ECE 786

Program assignment 2

Report
chandu yuvarajappa (cyuvara)

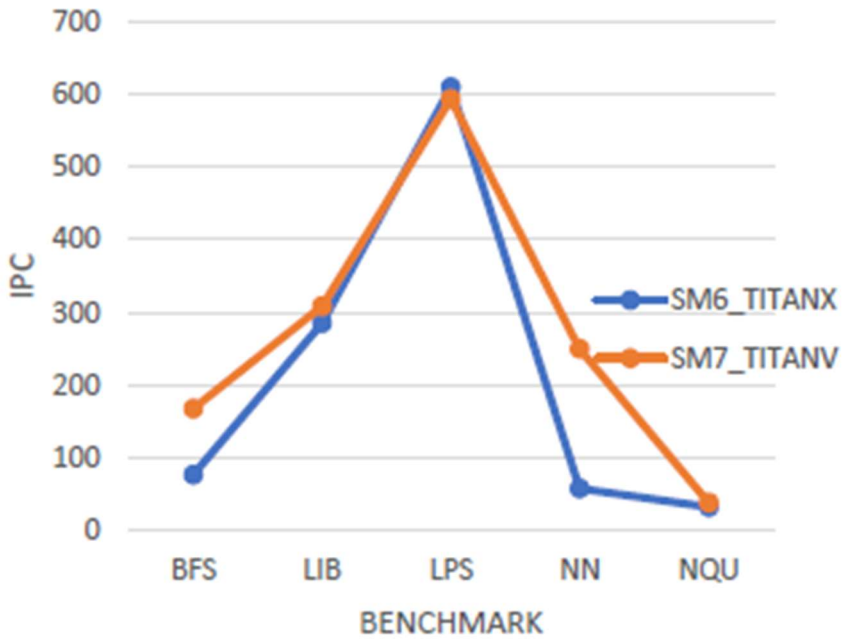
Task 1 :

It is necessary to convert the float(32 bits) point instruction into a power instruction for the task. Instructions.cc is where the actual add implementation for "vectorAdd" is carried out. Instructions.cc provides implementations for each operation, including add, and, branch, division, and multiplication, with the corresponding function names add impl, and impl, bra impl, div impl, and mul impl. Add impl has been updated for the floating point situation in our task. The real floating point add code has been modified to the floating point pow function as follows:

```
case F32_TYPE:
  //data.f32 = src1_data.f32 + src2_data.f32;
  data.f32 = cuda_math::__powf(src1_data.f32 , src2_data.f32);//project 2 experiment
  break;
```

So, this function would be called and the pow function would be executed anytime a floating point add is done in code.

Task 2:



LPS benchmark shows the greatest IPC while NQU shows the lowest IPC for both the configuration files SM6 TITANX and SM7 TITANV. In both configuration files, I included the option -gpgpu max insn 100000000 to restrict the amount of instructions that may be executed to a maximum of 100000000. (some of the benchmarks has less number of instructions than this). Benchmarks BFS, LIB, LPS, and NQU take about 6 minutes, 3 minutes, and under a minute, respectively, to execute. For NN, an execution time of up to 30 minutes is recorded.

Task 3:

My observations show that any warps are sent to the scheduler by using the following code in shader.cc. This function performs tasks like checking the scoreboard and validating the instruction type, validity, and SIMT stack.

```
void shader_core_ctx::issue_warp(register_set &pipe_reg_set,
                                 const warp_inst_t *next_inst,
                                 const active_mask_t &active_mask,
                                 unsigned warp_id, unsigned sch_id) {
  warp_inst_t **pipe_reg =
      pipe_reg_set.get_free(m_config->sub_core_model, sch_id);
  assert(pipe_reg);
```

To update SIMT stack following function in abstraction_hardware_model.cc is called from issue_warp()

```
void core_t::updateSIMTStack(unsigned warpId, warp_inst_t *inst) {
  simt_mask_t thread_done;
  addr_vector_t next_pc;
  unsigned wtid = warpId * m_warp_size;
  for (unsigned i = 0; i < m_warp_size; i++) {
    if (ptx_thread_done(wtid + i)) {
      thread_done.set(i);
      next_pc.push_back((address_type)-1);
    } else {
      if (inst->reconvergence_pc == RECONVERGE_RETURN_PC)
        inst->reconvergence_pc = get_return_pc(m_thread[wtid + i]);
      next_pc.push_back(m_thread[wtid + i]->get_pc());
    }
  }
  if(inst->op==BRA_OP) total_branches++;//project 2 experiment
  m_simt_stack[warpId]->update(thread_done, next_pc, inst->reconvergence_pc,
                               inst->op, inst->isize, inst->pc);
}
```

