

A Fast and Accurate Cost Model for FPGA Design Space Exploration in HPC Applications

*Developing an optimizing compiler for running
scientific code on FPGAs*

S Waqar Nabi and Wim Vanderbauwhede
School of Computing Science,
University of Glasgow

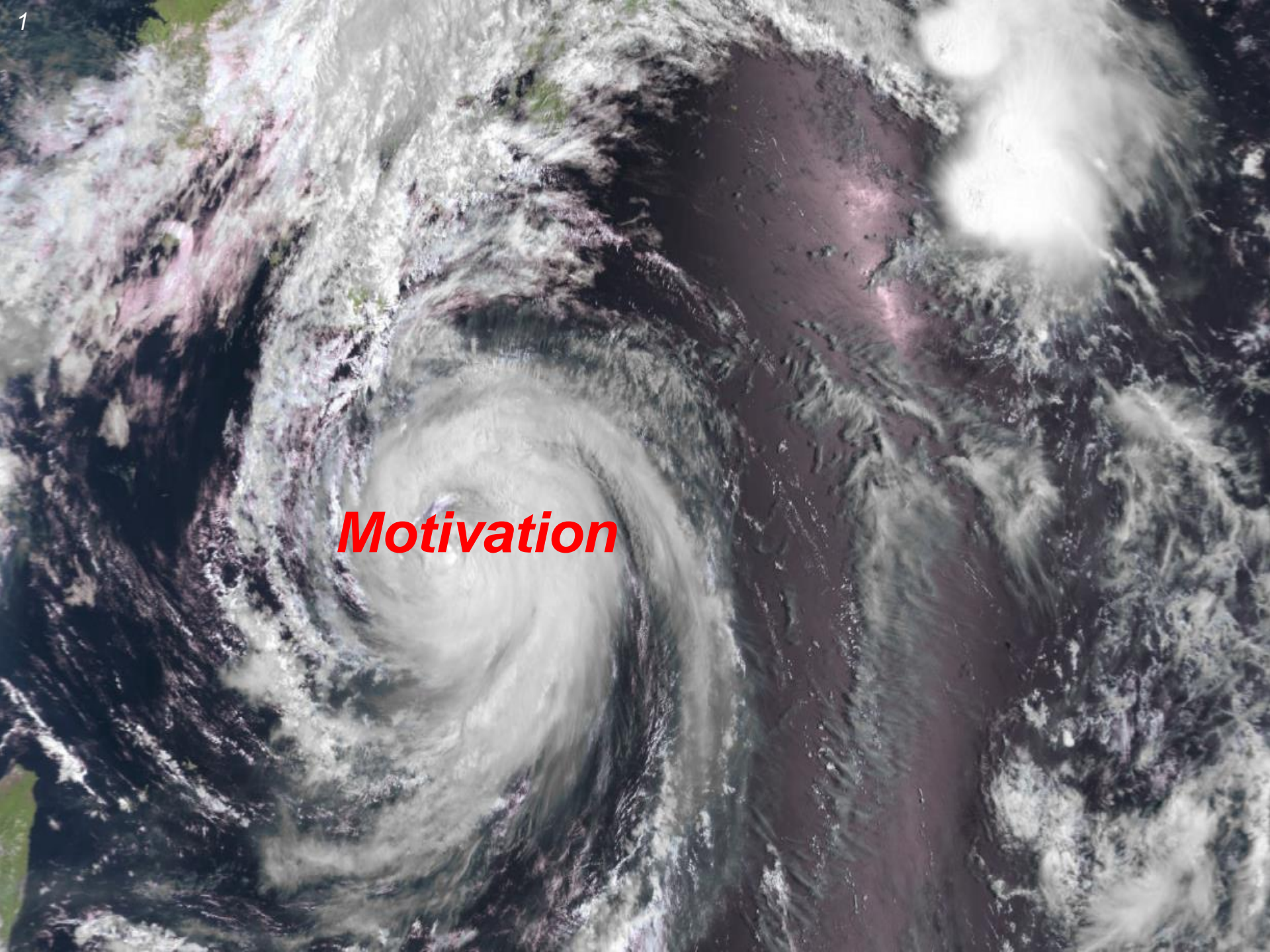


EPSRC

Engineering and Physical Sciences
Research Council

**Imperial College
London**





Motivation

- FPGAs *now* beginning to be used in mainstream computing
 - big-data and big-compute (HPC)
- *Can* provide better FLOPS/Watt for some types of applications
- Difficult to tune even with today's HLS tools like OpenCL, Maxeler, etc
 - E.g. 3 days vs 3 weeks
- **A still higher-level approach** is needed
 - automatic architectural exploration of the FPGA design-space
 - make FPGAs more accessible to scientists/HPC users
- The argument can be generalized to **heterogeneous** computing targets

Lots of promise, encouraging recent developments, but miles to go still...

- FPGAs *now* beginning to be used in mainstream computing
 - big-data and big-compute (HPC)
- *Can* provide better FLOPS/Watt for some types of applications

Today's high-level language is tomorrow's compiler target

- **A still higher-level approach** is needed
 - automatic architectural exploration of the FPGA design-space
 - make FPGAs more accessible to scientists/HPC users
- The argument can be generalized to **heterogeneous** computing targets

Lots of promise, encouraging recent developments, but miles to go still...

Why are we working on an FPGA cost-model?

- Our proposed *TyTra* compiler flow requires evaluation of multiple design-variants, in order to converge on the best one.
- It requires a **light-weight, reasonably accurate** cost-model.

A light-weight cost-model is the linchpin of our proposed FPGA optimizing compiler flow



OUTLINE OF TALK

Towards an optimizing compiler for running scientific code on FPGAs

1. The TyTra Framework

- Compiler flow
- The Intermediate Representation (IR)
- **Need for a cost-model**

2. Developing a Cost-Model

- Models of abstraction
- Cost-model for **resource-utilization** and **performance**

3. Observations

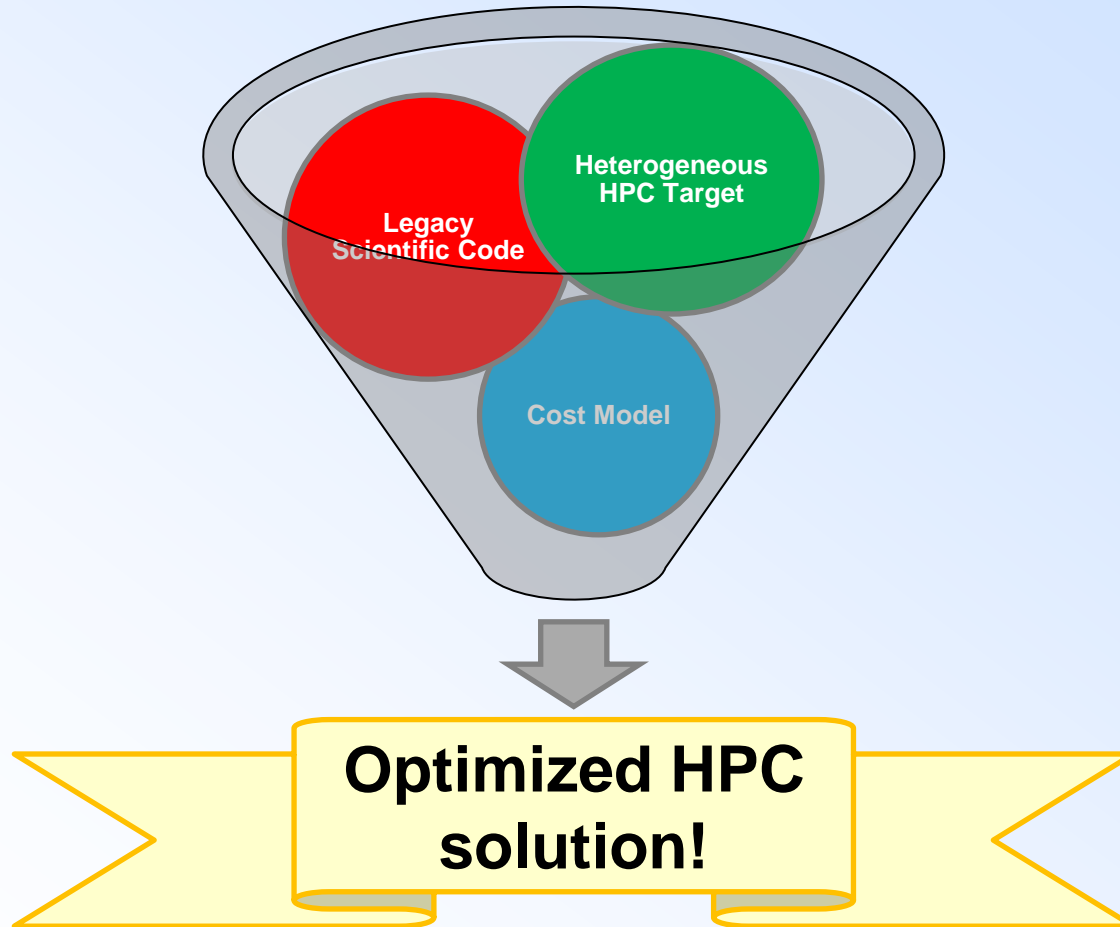
- Results: **design-space exploration potential**, **accuracy of cost-model**, **potential for improved performance**
- Limitations
- The way forward

An auto-tuning programming approach, for scientific computing, requiring a fresh approach to cost-modelling



THE *TYTRA* FRAMEWORK





The Cunning Plan...



