Implement Darknet on FPGA

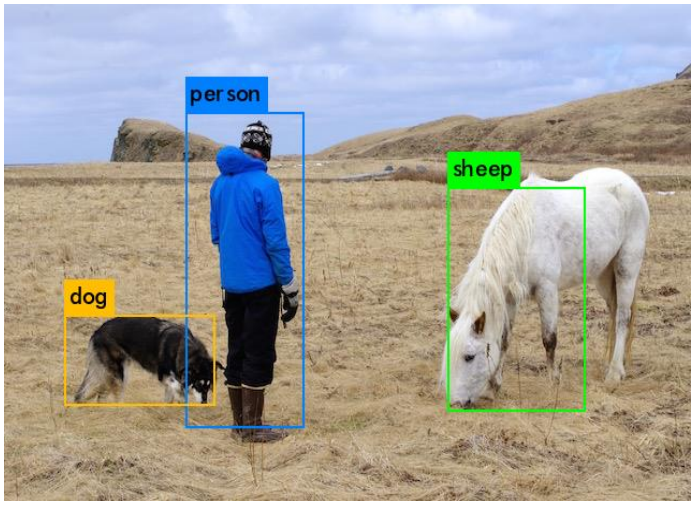
1 Darknet

Darknet is a current open source deep learning framework, it is lightweight and is easy to be deployed. Darknet is very convenient to realize object detection, classification and etc. Like Caffe, Darknet is based on C, and it is easy to modify its source code.

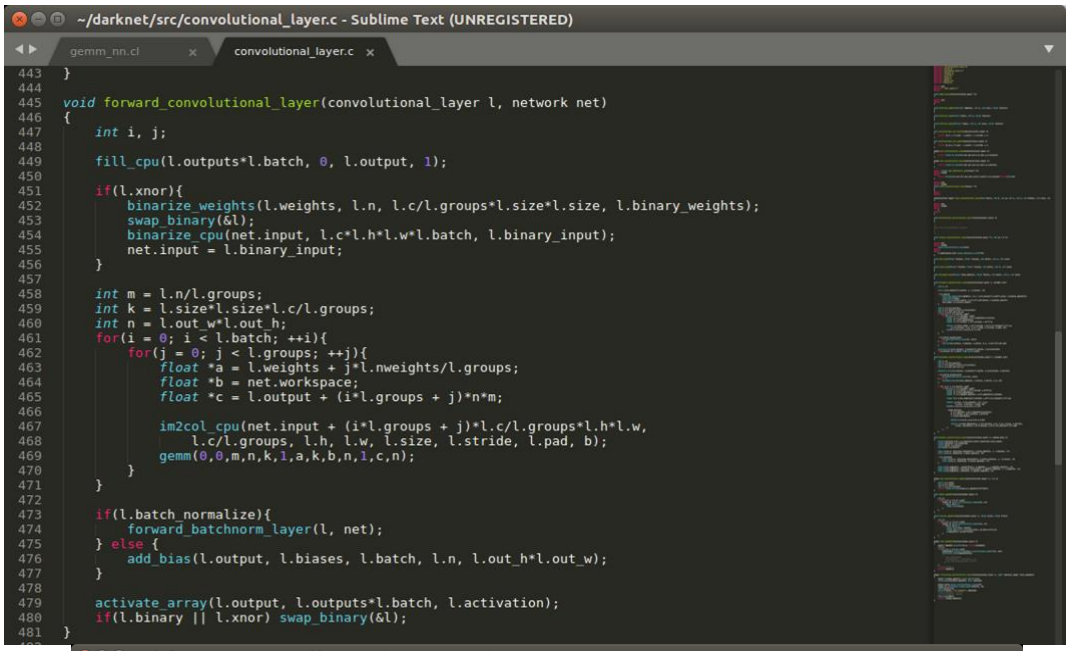
2 Using Darknet to realize object detection

