A Parallel Programming Tool-flow for SDR Signal Processing using Heterogeneous Architectures

Lerato J. Mohapi
Supervised By: Dr. Simon Winberg

University of Cape Town
Department of Electrical Engineering
Software Defined Radio Group

HPSPSA 2014

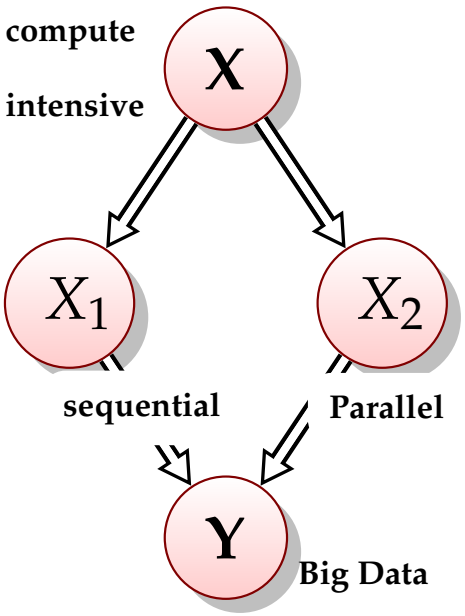
January 31, 2014

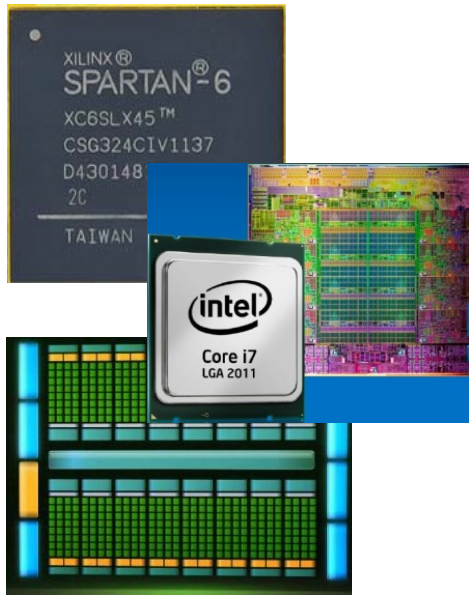
OUTLINE

INTRODUCTION

* A Domain Specific Language (DSL) for SDR: SDR-DSL

Algorithms | **Parallel Programming** | **Accelerators**



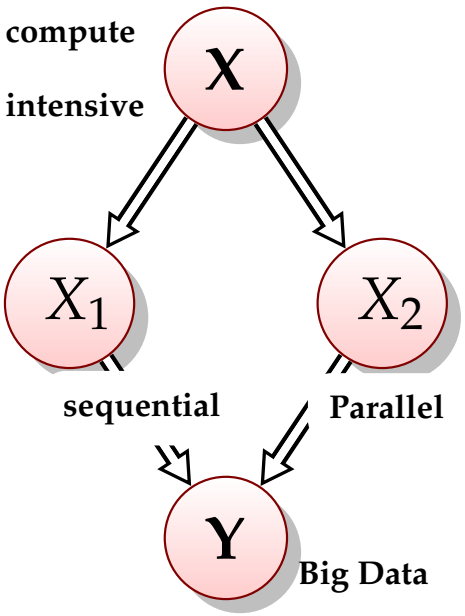


* Need to facilitate parallel programming of Heterogeneous Archs.

INTRODUCTION

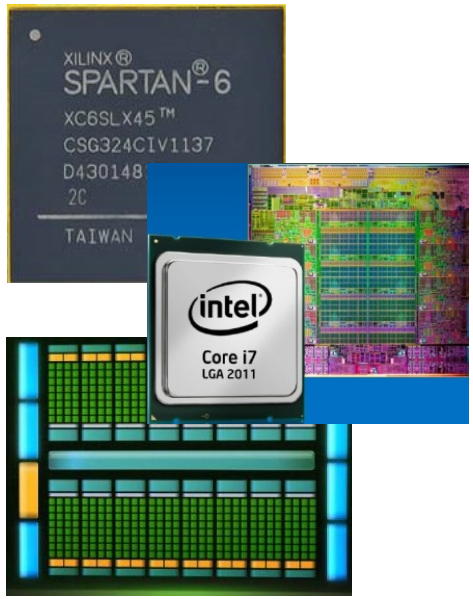
* DSL - Constructs tailored towards a Domain

Algorithms



DSL Compiler

Accelerators



* Expressiveness & Ease of use for heterogeneous archs.

INTRODUCTION: PROBLEM STRUCTURE

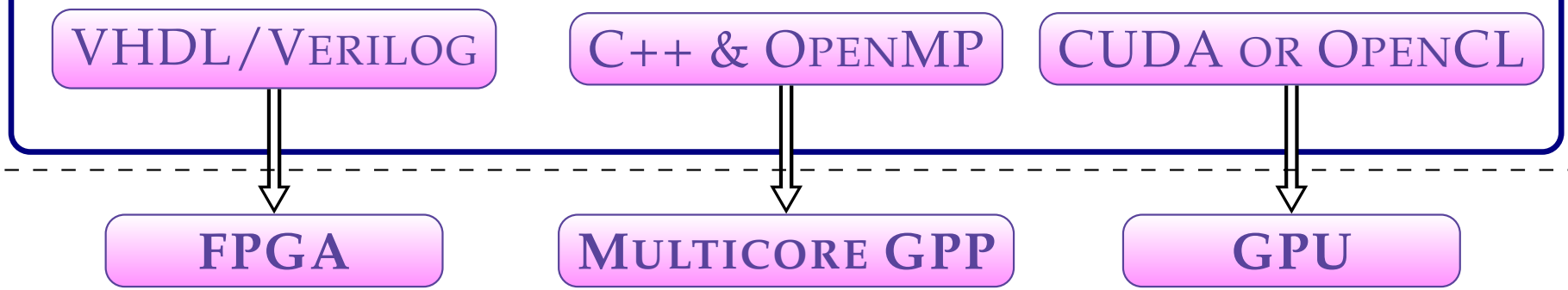
1. Software Defined Radio Domain

Domain Experts: How to Efficiently Parallelize DSP algos?

