T2SP Programming Guide

Hongbo Rong

2022-01-10

Contents

1 Overview

T2SP generates accelerators for dense tensor computes. Usually, such a compute is described as one or a few math equations. We can rewrite these equations to be recursive, the so-called Uniform Recursive Equations or UREs. With UREs, we can accurately control input and output data to flow in a pipeline fashion during the compute.

We can write basic UREs without tiling the compute domain. Then we tile the compute domain for data locality and reuse. It is straightforward to translate basic UREs into the final UREs after tiling.

Note that UREs are simply C statements. So after adding some helper code necessary for testing, we can compile them in any C compiler, print out the results, and check if the UREs are correct vs. the original math equations.

The compute domain is iterated with a loop nest. If the compute is to run on a GPU, we need decide which loops are block loops, and inside a block, which loops are thread loops: An iteration of the thread loops will be turned into a thread. If the compute is to run on an FPGA, however, we always use a single thread for the entire compute, and thus there is no block and thread loops.

Inside a thread, we will build a systolic array in a space-time transform. So we need decide which loops are space loops.

Finally, we add an I/O network for the systolic array. The I/O network is composed of multiple memory levels. The input/output data are loaded/unloaded through the I/O network into/out of the systolic array. Inside the systolic array, the data flow according to the UREs.

Now for the original compute, we have a complete T2SP program, or more accurately, a specification, since such a program only specifies what to implement (e.g. space-time transform and I/O network), but leave the actual implementation to the T2SP compiler. In other words, the program controls the compiler to generate the expected accelerator.

For an FPGA, the T2SP compiler generates OpenCL device code and a C interface, compile and run these generated code by invoking the downstream FPGA tools. The programmer can use the emulator tool to verify correctness of the generated code, and use the synthesis tool to produce a bit-stream. The programmer can write CPU host code and call the device code through the C interface, just like calling any normal C function on a CPU. That call would offload the compute to the emulator or an actual FPGA to run.

For a GPU, the T2SP compiler generates CM device code, and invoke the CM compiler to build a binary. The programmer need write CPU host code that offloads the device code to a GPU.

A beginner might encounter several hurdles in using T2SP: (1) how to write UREs? (2) how to tile loops? (3) how to decide block, thread, and space loops? and (4) how to design I/O? There are numerous valid answers for each of these questions. Fortunately, we do not have to be mathematicians, algorithm experts, or computer architects in order to effectively address the questions. Based on simple and intuitive rules, any programmer may master the programming quickly, and use the skills to accelerate real-world workloads.

We will describe the steps in more details below.