Darknet supports CPU and GPU, change the Makefile according to your condition. Here is a example of running darknet on CPU. It's the yolo algorithms and the input image is person.jpg. As can be seen, it will output the confidence of each object and the total detection time: sheep(60%),person(73%),dog(53%),time(2.88s). The object is surrounded by the bounding box.

```
zlmo@zlmo: ~/darknet
zlmo@zlmo:~$ cd darknet/
zlmo@zlmo:~/darknet$ ./darknet detector test cfg/voc.data cfg/yolov2-tiny-voc.cf
g ./tiny-yolo-voc.weights ./data/person.jpg
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16  0.150 BFL
OPs
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32  0.399 BFL
OPs
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64  0.399 BFL
OPs
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128  0.399 BFL
OPs
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256  0.399 BFL
OPs
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFL
OPs
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFL
OPs
   13 conv   1024  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x1024  3.190 BFL
OPs
   14 conv    125  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 125  0.043 BFL
OPs
   15 detection
mask_scale: Using default '1.000000'
Loading weights from ./tiny-yolo-voc.weights...Done!
./data/person.jpg: Predicted in 2.883106 seconds.
sheep: 60%
person: 73%
dog: 53%
zlmo@zlmo:~/darknet$
```



3 Source Code Analysis



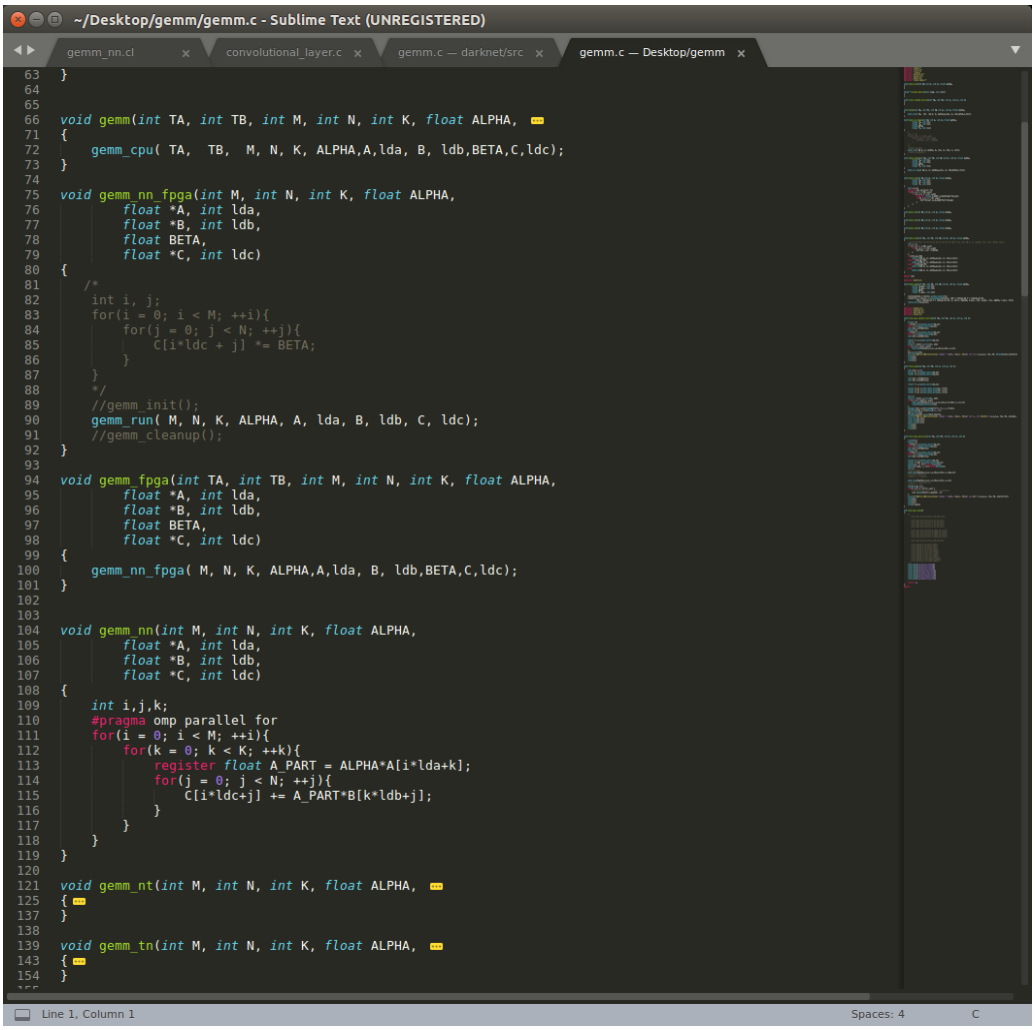
In CNNs, there are Conv-layers, pooling layers and fully connected layers, and the Conv-layers are computational intensive. Here we mainly modify the source code of Conv-layers. In Darknet, the cource code of Conv-layers is in convolutional_layer.c. And here we talk about running CNN on fpga, it mainly means we run the inference phase on fpga. In inference phase, Darknet mainly uses this function: forward_convolutional_layer(), it completes the matrix multiplication which is input matrix A * input matrix B=output matrix C. And the gemm() function