2. Parallel Programming

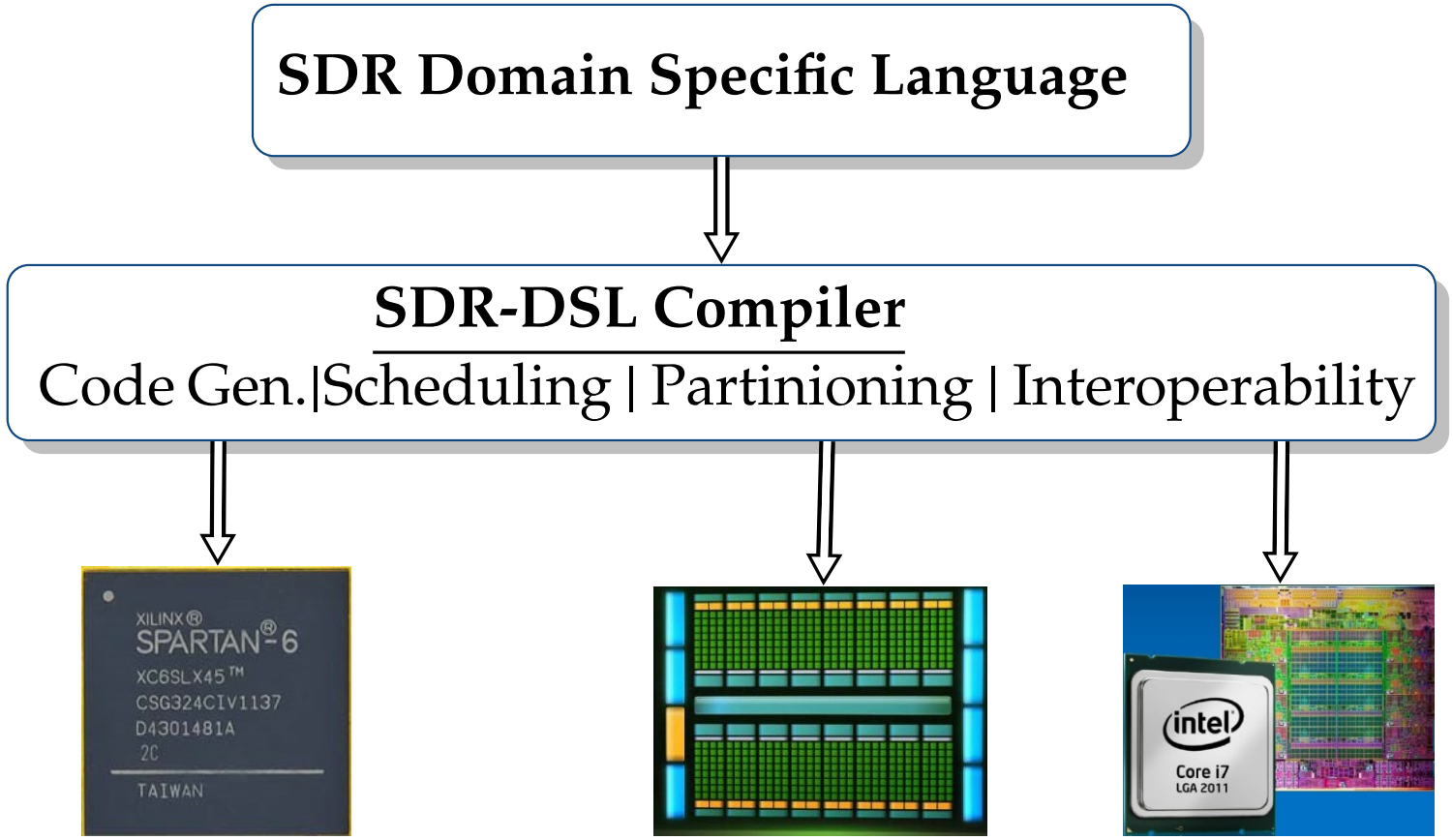
Algorithms and Accelerators => Verbose + Concurrent LPL