I have a cunning plan.

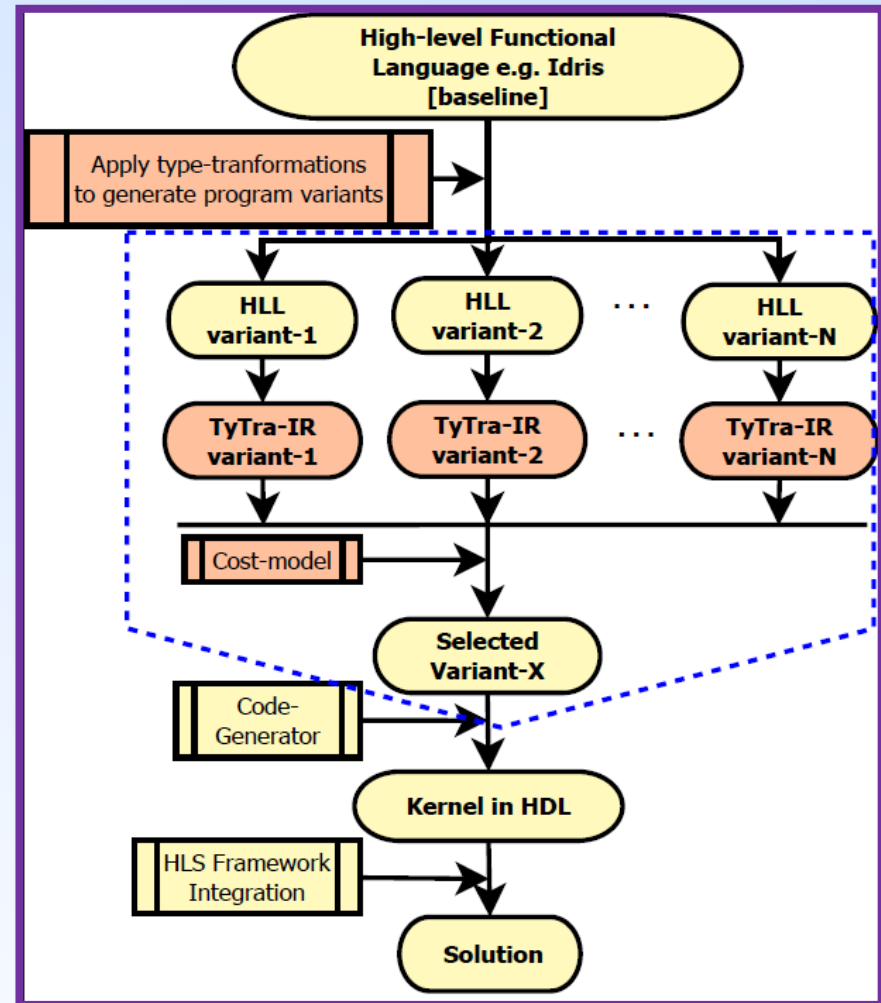
1. Use the **functional programming paradigm**
 - *type-transformations* to create design-variants
2. Have an **Intermediate Language** that can:
 - express the design-space
 - be costed directly and quickly
3. Create a **light-weight cost-model** that can estimate:
 - performance
 - resource-utilization

Exploit the “elegance” of functional abstraction to generate equivalent design variants, then lower it to an IR that can be costed

Key contributions

Following on from the cunning plan

1. Type transformations for generating program variants
2. A new(ish) intermediate language based on LLVM, and
3. *A light-weight cost-model*

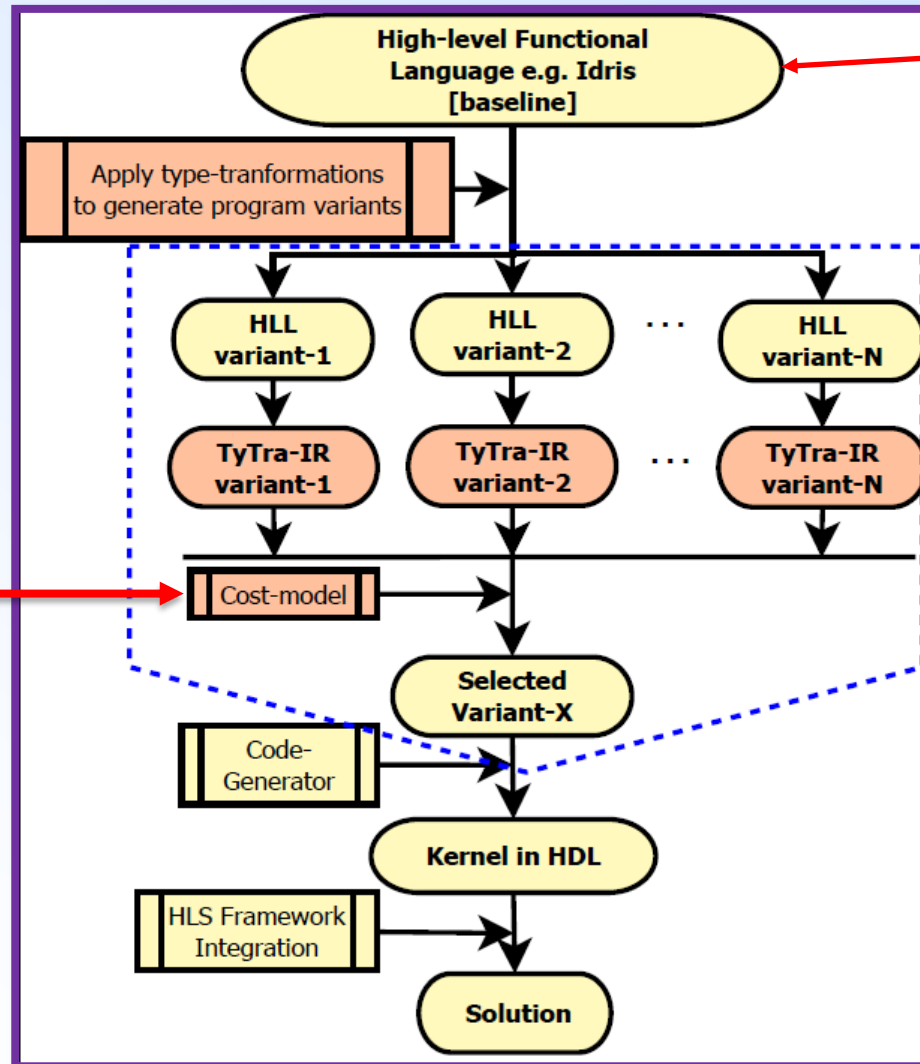


Generating variants, and connecting them to a cost-model (and generator) via an appropriate Intermediate Representation



The TyTra Flow

What's keeping us busy these days



this work

Refactored
Fortran Code

Legacy Fortran
Scientific Code

1. Type transformations for generating program variants
2. A new(ish) intermediate language based on LLVM, and
3. **A light-weight cost-model**

Generating variants, and connecting them to a cost-model (and code-generator) via an appropriate Intermediate Representation

Type Transformations

Why Functional Programming

- Describe **what something is**, not **what to do**
 - *not imperative*
 - *no “side-effects”*
- High-level **types** that describe functions as well as variables
- Transformation of **vector-types** can be done in a provably correct manner
- **Type-transformations** translate to **design-variants** on the FPGA

A functional paradigm with high-level functions allows creation of design-variants that are correct-by-construction.

Type Transformations

Illustration

Vector Types

- `typeA :Vect im int` *--1D integer vector sized im*
- `typeB :Vect km (Vect im int)` *--transformed 2D data*

Program Variants

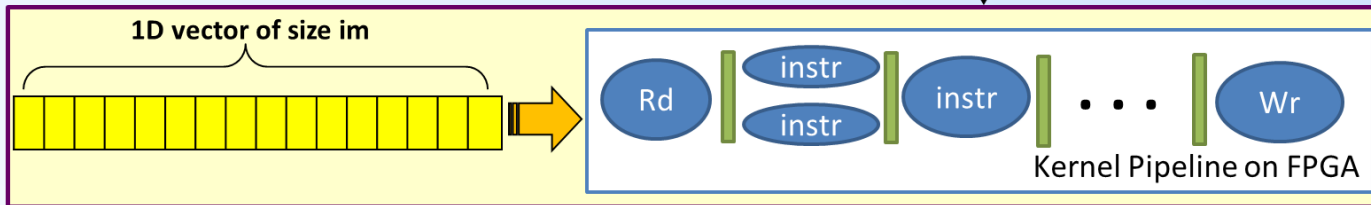
- `output = mappipe kernel_func input` *--original program*
- `inputTra = reshapeTo km input` *--reshaping data*
- `output = mappar (mappipe kernel_func) inputTra` *--new program*

Simple and provably correct vector transformations in the functional paradigm enable generation of “program-variants”

