Convolution Operation on FPGA-CPU Heterogeneous System

Wei Cheng

wchengad@connect.ust.hk

May 23, 2019

Overview

Introduction

Background

- Artificial Intelligence (AI), Computer Vision, Natural Language Processing have experienced great success in many practical applications, driven by the development of heterogeneous computing. Recently, multiple heterogeneous systems are applied into these applications, for example, CPU-GPU and CPU-FPGA.
- Convolution is one of the basis linear operation in the previously mentioned techniques.

Target

- To realize convolution operation in one of the heterogeneous structure, for example, implement a kernel filter (Sobel Filter) for image processing.
- CPU and FPGA cooperation, including control, simulation and memory transition.

Related Work

- Power *et al.* proposed a simulation platform *gem5-gpu* on tightly integrated CPU-GPU systems, *gem5-gpu* is open source and available at gem5-gpu.cs.wisc.edu. Convolution layer in CPU-GPU systems will be implemented using CUDA C, evaluations on different characteristics will be profiled in simulation.
- Liang *et al.* developed a FPGA-CPU simulation system *PAAS* based on a CPU simulator gem5 and HDL simulator Verilator.
- Motamedi *et al.* realized a FPGA based CNN developed on Altera DE5 Net platform.

PAAS Simulation System

I started my implementataion based on the simulation platform proposed by Tianyuan Liang and Prof. Zhang. Processor Accelerator Architecture Simulator (PAAS) for heterogeneous computing system

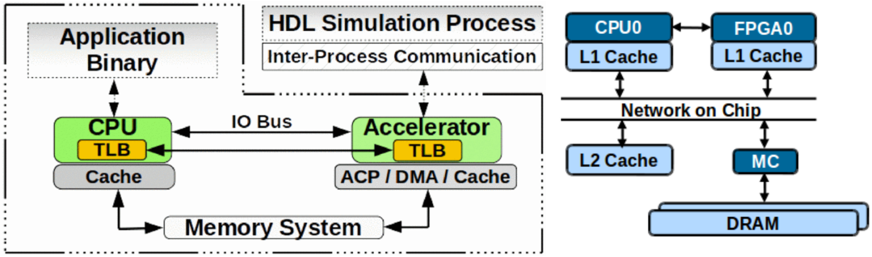
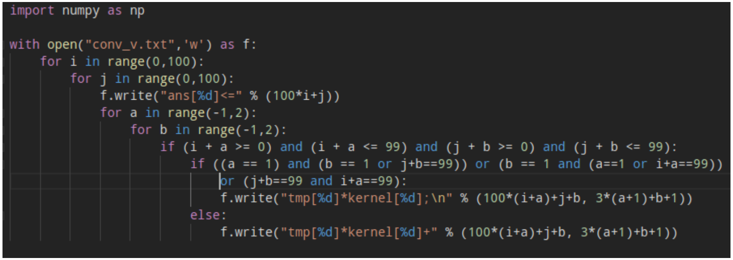


Figure: System pipeline of PAAS

Convolution in Verilog

I first use write a script to generate the following code in Verilog, which is a zero-padding 2D convolution with 100 $\times$ 100 input and 3 $\times$3 kernel.

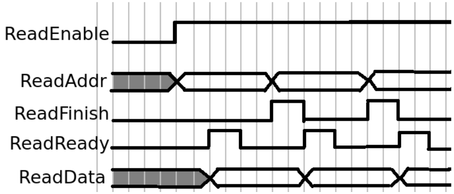


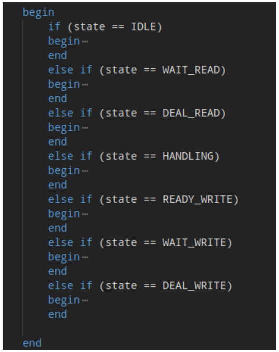
```python
import numpy as np

with open("conv_v.txt",'w') as f:
    for i in range(0,100):
        for j in range(0,100):
            f.write("ans[%d]<=" % (100*i+j))
            for a in range(-1,2):
                for b in range(-1,2):
                    if (i + a >= 0) and (i + a <= 99) and (j + b >= 0) and (j + b <= 99):
                        if ((a == 1) and (b == 1 or j+b==99)) or (b == 1 and (a==1 or i+a==99))
                        or (j+b==99 and i+a==99):
                            f.write("tmp[%d]*kernel[%d];\n" % (100*(i+a)+j+b, 3*(a+1)+b+1))
                        else:
                            f.write("tmp[%d]*kernel[%d]+" % (100*(i+a)+j+b, 3*(a+1)+b+1))
```

FPGA Finite State Machine

The FPGA always works in a FSM mode, for data transition, reset, and execution. For example, to comunicate with CPU, FPGA has to obey the following read/write timing.

