ReadMe

Code Explanation:

1) We first created our own class VectorOperations, which inherits from the SimObject Class as its base class.

```
class VectorOperations(SimObject):
    type = 'VectorOperations'
    cxx_header = "a2/vector_operations.hh"
    cxx_class = "gem5::VectorOperations"
    sch_cross = Param.Int(150, "time for cross")
    sch_norm = Param.Int(1500, "time for norm")
    sch_sub = Param.Int(15000, "time for sub")
    v1_0 = Param.Float("v1_0")
    v1_1 = Param.Float("v1_1")
    v1_2 = Param.Float("v1_2")
    v2_0 = Param.Float("v2_0")
    v2_1 = Param.Float("v2_1")
    v2_2 = Param.Float("v2_2")
```

The cxx_header files contains the .hh file where the C++ class structure of this class resides and the cxx_class file defines the class constructor and functions.
The next 9 lines are parameters to set custom values for the class members.
Since the events are called at ‘ticks’ we take input as Int.
Vector input can be floating point numbers as well, hence have used Float.

2) next we define the .hh file which contains the class structure:

```
class VectorOperations : public SimObject
{
  private:

    std::vector<std::vector<float>> v1;
    std::vector<std::vector<float>> v2;

    void FnVectorCrossProduct();
    void FnNormalizeVector();
    void FnVectorSubtraction();


    EventFunctionWrapper VectorCrossProduct;
    EventFunctionWrapper NormalizeVector;
    EventFunctionWrapper VectorSubtraction;

    int tsub;
    int tnor;
    int tcross;

    float v1_0;
    float v1_1;
    float v1_2;

    float v2_0;
    float v2_1;
    float v2_2;

  public:

    VectorOperations(const VectorOperationsParams &p);
    // VectorOperations(const VectorOperationsParams &p, std::vector<float> a, std::vector<float> b);
    virtual void startup() override;
};
```

- The vectors v1 and v2 are the input vectors on which operations are performed.
- EventWrapper wraps a function which is called whenever the event is scheduled to execute.
- The tsub, tnor .. v2_2 are class members which are used to take set user input.
- The constructor is called at tick 0 and sets default values
- startup() is a special function that is executed at tick 0.

3) Next we define the functions in the .cc file.
First the constructor:

- In the constructor we connect the events to the function which is executed whenever called. [this] captures the whole event and class the function named after it.
- We also set the parameters using the 'params.' tsub(params.sch_sub) means the class member tsub takes value stored in sch_sub(defined in python class)

```
VectorOperations::VectorOperations(const VectorOperationsParams &params) :
    SimObject(params),
    VectorSubtraction([this]{FnVectorSubtraction();}, name()) ,
    VectorCrossProduct([this]{FnVectorCrossProduct();}, name()),
    NormalizeVector([this]{FnNormalizeVector();}, name()),
    tsub(params.sch_sub),
    tnor(params.sch_norm),
    tcross(params.sch_cross),
    v1_0(params.v1_0),
    v1_1(params.v1_1),
    v1_2(params.v1_2),
    v2_0(params.v2_0),
    v2_1(params.v2_1),
    v2_2(params.v2_2)
{
```

We schedule the events in start-up function since start-up is executed at tick 0.

```
void
VectorOperations::startup()
{
    // startup is a special function executed at tick 0.
    // std::cout << "We are inside start-up function";

    // schedule the events
    schedule(VectorCrossProduct, tcross);
    schedule(NormalizeVector, tnor);
    schedule(VectorSubtraction, tsub);

}
```

Finally, define the three functions and functionalities. Below subtraction is defined:

```
void
VectorOperations::FnVectorSubtraction()
{
    // Perform vector cross
    std::vector<std::vector<float>> ans;
    // std::cout<<"Performing v1 - v2";
    ans = {{v1[0][0] - v2[0][0]}, {v1[1][0] - v2[1][0]}, {v1[1][0] - v2[1][0]} };
    DPRINTF(RESULTSUB, "\n Subtraction of the vector is: %f %f %f  \n", ans[0][0], ans[1][0], ans[2][0]);
}
```