in forward_convolutional_layer() realizes this multiplication. The parameters in the gemm() function specifically refer to TA: whether matrix A is transposed, TB: whether matrix B is transposed, M: row numbers of matrix A, N: column numbers of matrix B, K: column numbers of matrix B, *A: input Matrix A, *B: input matrix B, float *C: output matrix C, lda: columns of A, ldb: columns of B, ldc: columns of C. gemm() function realizes A(M,K)*B(K, N) = C (M, N) operation. It can be seen that the gemm() function further calls gemm_cpu(), and gemm_cpu() function further calls gemm_nn(), gemm_tn(), gemm_nt() and gemm_tt(), gemm_nn() is used by default in darknet. The gemm_nn() function is called every time the convolution layer is running, so this is the core function of the convolutional layer matrix operation. When transforming to fpga, modify the gemm_nn() function to the kernel.cl function, and let fpga be responsible for this part of the operation.

4 Modify Source Code

As shown in the figure below, I first replace the original gemm() function with gemm_fpga() in the initial convolutional_layer.c, and realize gemm_fpga() in gemm.c.

```
505
506             if (l.size == 1) {
507                 b = im;
508             } else {
509                 im2col_cpu(im, l.c/l.groups, l.h, l.w, l.size, l.stride, l.pad, b);
510             }
511             //gemm(0,0,m,n,k,1,a,k,b,n,1,c,n);
512             gemm_fpga(0,0,m,n,k,1,a,k,b,n,1,c,n);
513         }
514     }
515
```



~/Desktop/gemm/gemm.c - Sublime Text (UNREGISTERED)

gemm_nn.cl | convolutional_layer.c | gemm.c — darknet/src | gemm.c — Desktop/gemm

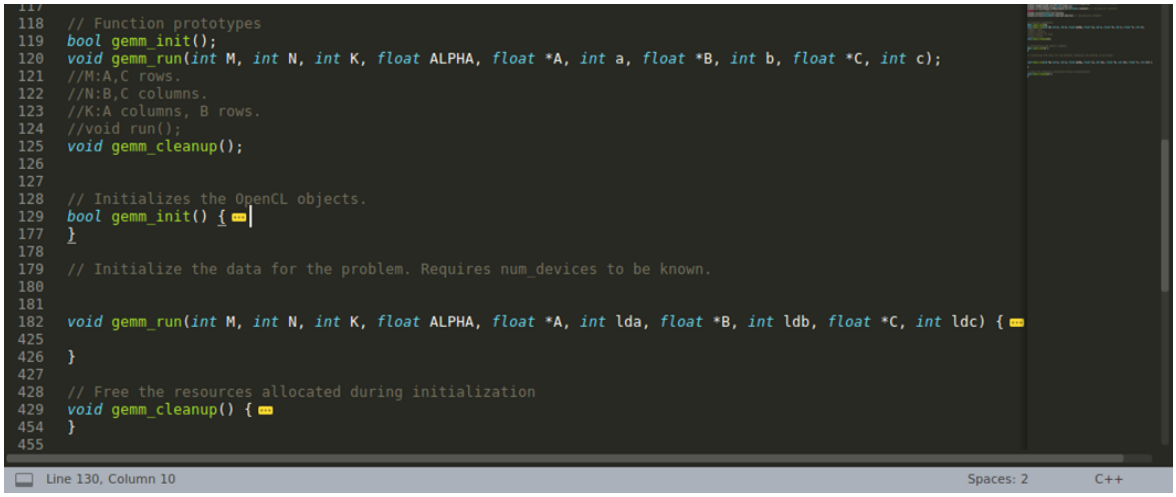
```
63  }
64
65
66  void gemm(int TA, int TB, int M, int N, int K, float ALPHA,  ...
71  {
72      gemm_cpu( TA,  TB,  M, N, K, ALPHA,A,lda, B, ldb,BETA,C,ldc);
73  }
74
75  void gemm_nn_fpga(int M, int N, int K, float ALPHA,
76          float *A, int lda,
77          float *B, int ldb,
78          float BETA,
79          float *C, int ldc)
80  {
81      /*
82      int i, j;
83      for(i = 0; i < M; ++i){
84          for(j = 0; j < N; ++j){
85              C[i*ldc + j] *= BETA;
86          }
87      }
88      */
89      //gemm_init();
90      gemm_run( M, N, K, ALPHA, A, lda, B, ldb, C, ldc);
91      //gemm_cleanup();
92  }
93
94  void gemm_fpga(int TA, int TB, int M, int N, int K, float ALPHA,
95          float *A, int lda,
96          float *B, int ldb,
97          float BETA,
98          float *C, int ldc)
99  {
100     gemm_nn_fpga( M, N, K, ALPHA,A,lda, B, ldb,BETA,C,ldc);
101 }
102
103
104 void gemm_nn(int M, int N, int K, float ALPHA,
105         float *A, int lda,
106         float *B, int ldb,
107         float *C, int ldc)
108 {
109     int i,j,k;
110     #pragma omp parallel for
111     for(i = 0; i < M; ++i){
112         for(k = 0; k < K; ++k){
113             register float A_PART = ALPHA*A[i*lda+k];
114             for(j = 0; j < N; ++j){
115                 C[i*ldc+j] += A_PART*B[k*ldb+j];
116             }
117         }
118     }
119 }
120
121 void gemm_nt(int M, int N, int K, float ALPHA,  ...
125 { ...
137 }
138
139 void gemm_tn(int M, int N, int K, float ALPHA,  ...
143 { ...
154 }
```