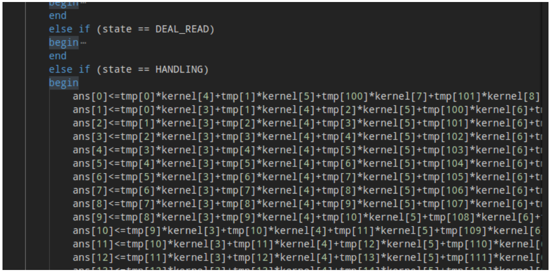




```
begin
    if (state == IDLE)
    begin…
    end
    else if (state == WAIT_READ)
    begin…
    end
    else if (state == DEAL_READ)
    begin…
    end
    else if (state == HANDLING)
    begin…
    end
    else if (state == READY_WRITE)
    begin…
    end
    else if (state == WAIT_WRITE)
    begin…
    end
    else if (state == DEAL_WRITE)
    begin…
    end
end
```

FPGA Handling

Then we should apply our generate 2D convolution code to the HANDLING state.



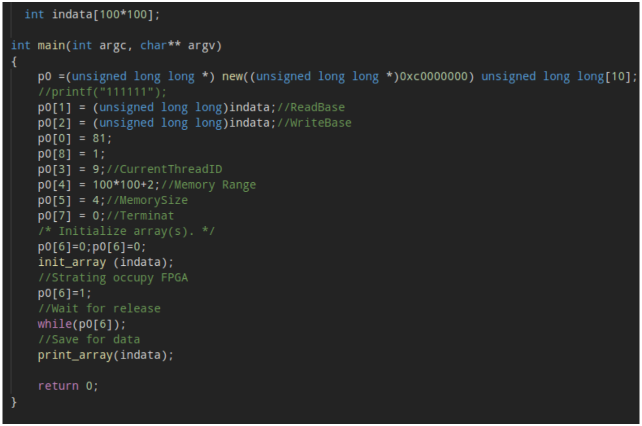
```
    begin…
    end
    else if (state == DEAL_READ)
    begin…
    end
    else if (state == HANDLING)
    begin
        ans[0]<=tmp[0]*kernel[4]+tmp[1]*kernel[5]+tmp[100]*kernel[7]+tmp[101]*kernel[8]
        ans[1]<=tmp[0]*kernel[3]+tmp[1]*kernel[4]+tmp[2]*kernel[5]+tmp[100]*kernel[6]+t
        ans[2]<=tmp[1]*kernel[3]+tmp[2]*kernel[4]+tmp[3]*kernel[5]+tmp[101]*kernel[6]+t
        ans[3]<=tmp[2]*kernel[3]+tmp[3]*kernel[4]+tmp[4]*kernel[5]+tmp[102]*kernel[6]+t
        ans[4]<=tmp[3]*kernel[3]+tmp[4]*kernel[4]+tmp[5]*kernel[5]+tmp[103]*kernel[6]+t
        ans[5]<=tmp[4]*kernel[3]+tmp[5]*kernel[4]+tmp[6]*kernel[5]+tmp[104]*kernel[6]+t
        ans[6]<=tmp[5]*kernel[3]+tmp[6]*kernel[4]+tmp[7]*kernel[5]+tmp[105]*kernel[6]+t
        ans[7]<=tmp[6]*kernel[3]+tmp[7]*kernel[4]+tmp[8]*kernel[5]+tmp[106]*kernel[6]+t
        ans[8]<=tmp[7]*kernel[3]+tmp[8]*kernel[4]+tmp[9]*kernel[5]+tmp[107]*kernel[6]+t
        ans[9]<=tmp[8]*kernel[3]+tmp[9]*kernel[4]+tmp[10]*kernel[5]+tmp[108]*kernel[6]+
        ans[10]<=tmp[9]*kernel[3]+tmp[10]*kernel[4]+tmp[11]*kernel[5]+tmp[109]*kernel[6
        ans[11]<=tmp[10]*kernel[3]+tmp[11]*kernel[4]+tmp[12]*kernel[5]+tmp[110]*kernel[
        ans[12]<=tmp[11]*kernel[3]+tmp[12]*kernel[4]+tmp[13]*kernel[5]+tmp[111]*kernel[
```

Verilator

```
verilator -Wall –cc our.v –exe sim_main.cpp
make -j -C obj_dir -f Vour.mk Vour
```

CPU Work Load

The CPU will prepare the input image, control the occupancy of FPGA module, and read back the processed data, when FPGA finishes his task.