3. Heterogeneous Architectures



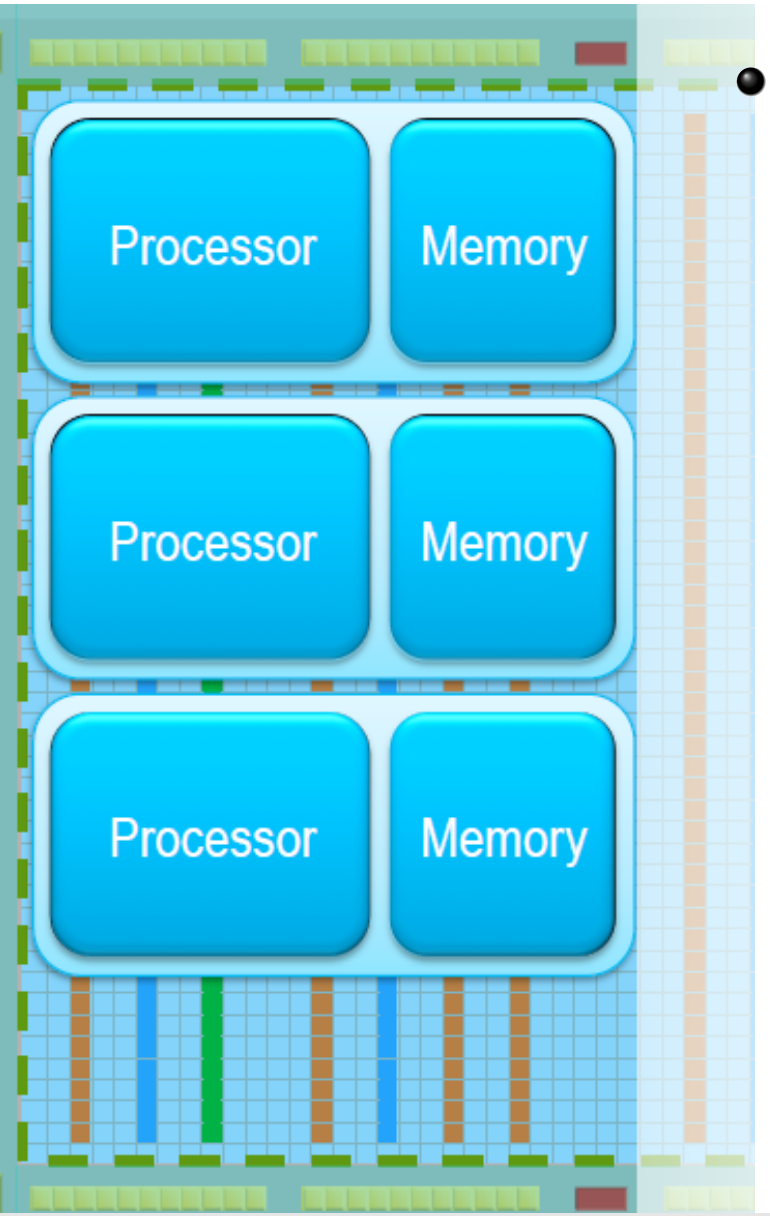
INTRODUCTION: OBJECTIVES AND SCOPE

- intuitive Parallel Programming tool-flow for HCA



  - FPGA <——————————> GPU <———> Multicore GPP
- Single source => Multiple heterogeneous targets
  - Auto Parallelize and Validate DSP algorithms

INTRODUCTION: SDR AND HETEROGENIETY



- Hardware/Application Level Heterogeniety
  - Each PE Optimized to perform different task
  - Each Processor exploit parallelism
  - Multiple chips, single platform
  - Sequencial and Parallel instruction sets

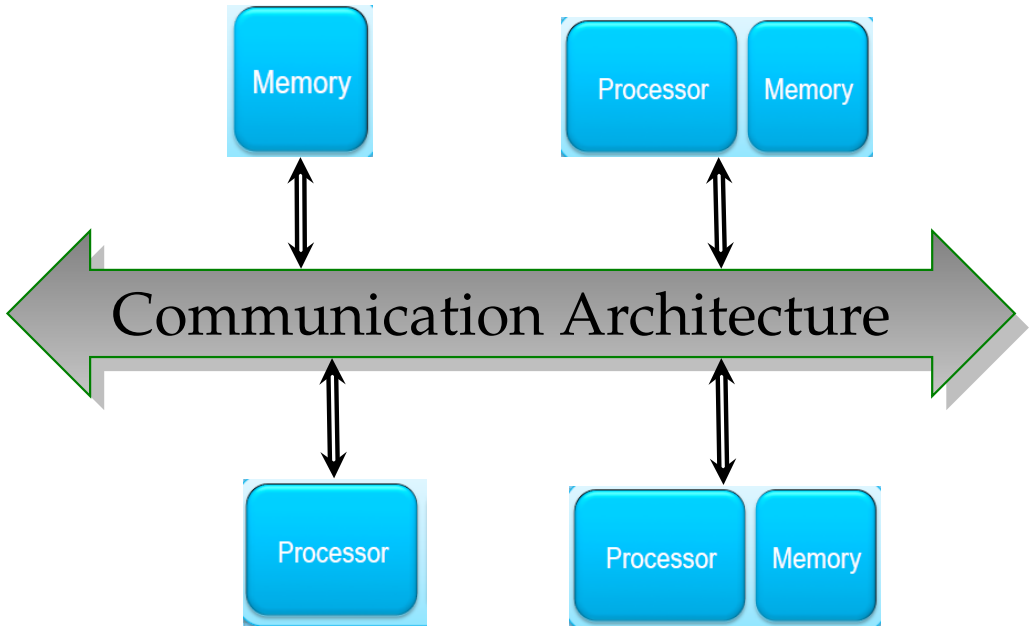
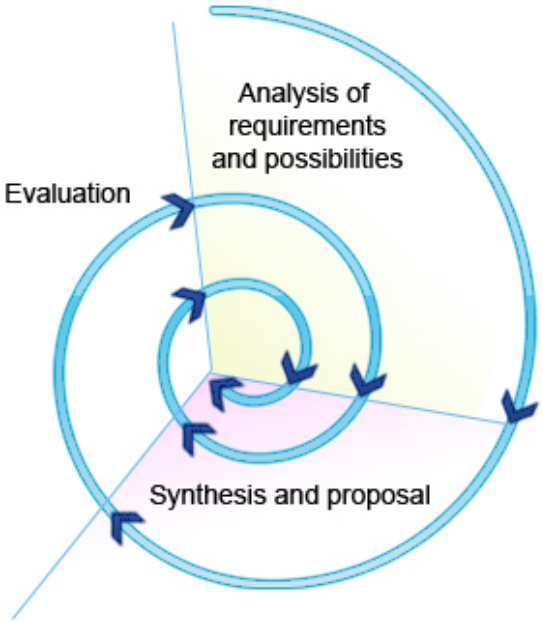