Line 1, Column 1 Spaces: 4 C

Adding two interfaces gemm_fpga() and gemm_nn_fpga() to gemm.c as shown below. All parameters in gemm_fpga() are the same as gemm(), then gemm_fpga() calls gemm_nn_fpga(), while gemm_nn_fpga() further calls gemm_run(), and gemm_run() uses fpga to repace the matrix operation taken by gemm_nn() before.

4.1 gemm_run()

The gemm_run() is responsible for realising the matrix operation of A*B=C on fpga. This function is implemented with opencl. I modify the official matrix multiplication example to realize gemm_run(). As shown in the figure below, I modify the original host program to get gemm_fpga.cpp, which mainly contains three interfaces, gemm_init(), gemm_run() and gemm_cleanup(). Gemm_init() is responsible for some initialization of opencl, gemm_cleanup() is responsible for the release of resources, and gemm_run() is mainly responsible for matrix operations.

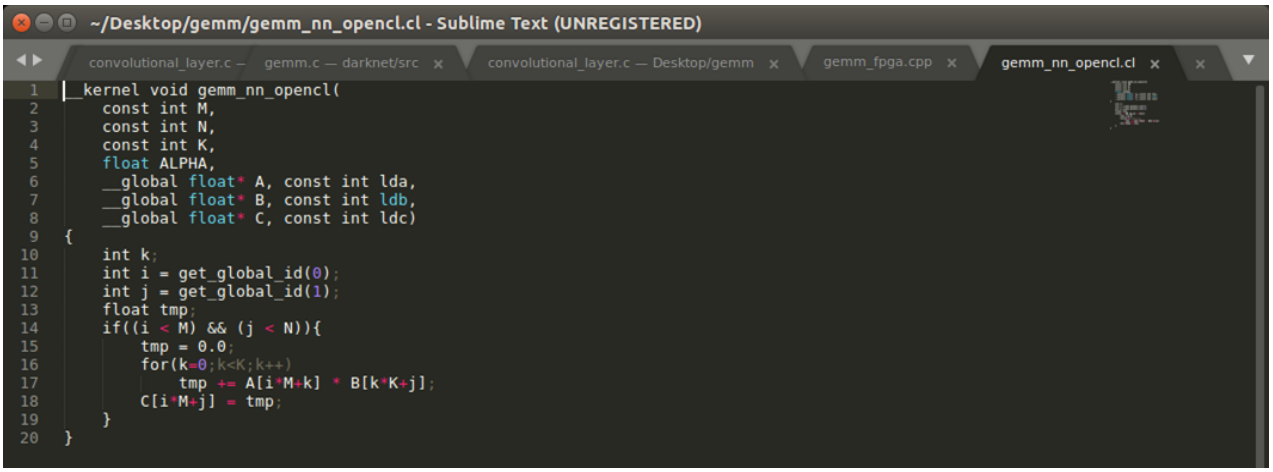


The actual implementation of these three functions is much complicated. You can refer to the gemm_fpga.cpp which I already completed. Here only some of the code in the gemm_run() function is explained. As shown in the figure, this section passes parameters of gemm_run() in host file to the kernel file, and the parameters should in order.

```
status = clSetKernelArg(kernel[i], argi++, sizeof(A_height), &A_height);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(B_width), &B_width);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(A_width), &A_width);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(ALPHA), &ALPHA);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_a_buf[i]);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(lda), &lda);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_b_buf[i]);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(ldb), &ldb);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &output_buf[i]);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(ldc), &ldc);
checkError(status, "Failed to set argument %d", argi - 1);
```

4.2 Kernel function

