Translating CUDA to OpenCL for Hardware Generation using Neural Machine Translation

Yonghae Kim
School of Computer Science
Georgia Institute of Technology
Atlanta, GA USA
yonghae@gatech.edu

Hyesoon Kim
School of Computer Science
Georgia Institute of Technology
Atlanta, GA USA
hyesoon@cc.gatech.edu

Abstract— **Hardware generation from high-level languages like C/C++ has been one of the dreams of software and hardware engineers for decades. Several high-level synthesis (HLS) or domain-specific languages (DSLs) have been developed to reduce the gap between high-level languages and hardware descriptive languages. However, each language tends to target some specific applications or there is a big learning curve in learning DSLs, which ends up having many program languages and tool chains.**

To address these challenges, we propose the use of a source-to-source translation to pick and choose which framework to use so that the hardware designer chooses the best target HLS/DSL that can be synthesized to the best performing hardware. In this work, we present source-to-source translation between CUDA to OpenCL using NMT, which we call PLNMT. The contribution of our work is that it develops techniques to generate training inputs. To generate a training dataset, we extract CUDA API usages from CUDA examples and write corresponding OpenCL API usages. With a pair of API usages acquired, we construct API usage trees that helps users find unseen usages from new samples and easily add them to a training input. Our initial results show that we can translate many applications from benchmarks such as CUDA SDK, polybench-gpu, and Rodinia. Furthermore, we show that translated kernel code from CUDA applications can be run in the OpenCL FPGA framework, which implies a new direction of HLS.

Index Terms—**High-level Synthesis, Program Translator, Neural Machine Translation**

I. Introduction

Due to increasing design complexity and the need of higher productivity, generating hardware design from high-level languages has been motivated by software and hardware engineers for decades. To reduce the gap between high-level languages and hardware descriptive languages, several high-level synthesis (HLS), including OpenCL or domain-specific languages (DSLs) like Chisel, Bluespec, and SystemC, have been developed. However, a number of program languages and tool chains still have a big learning curve, and each infrastructure requires a huge amount of man power to develop and maintain it.

Source-to-source translation is one way of addressing the challenges, commonly used to translate one legacy code to target code in another language, such as Fortran to C. However, developing a new translator usually demands expertise in a compiler and language, and requires enormous engineering efforts, which defeats the purpose of reducing the efforts of infrastructure maintenance.

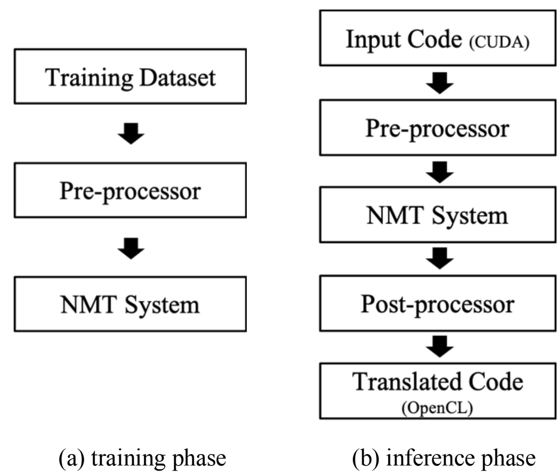

Fig. 1. Overview of workflows

To reduce the challenge for the development and maintenance of a source code translator, we propose using natural language translation. Several recent works have exploited neural machine translation (NMT) techniques to translate program. A recent work [1] proposes a novel tree-to-tree neural network and demonstrates higher accuracy for program translation, but it has limited set of variables and restricts the vocabulary size. The other work [2] utilizes NMT techniques to deal with cross-architecture code similarity comparison. However, it does not handle high-level language translation.

In this work, we present source-to-source translation between CUDA to OpenCL using NMT, which we call PLNMT. We chose these languages for two reasons: (1) CUDA and OpenCL share many similarities, so they provide a good platform to develop the techniques of PLNMT. Based on the knowledge/techniques from this translation, we will expand our work to other program languages. (2) OpenCL is one of the HLS frameworks.

While the existing works translate CUDA to OpenCL at an AST level [3] or uses wrapper functions to generate binary executable [4], we perform actual source-to-source translation using