Figure: Overview of interoperability in HCA

METHODOLOGY OUTLINE

- SDR-DSL Development Steps:
  1. Refine Requirements
  2. Study SDR DSP algorithms
  3. Build SDR-DSL Compiler: Delite
  4. Validate
  5. Case study: Wideband Channelization



- Several individual DSP modules, thus spiral model of software development
- Permits iterative SDR-DSL development and cyclic progress assessment

METHODOLOGY: DELITE APPROACH

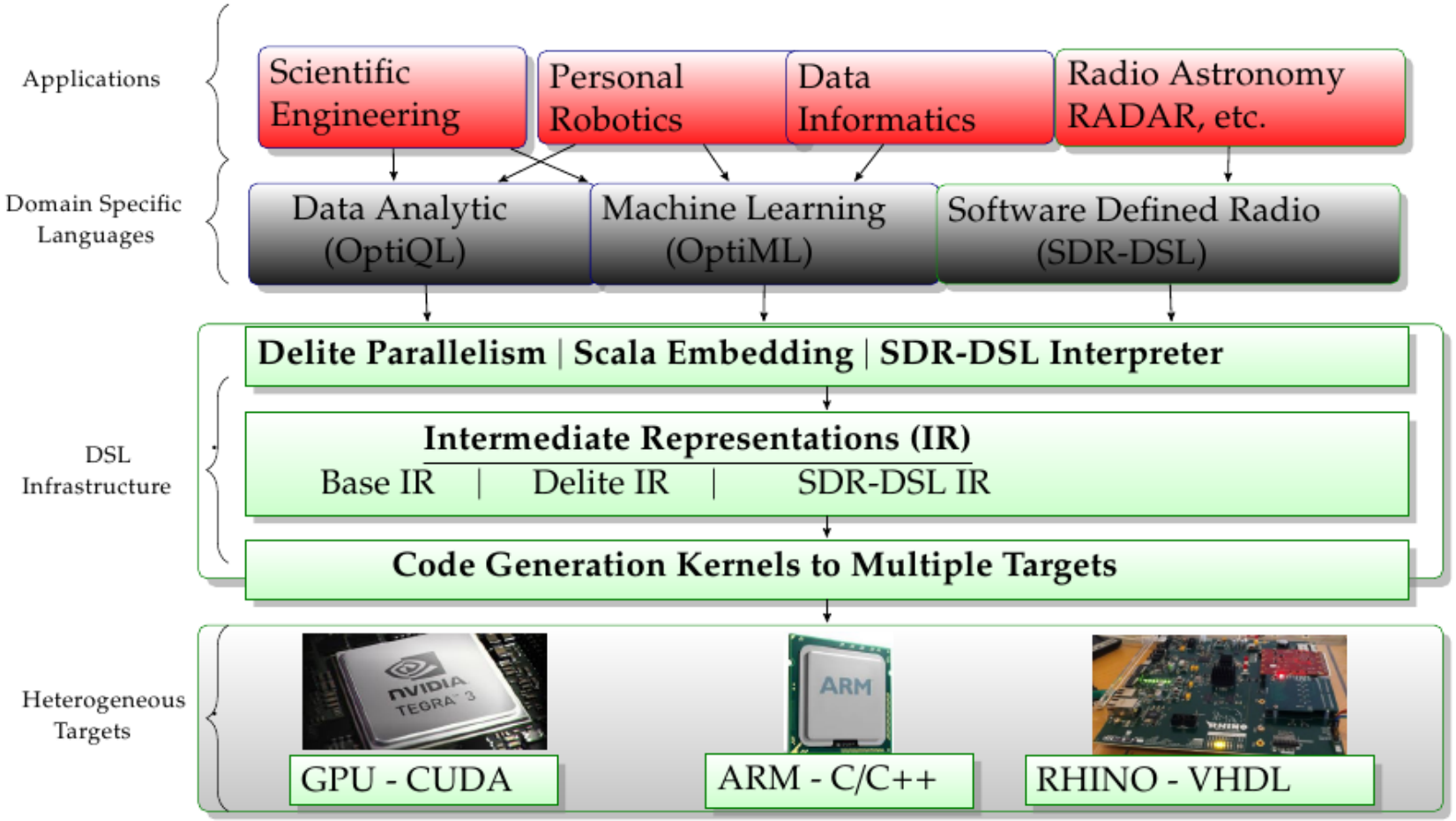
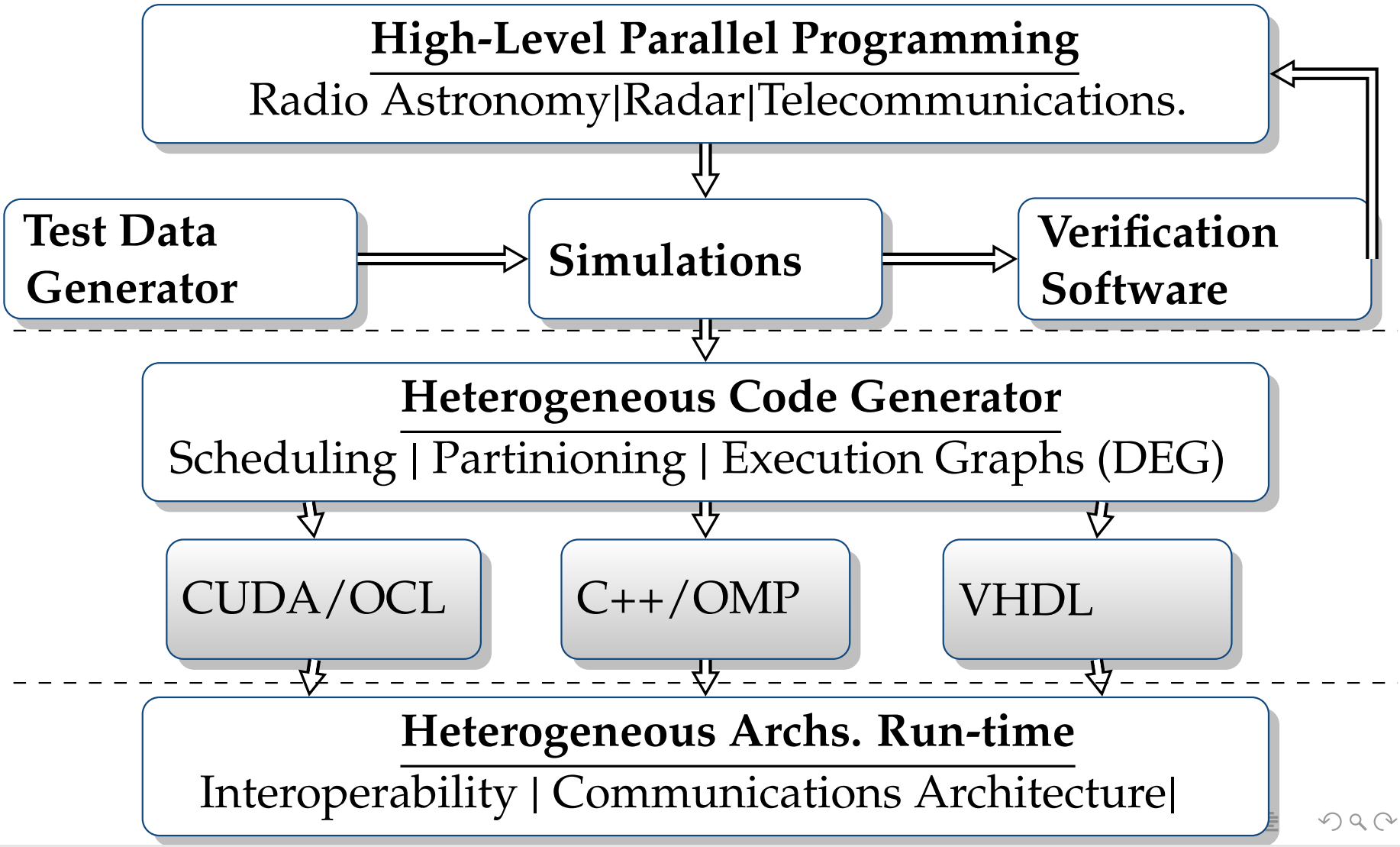


