Notes on TyBEC, Waqar Nabi, Glasgow, Dec 2016.

[Syntax 2](#_Toc469055863)

[TODO 2](#_Toc469055864)

[Bugs 2](#_Toc469055865)

[Short-term 2](#_Toc469055866)

[Medium-term 2](#_Toc469055867)

[Kitchen Sink 4](#_Toc469055868)

# General Notes

# Syntax

See publish/project-report

# TODO

## Bugs

## Short-term

1. Thoroughly investigate implementation of arithmetic operations, integration of Flopoco.
   1. A much more protracted extension of this is to automate selection of optimal number representations (and corresponding arithmetic units). [Refer to work at Imperial college on this]
      1. *On-going work by Gregor on this*
2. Floats
3. Multi-cycle FUs (e.g. floats), and balancing FG-pipeline in view of them
4. Balancing CG-pipeline
5. Smart Buffering
   1. Dealing with stencils
   2. Dealing with stencils in between CG-pipeline PEs (e.g. loops in dyn)
6. Streams that feed CG-pipeline PEs down the line (not first one)
7. map on one path, reduce on another?
8. Dealing with multiple pipelines when integrating with AOCL (
   1. single opencl buffer in main memory, manages as an address-space in the FPGA
9. Remove redundant meta-information in tytra-ir, e.g. when declaring streams and memories. do I need all of that?
10. Macro definitions NDim1, 2, NLinear etc is crude!

## Medium-term

1. Auto-integration (or at least auto-wrapper generation) for integration with AOCL
2. A more thorough investigation of the use of instruction processors (that is along the 3rd dimension), including vector instruction processors, chained (pipelined) instruction processors, and mixed architecture (normal FPGA pipelines + instruction processors).
3. Resource Balancing on an FPGA (e.g. Registers, LUTs, DSPs, Mem, all used optimally, at almost equal percentage usage, i.e. we do tradeoffs whereever possible and increases utilization)... See w.r.t. 112.157
4. Consider pipeline EXTENSION rather than just replication, e.g. see 112.157....
   1. Though the same reference seems to indicate that the better option is multiple replicated pipelines side by side (NOT end to end), so can perhaps just refer to this as a possibility and move on?
5. Explore optimizations that reduce power, exp and div, at the cost of additional mults, add or div

# TIR Syntax, Rules, Limitations

1. Reductions: Can only be at the edge of kernels?
2. Specifying reductions: same operand on both sides of primitive instruction?
3. Will FAIL if there are multiple edges of DIFFERENT TYPES between two nodes (Search for this text in code to see where this will happen)
4. Will fail on multiple edges altogether? (see NOTE in 2.5 pass)
5. I am currently assuming that ALL my multi-latency primitive units are pipelined, that is, they can be fed on every cycle (so LFI = 1).
6. I require all reduction operations to be annotated with the reduction size. Otherwise, I need to propagate backwards until I get to a memory object, which breaks the reductive nature of my DFG analysis.
7. I have to test the framework for cases like this:
   1. Two (more) outputs, one is reduction, other is normal
8. Main can have strictly 1 kernel (the top level kernel)
9. All array sizes fed to the top-kernel as input have same size (so I can choose any one of them to estimate overall performance figures).

# Kitchen Sink

2017.02.21

* From LLVM-LRM: "The ‘alloca‘ instruction allocates memory on the stack frame of the currently executing function, to be automatically released when this function returns to its caller. "

;-- So we use the ALLOCA instruction as a way to create scratchpad/

;-- private memory, so that it remains compatible with the

;-- LLVM framework

;-- For now, we only allow CONSTANT SCALARS to be declared this way

;-- But LLVM-LRM has no such restriction

**Streams, 1D vs 2D**

One option is to do the following:

* ;-- We always use linearized memory-objects/arrays
* ;-- Any dimensioning is done by using the appropriate streams
* ;-- this allows us to "reshape" the memory object in-place

**BUT,** for now, I am going to continue using 1D streams as well, and expect the IR user to manage the dimensionality in a linear array

What I do have is this:

;-- new syntax for creating 2D counters synched to 1D streams

* My DFG creation works in two passes:
  + Pass1: Create abstract DFG (nodes and edges, but no DOT)
  + Local scheduler (infers Buffers, inserts them into abstract DFG – both nodes and edges)
  + Pass2: Create DOT DFG, including buffers…