Type Transformations

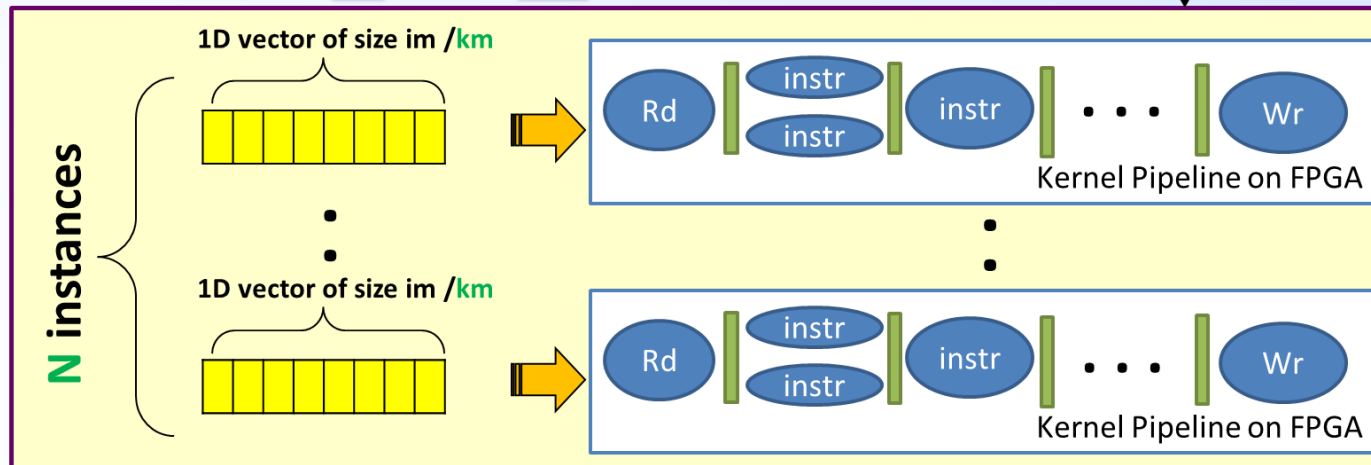
Illustration

`output = mappipe kernel_func input1D`



`input2D = reshapeTo km input1D`

`output = mappar (mappipe kernel_func) input2D`



The program-variants from high-level transformation translate into **design-variants** on the FPGA

Key contributions

1. Type transformations for generating program variants
2. A new(ish) intermediate language based on LLVM, and
3. A light-weight cost-model

Generating variants, and connecting them to a cost-model (and generator) via an appropriate Intermediate Representation

A New Intermediate Language

baseline

```
1 @main.a = addrSpace(12) ui18,  
2         !"istream", !"CONT", !0, !"strobj_a"  
3 @...[other ports]  
4 define void @f1 ( ...args...) pipe {  
5     ui18 %1 = add ui18 %a, %b  
6     ui18 %2 = add ui18 %c, %c  
7     ui18 %3 = mul ui18 %1, %2  
8     ui18 %y = add ui18 %3, @k }  
9 define void @main () {  
10    call @f1(...args...) pipe }
```



functionally
equivalent
variant

```
1 @main.a_01 = ...  
2 @main.a_02 = ...  
3 @...[other ports]  
4 define void @f1 ( ...args...) pipe {...}  
5 define void @f2 (...args...) par {  
6     call @f1(...args...) pipe  
7     call @f1(...args...) pipe  
8     call @f1(...args...) pipe  
9     call @f1(...args...) pipe }  
10 define void @main () {  
11    call @f2(...args...) par }
```

Design-variants are lowered into an Intermediate-Representation, making it easier to estimate cost, performance, and then generate HDL code

A New Intermediate Language

Expressing Configurations in the IR

```

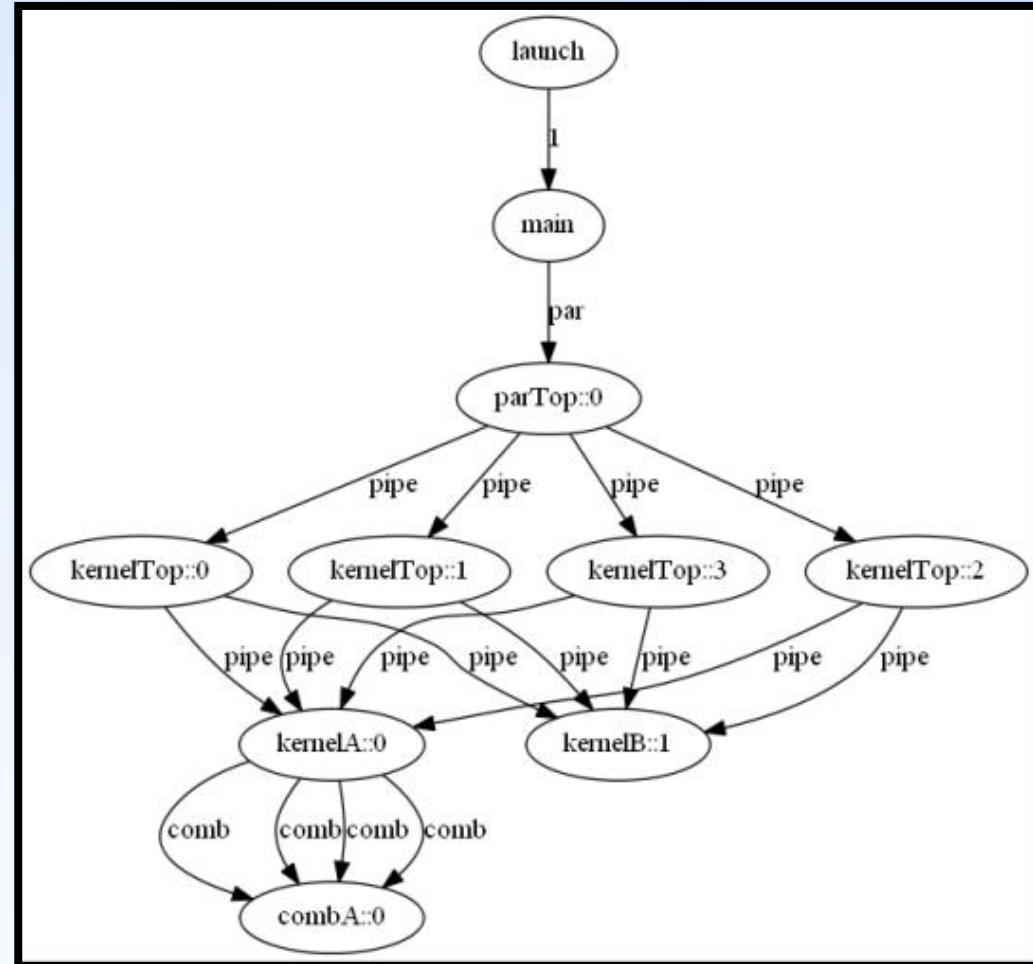
;1. Pipeline with
;   combinatorial blocks
pipe {
  instr
  instr
  combA()
  ... }

;2. Data-parallel
;   pipelines
par {
  pipeA()
  pipeA()
  ... }

;3. Coarse-grained
;   pipeline
pipe {
  pipeA()
  pipeB()
  ... }

;4. Data-parallel
;   Coarse-grained pipeline
par {
  pipeTop()
  pipeTop()
  ... }
;where
pipeTop{
  pipeA()
  pipeB()
  ... }

```



Nesting functions of types `pipe`, `par`, `seq` and `comb` in different combinations enables expression of different design configurations

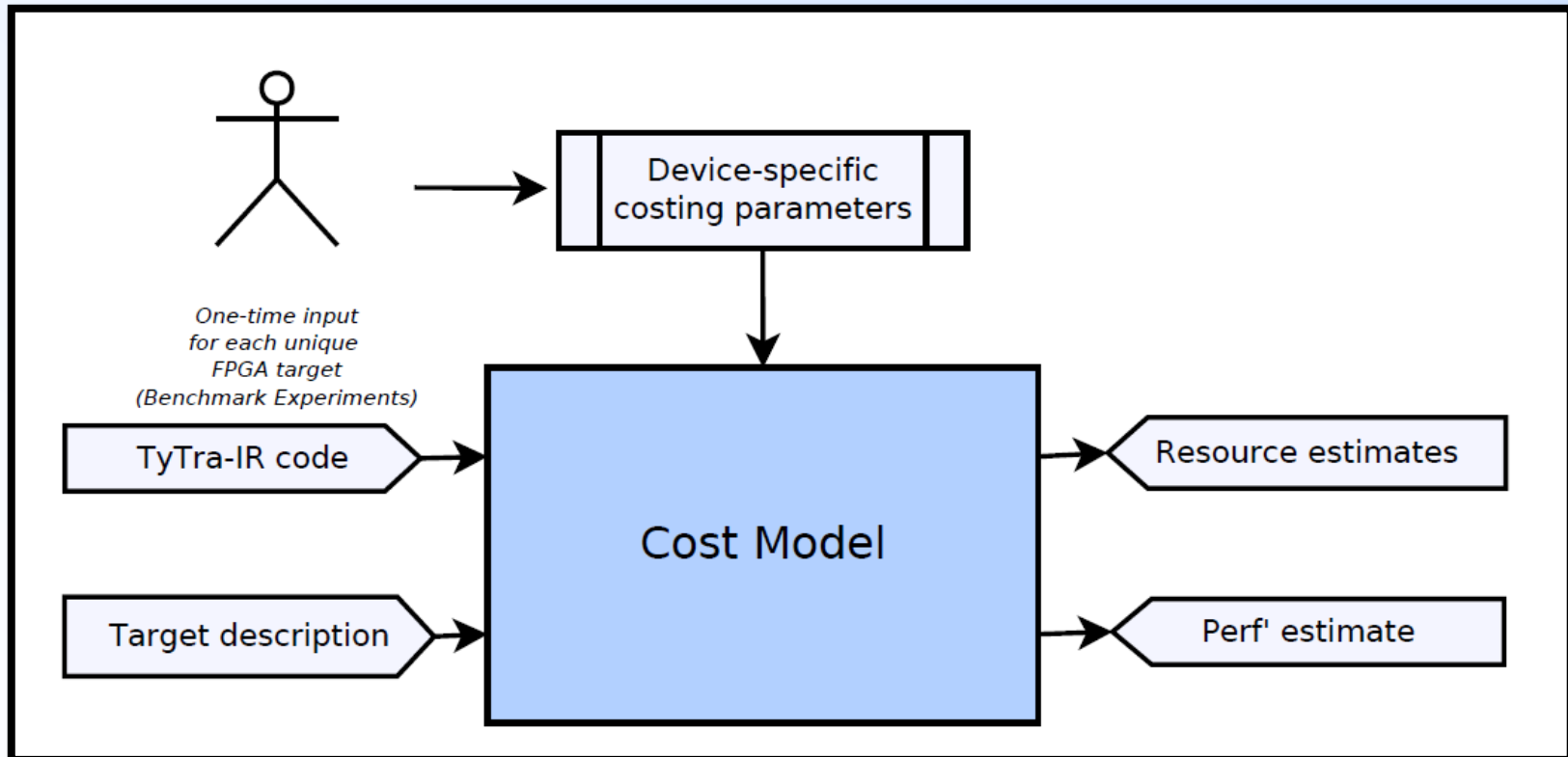
Now that we have design variants...

