First International Workshop on Heterogeneous High-performance Reconfigurable Computing (H²RC'15)
Sunday, November 15, 2015
Austin, TX

University of Glasgow | School of Computing Science

Towards Automated Design Space Exploration and Code Generation using Type Transformations

www.tytra.org.uk

S Waqar Nabi & Wim Vanderbauwhede

University of Glasgow
Imperial College London
Heriot Watt University

EPSRC
Engineering and Physical Sciences Research Council

Using Safe Transformations and a Cost-Model for HPC on FPGAs

University of Glasgow | School of Computing Science

- The TyTra project **context**
  - Our approach, *blue-sky* target, *down-to-earth* target, where we are now, how we are different
- Key **contributions**
  - (1) Type transformations to create design-variants, (2) a new Intermediate Language, and (3) an FPGA Cost model
- The **cost model**
  - Performance and resource-usage estimates, some results

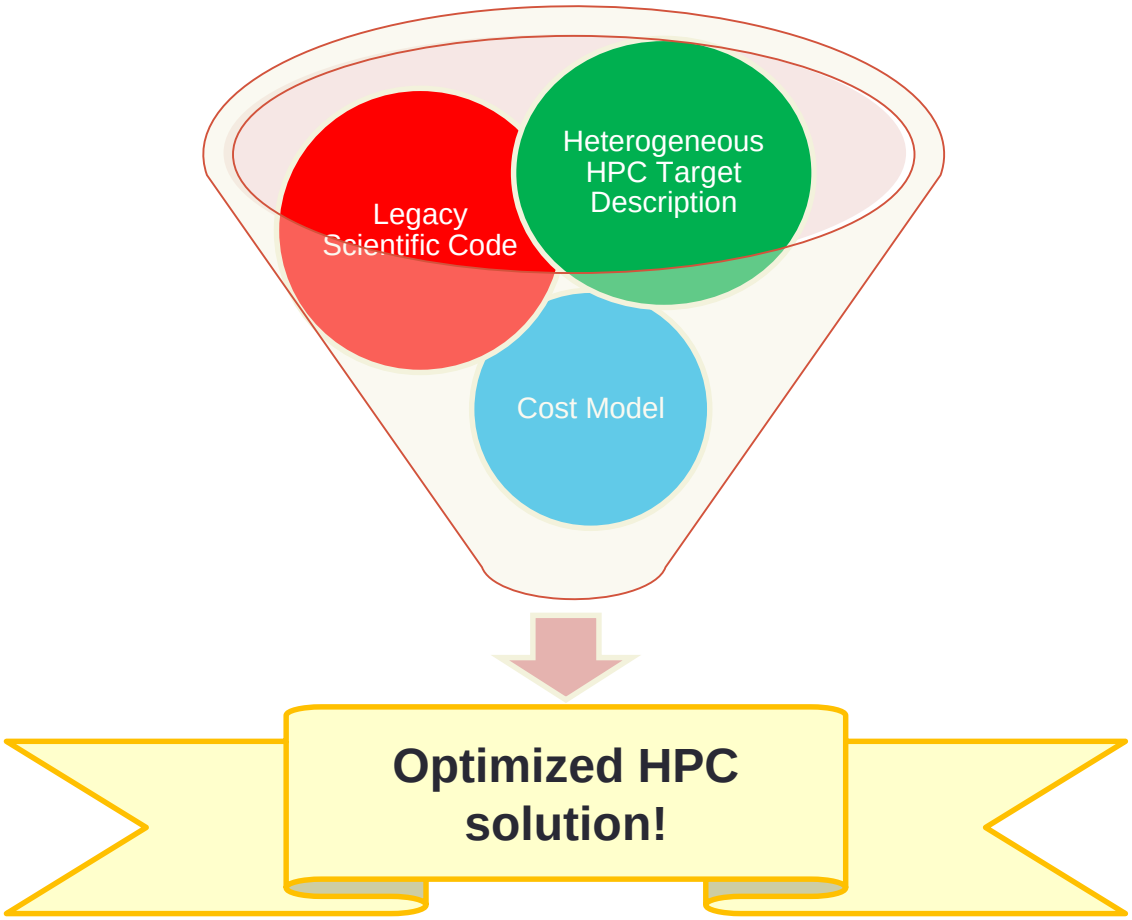
Using safe transformations and an associated light-weight cost-model opens the route to a fully automated design-space exploration flow

THE CONTEXT

Our approach, blue-sky target, down-to-earth target, where we are now, how we are different

Blue Sky Target

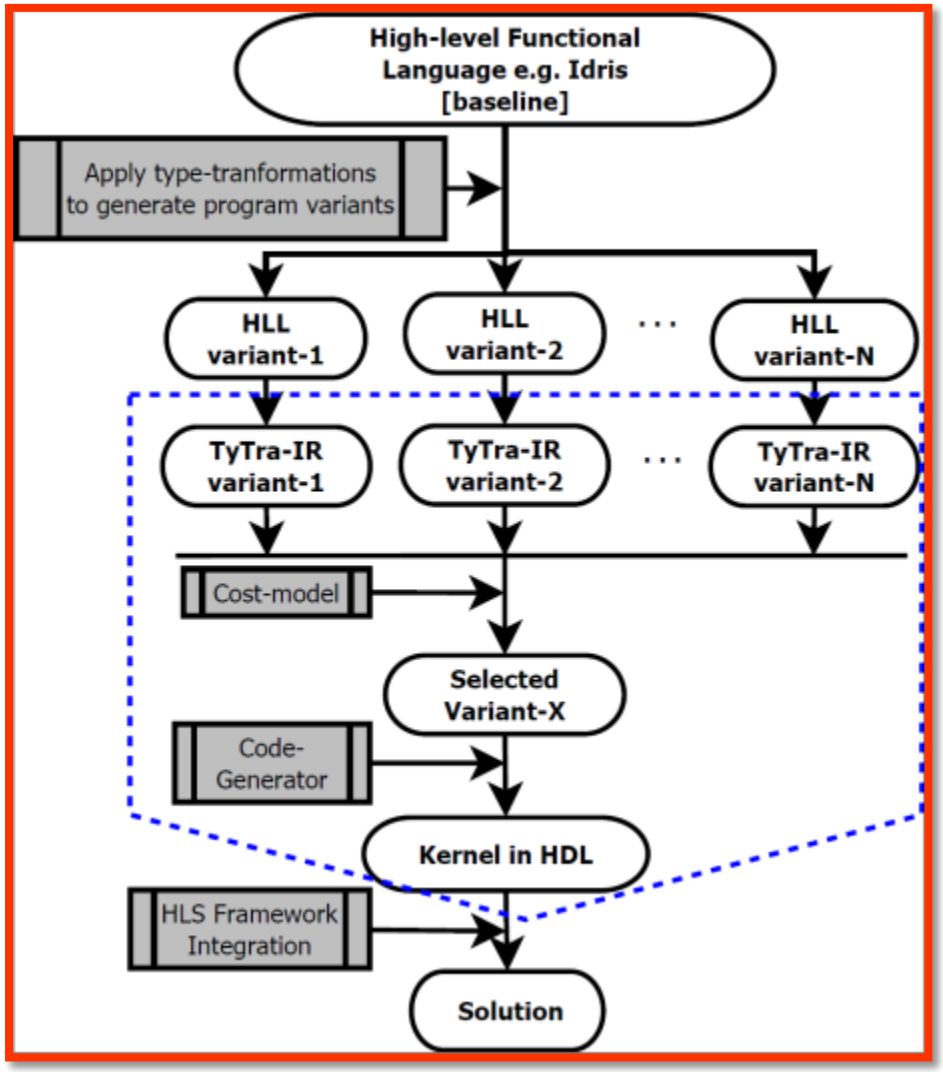
Blue Sky Target



The goal that keeps us motivated!

(The pragmatic target is somewhat more modest…)

The Short-Term Target



Our focus is on FPGA targets, and we currently require design entry in a Functional Language using High-Level Functions (maps, folds) [a kind of DSL]

The cunning plan…

1. Use the **functional programming paradigm** to (auto) generate **program-variants** which translate to **design-variants** on the FPGA.

2. Create an **Intermediate Language** that:
   - Is able to capture points entire design-space
   - Allows a light-weight cost-model to be built around it
   - Is a convenient target for front-end compiler

3. Create a light-weight **cost-model** that can **estimate** the performance and resource-utilization for each **variant**.

A performance portable code-base that builds on a purely software programming paradigm.

University of Glasgow | School of Computing Science

And you may very well ask…

The jury is still out…

How our work is different

- Our observations on limitations of current tools and flows:

1. Design-entry in a ***custom*** high-level language which nevertheless has <u>hardware-specific semantics</u>
2. Architecture of the FPGA-solution specified by programmer; compilers cannot optimize it.
3. Solutions create soft-processors on the FPGA; not optimized for HPC (orientation towards embedded applications)
4. Design-space exploration requires prohibitively long time
5. Compiler is application specific (e.g. DSP applications)

We are not there yet, but in principle, our approach entirely eliminates the first four, and mitigates the fifth.

KEY CONTRIBUTIONS

(1) Type transformations for generating program variants, (2) a new Intermediate Language, and (3) a light-weight Cost Model

1. Type Transformations to Generate Program Variants

University of Glasgow | School of Computing Science

- Functional Programming

- Types
  - More general than types in C
  - Our focus is on *types of functions* that perform **array operations**
    - *reshape, maps and folds*

- Type <u>transformations</u>
  - Can be derived automatically
  - Provably correct
  - Essentially **reshape** the arrays

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

Illustration of Variant Generation through Type-Transformation

University of Glasgow | School of Computing Science

- `typeA :Vect (im*jm*km) dataType --1D data`
  - *Single execution thread*

- `typeB :Vect km (Vect im*jm dataType)--transformed 2D data`
  - *(km concurrent execution threads)*

- `output = map`$_{pipe}$ `kernel_func input  --original program`

- `inputTr = reshapeTo km input  --reshaping data`

- `output = map`$_{par}$ `(map`$_{pipe}$ `kernel_func) inputTr --new program`

Simple and provably correct transformations in a high-level functional language translates to design-variants on the FPGA.

2. A New Intermediate Language

University of Glasgow | School of Computing Science

- Strongly and statically typed
- All computations expressed as SSA (Single-Static Assignments)
- Largely (and deliberately) based on the LLVM-IR

- Manage-IR
  - Deals with
    - memory objects (arrays)
    - streams (loops over arrays)
    - offset streams
    - loops over work-unit
    - block-memory transfers

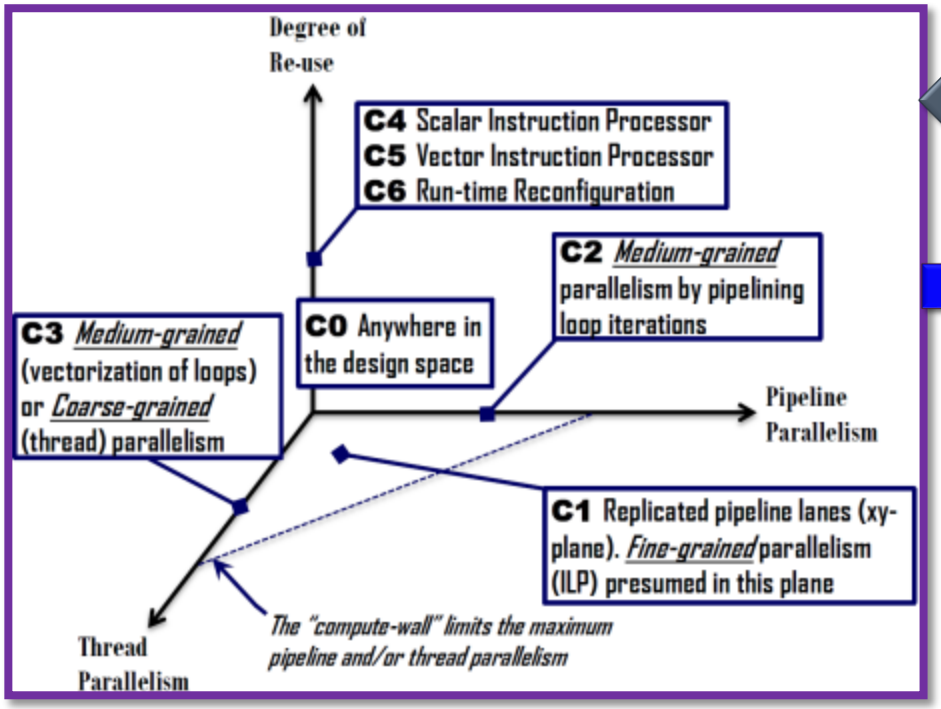
- Compute-IR
  - Streaming model
  - SSA instructions define the datapath