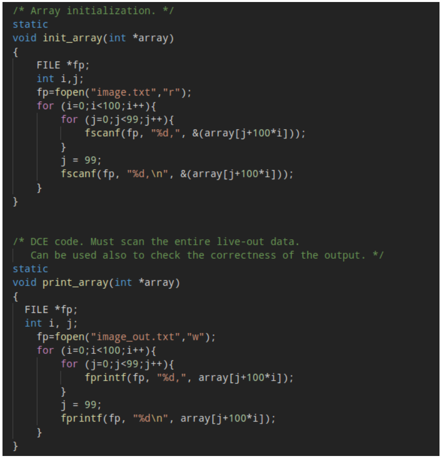


```
  int indata[100*100];

int main(int argc, char** argv)
{
    p0 =(unsigned long long *) new((unsigned long long *)0xc0000000) unsigned long long[10];
    //printf("111111");
    p0[1] = (unsigned long long)indata;//ReadBase
    p0[2] = (unsigned long long)indata;//WriteBase
    p0[0] = 81;
    p0[8] = 1;
    p0[3] = 9;//CurrentThreadID
    p0[4] = 100*100+2;//Memory Range
    p0[5] = 4;//MemorySize
    p0[7] = 0;//Terminat
    /* Initialize array(s). */
    p0[6]=0;p0[6]=0;
    init_array (indata);
    //Strating occupy FPGA
    p0[6]=1;
    //Wait for release
    while(p0[6]);
    //Save for data
    print_array(indata);

    return 0;
}
```

I/O Trick



```c
/* Array initialization. */
static
void init_array(int *array)
{
    FILE *fp;
    int i,j;
    fp=fopen("image.txt","r");
    for (i=0;i<100;i++){
        for (j=0;j<99;j++){
            fscanf(fp, "%d,", &(array[j+100*i]));
        }
        j = 99;
        fscanf(fp, "%d,\n", &(array[j+100*i]));
    }
}


/* DCE code. Must scan the entire live-out data.
   Can be used also to check the correctness of the output. */
static
void print_array(int *array)
{
  FILE *fp;
  int i, j;
    fp=fopen("image_out.txt","w");
    for (i=0;i<100;i++){
        for (j=0;j<99;j++){
            fprintf(fp, "%d,", array[j+100*i]);
        }
        j = 99;
        fprintf(fp, "%d\n", array[j+100*i]);
    }
}
```

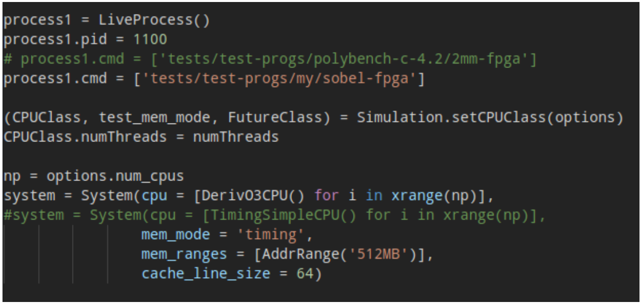
Syscall-Emulation Simulation

Before, put everything together, we should compile the CPU work load to executable binary file.

Compile CPU work load

g++ sobel-fpga.c -o sobel-fpga –static

Then in the top level Python script, we should assign the work load binary file to the gem5 CPUs.



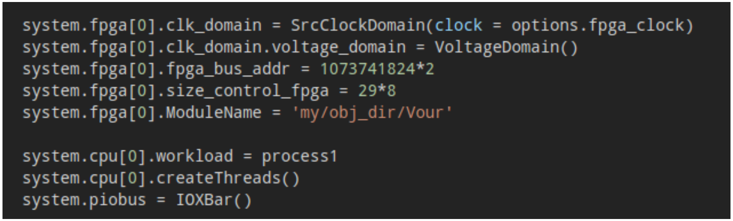
```
process1 = LiveProcess()
process1.pid = 1100
# process1.cmd = ['tests/test-progs/polybench-c-4.2/2mm-fpga']
process1.cmd = ['tests/test-progs/my/sobel-fpga']

(CPUClass, test_mem_mode, FutureClass) = Simulation.setCPUClass(options)
CPUClass.numThreads = numThreads

np = options.num_cpus
system = System(cpu = [DerivO3CPU() for i in xrange(np)],
#system = System(cpu = [TimingSimpleCPU() for i in xrange(np)],
                mem_mode = 'timing',
                mem_ranges = [AddrRange('512MB')],
                cache_line_size = 64)
```

Syscall-Emulation Simulation

And also the executable FPGA file should be assigned to the system.



```
system.fpga[0].clk_domain = SrcClockDomain(clock = options.fpga_clock)
system.fpga[0].clk_domain.voltage_domain = VoltageDomain()
system.fpga[0].fpga_bus_addr = 1073741824*2
system.fpga[0].size_control_fpga = 29*8
system.fpga[0].ModuleName = 'my/obj_dir/Vour'

system.cpu[0].workload = process1
system.cpu[0].createThreads()
system.piobus = IOXBar()
```