NMT. The summary of our contributions is as follows. We develop techniques to generate training inputs. Both CUDA and OpenCL require host and kernel code, and most APIs have a one-to-one correspondence with each other. We first extract CUDA API usages from CUDA samples and write corresponding OpenCL API sages. Then, we construct API usage trees which make it easier to find unseen usages from new samples and add them to a training dataset. Lastly, we train the NMT model that learns the API mapping and structural similarity between CUDA and OpenCL. Fig. 1 shows the overview of training and inference workflows. To handle the differences between natural languages and program languages, we use a pre/post-processor. The pre-processor performs lexical analysis to tokenize code and identify types of tokens. Then, it renames tokens to abstract symbols to enable arbitrary variable names to be translated. In an inference phase, the NMT system takes as input pre-processed CUDA code and generates OpenCL code which retains abstract symbols. Finally, with the symbol table acquired from pre-processing, the post-processor replaces abstract symbols with initial names and restructure tokens following syntactic rules. Fig. 2 presents a translation example.

```
void mm2Cuda(float* A, float* B, float* C)
{
 float *A_gpu;
 ...
 cudaMalloc((void **)&A_gpu, sizeof(float) * NI * NK);
 cudaMemcpy(A_gpu, A, sizeof(float) * NI * NK,
                              cudaMemcpyHostToDevice);
 ...
```

(a) CUDA host code

```
__global__ void mm2_kernel1(float *A, float *B, float *C)
{
 int j = blockIdx.x * blockDim.x + threadIdx.x;
 ...
  for (k = 0; k < NK; k++)
  {
    C[i * NJ + j] += A[i * NK + k] * B[k * NJ + j];
 ...
```

(b) CUDA kernel code

```
_line_not_to_translate
_line_not_to_translate
_tp0 _op0 _id0 ;
...
cudaMalloc ( ( _tp0 _op0 ) _op1 _id0 , sizeof ( _tp1 ) _op2
                                        _id1 _op2 _id2 ) ;
cudaMemcpy ( _id0 , _id1 , sizeof ( _tp0 ) _op0 _id2
          _op0 _id3 , cudaMemcpyHostToDevice ) ;
...
```

(c) Pre-processed CUDA host code

```
__global__ _tp0 _id0 ( _tp1 _op0 _id1 , _tp1 _op0 _id2 ,
                                        _tp1 _op0 _id3 )
_line_not_to_translate
_tp0 _id0 _op0 blockIdx.x _op1 blockDim.x _op2 threadIdx.x ;
...
_line_not_to_translate
_line_not_to_translate
_line_not_to_translate
...
```

(d) Pre-processed CUDA kernel code

```
_line_not_to_translate
_line_not_to_translate
cl_mem _id0 ;
...
_id0 = clCreateBuffer ( context , CL_MEM_READ_WRITE ,
          sizeof ( _tp1 ) _op2 _id1 _op2 _id2 , NULL , NULL ) ;
clEnqueueWriteBuffer ( command_queue , _id0 , CL_TRUE , 0 ,
  sizeof ( _tp0 ) _op0 _id2 _op0 _id3 , _id1 , 0 , NULL , NULL ) ;
...
```

(e) Host code translated by NMT system

```
__kernel _tp0 _id0 ( __global _tp1 _op0 _id1 , __global _tp1
                              _op0 _id2 , __global _tp1 _op0 _id3 )
_line_not_to_translate
_tp0 _id0 _op0 get_group_id ( 0 ) _op1 get_group_id ( 0 )
                                      _op2 get_local_id ( 0 ) ;
...
_line_not_to_translate
_line_not_to_translate
_line_not_to_translate
...
```

(f) Kernel code translated by NMT system

```
void mm2Cuda ( float * A , float * B , float * C )
{
 cl_mem A_gpu ;
 ...
 A_gpu=clCreateBuffer(context, CL_MEM_READ_WRITE,
                              sizeof(float)*NI*NK, NULL, NULL);
 clEnqueueWriteBuffer(command_queue, A_gpu, CL_TRUE, 0,
                         sizeof(float)*NI*NK, A, 0, NULL, NULL);
 ...
```

(g) Post-processed OpenCL host code

```
__kernel void mm2_kernel1(__global float*A, __global float* B,
                                       __global float* C)
{
 int j = get_group_id(0)*get_group_id(0)+get_local_id(0);
 ...
 for ( k = 0 ; k < NK ; k ++ )
 {
  C[i * NJ + j] += A[i * NK + k] * B[k * NJ + j] ;
 ...
```

(h) Post-processed OpenCL kernel code

Fig. 2. Translation example of 2-D matrix multiplication code

REFERENCES

[1] X. Chen, C. Liu, D. X. Song, "Tree-to-tree Neural Networks for Program Translation," NeurIPS, 2018.

[2] F. Zuo, X. Li, Z. Zhang, P. Young, L. Luo, Q. Zeng, "Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs," CoRR, 2018, abs/1808.04706.

[3] G. Martinez, M. Gardner, and W. chun Feng, "CU2CL: A CUDA-to-OpenCL Translator for Multi- and Many-Core Architectures," Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS), 2011, pages 300–307.

[4] J. Kim, T. T. Dao, J. Jung, J. Joo, and J. Lee, "Bridging OpenCL and CUDA: a comparative analysis and translation," SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, 2015, pp. 1-12.