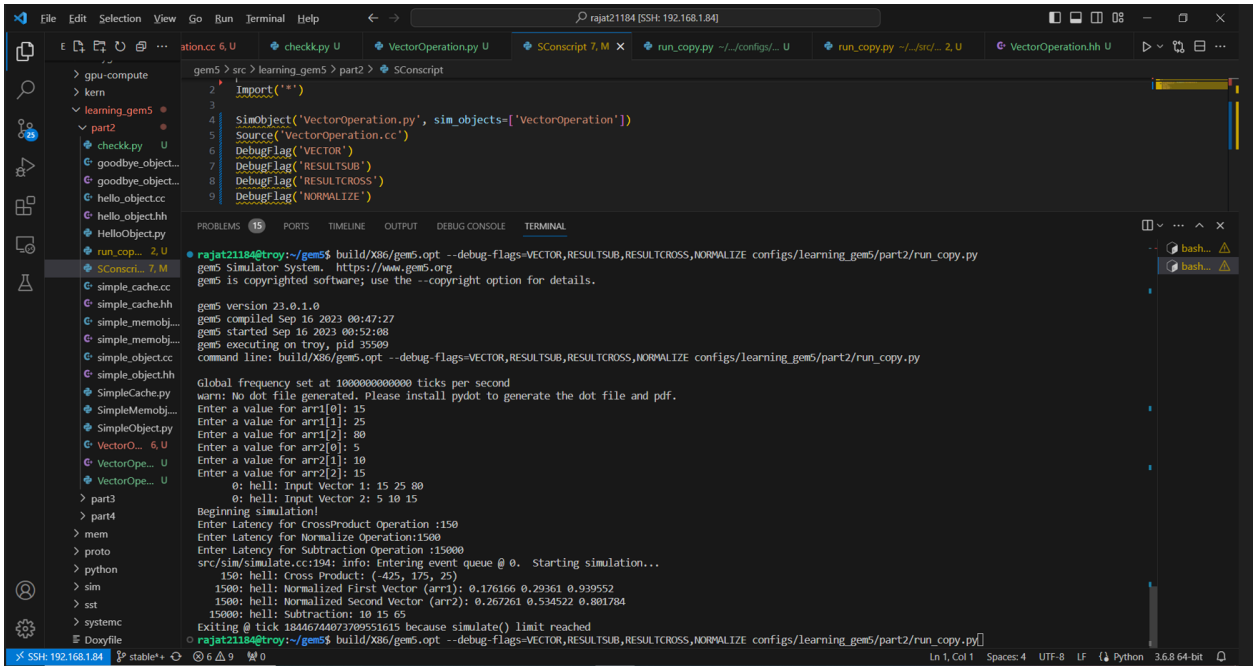


Some changes were made to VectorOperation.py, VectorOperations.cc and the so that the functionality for obtaining and using use inputs is implemented. Input is taking the value for arr1 values and arr2 values and taking the latency input for RESULTCROSS ,NORMALIZE ,RESULTSUB

Command used:
python3 `which scons` build/X86/gem5.opt -j4

build/X86/gem5.opt --debug-flags=VECTOR,RESULTSUB,RESULTCROSS,NORMALIZE configs/learning_gem5/part2/run_copy.py

Output


File Edit Selection View Go Run Terminal Help
rajat21184 [SSH: 192.168.1.84]
checkk.py U
VectorOperation.py U
SConscript 7, M
run_copy.py ~/.../configs/... U
run_copy.py ~/.../src/... 2, U
VectorOperation.hh U
gem5 > src > learning_gem5 > part2 > SConscript
Import('*')
SimObject('VectorOperation.py', sim_objects=['VectorOperation'])
Source('VectorOperation.cc')
DebugFlag('VECTOR')
DebugFlag('RESULTSUB')
DebugFlag('RESULTCROSS')
DebugFlag('NORMALIZE')
PROBLEMS 15 PORTS TIMELINE OUTPUT DEBUG CONSOLE TERMINAL
rajat21184@troy:~/gem5$ build/X86/gem5.opt --debug-flags=VECTOR,RESULTSUB,RESULTCROSS,NORMALIZE configs/learning_gem5/part2/run_copy.py
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.
gem5 version 23.0.1.0
gem5 compiled Sep 16 2023 00:47:27
gem5 started Sep 16 2023 00:52:08
gem5 executing on troy, pid 35509
command line: build/X86/gem5.opt --debug-flags=VECTOR,RESULTSUB,RESULTCROSS,NORMALIZE configs/learning_gem5/part2/run_copy.py
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
Enter a value for arr1[0]: 15
Enter a value for arr1[1]: 25
Enter a value for arr1[2]: 80
Enter a value for arr2[0]: 5
Enter a value for arr2[1]: 10
Enter a value for arr2[2]: 15
0: hell: Input Vector 1: 15 25 80
0: hell: Input Vector 2: 5 10 15
Beginning simulation!
Enter Latency for CrossProduct Operation :150
Enter Latency for Normalize Operation:1500
Enter Latency for Subtraction Operation :15000
src/sim/simulate.cc:194: info: Entering event queue @ 0. Starting simulation...
150: hell: Cross Product: (-425, 175, 25)
1500: hell: Normalized First Vector (arr1): 0.176166 0.29361 0.939552
1500: hell: Normalized Second Vector (arr2): 0.267261 0.534522 0.801784
15000: hell: Subtraction: 10 15 65
Exiting @ tick 18446744073709551615 because simulate() limit reached
rajat21184@troy:~/gem5$ build/X86/gem5.opt --debug-flags=VECTOR,RESULTSUB,RESULTCROSS,NORMALIZE configs/learning_gem5/part2/run_copy.py
SSH: 192.168.1.84
Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.6.8 64-bit