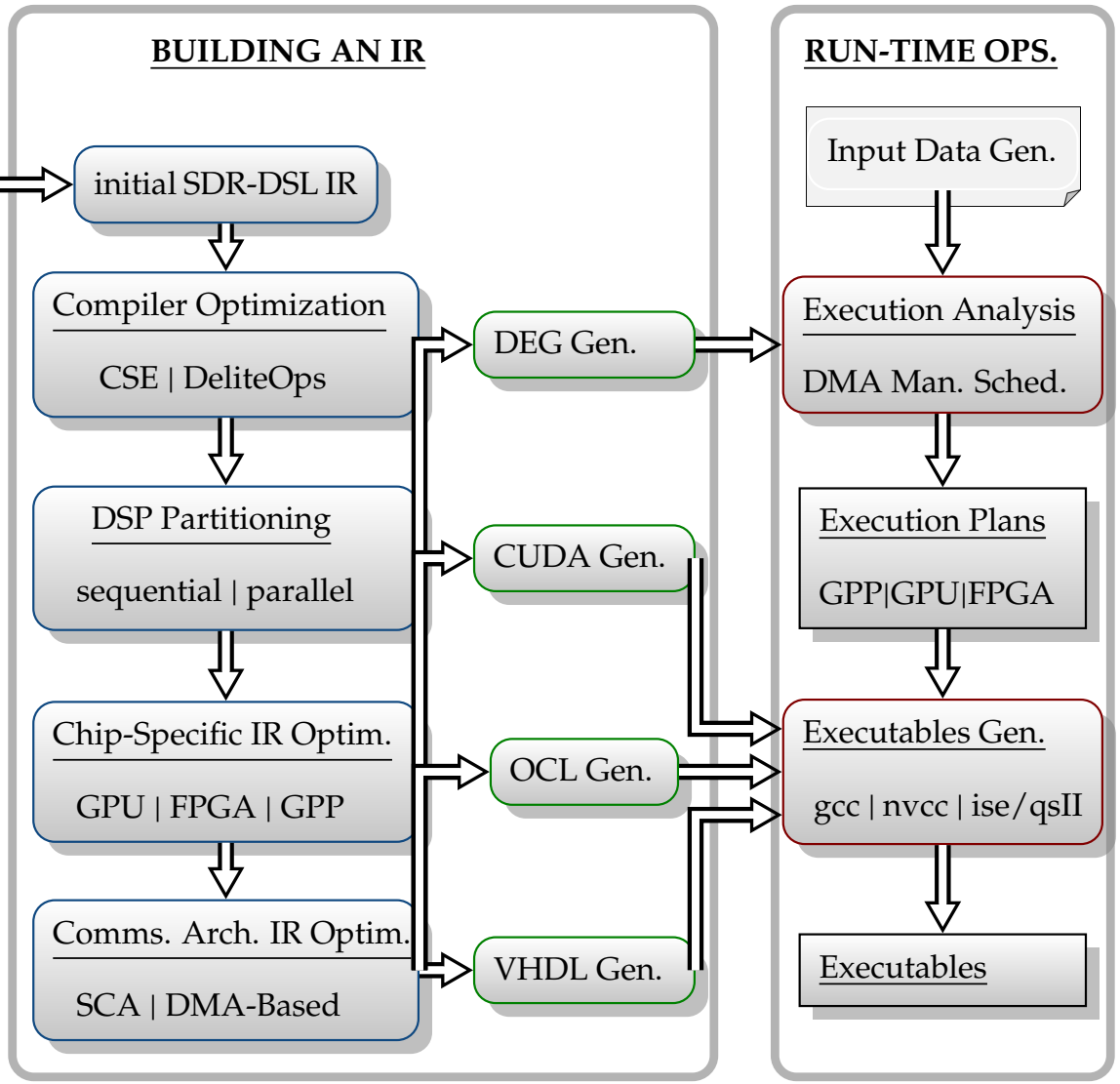
Figure: Delite framework - Common parallel execution patterns to ease DSL development - Runtime management (Sched. & Synch.)