How do we know which variants are valid (fit on the FPGA)?

How do we know which one performs the best?



DEVELOPING THE COST- MODEL




A set of standardized experiments for each new target feeds empirical data to the cost model, and the rest comes from the IR description.

1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space and cost-space model
5. Memory execution model
6. Data access pattern model

Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space

Pre-requisite: Models Of Abstraction

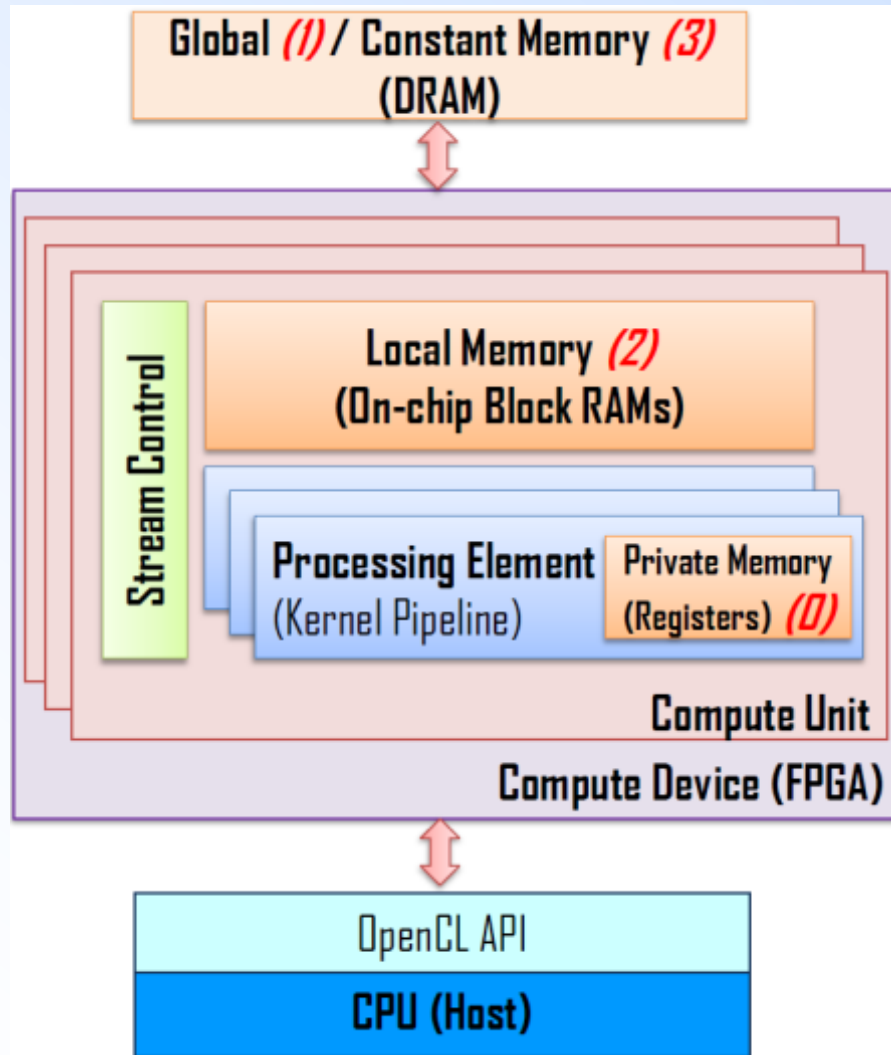
1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. Memory execution model
6. Data access pattern model



(More or less) based
on OpenCL standard

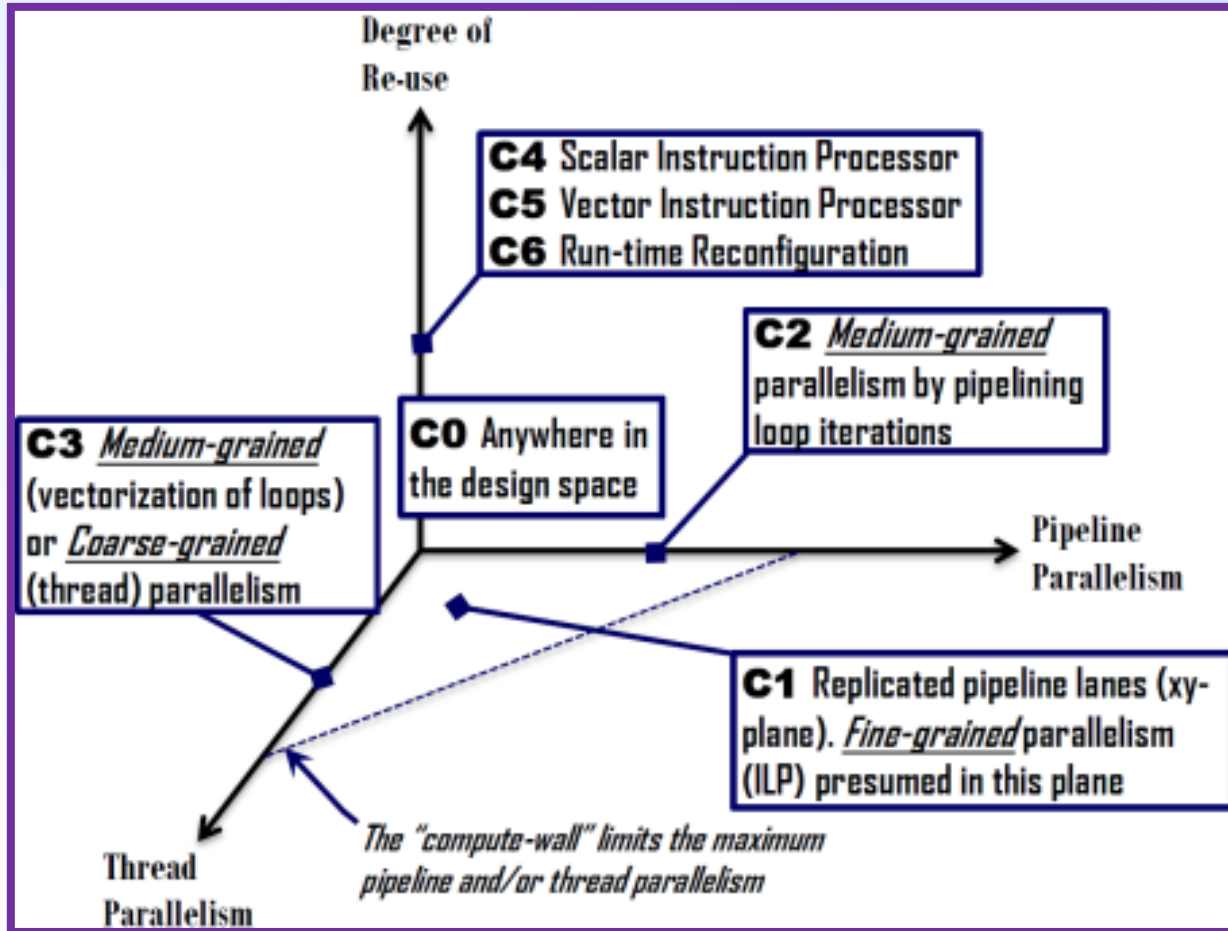
Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space

Platform And Memory Model



1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. Memory execution model
6. Data access pattern model

Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space

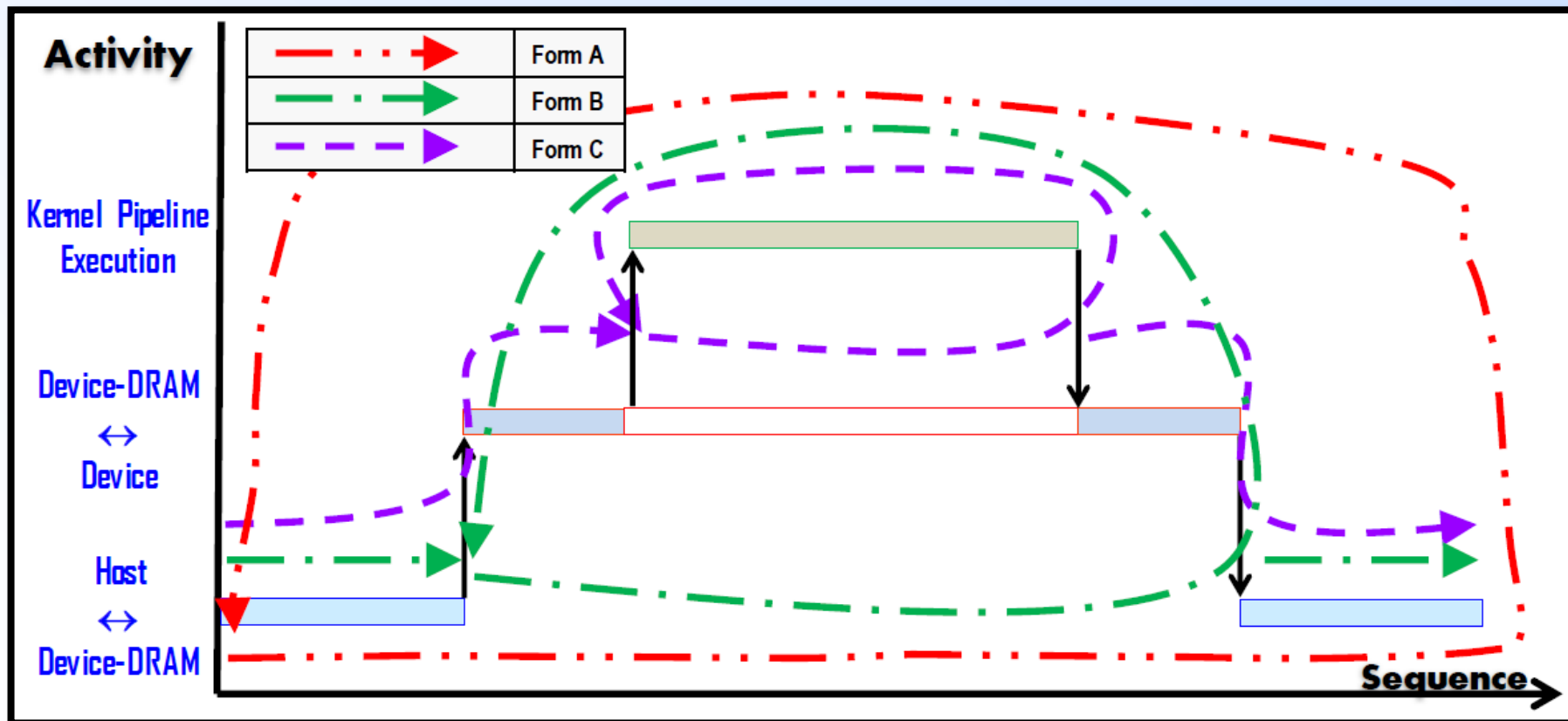


A way to look at the design-space for FPGA implementation. This still does not

1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. **Memory execution model**
6. Data access pattern model

Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space

Performance Estimate Dependence On Memory Execution Model



The manner in which the FPGA memory-hierarchy is accessed across the execution of an application has a huge impact on performance

1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. Memory execution model
6. **Data access pattern model**

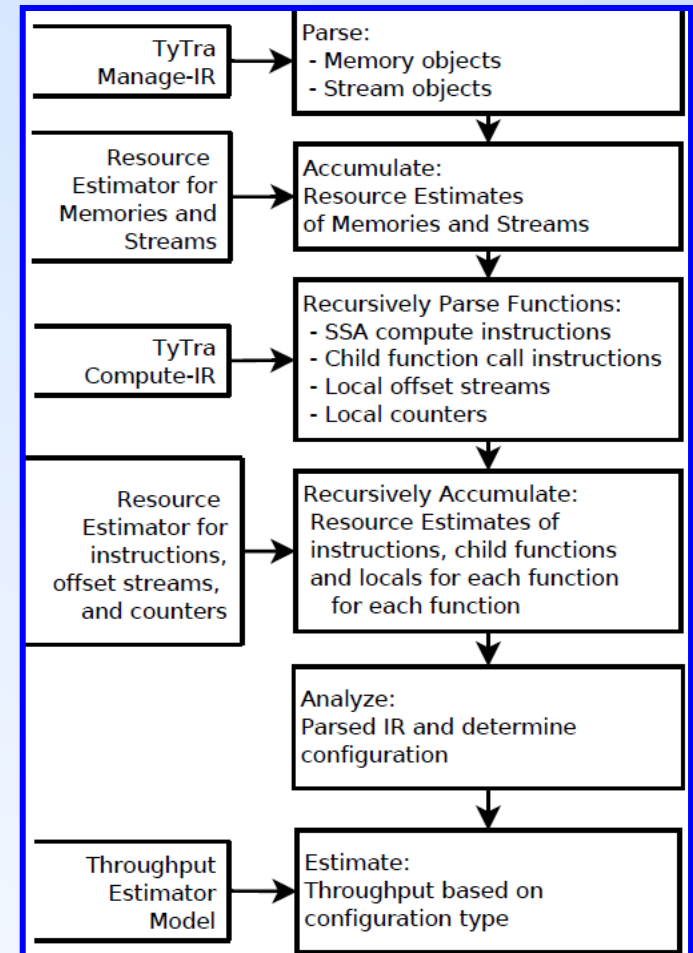
Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space

1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. Memory execution model
6. Data access pattern model
 - Contiguous access
 - (Fixed) Strided access

The data-access pattern has significant impact on performance of memory-bound applications

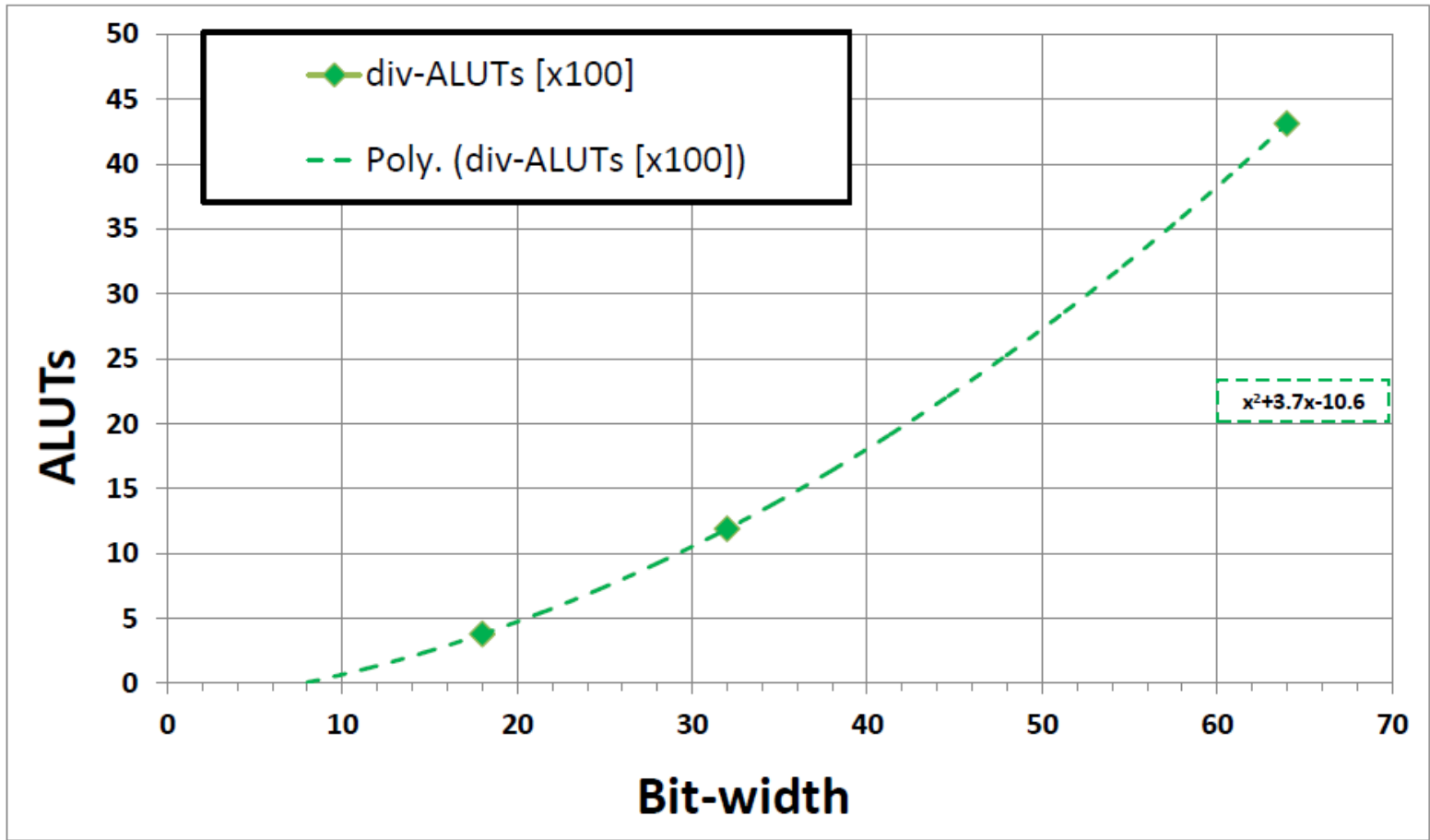
Two Types of Estimates

- Resource-Utilization Estimates
 - ALUTs, REGs, DSPs
- Performance Estimates
 - Memory-bound or compute-bound?
 - **Memory-bound: The sustained memory bandwidth**
 - **Compute-bound: FPGA pipeline throughput**