2. A New Intermediate Language

University of Glasgow | School of Computing Science

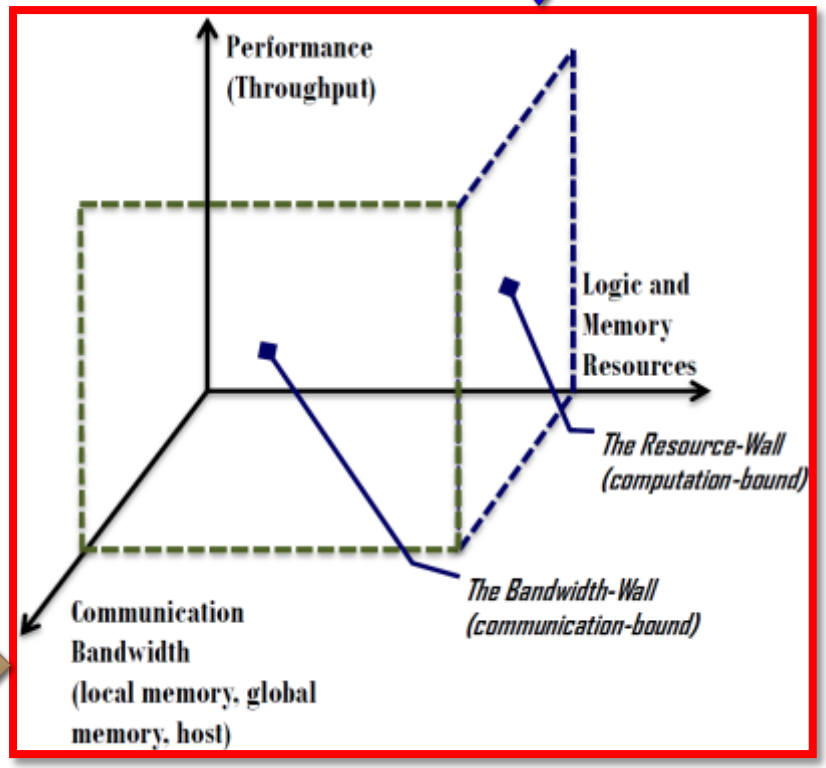
```
@main.a = addrSpace(12) ui18,
          !"istream", !"CONT", !0, !"strobj_a"
@...[other ports]
define void @f1 ( ...args...) pipe {
  ui18 %1 = add ui18 %a, %b
  ui18 %2 = add ui18 %c, %c
  ui18 %3 = mul ui18 %1, %2
  ui18 %y = add ui18 %3, @k }
define void @main () {
  call @f1(...args...) pipe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