I've added the following code to this method .

```
if(inst->op==BRA_OP) total_branches++;//project 2 experiment
m_simt_stack[warpId]->update(thread_done, next_pc, inst->reconvergence_pc,
                             inst->op, inst->isize, inst->pc);
```

Every time a warp is issued, the updateSIMTStack job is called, so I utilize this function to determine whether a branch instruction is present and to increase the "tot branches" counter (this counter gives the number of warps executed conditional branch instructions irrespective whether they diverged or not). Every warp calls the following code to update the associated stack pointer.

Function call in updateSIMTStack:

```
m_simt_stack[warpId]->update(thread_done, next_pc, inst->reconvergence_pc,
                             inst->op, inst->isize, inst->pc);
```

In this function, the real stack change takes place. In this code, I've utilized a previously declared "warp diverged" variable to track if the warp displays divergence. If op type is BRANCH and

warp diverged==1, the function's "tot div branches" counter is also increased. The files gpu sim.h, abstraction hardware model.cc, and gpgpusim entrypoint.cc include declarations for the counters tot branches and tot div branches as well as code that uses those counters. The variables stated are accessible to both files as gpu sim.h is included in both abstraction hardware model.cc and gpgpusim entrypoint.cc. The variables are reported in the print simulation time() function of the gpgpusim entrypoint.cc file.

```
  printf("gpgpu_silicon_slowdown = %ux\n",
         the_gpgpusim->g_the_gpu->shader_clock() * 1000 / cycles_per_sec);
  printf("# warps executed conditional branch instructions and have divergence=%d\n",total_divergence_branches);
  printf("# warps executed conditional branch instructions(no matter they have divergence or not)=%d\n",total_branches);
  fflush(stdout);
```

With SM7_TITANV

```
gpgpu_silicon_slowdown = 2727272x
# of warps executed conditional branch instructions and have divergence: 54356
# of warps executed conditional branch instructions(no matter they have divergence or not): 247600
GPGPU-Sim: synchronize waiting for inactive GPU simulation
```

with SM6_TITANX

```
gpgpu_silicon_slowdown = 1645760x
# of warps executed conditional branch instructions and have divergence: 54356
# of warps executed conditional branch instructions(no matter they have divergence or not): 247600
GPGPU-Sim: synchronize waiting for inactive GPU simulation
```

Task 4:

The shader.cc file's ldst unit::cycle() function has been updated by me to count both global and local accesses. I picked this function to carry out the operation since it is called each time an instruction is sent to load a storage unit.

In the function, I have increased the global mem access counter if the access is either GLOBAL ACC R || GLOBAL ACC W. Otherwise, I have increased the local mem access counter if the access is either LOCAL ACC R || LOCAL ACC W.

```
mem_fetch *mf = m_response_fifo.front();
if(mf->get_access_type() == GLOBAL_ACC_R || mf->get_access_type() == GLOBAL_ACC_W )global_memory_access++;//project 2 eperiment
if(mf->get_access_type() == LOCAL_ACC_R || mf->get_access_type() == LOCAL_ACC_W )local_memory_access++;//project 2 experiment
if (mf->get_access_type() == TEXTURE_ACC_R) {
```

The gpu sim.h file declares, shader.cc file uses, and gpgpusim entrypoint.cc file monitors and accesses the counters global mem access and local mem access. The variables declared are accessible from both shader.cc and gpgpusim entrypoint.cc as gpu sim.h is included in both programs. Both variables are reported in the print simulation time() function of gpgpusim entrypoint.cc.

```
        the_gpgpusim->g_the_gpu->shader_clock() * 1000 / cycles_per_sec);
printf("# of global memory access:%d\n",global_memory_access);//Project 2 experiment
printf("# of local memory access:%d\n",local_memory_access);//Project 2 experiment
fflush(stdout);
```
:w

With SM6_TITANX

```
gpgpu_silicon_slowdown = 1661195x
# of global memory access:476746
# of local memory access:0
GPGPU-Sim: synchronize waiting for inactive GPU simulation
```

With SM7_TITANV

```
gpgpu_silicon_slowdown = 2727272x
# of global memory access:404000
# of local memory access:0
GPGPU-Sim: synchronize waiting for inactive GPU simulation
```