4) Next, the sconscript.

```
a2 > SConscript
  Import('*')

  SimObject('VectorOperations.py', sim_objects=['VectorOperations'])
  Source('vector_operations.cc')

  DebugFlag('VECTOR')
  DebugFlag('RESULTCROSS')
  DebugFlag('NORMALIZE')
  DebugFlag('RESULTSUB')
```

- Since we have created our own object deriving from sim object we tell gem5 about it using sim_objects.
- Whenever debug flags are used, those lines involving the flag are outputted on the terminal.
- Gem5 automatically generates "debug/<flag_name>.hh" for each debug flag. These files are then used in .cc files to identify the flags using #include.

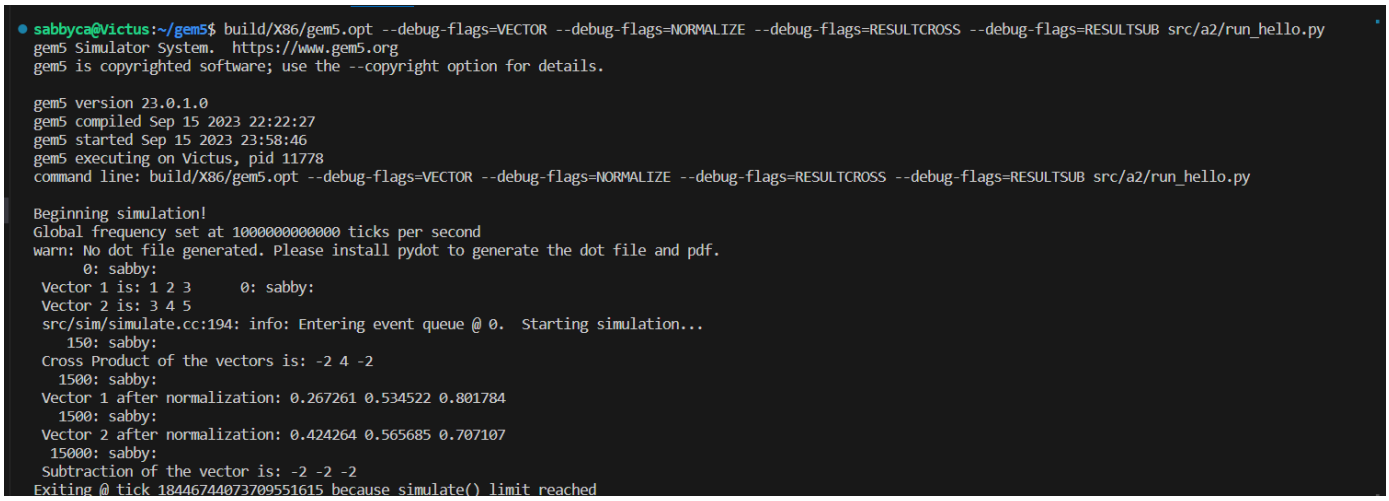
5) The main runner file:

- In the main runner file define a root. Then define a object of type VectorOperations.
- Next we need to set values of user params of the object.
- For that I have used argparse as defined in gem5 documentation.