```
@main.a_01 = ...
@main.a_02 = ...
@...[other ports]
define void @f1 ( ...args...) pipe {...}
define void @f2 (...args...) par {
  call @f1(...args...) pipe
  call @f1(...args...) pipe
  call @f1(...args...) pipe
  call @f1(...args...) pipe }
define void @main () {
  call @f2(...args...) par }
```



Design Space

The Cost Model



Estimation Space

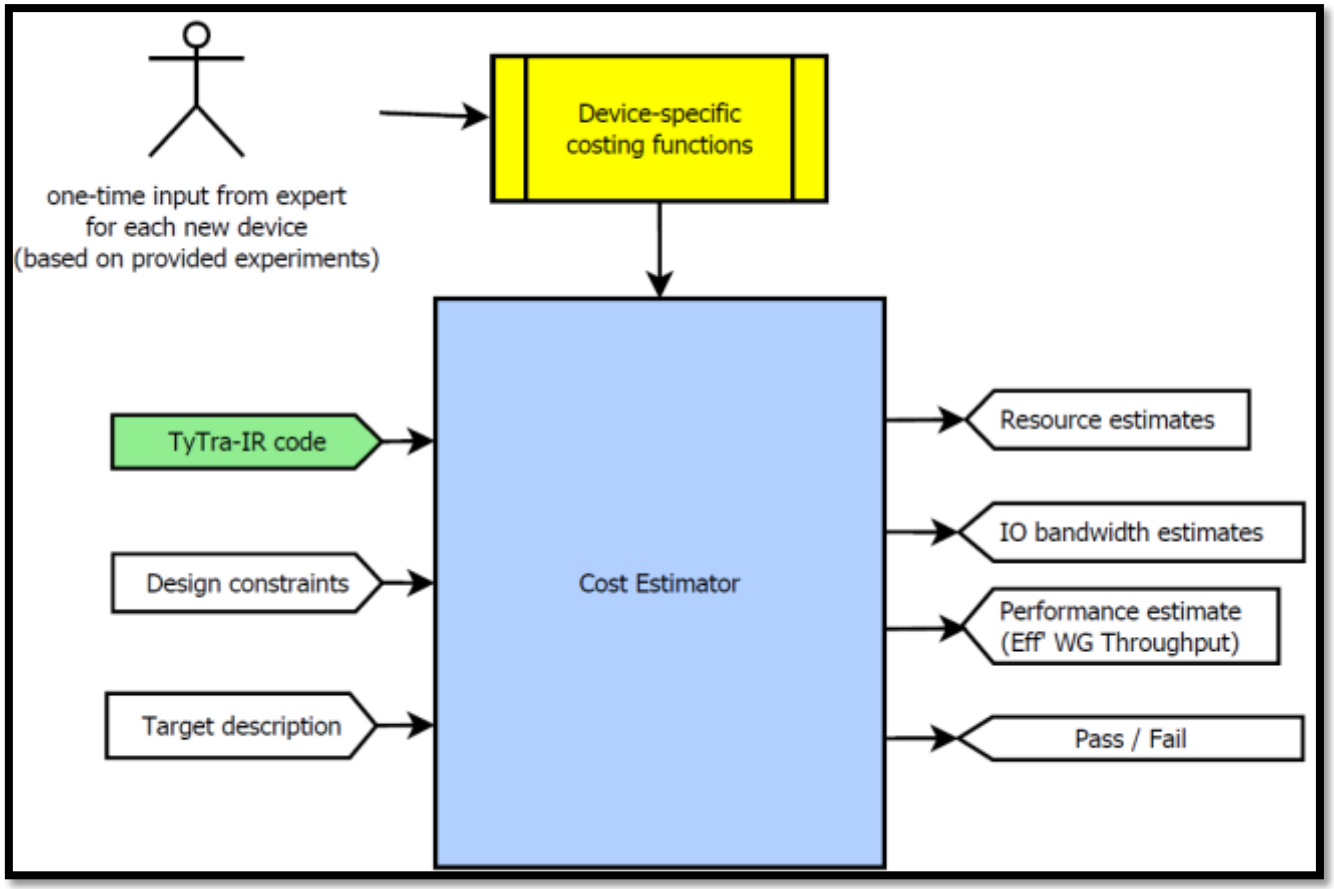
3. Cost Model

THE FPGA COST-MODEL

Performance Estimate, Resource-utilization estiamte, Experimental Results

The Cost-Model Use-Case

University of Glasgow | School of Computing Science

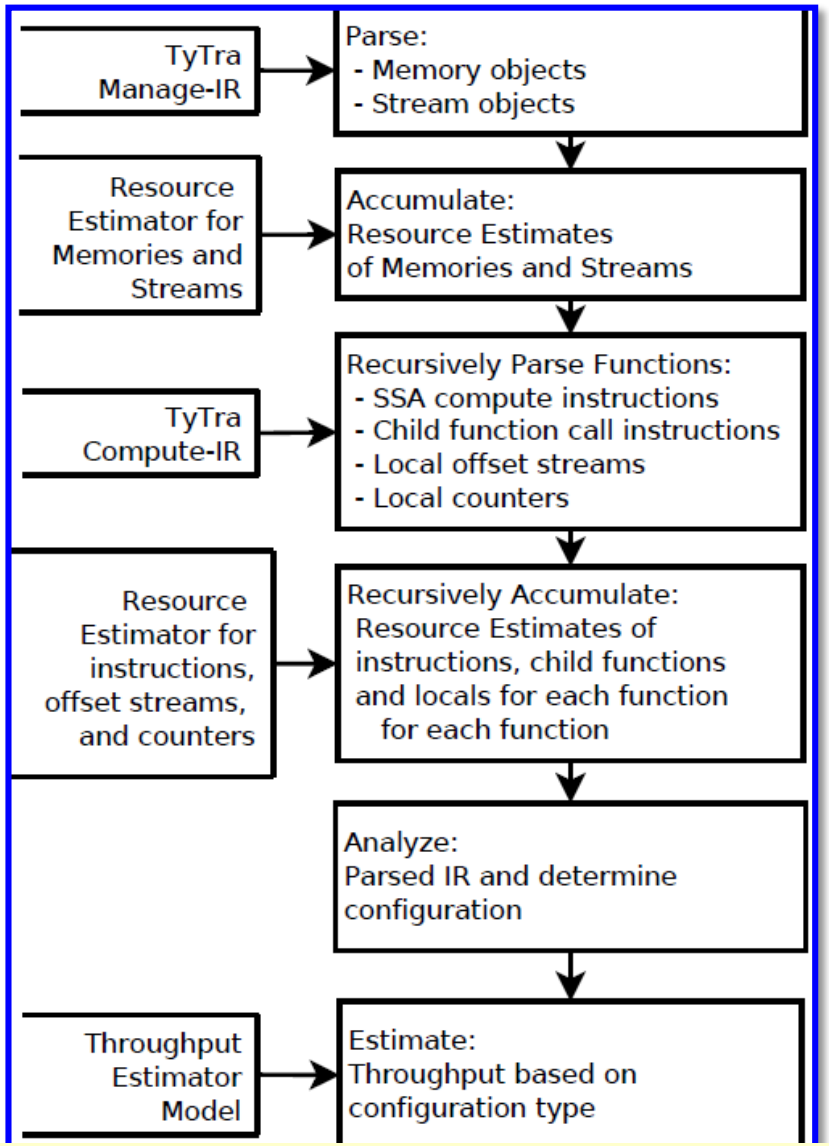


A set of standardized experiments feed target-specific empirical data to the cost model, and the rest comes from the IR description.

University of Glasgow | School of Computing Science

Two Types of Estimates

- Resource-Utilization Estimates
  - ALUTs, REGs, DSPs
- Performance Estimates
  - Estimating memory-access bandwidth for specific data patterns
  - Estimating FPGA operating frequency



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

1. Resource Estimates

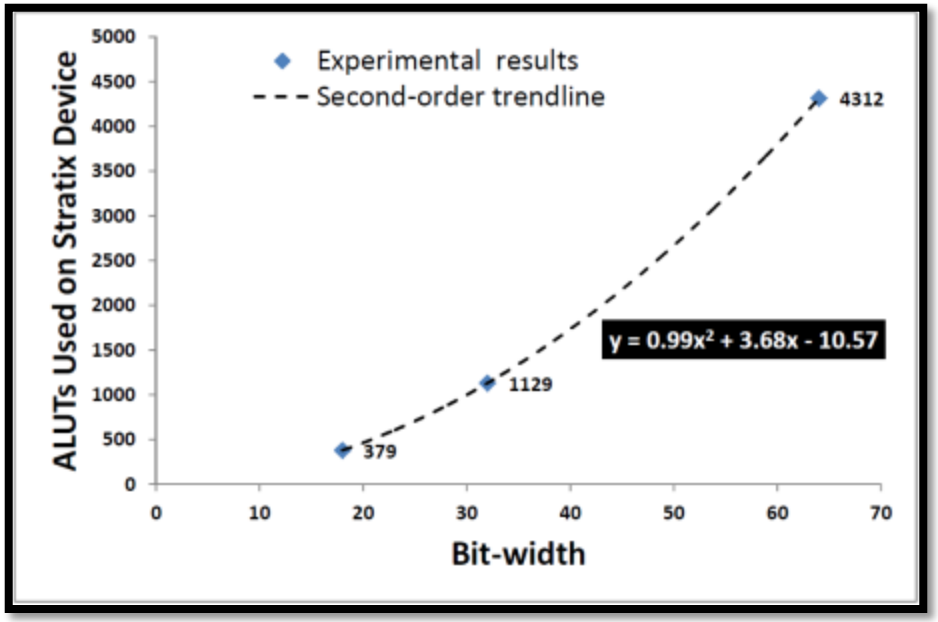
University of Glasgow | School of Computing Science

- Observation
  - Regularity of FPGA fabric allows some very simple first or second order expressions to be built up for most instructions based on a few experiments.
- Key Determinants
  - Primitive (SSA) instructions used in IR of the kernel functions
  - Data-types
  - Structure of various functions (par, comb, par, seq)
  - Control logic over-head

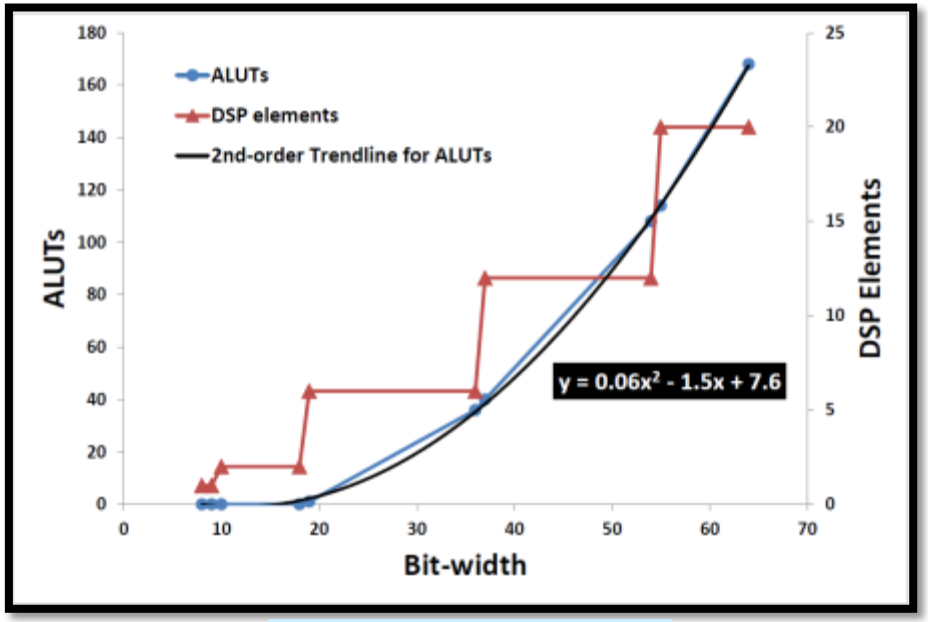
A set of one-time simple synthesis experiments on the target device helps us create a very accurate resource-utilization cost model

Resource Estimates - Example

University of Glasgow | School of Computing Science



Integer Division



Integer Multiplication

Light-weight cost expressions associated with every legal SSA instruction in the TyTra-IR

2. Performance Estimate

University of Glasgow | School of Computing Science

- Effective Work-Unit Throughput (EWUT)
  - *Work-Unit = 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?

Performance model is trickier, especially calculating estimates of sustained memory bandwidth and FPGA operating frequency.

2. Performance Estimate

University of Glasgow | School of Computing Science

- Effective Work-Unit Throughput (EWUT)
  - *Work-Unit = Executing the kernel over the entire index-space*

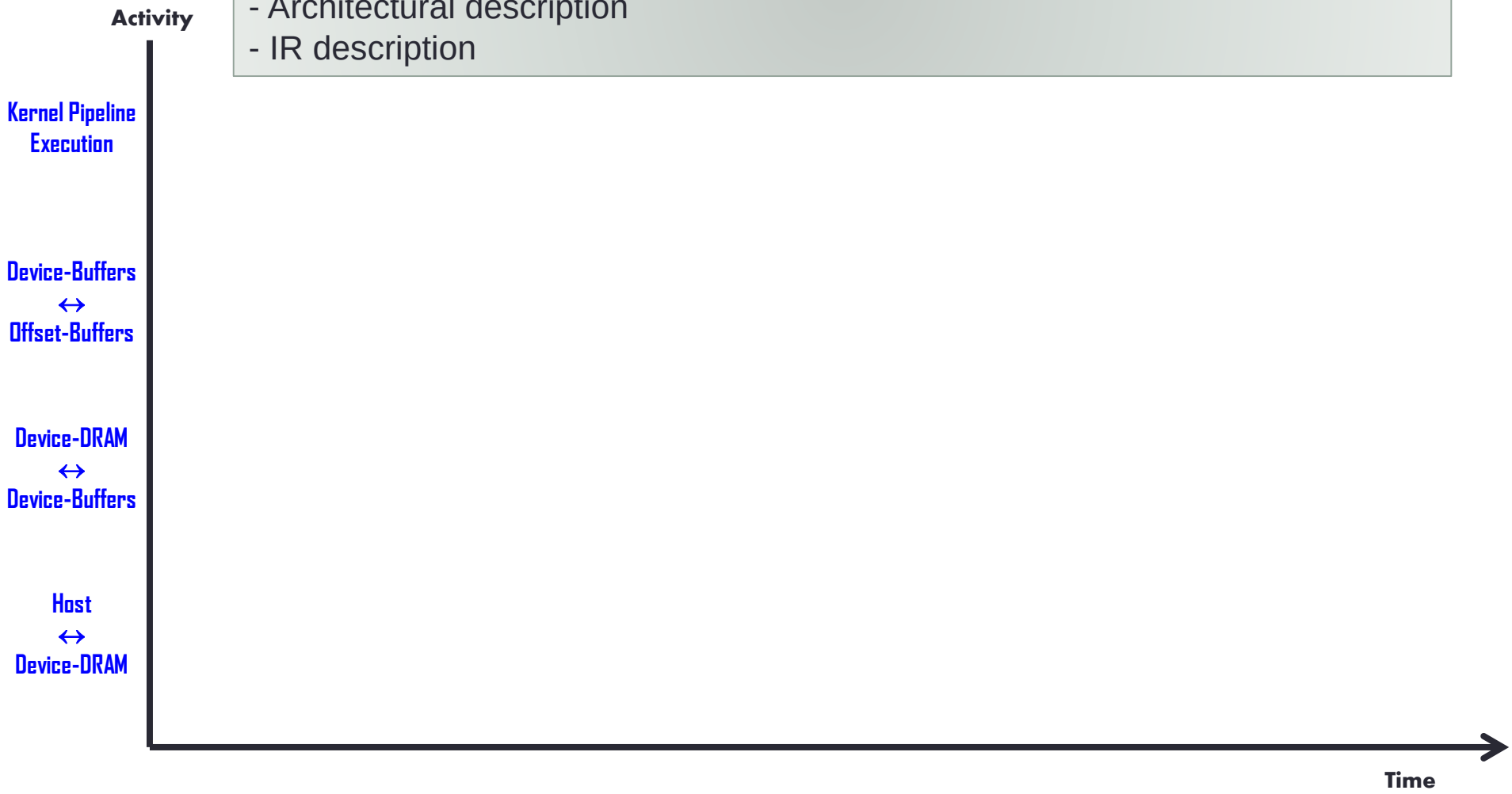
- Key Determinants
  - **Memory execution model**
  - Sustained memory bandwidth for the target architecture and design-variant
    - Data-access pattern
  - Design configuration of the FPGA
  - Operating frequency of the FPGA
  - Compute-bound or IO-bound?

Performance model is trickier, especially calculating estimates of sustained memory bandwidth and FPGA operating frequency.

Performance Estimate
Dependence on Memory Execution Model

University of Glasgow | School of Computing Science

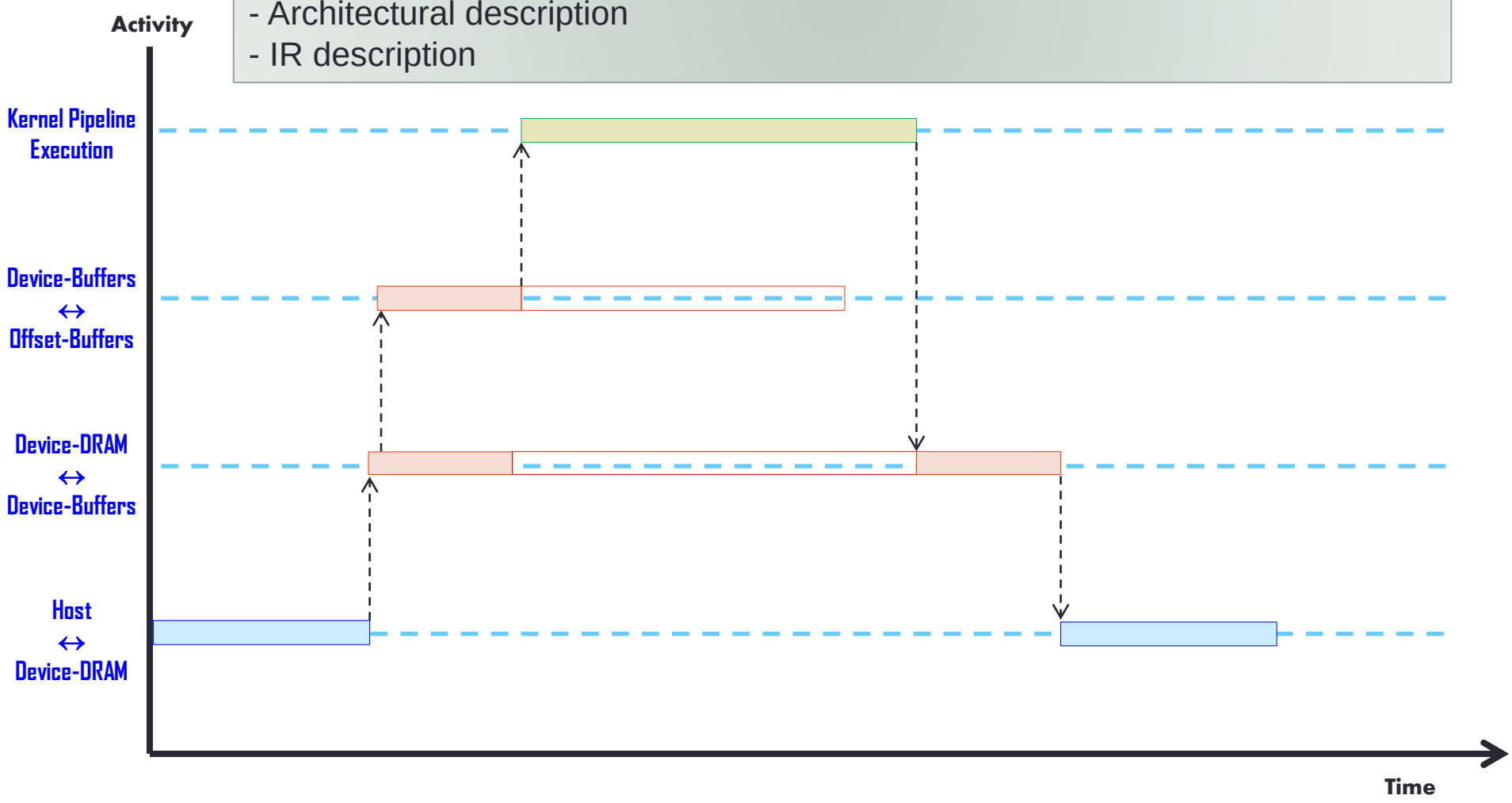
Three **Types** of memory executions
A given design-variant can be categorized based on:
- Architectural description
- IR description



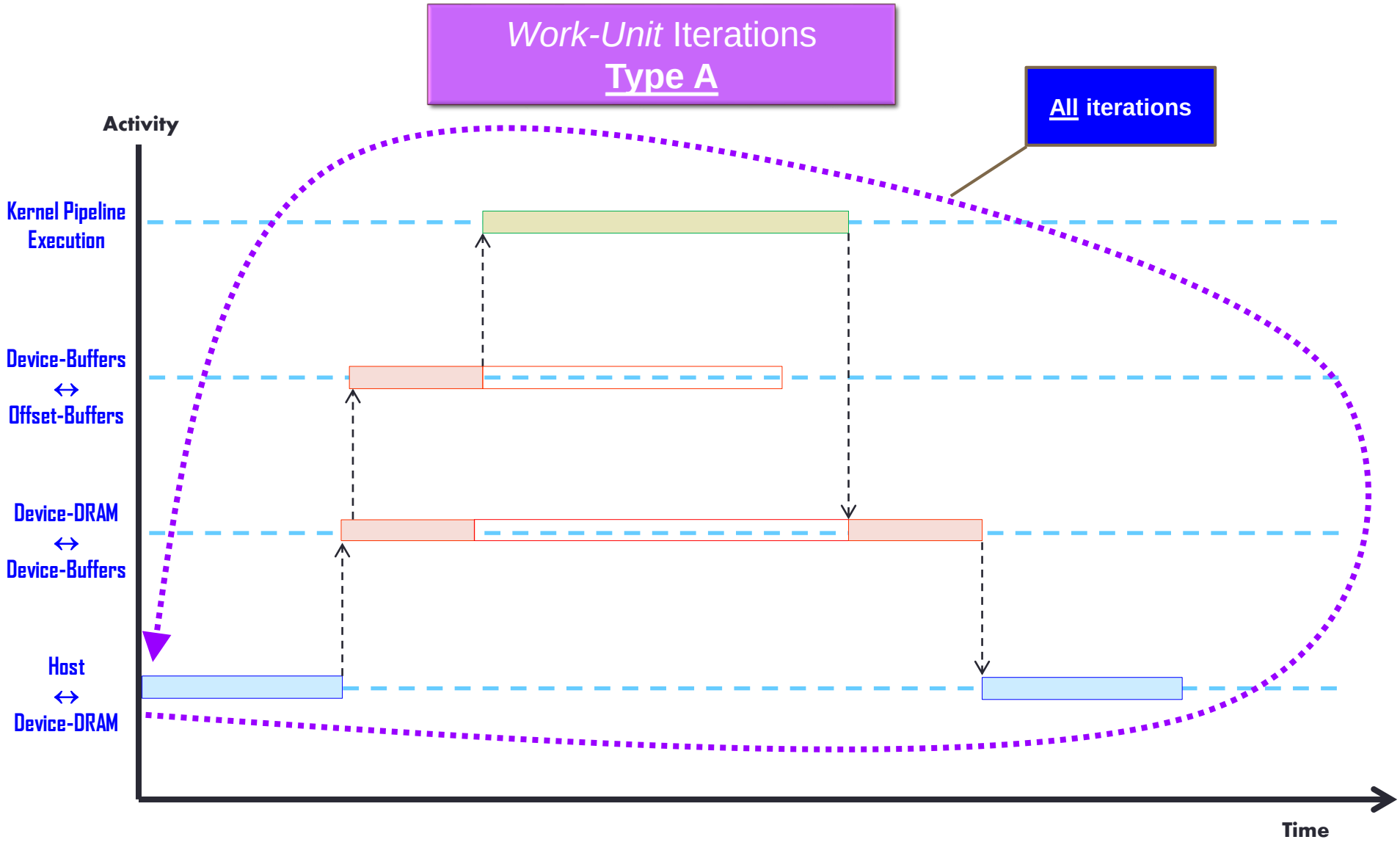
Performance Estimate
Dependence on Memory Execution Model

University of Glasgow | School of Computing Science

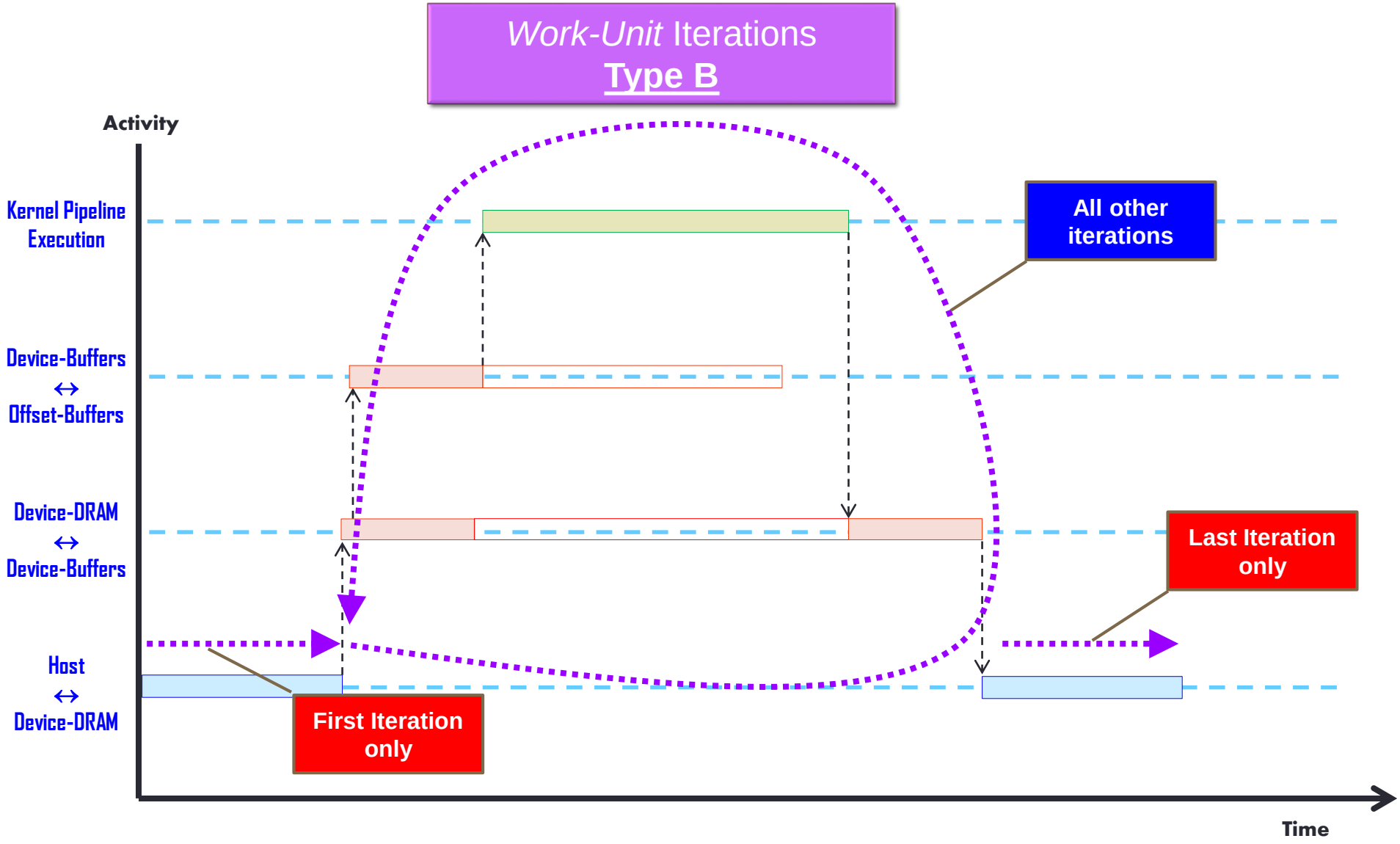
Three **Types** of memory executions
A given design-variant can be categorized based on:
- Architectural description
- IR description



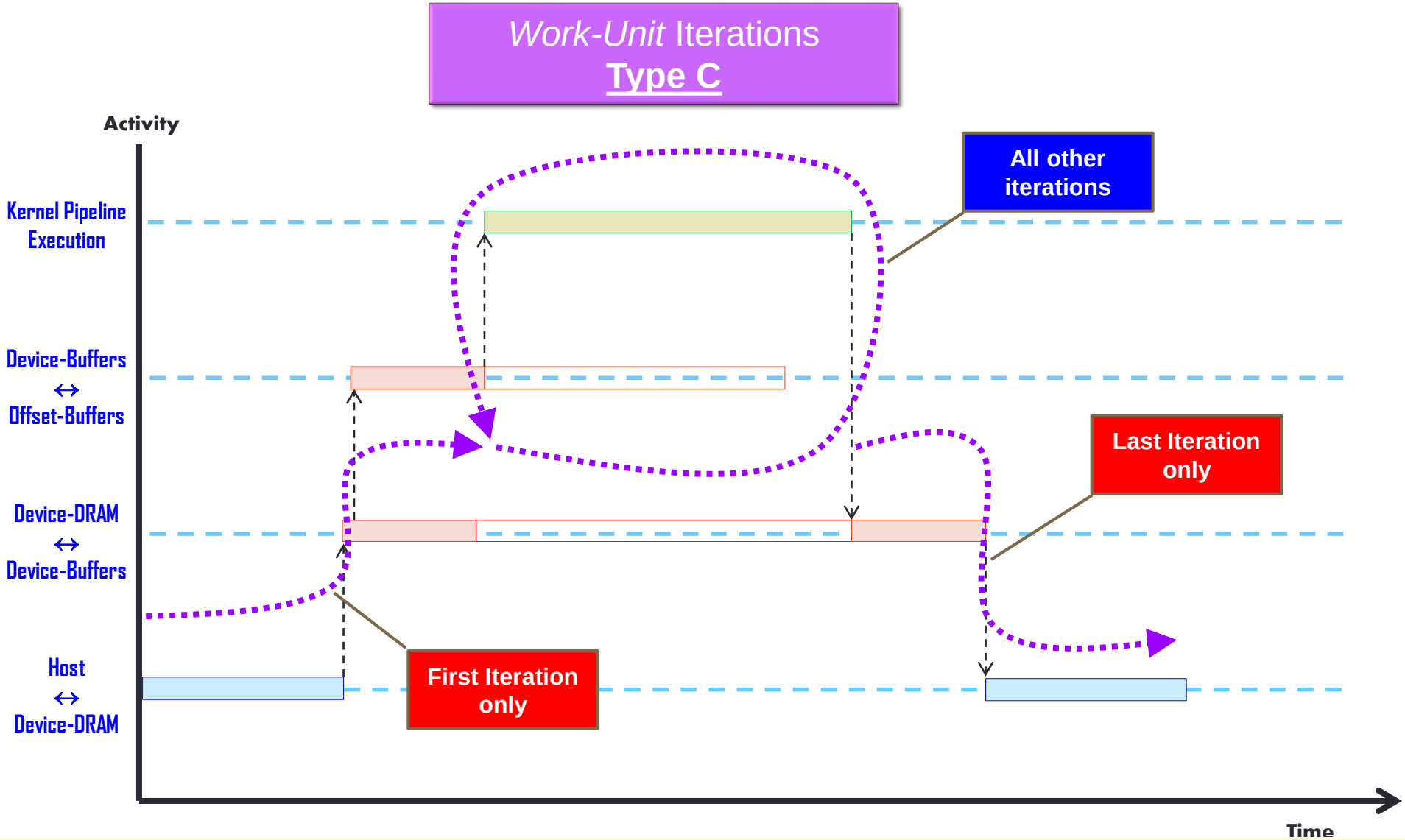
Performance Estimate
Dependence on Memory Execution Model



Performance Estimate
Dependence on Memory Execution Model



Performance Estimate
Dependence on Memory Execution Model



Once a design-variant is categorized, performance can be estimated accordingly

2. Performance Estimate

University of Glasgow | School of Computing Science

- Effective Work-Unit Throughput (EWUT)
  - *Work-Unit = 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?

Performance model is trickier, especially calculating estimates of sustained memory bandwidth and FPGA operating frequency.

University of Glasgow | School of Computing Science

Performance Estimate Dependence on **Data Access Pattern**