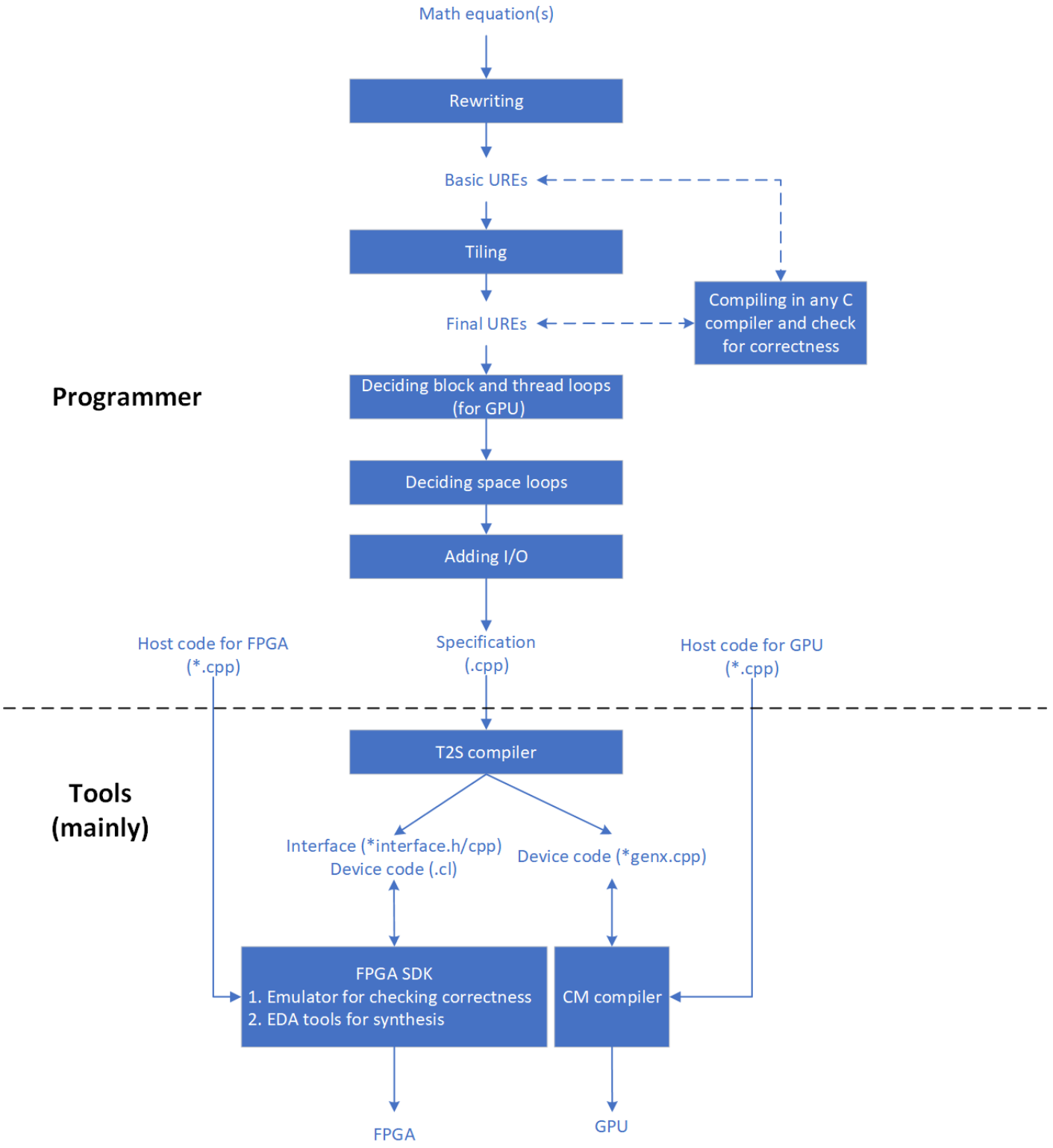

Figure 1: Overall-flow

2 Rewriting math equations into basic UREs

Based on the work of Gusev and Evans [1][2], we can translate math equations into UREs, following the simple steps below. This process is pretty much mechanical and intuitive, and only middle-school level of math knowledge is needed. So do not be scared by the math symbols.

Use auto-regressive filter for example:

Step	Equations		In
1: Iterative form	$y_i = \sum_{j=1}^{n} a_j * y_{i-j} + x_i$	$i = 1, ..., n$	
2: Recursive form	$y_i = y_i + a_j * y_{i-j}$	$i, j = 1, ..., n$	$y_i = x_i$
3: DSA	$y_i^j = y_i^{j-1} + a_j * y_{i-j}^n$	$i, j = 1, ..., n$	$y_i^0 = x_i$
4: Full index form	$y_i^j = y_i^{j-1} + a_j^0 * y_{i-j}^n$	$i, j = 1, ..., n$	$y_i^0 = x_i^0$
5: UREs	$A_i^j = A_{i-1}^j$ $Y_i^j = Y_{i-1}^{j-1}$ $y_i^j = y_i^{j-1} + A_i^j * Y_i^j$	$i, j = 1, ..., n$	$A_{i-1}^j = a_j^0$ $Y_{i-1}^{j-1} = y_i^n$ $y_i^0 = x_i^0$

2.1 Step 1: Iterative form

First, write down the math equations of the original problem. Usually, these equations are to iterate over a domain (like $i, j = 1, ..., n$) and accumulate values for some variables (like y).

2.2 Step 2: Recursive form

here it is

```
    // Or so
    /* or so */
    for (int i=1; i< 99; i++) {
        int j=10;
    // dkfjldjfl
    //lkdfjdsf
    }
```

And after

for1

for2

3 References

[1] Marjan Gusev and David J. Evans. Algorithm Transformations for the Data Broadcast Elimination Method. Parallel Algorithm Research Centre. Loughborough University of Technology. Loughborough, Leicestershire LE11 3TU. Internal Report 646 Computer Department.

[2] Marjan Gusev and David J. Evans. Elimination of the Computational Broadcast : An Application to the QR Decomposition Algorithm. Parallel Algorithm Research Centre. Loughborough University of Technology. Loughborough, Leicestershire LE11 3TU. Internal Report 676 Computer Department.