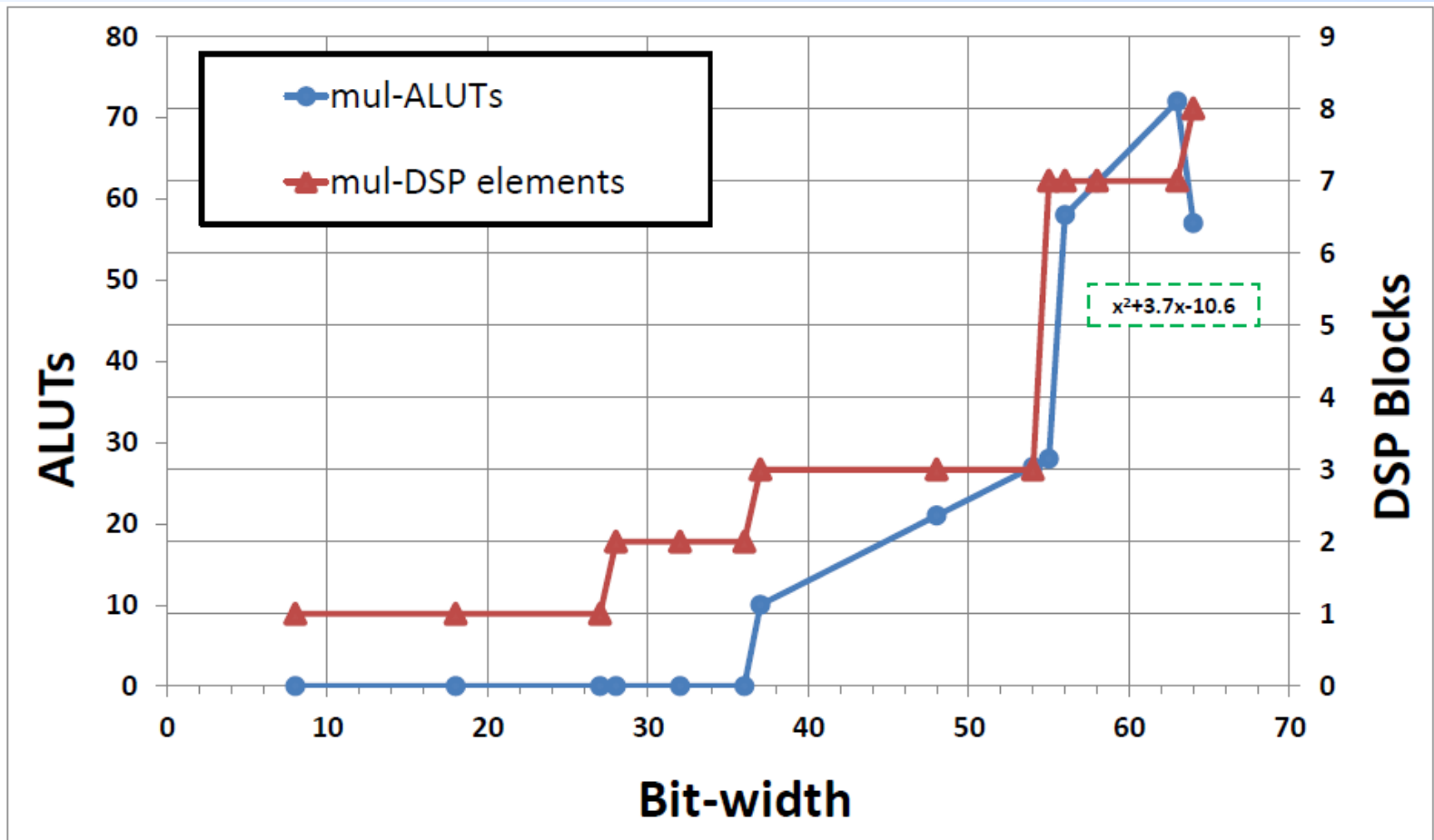


Both estimates needed to allow compiler to choose the best design variant.

- Estimate cost of primitive instructions
 - Instructions should be cost-able across valid data types
- Accumulate costs based on parallelism configuration
 - which is expressed by nesting of functions of types `par`, `pipe`, `seq`



Light-weight cost expressions associated with every legal SSA instruction in the TyTra-IR, e.g. integer division



Light-weight cost expressions associated with every legal SSA instruction in the TyTra-IR, e.g. integer multiplication

- Effective Work-Instance Throughput (EWIT)
 - Work-Instance = Executing the kernel over the entire index-space
- Key Determinants
 - Memory execution model
 - Sustained memory bandwidth for the target architecture and design-variant
 - Data-access pattern
 - Design configuration of the FPGA
 - Operating frequency of the FPGA
 - Compute-bound or IO-bound?

The performance estimate requires design to be classified based on the abstractions we developed earlier

Performance Estimates The Expressions

$$EWUT_A = \frac{1}{\frac{N_{GS} \cdot N_{WPT}}{H_{PB} \cdot \rho_H} + \frac{N_{off}}{G_{PB} \cdot \rho_G} + \frac{K_{PD}}{F_D} + \max\left(\frac{N_{GS} \cdot N_{WPT}}{G_{PB} \cdot \rho_G}, \frac{N_{GS} \cdot N_{WPT} \cdot N_{TO} \cdot N_I}{F_D \cdot K_{NL} \cdot D_V}\right)}$$

$$EWUT_B = \frac{1}{\frac{N_{GS} \cdot N_{WPT}}{N_{WU} \cdot H_{PB} \cdot \rho_H} + \frac{N_{off}}{G_{PB} \cdot \rho_G} + \frac{K_{PD}}{F_D} + \max\left(\frac{N_{GS} \cdot N_{WPT}}{G_{PB} \cdot \rho_G}, \frac{N_{GS} \cdot N_{WPT} \cdot N_{TO} \cdot N_I}{F_D \cdot K_{NL} \cdot D_V}\right)}$$

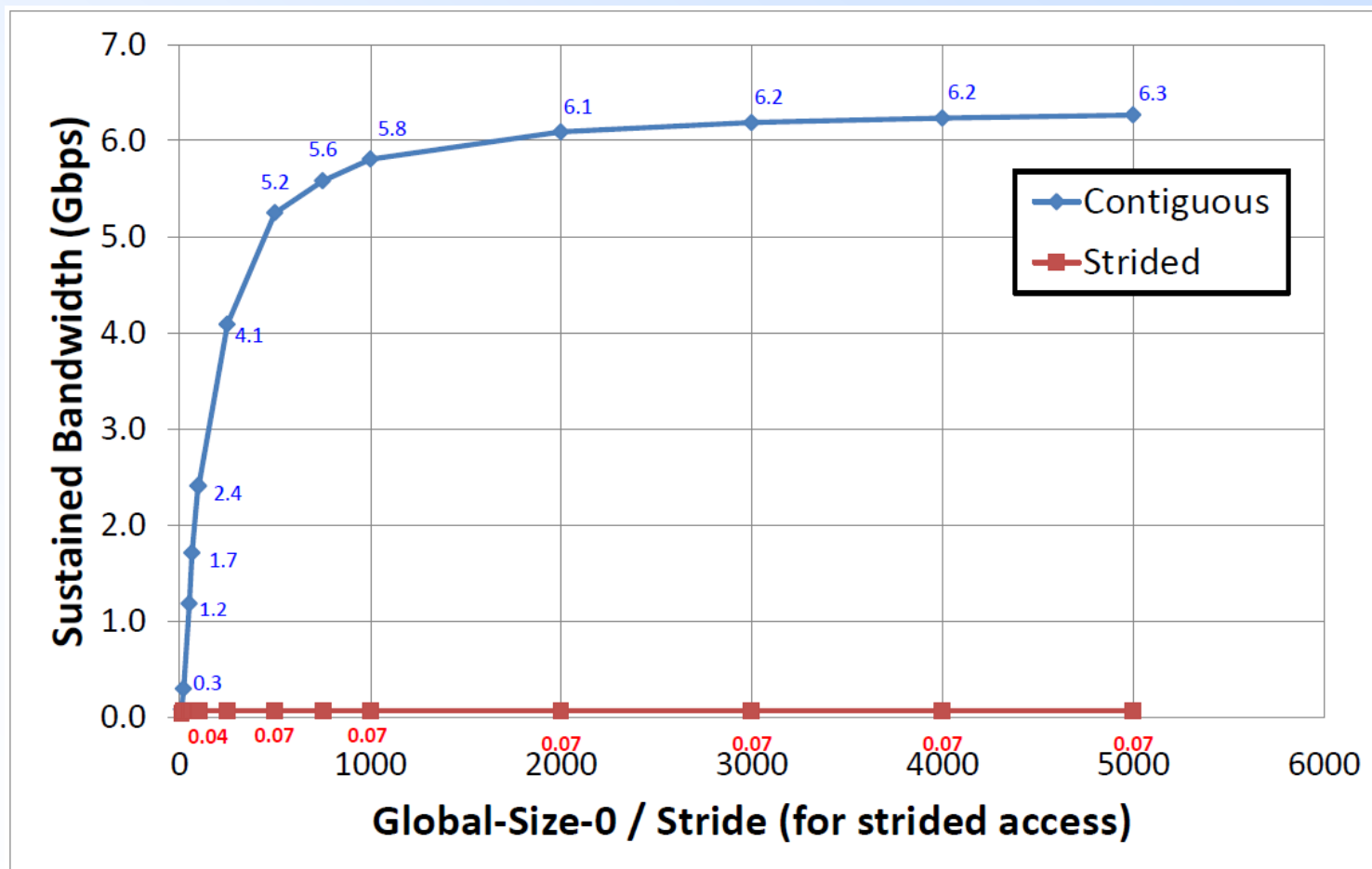
$$EWUT_C = \frac{1}{\frac{N_{GS} \cdot N_{WPT}}{N_{WU} \cdot H_{PB} \cdot \rho_H} + \frac{N_{off}}{G_{PB} \cdot \rho_G} + \frac{K_{PD}}{F_D} + \frac{N_{GS} \cdot N_{WPT} \cdot N_{TO} \cdot N_I}{F_D \cdot K_{NL} \cdot D_V}}$$