- We have defined a ***rho*** ***(ρ)*** factor defined as a *scaling factor* of the **peak** memory bandwidth
  - *Varies from 0-1*
  - *Based on data-access pattern*
  - *Derived empirically through one-time standardized experiments on target node*

2. Performance Estimate

University of Glasgow | School of Computing Science

- Effective Work-Unit Throughput (EWUT)
  - *Work-Unit = 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?**

Determined from the IR description of design-variant

Performance model is trickier, especially calculating estimates of sustained memory bandwidth and FPGA operating frequency.

University of Glasgow | School of Computing Science

Performance Estimates
The Parameters and their Evaluation

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	*Experiments* with different data-patterns on the target node.
G_{PB}	Node architecture	The device DRAM peak bandwidth.	Architecture description fed to compiler
ρ_H	Device architecture & design-variant	Scaling factor, host-device bandwidth	*Experiments* 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

University of Glasgow | School of Computing Science

Performance Estimates

Parameters from Architecture Description

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	*Experiments* with different data-patterns on the target node.
G_{PB}	Node architecture	The device DRAM peak bandwidth.	Architecture description fed to compiler
ρ_H	Device architecture & design-variant	Scaling factor, host-device bandwidth	*Experiments* 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

University of Glasgow | School of Computing Science

Performance Estimates
Parameters Calculated Empirically

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	*Experiments* with different data-patterns on the target node.
G_{PB}	Node architecture	The device DRAM peak bandwidth.	Architecture description fed to compiler
ρ_H	Device architecture & design-variant	Scaling factor, host-device bandwidth	*Experiments* 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

University of Glasgow | School of Computing Science

Performance Estimates

Parameters derived from IR description of Kernel

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	*Experiments* with different data-patterns on the target node.
G_{PB}	Node architecture	The device DRAM peak bandwidth.	Architecture description fed to compiler
ρ_H	Device architecture & design-variant	Scaling factor, host-device bandwidth	*Experiments* 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

University of Glasgow | School of Computing Science

Performance Estimates
The Expressions

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

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

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

University of Glasgow | School of Computing Science

Performance Estimates
The Expressions

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

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

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

University of Glasgow | School of Computing Science

Performance Estimates
The Expressions

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

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

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

University of Glasgow | School of Computing Science

Performance Estimates
The Expressions

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

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

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

University of Glasgow | School of Computing Science

Performance Estimates
The Expressions

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

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

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

University of Glasgow | School of Computing Science

Performance Estimates
The Expressions

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

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

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

University of Glasgow | School of Computing Science

Performance Estimates
Experimental Results (Type C)

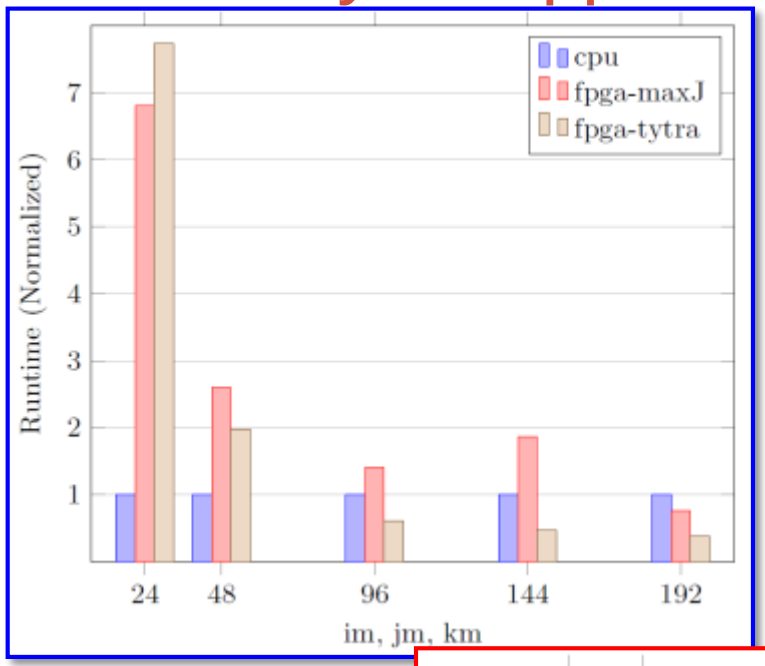
Resource	C2(E)	C2(A)	C1(E)	C1(A)
ALUTs	528	546	5764	5837
REGs	534	575	4504	4892
BRAM(bits)	5418	5400	11304	11250