SDR-DSL COMPILER



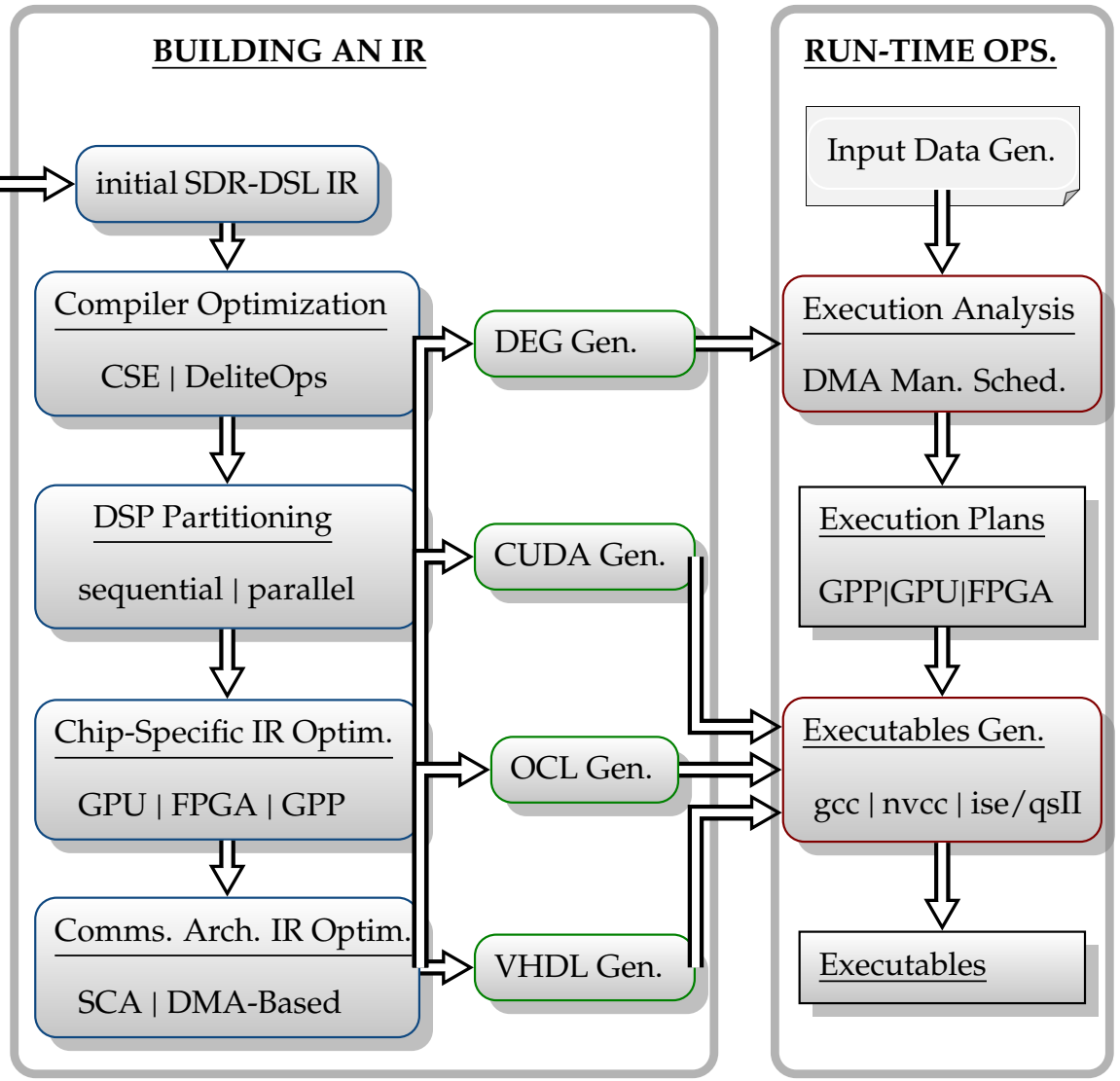
THE TOOL-FLOW

```
hilbert_fft [input_sig] gpu validate
    fft [input_sig] [var_fft]
    remove_lsb [var_fft] [var_low]
    return ifft [var_lower] [var_ifft]
end hilbert_fft
```



