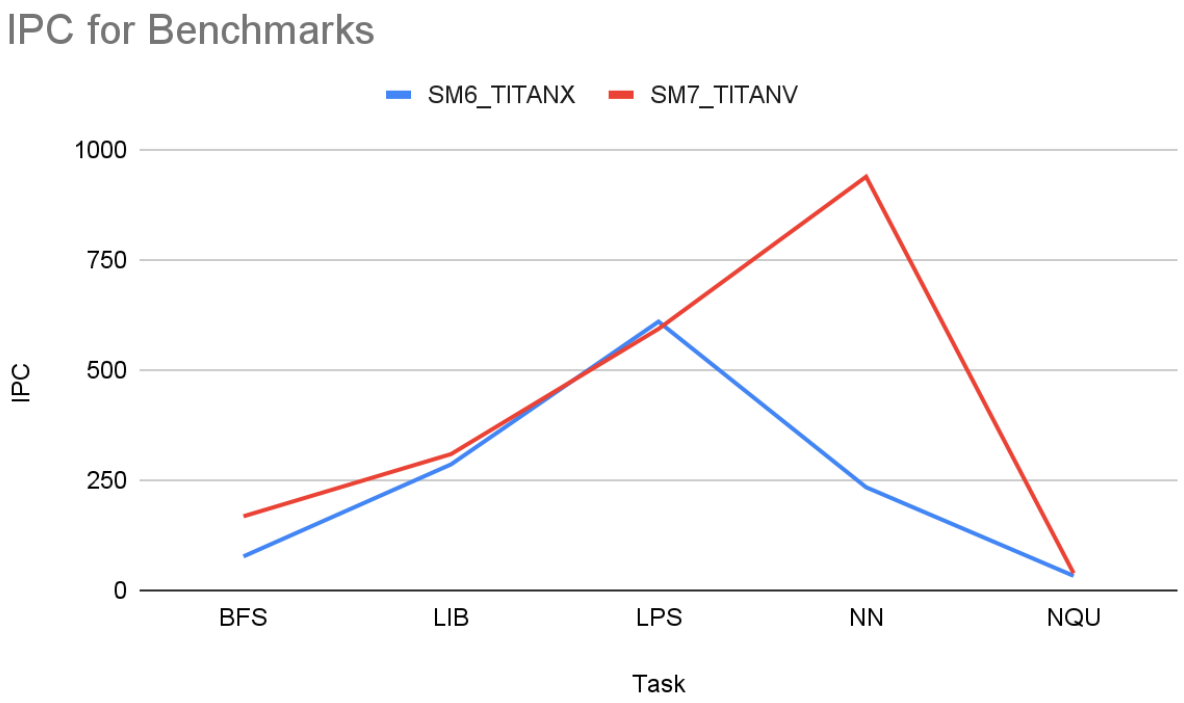
Programming Assignment 2
ECE 786
Shawn Salekin

Task 1 (ISA)

I made changes to the `add_impl` in `instructions.cc`. Specifically, I changed the add operations implementations where the operands are floating points. The instructions are performed on the hardware, as opposed to being emulated completely, which is what the functional simulator does, based on gpgpu-sim documentation.

Task 2 (Benchmark)

IPC vs benchmark graph after running them.



Looking at the graph, it is obvious that SM7_TITANV has a higher IPC count for almost all the benchmarks. This could be attributed to a couple of things:

1. SM6_TITANX configs have the L1 data cache disabled, which results in a higher number of pipeline stalls and thus reduced IPC
2. SM7_TITANV has more functional units, specifically, 4 DP units, 4 INT units and 1 Tensor core

There are other factors as well, but these two emerge as good reasons why SM7_TITANV has a better IPC than SM6_TITANX.

Task 3 (Branch Divergence)

I made changes to `abstract_hardware_model[.c, .hh]` files and `src/gpgpu-sim/shader[.cc, .h]` files.

Specifically, we added a flag in the simt_stack class which indicates whether the warp has seen any divergence. When `simt_stack::update` function, that flag is updated when the existing logic figures out there is indeed a divergence. We leverage that code to update the divergence flag for that simt_stack instance in the `scheduler_unit::cycle()`. Since this function is called for every instruction that is executed for all the warps, it is a safe place to check for divergence. Similarly, we check whether an instruction is a conditional branch instruction in that same function using existence of predication guard and opcode.

Both of these events are stored in a simple map keyed by warp_id, as to keep track of which warps have had divergence and/or conditional branch. At the end, the report collects the size of the map (since map does not allow duplicates), which should give us the correct number of divergent and conditional branched warps.

This is from the test run on LPS benchmark:

```
# of warps executed conditional branch instructions and have divergence: 8
# of warps executed conditional branch instructions(no matter they have divergence or not): 8
```

Task 4 (Memory Access)

Memory access is counted in `ldst_unit::memory_cycle()` function in shader.cc file. Since this function is called every time there is an access to the L1 Data cache, we can reliably collect the access type and the number of times global and local memory was accessed. We implemented two counters to keep track of local and global accesses. There is a simple check at the beginning of the aforementioned function and based on local or global, the corresponding variable is updated.

It is worth noting that in both local and global memory access happen through L1 Data Cache (according to the documentation) which is why we made the decision implement the counter there.

This is from a test ran on LPS benchmark:

```
total global accesses=448400
total local accesses=0
```