DSPs	0	0	0	0
Cycles/Kernel	292	308	180	185
EWGT	57K	43K	92K	72K

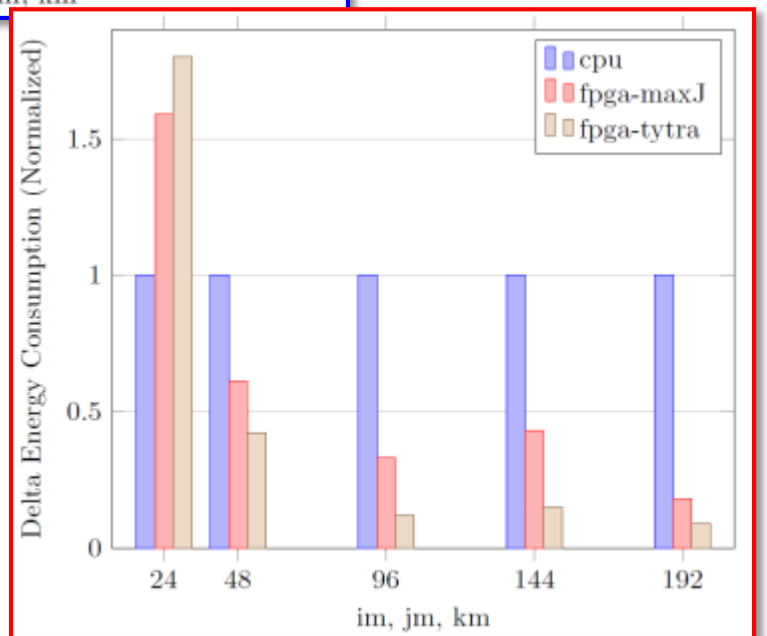
Estimated (E) vs actual (A) cost and throughput for C2 and C1 configurations of a successive over-relaxation kernel.

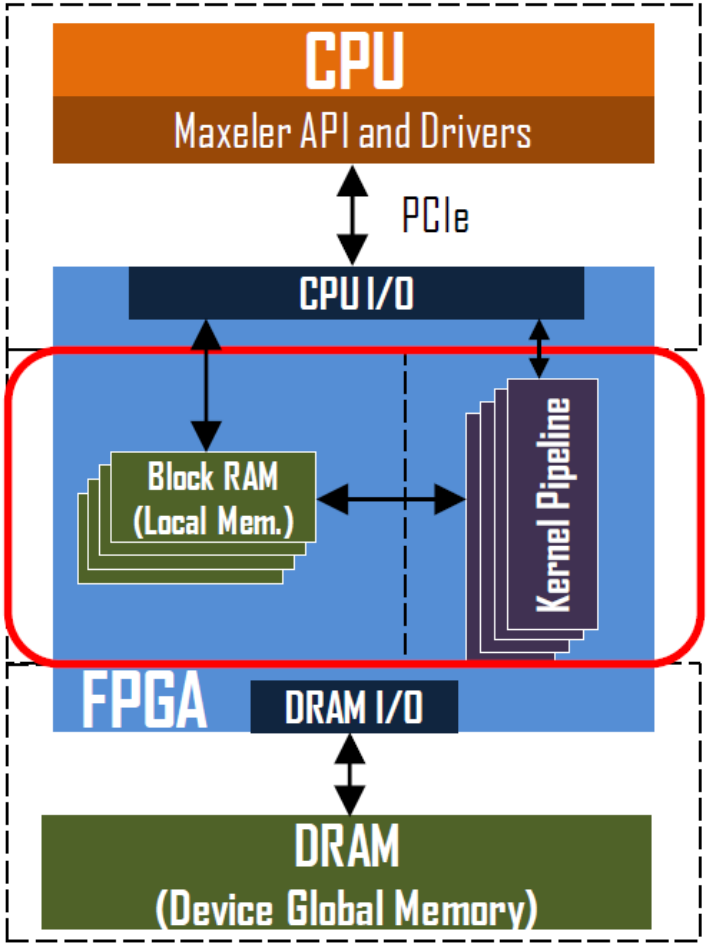
*[Note that the **cycles/kernel** are estimated very accurately, but the **Effective Workgroup Throughput (EWGT)** is off because of inaccuracy of frequency estimate for the FPGA]*

Does the TyTra Approach Work?

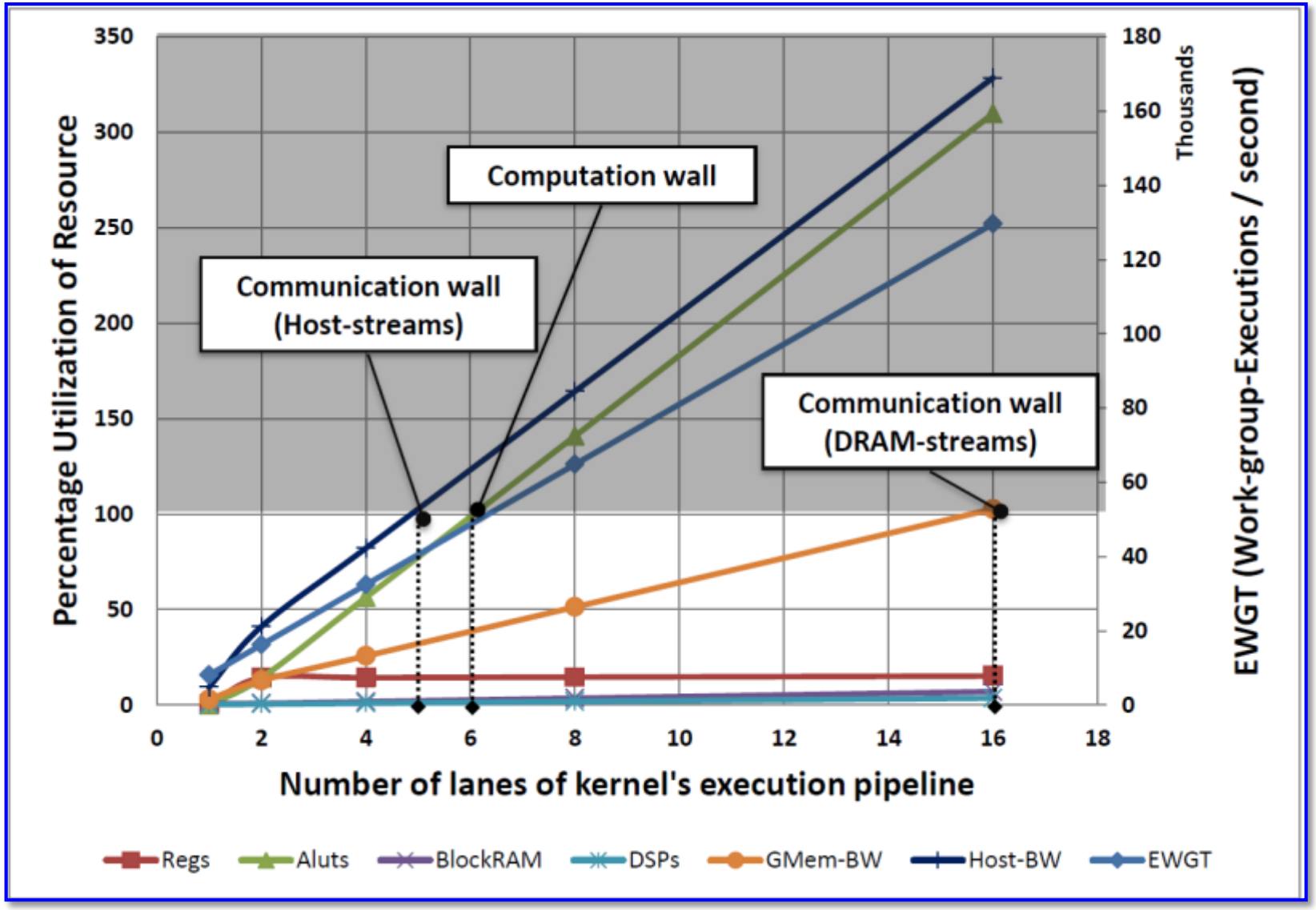
University of Glasgow | School of Computing Science







Design-Space Exploration?



CONCLUSION

The route to Automated Design Space Exploration on FPGAs for HPC Applications

University of Glasgow | School of Computing Science

- The larger aim is to create a turn-key compiler for:

  Legacy scientific code → Heterogeneous HPC Platform

  - Current focus is on FPGAs, and on using a Functional Language design entry

- Our main contributions are:
  - Type transformations to create design-variants,
  - New Intermediate Language, and
  - FPGA Cost model

- Our FPGA Cost Model
  - Works on the TyTra-UIR, is light-weight, accurate (enough), and allows us to evaluate design-variants

Using safe transformations on a functional language paradigm and a light-weight cost-model to brings us closer to a turn-key HPC compiler for legacy code


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.

EXTRAS

Parallel Approaches

- What we do is very similar to:
  - Loop optimizations to accelerate a scientific application
  - Using skeletons to create a high-level abstraction for parallel programming
  - Tools that automatically explore design-space

Our Approach to a Light-Weight Cost Model

University of Glasgow | School of Computing Science

- An IR sufficiently low-level to expose the parameters needed for the
  - The TyTra-IR has sufficient structural information to associate it directly with resources on an FPGA
- Because TyTra-IR is a customized language, we can ensure that:
  - All legal instructions (and structures) have a cost associated with them
  - As long as the front-end compiler can target a HLL on the TyTra-IR, we can cost HL program variants
  - Costing resources on specific FPGA devices, and estimating memory bandwidth for various patterns on the target node, requires some empirical data.
    - We are working on creating a set of standardized experiments that

We are not there yet, but in principle, our approach entirely eliminates all these limitations.

University of Glasgow | School of Computing Science

Quite a few avenues…

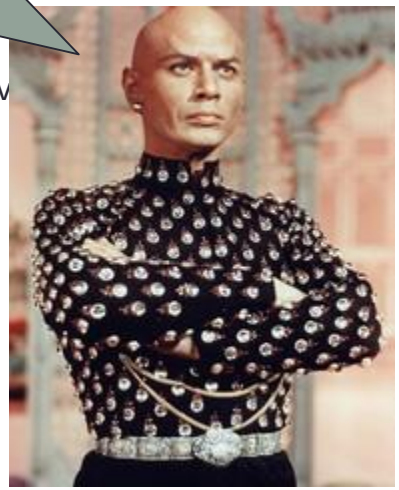
- Experiment with more kernels, their program-variants, estimated vs actual costs, (correct) code-generation. Use (CHStone) benchmarks.
- Computation-aware caches, optimized for halo-based scientific computations
- Integrate with Altera-OpenCL platform for host-device communication
- Back-end optimizations, LLVM passes, LLVM → TyTra-IR translation
- Route to TyTra-IR from SAC
- Integrate Tytra-FPGA flow with SAC→GPU(OpenCL flow) for heterogeneous targets
- Use of Multi-party Session Types to ensure correctness of transformations
  - Even code-generation for clusters?
- Abstract descriptions of target hardware
- SystemC-TLM model to profile application and high-level partitioning in a heterogeneous environment

University of Glasgow | School of Computing Science

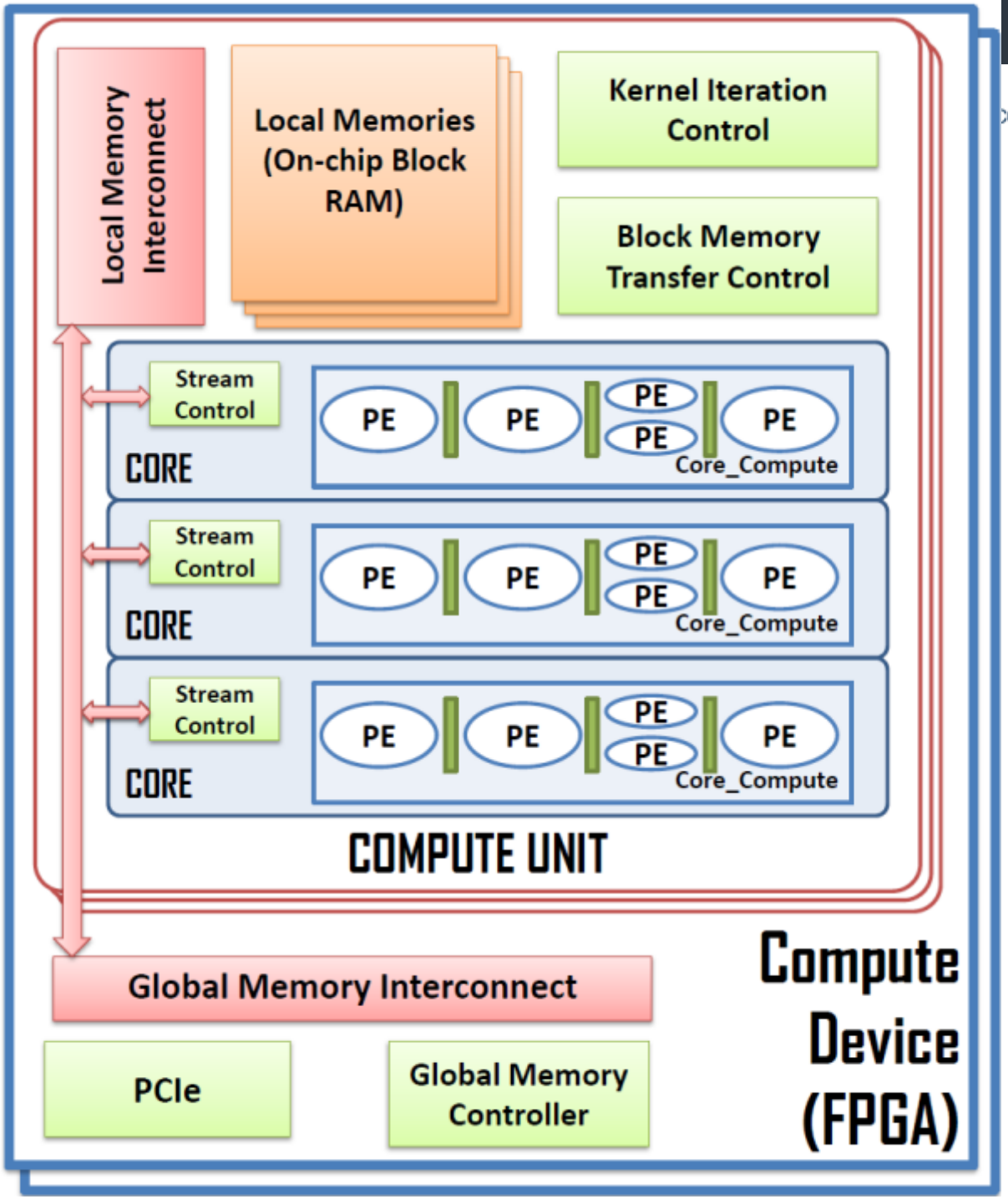
Quite a few avenues…

- Experiment with more kernels, their program-variants, estimated vs actual costs, (correct) code-generation. Use (CHStone) benchmarks.
- Computation-aware caches, optimized for halo-based scientific computations
- Integrate with Altera-OpenCL platform for host-device communication
- Back-end optimizations, LLVM passes, LLVM → TyTra-IR translation
- Route to TyTra-IR from SAC
- Integrate Tytra-FPGA flow with SAC→GPU(
- Use of Multi-party Session Types to ensure
  - Even code-generation for clusters?
- Abstract descriptions of target hardware
- SystemC-TLM model to profile application and high-level partitioning in a heterogeneous env

etcetera, etcetera, etcetera



The platform model for TyTra (FPGA)



The Manage-IR; *Memory Objects*

University of Glasgow | School of Computing Science