Parameters that Make up the Expression

Parameter	Key Dependence	Short Description	Evaluation Method
H_{PB}	Node architecture	The host-device peak bandwidth (typically PCI Express).	Architecture description fed to compiler
ρ_H	Node architecture & design-variant	Scaling factor, host-device bandwidth	<i>Experiments</i> with different data-patterns on the target node.
G_{PB}	Node architecture	The device DRAM peak bandwidth.	Architecture description fed to compiler
ρ_D	Device architecture & design-variant	Scaling factor, host-device bandwidth	<i>Experiments</i> with different data-patterns on the target node.
N_{GS}	Program	Global-size of work-items in NDRange	Compiler parse of IR
N_{WPT}	Program	Words per tuple per work-item	Compiler parse of IR
N_{NDR}	Program	Repetition of kernel over NDRange	Compiler parse of IR
N_{off}	Program	Maximum offset in a stream	Compiler parse of IR
K_{PD}	Program & design-variant	Pipeline depth of kernel	Compiler parse of IR
F_D	Program & design-variant	Device's operating frequency	Compiler costing of IR
N_{TO}	Program & design-variant	Cycles per instruction	Compiler parse of IR
N_I	Program & design-variant	Instructions per PE	Compiler parse of IR
K_{NL}	Program & design-variant	Number of parallel kernel lanes	Compiler parse of IR
D_V	Program & design-variant	Degree of vectorization per lane	Compiler parse of IR

The variables that make up the expression for estimating performance (throughput) are either directly available from IR, or require an empirical model

Effect of Access Pattern with Different Array Sizes



An illustration showing impact of data-access pattern on an sdaccel programmed Alpha-Data FPGA board with Virtex7 device



**Observations
and Results**

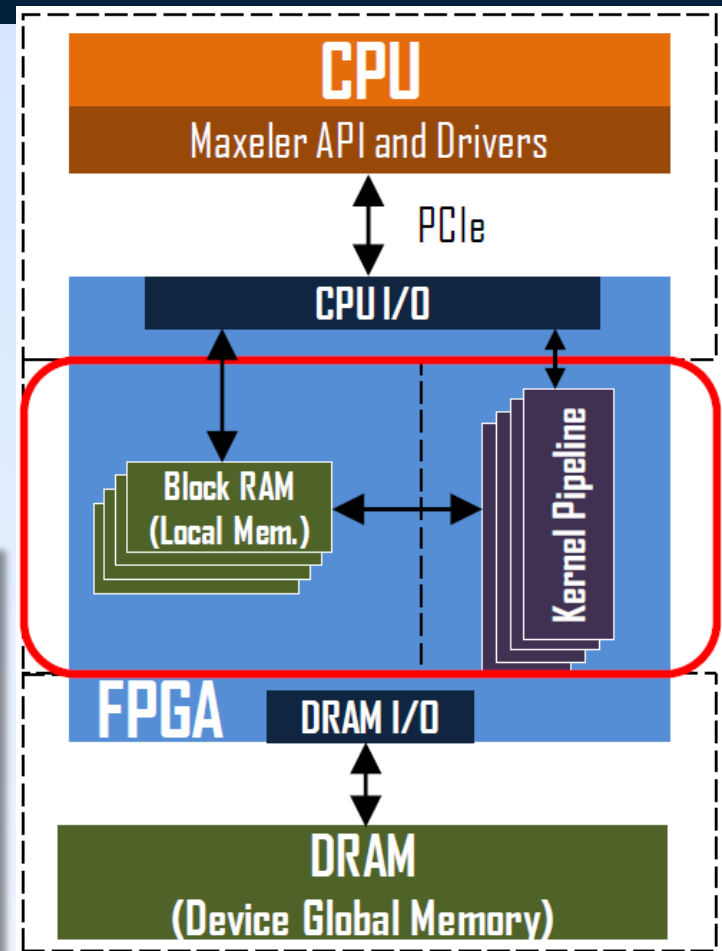
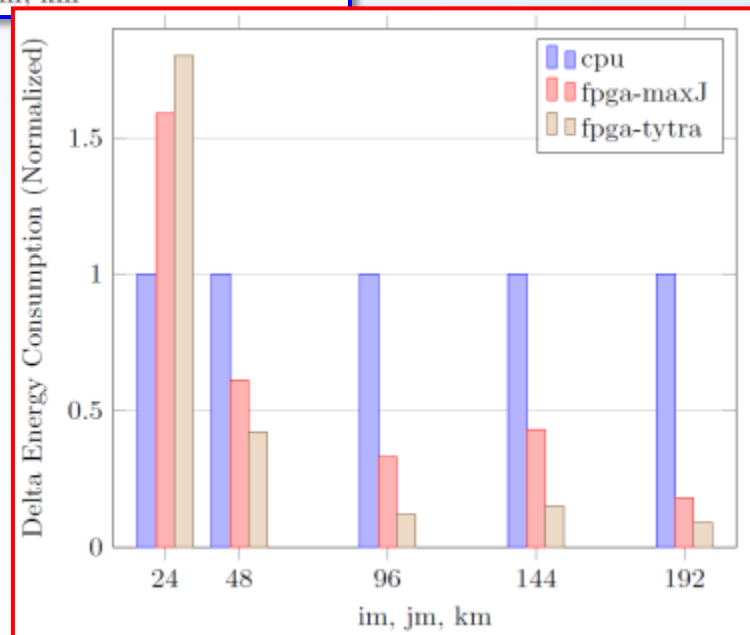
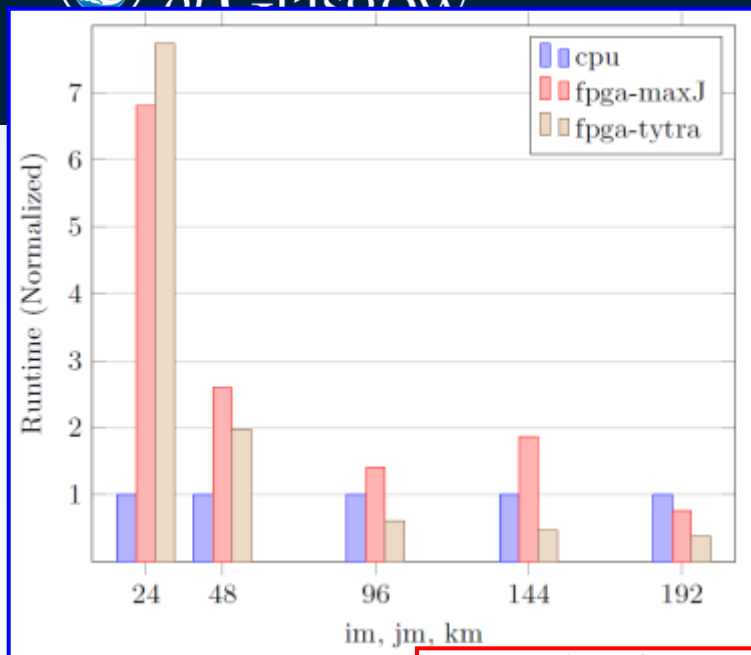
Performance Estimates