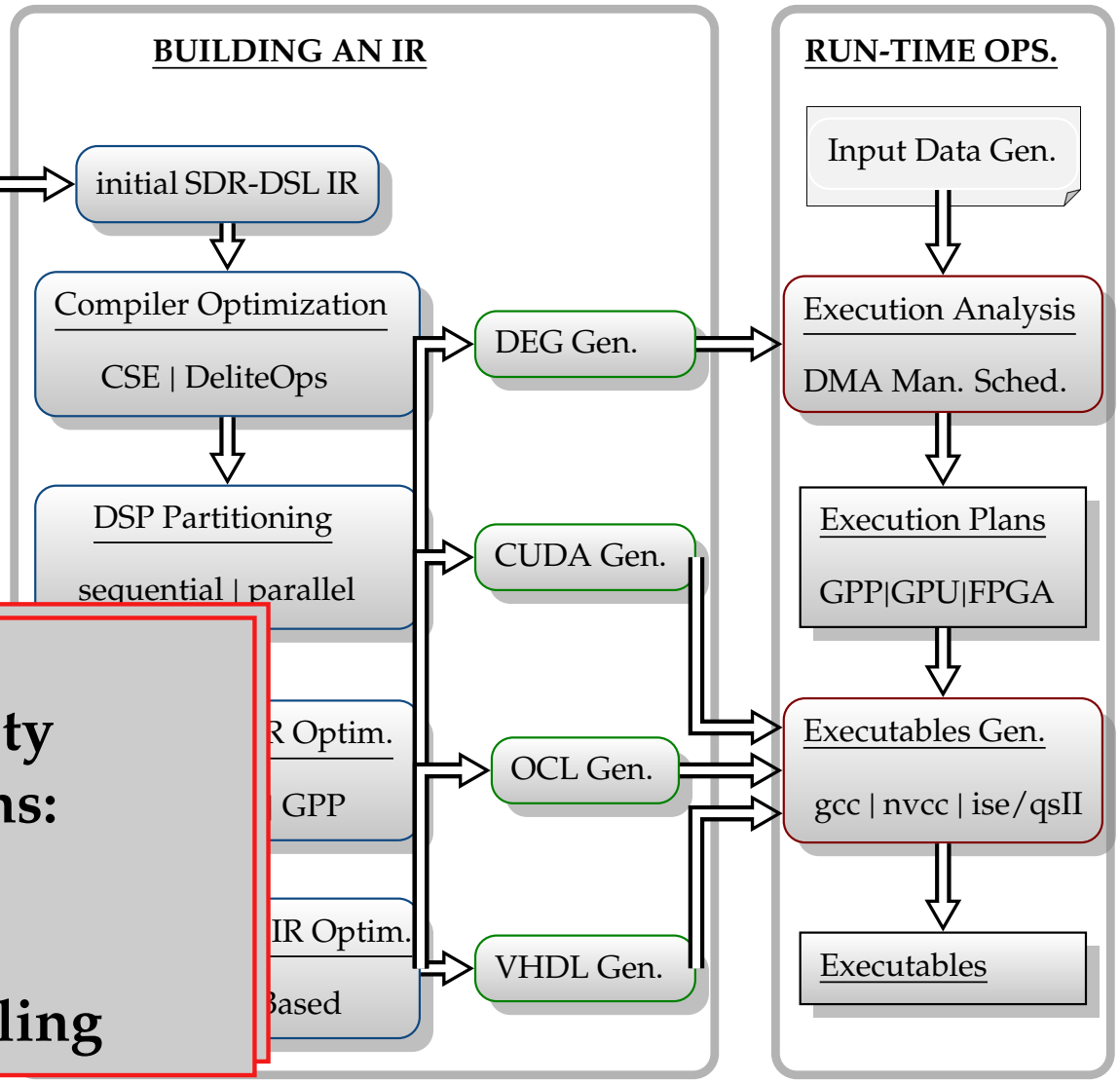
THE TOOL-FLOW

```
hilbert_fft [input_sig] gpu validate
    fft [input_sig] [var_fft]
    remove_lsb [var_fft] [var_low]
    return ifft [var_lower] [var_ifft]
end hilbert_fft
```



THE TOOL-FLOW

```
hilbert_fft [input_sig] gpu validate
    fft [input_sig] [var_fft]
    remove_lsb [var_fft] [var_low]
    return ifft [var_lower] [var_ifft]
end hilbert_fft
```



Focus on:
1. IR+ DSL Expressivity
2. Delite Optimizations:
- DeliteOpMap
- DeliteOpReduce
- Loop fusing and Peeling