```
@mem_a = addrSpace(3) <NLinear x ui18>,
          !"hmap" , !"NULL"  ,
          !"init" , !"a.dat" ,
          !"readPorts"  , !1 ,
          !"writePorts" , !1
```

TyTra-IR	OpenCL view	LLVM-SPIR View	Hardware (FPGA)
Cmem Constant Memory	Constant Memory	3: Constant	
Imem Instruction Memory	Constant Memory		DistRAM / BRAM
Pipemem Pipeline registers			DistRAM
Pmem Private Memory (Data Mem for Instruc' Proc')	Private Memory	0: Private	DistRAM
Cachemem Data (and Constant) Cache			DistRAM / BRAM
Lmem Local (shared) memory	Local Memory	4: Local	M20K (BRAM) or Dist RAM
Gmem Global memory	Global Memory	1: Global	On-board DRAM
Hmem Host memory	Host Memory		Host communication

The Manage-IR; *Stream Objects*

University of Glasgow | School of Computing Science

```
@strobj_a = addrSpace(10),
            !"dir"      , !"in"           ,
            !"memConn"  , !"@mem_a"       ,
            !"length"   , !NLinear        ,
            !"startAdd" , !0              ,
            !"signal"   , !"yes"          ,
            !"nDim"     , !2              ,
            !"dimShape" , ![NDim1, NDim2] ,
            !"startInd" , ![0, 0]         ,
            !"majorDim" , !1
```

- Can have a 1-1 or Many-1 relation with memory objects
- Have a 1-1 relation with arguments to `pipe` functions (i.e. port connections to `compute-cores`)

The Manage-IR; `repeat` blocks

- Repeatedly call a kernel without referring back to the host (*outer*-loop)
- May involve block memory transfers between iterations