Accuracy

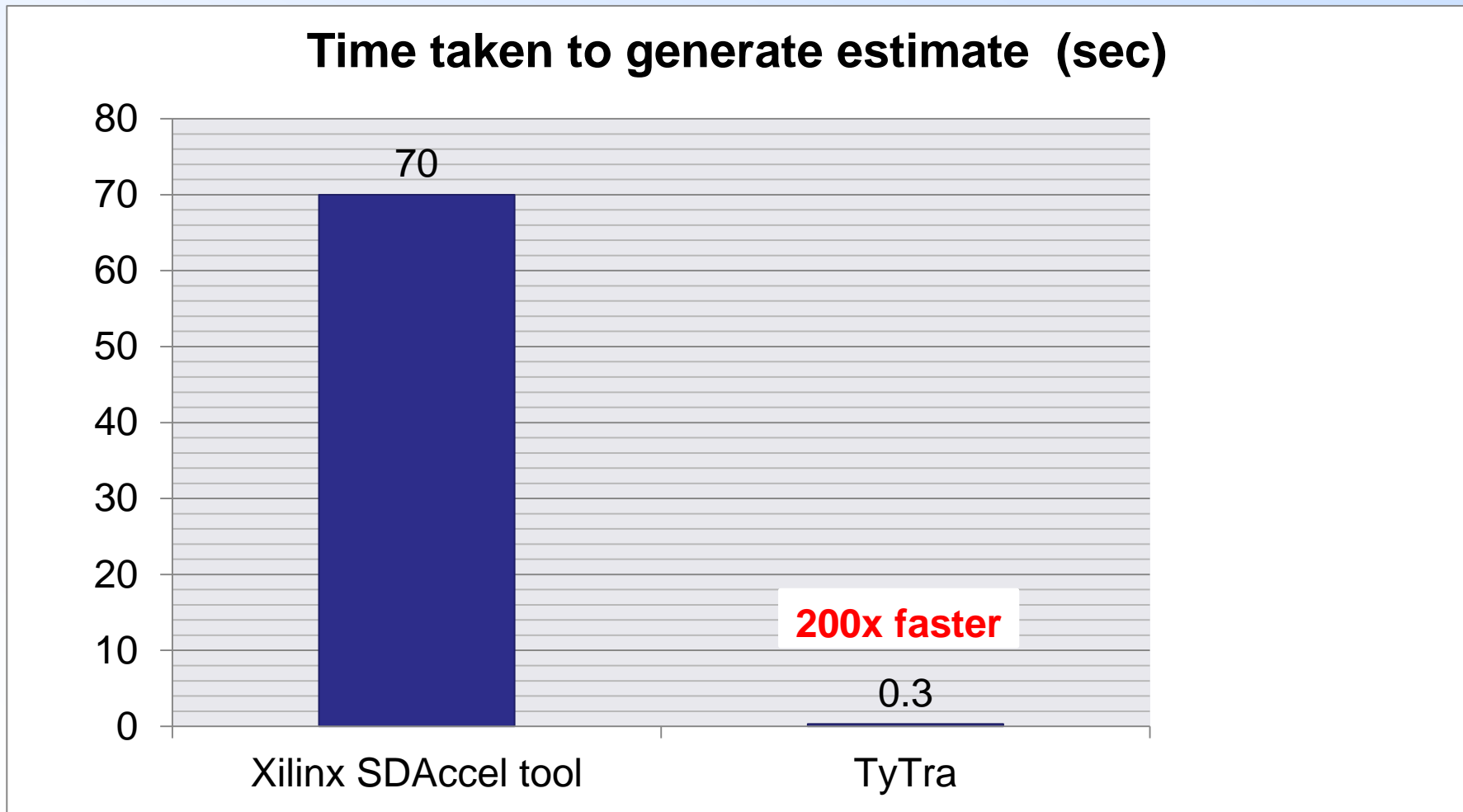
Kernel		ALUT	REG	BRAM	DSP	CPWI
Hotspot (Rodinia)	Estimated	391	1305	32.8K	12	262.3K
	Actual	408	1363	32.7K	12	262.1K
	% error	4	4.2	0.3	0	0.07
LavaMD (Rodinia)	Estimated	408	1496	0	26	111
	Actual	385	1557	0	23	115
	% error	6	3.9	0	13	3.4
SOR	Estimated	528	534	5418	0	292
	Actual	534	575	5400	0	308
	% error	1.1	7.1	0.3	0	5.2

*Preliminary results show estimated vs actual values are quite close. Frequency estimate is still a tricky (hence **Clocks Per Work-Instance**)*

Does the TyTra approach work?

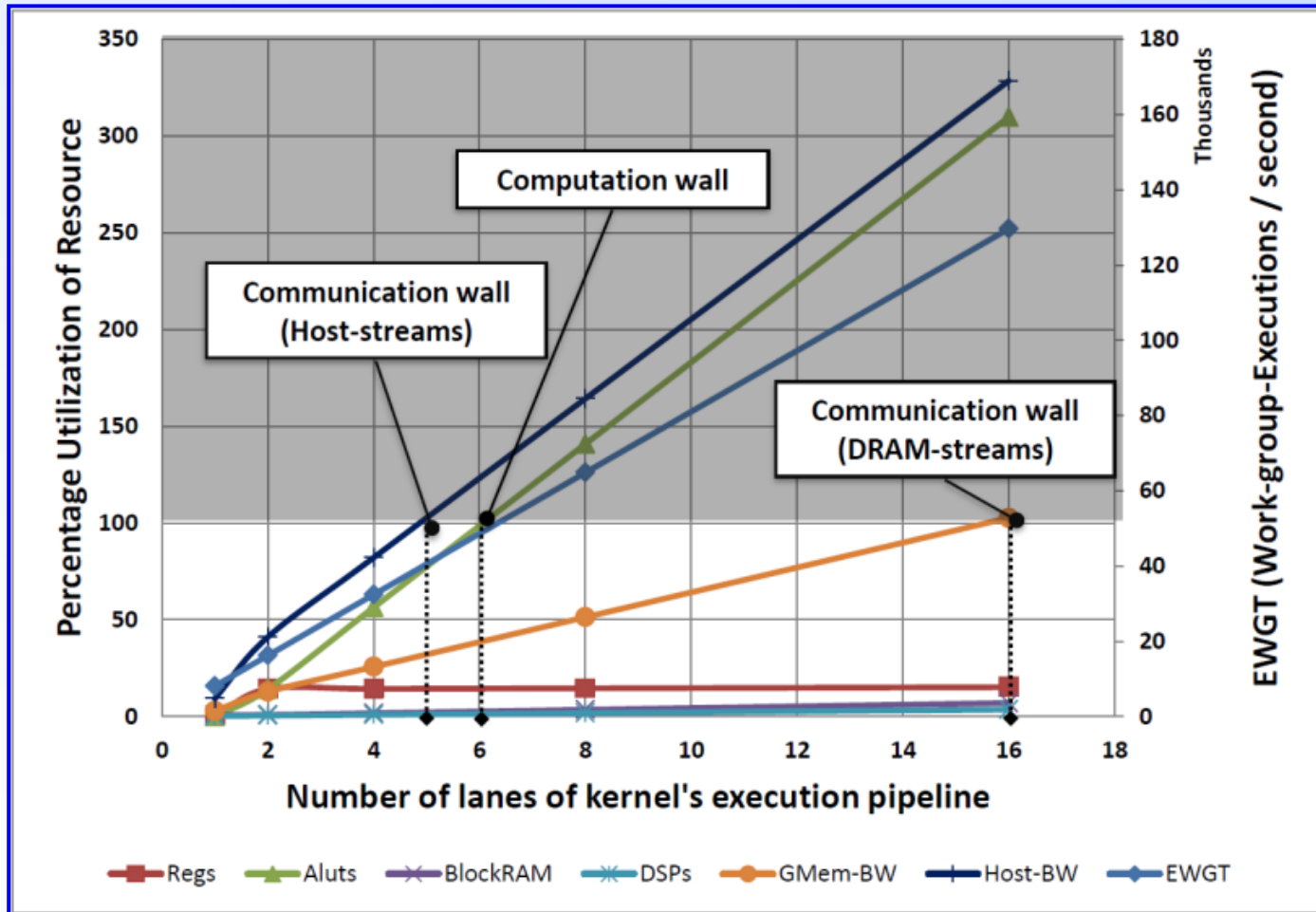


How Fast Is The Cost Model



The requirement of the cost-model to be light-weight is very important if we want to evaluate many design-variants

Design-space Exploration?



Estimates for multiple variants allows us to converge on the best option, and can also give optimization hint back to the compiler/programmer

On-going Challenges

Memory Bandwidth Estimates

- Estimating the memory bandwidth for:
 - a particular application
 - configured as a particular design-variant
 - being compiled for a particular HPC target platform
- If estimate show application will be memory-bound, then:
 - Can we optimize memory access to get better overall performance?
- We are currently working on a memory-bandwidth benchmark for FPGAs

Estimating the correct sustained memory-bandwidth is an important challenge (among others) that we are currently working on


Limitations → Future Work

- Experiment with simple kernels
- Cost-model currently for integers only
- (Lack of) Re-usable, user-friendly and publicly available benchmarks
- Non-optimized number representations
- No automated integration of generated HDL code with HLS tools
 - Manually we have integrated our generated code with Maxeler HLS
- Estimating resources for memory controllers/base platform
 - Also, more accurate estimates of frequency

Stay tuned...

Auto-tuning scientific computing for FPGAs - A fresh approach to cost-modelling

1. A functional language paradigm based *TyTra* Framework
 - Type transformations, variants, IR and need for a cost model
2. Making a light-weight cost-model
 - Models of Abstractions, the cost model for resources and performance, the key variables
3. Experimental results and observations
 - Accuracy, exploration, potential for improvement
 - Limitations and the way forward



*The woods are lovely, dark and deep,
But I have promises to keep,
And lines to code before I sleep,
And lines to code before I sleep.*

syed.nabi@glasgow.ac.uk
<http://dcs.gla.ac.uk/~waqar/>