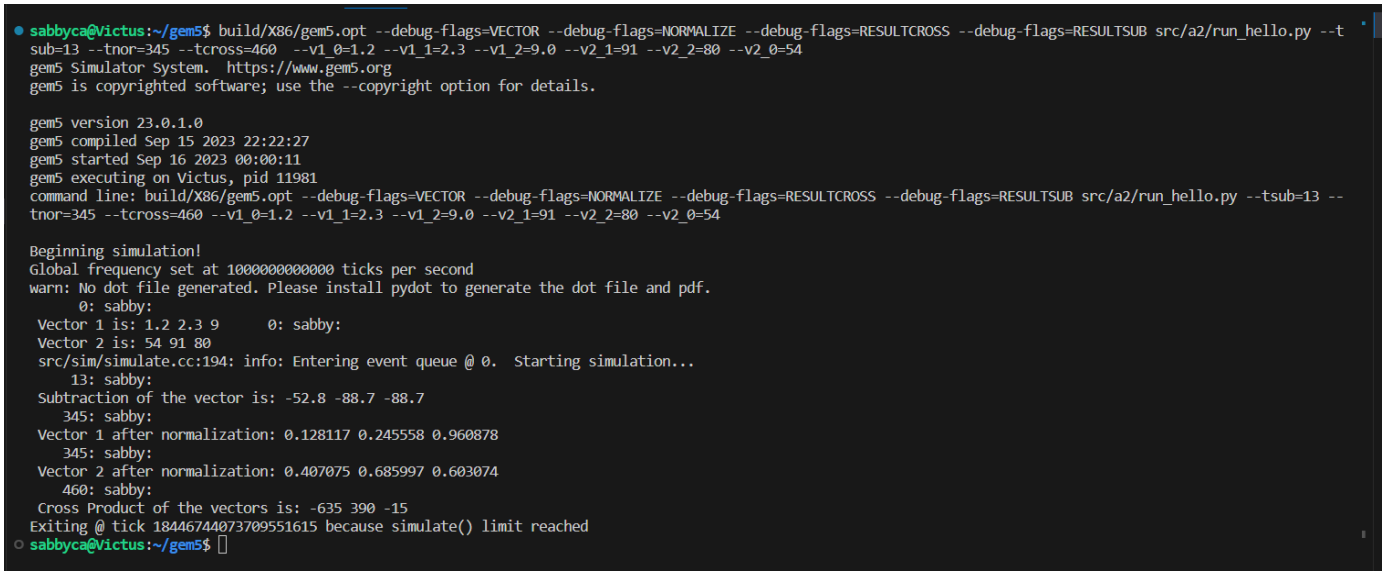
```
parser.add_argument("--v2_0", default=3, help="Flag for setting value of v2_0.")
parser.add_argument("--v2_1", default=4, help="Flag for setting value of v2_1.")
parser.add_argument("--v2_2", default=5, help="Flag for setting value of v2_2.")


args = parser.parse_args()

root.sabby = VectorOperations()

root.sabby.sch_sub = args.tsub
root.sabby.sch_norm = args.tnor
```

Sample Output:

Without passing any custom parameters for schedule time and vector this is the output:



When custom parameters are used for schedule time and vector this is the output:



How to run the code:

First, create a folder name ‘a2’ in the src folder of gem5. Now extract the zip and add all files into this folder.

Next, Build scons using:

```
scons build/X86/gem5.opt
```

If you want to use default parameters, then use:

```
build/X86/gem5.opt --debug-flags=VECTOR --debug-flags=NORMALIZE
--debug-flags=RESULTCROSS --debug-flags=RESULTSUB  src/a2/run_hello.py
```

If you want to specify the value of parameters, use the command below. If you do not want to specify all the parameters, then simply remove the –<name> from the command below and the code will take default values for those.

```
build/X86/gem5.opt --debug-flags=VECTOR --debug-flags=NORMALIZE
--debug-flags=RESULTCROSS --debug-flags=RESULTSUB src/a2/run_hello.py
--tsub=13 --tnor=345 --tcross=460  --v1_0=1.2 --v1_1=2.3 --v1_2=9.0
--v2_1=91 --v2_2=80 --v2_0=54
```

Where,
tsub = schedule time for event subtraction
tcross = schedule time for event cross product
tnor = schedule time for event Normalize
vi_j = value for vector i's jth index where i = {1,2} and j = {0,1,2}

References:

https://www.gem5.org/documentation/learning_gem5/part2/events/
https://www.gem5.org/documentation/learning_gem5/part2/helloobject/
https://www.gem5.org/documentation/learning_gem5/part2/debugging/
https://www.gem5.org/documentation/learning_gem5/part1/cache_config/ (for taking command line input, the argparse section is mentioned here)