```
repeat k=1:NKIter
{
  call @main()
  @mem_a =  @mem_y, !"tir.lmem.copy",
            !"srcStartAddr" , !0,
            !"destStartAddr", !0,
            !"trSizeWords"  , !NLinear
}
```

University of Glasgow | School of Computing Science

The Manage-IR; stream windows

- Access offsets in streams
- Use on-chip buffers for storing data read from memory

```
%a_e = ui18 @main.a, !tir.stream.offset, !+1
%a_w = ui18 @main.a, !tir.stream.offset, !-1
%a_n = ui18 @main.a, !tir.stream.offset, !-NDim1
%a_s = ui18 @main.a, !tir.stream.offset, !+NDim1
```

University of Glasgow | School of Computing Science

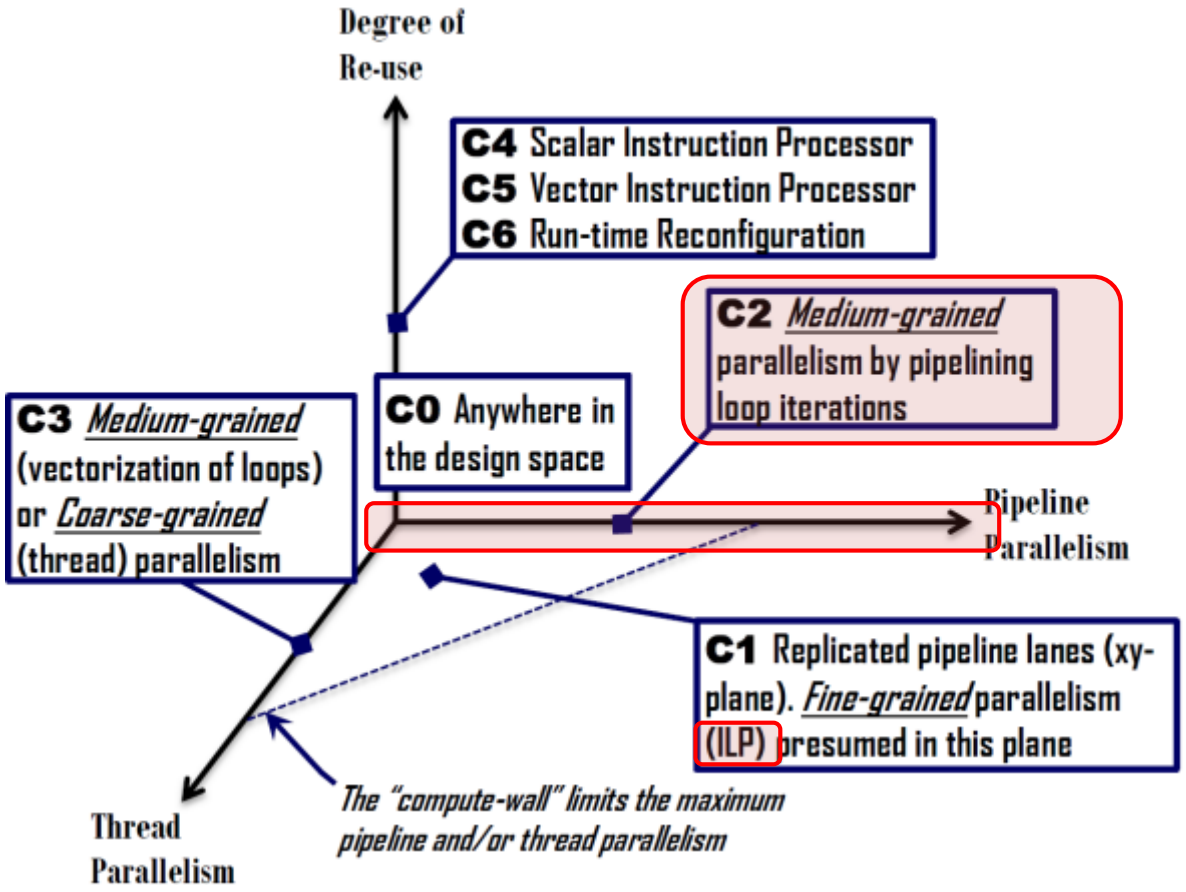
The Compute-IR

- Structural semantics
  - `@function_name (…args…) par`
  - `@function_name (…args…) seq`
  - `@function_name (…args…) pipe`
  - `@function_name (…args…) comb`
    - Nesting these functions gives us the expressiveness to explore various parallelism configurations
- Streaming ports
- Counters and nested counters
- SSA data-path instructions

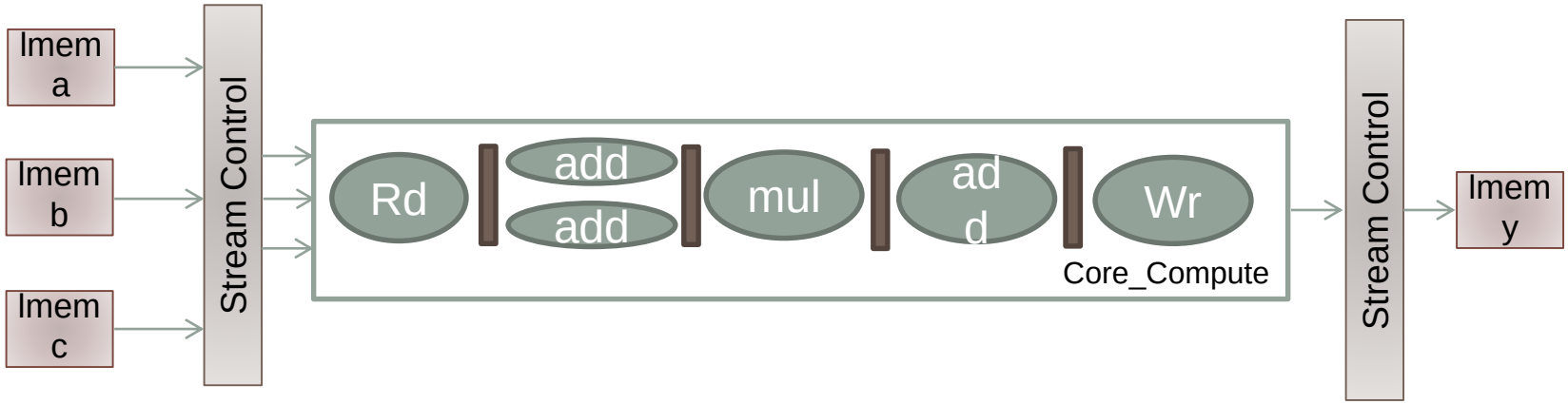
Example: Simple Vector Operation
The Kernel

```
! ----------- the kernel ----------------
! A dummy, stupid kernel with minimal operations allowing us to
! exlore vectors/streaming, pipelining, multi-threading,
! Y = K (constant) + {(A + B) * (C + C)}
! where A, B, C and Y are vectors

! using verbose way of vector ops to expose loop
do n = 1,ntot
        y(n) =  K + ( (a(n)+b(n)) * (c(n)+c(n)) )
end do
```

Version 1 – Single Pipeline (C2)

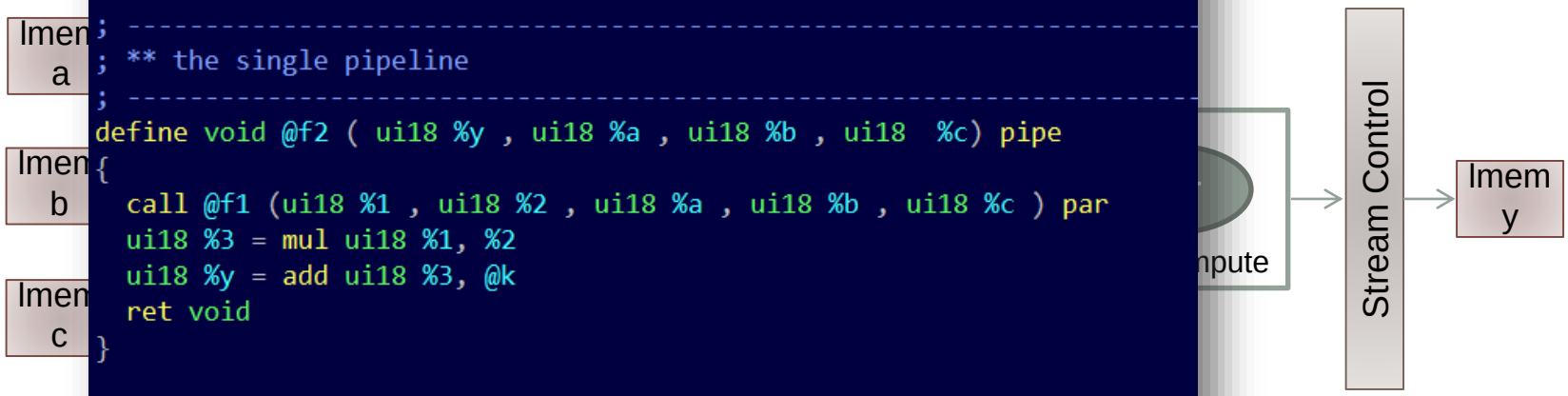


Version 1 – Single Pipeline (C2)



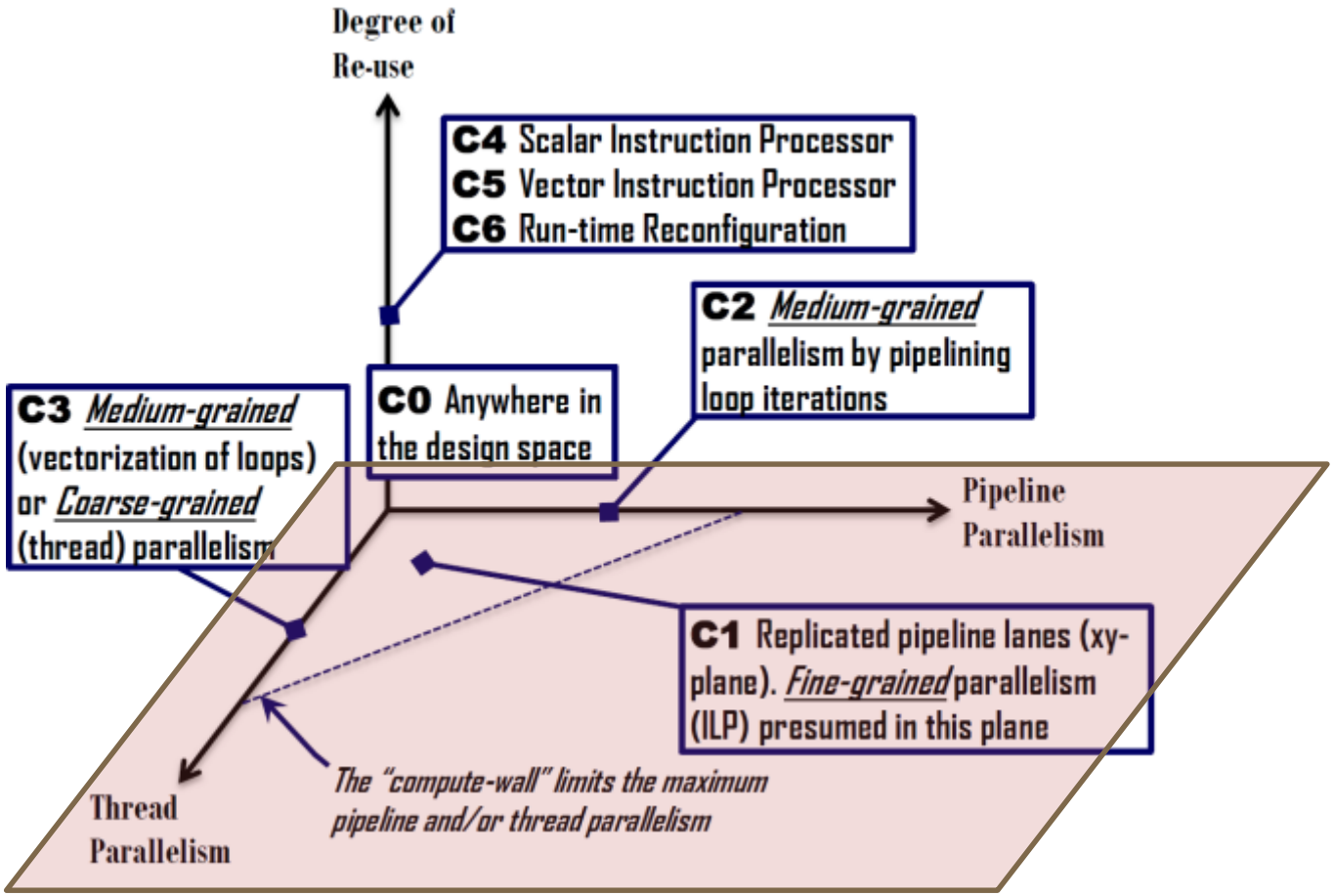
Vers