Here, the implementation of the kernel function is directly converted from the book. This implementation is relatively simple, according to the latter test, this kernel function may has a

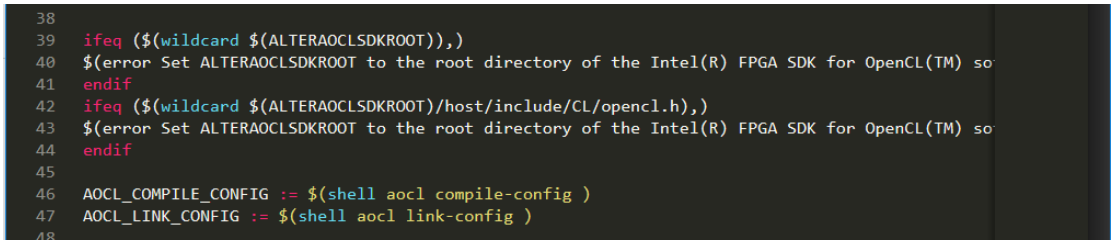


little problem. After the kernel function is written, I compile it and generates the **aocx** file. **This compiling step will take about 2 hours.**

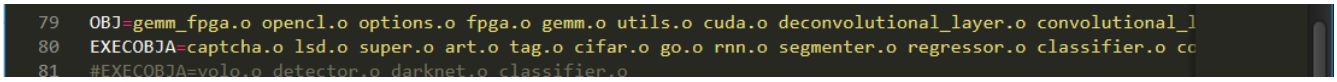
5 Compile

Place the written gemm_fpga.cpp and its corresponding header file gemm_fpga.h in the darknet/src directory, and place the kernel aocx file in the darknet/ directory. Then we need modify the Makefile, as shown below.

First check the openCl path:



Then add gemm_fpga.o on OBJ, this aims to compile the previously written gemm_fpga.cpp to an executable file.



Finally, adding openCL path at compile time, and add a new line:

$(OBJDIR)%.o: %.cpp $(DEPS)

$(CC) $(COMMON) $(CPPFLAGS) $(CXXFLAGS) $(AOCL_COMPILE_CONFIG) -c $< -o $@ $(AOCL_LINK_CONFIG) $(foreach L,$(LIBS),-l$L),

this line is responsible for compiling gemm_fpga.cpp into gemm_fpga.o, because the original Makefile can only compile the .c file, and the .cpp file can be compiled after the addition. In addition, the compiler in Makefile should be replaced by g++ before gcc, because opencl is based on g++, darknet is based on gcc.

I 'm not going to show the other little changes in Darknet, please refer to the project on Github(such as add head file to darknet.h and etc.).

6 Test

Test results are are shown below.

After running the test command, fpga can be called normally and the fpga information is

printed, this means darknet has loaded fpga.

However, it produced -nan error in the eighth layer, it might be that the operation value is too large, resulting in the final failure and only output execution time. This should be a problem in the kernel function, and because the entire process is still rough, the execution time is still slow. After further writing a better kernel function,it should be able to output the target information normally and achieve target detection.