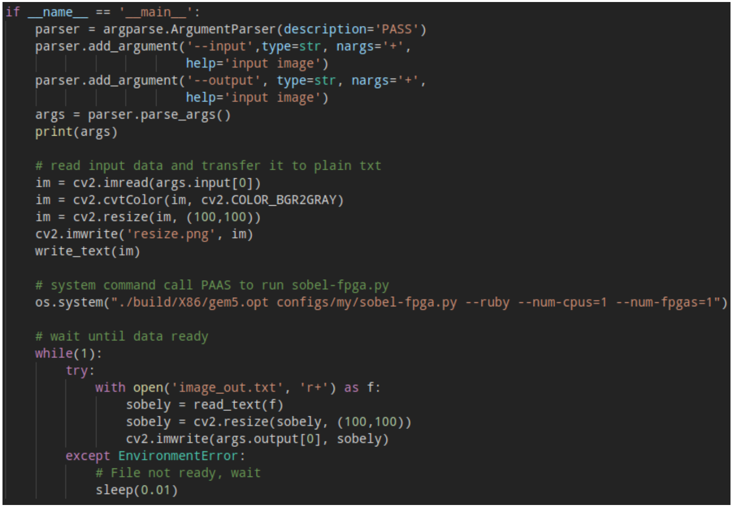
Finally, we should call the PAAS platform to perform the syscall-emulation simulation.

Call modified gem5

./build/X86/gem5.opt configs/my/sobel-fpga.py
–ruby –num-cpus=1 –num-fpgas=1

Run the Entire System

Python code which perform the dirty trick and call the simulation.



```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='PASS')
    parser.add_argument('--input',type=str, nargs='+',
                        help='input image')
    parser.add_argument('--output', type=str, nargs='+',
                        help='input image')
    args = parser.parse_args()
    print(args)

    # read input data and transfer it to plain txt
    im = cv2.imread(args.input[0])
    im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    im = cv2.resize(im, (100,100))
    cv2.imwrite('resize.png', im)
    write_text(im)

    # system command call PAAS to run sobel-fpga.py
    os.system("./build/X86/gem5.opt configs/my/sobel-fpga.py --ruby --num-cpus=1 --num-fpgas=1")

    # wait until data ready
    while(1):
        try:
            with open('image_out.txt', 'r+') as f:
                sobely = read_text(f)
                sobely = cv2.resize(sobely, (100,100))
                cv2.imwrite(args.output[0], sobely)
        except EnvironmentError:
            # File not ready, wait
            sleep(0.01)
```

Simulation Result

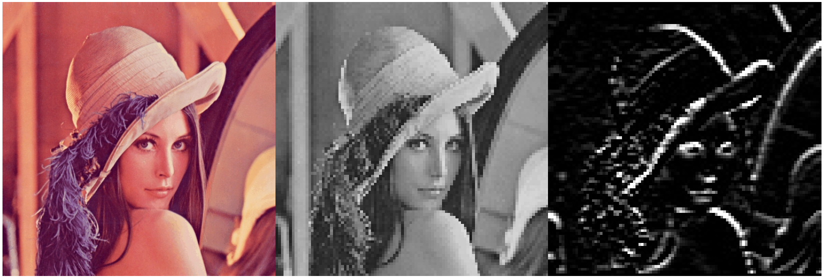


Figure: Left: Origin image; Middle: Preprocessed image sent to simulator. Right: Output image.

Conclusion

Contribution

- Completed a whole simulation pipeline on CPU-FPGA heterogeneous system, which amplifies the strength of both architecture and achieved correct result.

Gain/Comprehension

- Get familiar with GEM5 simulation and work flow of its cooperation between hardware accelerators;
- Although my project seems easy and contains no detailed analysis on memory settings, etc, the progress of put everything together and make the whole system run is not trivial, which is exact meaning of engineering.

Code,
https://github.com/wchengad/course_2019_spring/tree/master/ELEC5140

References

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu,Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator.ACMSIGARCH Computer Architecture News, 39(2):17, 2011.

Tingyuan Liang, Liang Feng, Sharad Sinha, and Wei Zhang. Paas: A system level simulator for hetero-geneous computing architectures. In2017 27th International Conference on Field Programmable Logicand Applications (FPL), pages 18. IEEE, 2017.

Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. Design space exploration offpga-based deep convolutional neural networks. In2016 21st Asia and South Pacific Design AutomationConference (ASP-DAC), pages 575580. IEEE, 2016.

Jason Power, Joel Hestness, Marc S Orr, Mark D Hill, and David A Wood. gem5-gpu: A heterogeneouscpu-gpu simulator.IEEE Computer Architecture Letters, 14(1):3436, 2015.

The End