RESULTS: TARGET CASE STUDY

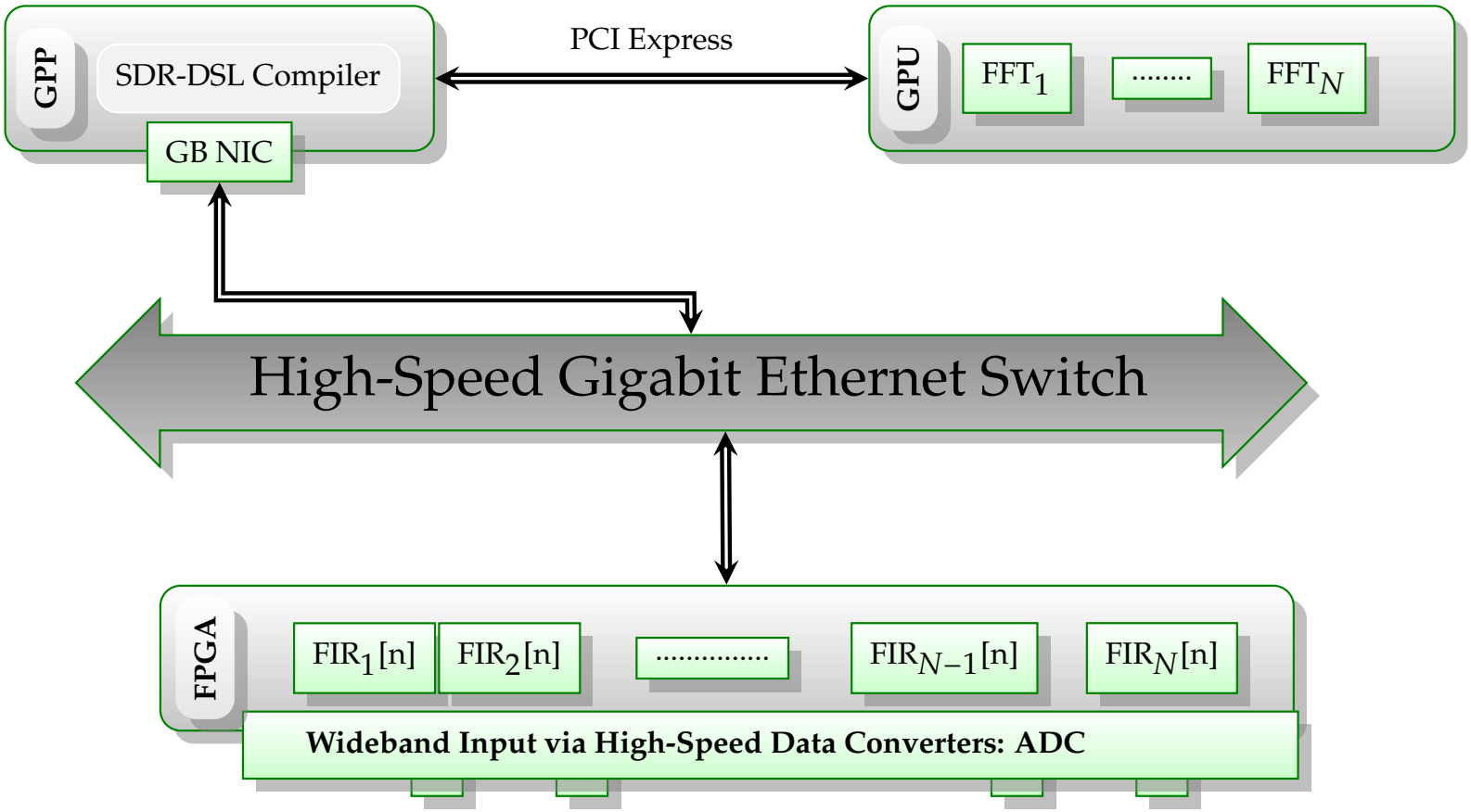


Figure: Real-time HCA for Wideband SDR channelization using FPGA-Based pipelined FIR filters,and GPU-based Spectral Analysis using FFTs

RESULTS: PROGRAMMING AND VALIDATION

Listing 1: *FIR filter implementation*

```
object HTransformerRunner extends OptiMLApplicationRunner
with FIRFilter  with KaizerWin with SineGenerator
trait FIRFilter extends OptiMLApplication
{
  def main() = {
    // Get Coefficients of Length = 63
    val FCoefficients =
      ConstFloatingPointFilterCoefficients()
    // Prep. Data Length and Tap Order
    val SLength = 993
    val FTapOrder = 62
    val K = SLength - FTapOrder + 1
    val FInput = GenerateSignal(SLength)
    val FOutput= DenseVector[Double](K,true)

    for(SHIndex <- 0 until K){
        var tmpvals = FInput(SHIndex::FTapOrder+SHIndex)
        FOutput(SHIndex) =
      FIRFilter(FCoefficients,tmpvals, FTapOrder)
        println(FOutput(SHIndex))
    }
  }
```

Expressivity

DSP Constructs embedded in Scala

Validation

Test Data + DeliteFileReader
- Need For Plotting

RESULTS: CODE GENERATION

Listing 2: *Generated CUDA Kernel*

```
// CUDA Kernel Generated by Delite
texture<float, 1,cudaReadModeElementType> KaizerCoeffs;

__global__ void fir(float *d_VAL_A,float* Output_Data, const
      unsigned int TapOrder,const unsigned int J){
    // Shared memory,the size is determined by the host application
    extern  __shared__  float OutputData[];
...
    // Perform FIR
    for (j=0; j<J; j++){
        for (i=0; i<TapOrder; ++i){
          sum += (float)((float) tex1D(KaizerCoeffs, (unsigned
      int)i ) * d_VAL_A[(unsigned int)(i+tid+TapOrder*j +
      bid*J*TapOrder)]);
        }
        __syncthreads();
        OutputData[tid+TapOrder*j] = (float)sum;
        __syncthreads();
        sum = (double)0.0;
    }
    ...
}
```

CUDA Kernels

The Kernel
- Implements FIR
- Declare Textures
- Declare Shared memory
- Synchronize Threads

CONCLUSIONS AND FURTHER WORK

- Work Done
  - Requirements refined, and SDR Domain Knowledge
  - SDR-DSL Syntax and Initial IR Implemented Using Delite (Borrowed Some OptiML Constructs)
  - Through its data types and execution semantics, SDR-DSL will reflects the desired abstraction hierarchy in SDR
- Work to be Done
  - Modify Code Generator
  - SDR-DSL Tool-flow Planned to include SDF and CSDF
  - SDR-DSL facilitates SDR signal processing on heterogeneous architectures

In Summary: SDR-DSL is expected to be hetegeneous PPTF that divides signal processing algorithms into sequential and parallel patterns while allowing designers to realize the performance of their applications by exploring different design parameters.

********** **Thank you!** **********

********** **Questions?** **********