```
; ---------------------------------------------------------------------
; a par block exposing ILP in the pipepline
; ---------------------------------------------------------------------
define void @f1( ui18 %1 , ui18 %2 , ui18 %a , ui18 %b , ui18 %c ) par
{
  ui18 %1  = add ui18 %a, %b
  ui18 %2  = add ui18 %c, %c
  ret void
}

; ---------------------------------------------------------------------
; ** the single pipeline
; ---------------------------------------------------------------------
define void @f2 ( ui18 %y , ui18 %a , ui18 %b , ui18  %c) pipe
{
  call @f1 (ui18 %1 , ui18 %2 , ui18 %a , ui18 %b , ui18 %c ) par
  ui18 %3 = mul ui18 %1, %2
  ui18 %y = add ui18 %3, @k
  ret void
}

; ---------------------------------------------------------------------
; ** Main **
; ---------------------------------------------------------------------
define void @main ()
{
entry:
  call @f2( ui18 @y , ui18 @a , ui18 @b , ui18  @c) pipe
  ret void
}
```

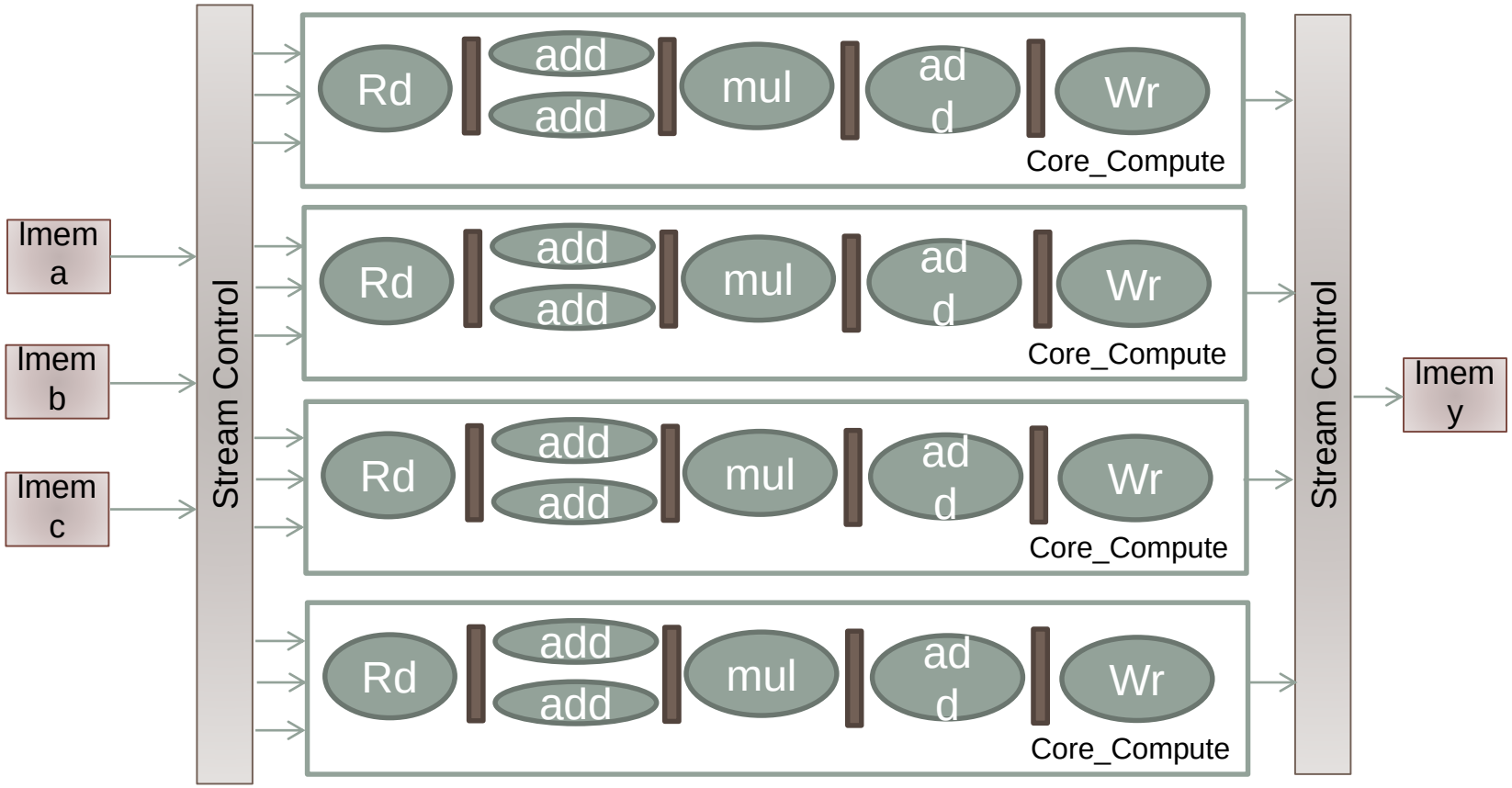


The parser can also automatically find ILP and schedule in an ASAP fashion

Version 2 – 4 Parallel Pipelines (C1)

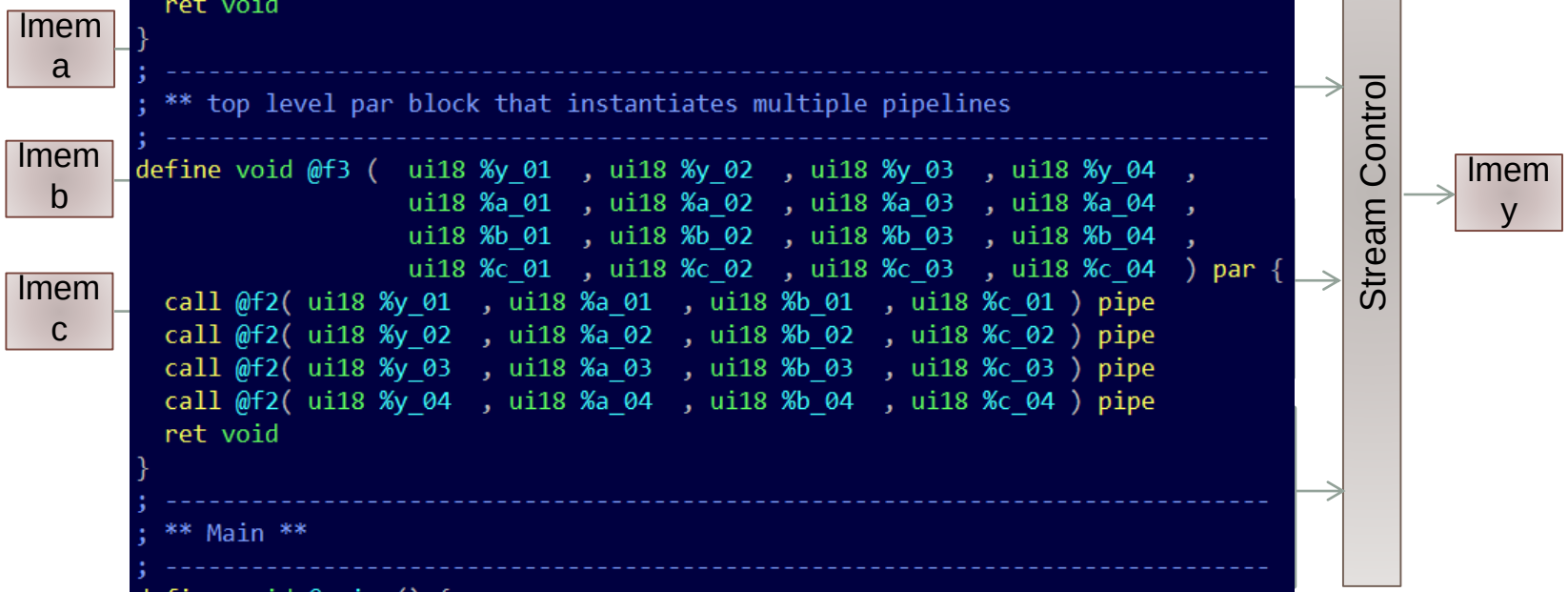


Version 2 – 4 Parallel Pipelines

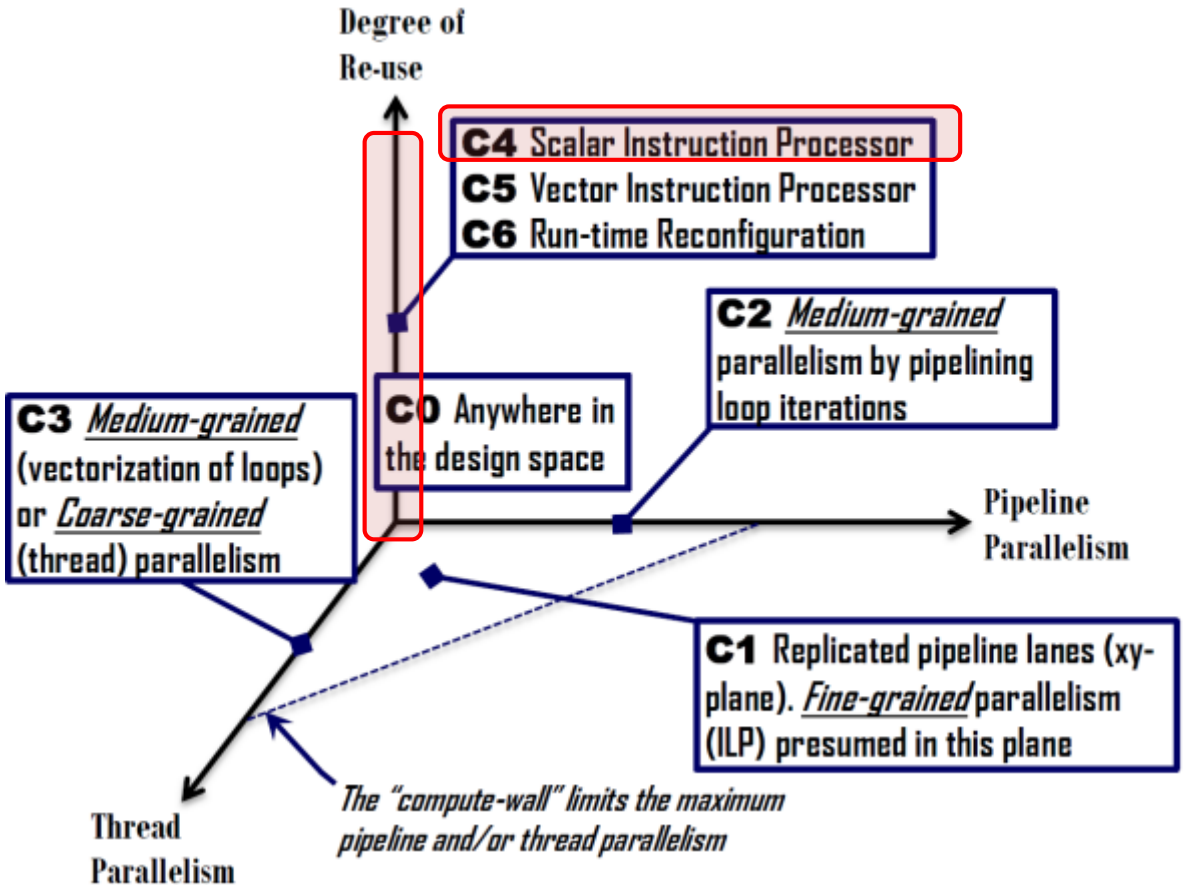


Vers

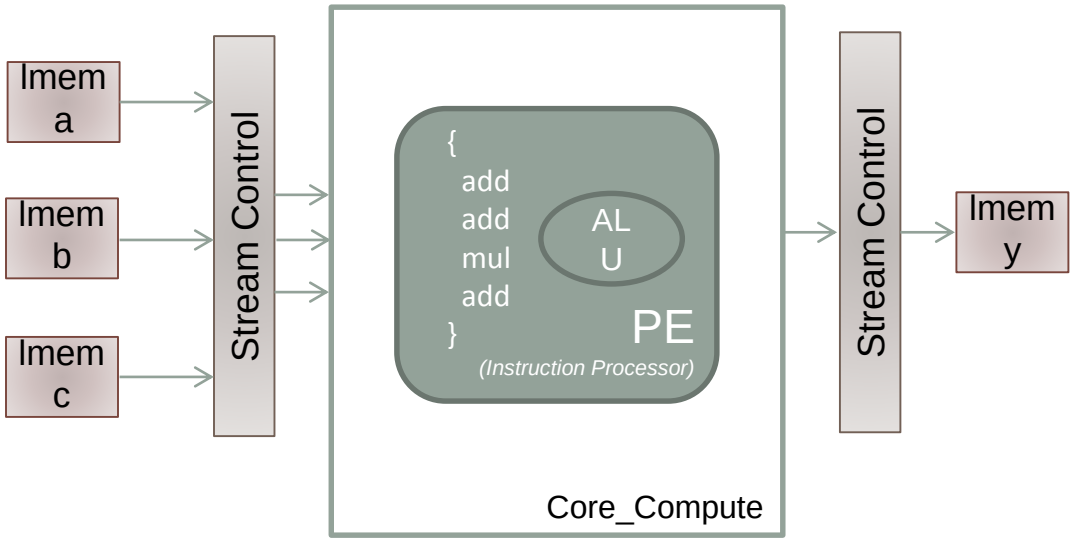
```
; ---------------------------------------------------------------------------
; a par block template for exposing ILP in the pipepline
; ---------------------------------------------------------------------------
define void @f1 (  ui18 %1, ui18 %2, ui18 %a, ui18 %b, ui18 %c ) par {
  ui18 %1  = add ui18 %a, %b
  ui18 %2  = add ui18 %c, %c
  ret void
}
; ---------------------------------------------------------------------------
; ** the single pipeline template function
; ---------------------------------------------------------------------------
define void @f2 ( ui18 %y , ui18 %a , ui18 %b , ui18 %c  ) pipe {
  call @f1 ( ui18 %1 , ui18 %2 , ui18 %a , ui18 %b , ui18 %c ) par
  ui18 %3 = mul ui18 %1, %2
  ui18 %y = add ui18 %3, @k
  ret void
}
; ---------------------------------------------------------------------------
; ** top level par block that instantiates multiple pipelines
; ---------------------------------------------------------------------------
define void @f3 (  ui18 %y_01  , ui18 %y_02  , ui18 %y_03  , ui18 %y_04  ,
                   ui18 %a_01  , ui18 %a_02  , ui18 %a_03  , ui18 %a_04  ,
                   ui18 %b_01  , ui18 %b_02  , ui18 %b_03  , ui18 %b_04  ,
                   ui18 %c_01  , ui18 %c_02  , ui18 %c_03  , ui18 %c_04  ) par {
  call @f2( ui18 %y_01  , ui18 %a_01  , ui18 %b_01  , ui18 %c_01 ) pipe
  call @f2( ui18 %y_02  , ui18 %a_02  , ui18 %b_02  , ui18 %c_02 ) pipe
  call @f2( ui18 %y_03  , ui18 %a_03  , ui18 %b_03  , ui18 %c_03 ) pipe
  call @f2( ui18 %y_04  , ui18 %a_04  , ui18 %b_04  , ui18 %c_04 ) pipe
  ret void
}
; ---------------------------------------------------------------------------
; ** Main **
; ---------------------------------------------------------------------------
define void @main () {
  call @f3  ( ui18 @y_01  , ui18 @y_02  , ui18 @y_03  , ui18 @y_04  ,
              ui18 @a_01  , ui18 @a_02  , ui18 @a_03  , ui18 @a_04  ,
              ui18 @b_01  , ui18 @b_02  , ui18 @b_03  , ui18 @b_04  ,
              ui18 @c_01  , ui18 @c_02  , ui18 @c_03  , ui18 @c_04  ) par
}
```



Version 3 – Scalar Instruction Processor (C4)



Version 3 – Scalar Instruction Processor (C4)

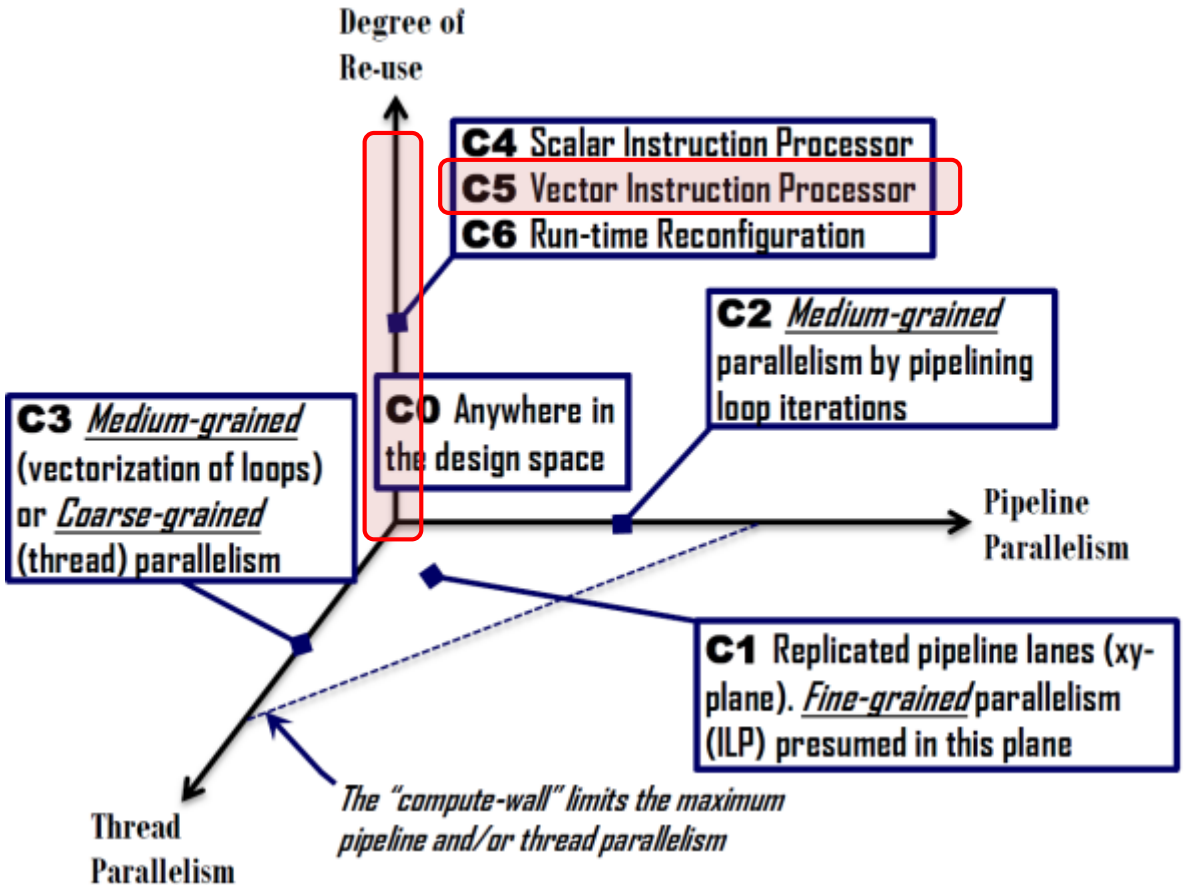


The ALU would be customized for the instructions mapped to this PE at compile-time

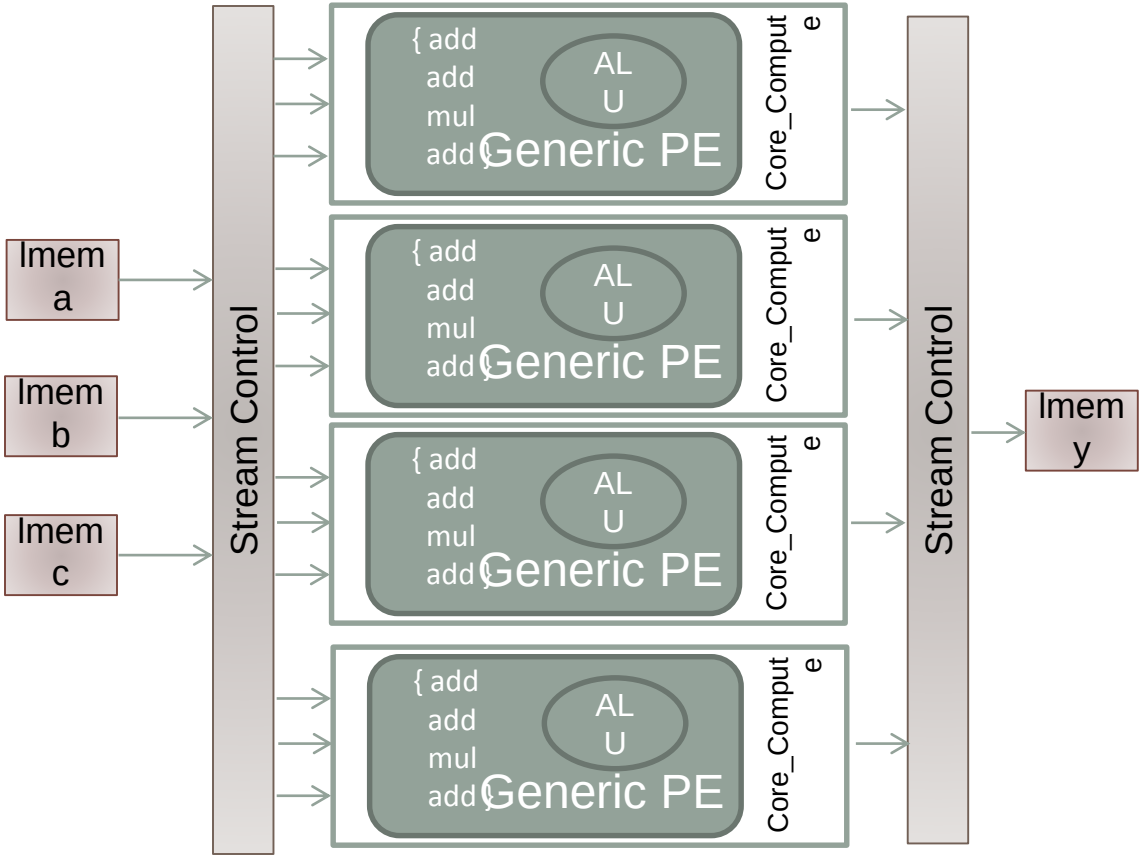
Version 3 – Single Sequential Processor

```
; ------------------------------------------------------------
; ** top level seq block
; ------------------------------------------------------------
define void @f1 (  ui18 %y ,ui18 %a , ui18 %b , ui18  %c) seq {
  ui18 %1 = add ui18 %a, %b
  ui18 %2 = add ui18 %c, %c
  ui18 %3 = mul ui18 %1, %2
  ui18 %y = add ui18 %3, @k
  ret void
}

; ------------------------------------------------------------
; ** Main **
; ------------------------------------------------------------
define void @main () {
  call @f1(  ui18 @y , ui18 @a , ui18 @b , ui18  @c) seq
  ret void
}
```

Version 4 – Multiple Processors / Vectorization (C5)



Version 4 – Multiple Processors / Vectorization (C5)



Version 4 – Multiple Sequential Proc

```
; ---------------------------------------------------------------------------
; ** the sequential block template
; ---------------------------------------------------------------------------
define void @f1 (  ui18 %y , ui18 %a , ui18 %b , ui18 %c  )  seq {
  ui18 %1 = add ui18 %a, %b
  ui18 %2 = add ui18 %c, %c
  ui18 %3 = mul ui18 %1, %2
  ui18 %y = add ui18 %3, @k
  ret void
; ---------------------------------------------------------------------------
; ** top level par block that instantiates sequential blocls (vector lanes)
; ---------------------------------------------------------------------------
define void @f2 (  ui18 %y_01  , ui18 %y_02  , ui18 %y_03  , ui18 %y_04  ,
                   ui18 %a_01  , ui18 %a_02  , ui18 %a_03  , ui18 %a_04  ,
                   ui18 %b_01  , ui18 %b_02  , ui18 %b_03  , ui18 %b_04  ,
                   ui18 %c_01  , ui18 %c_02  , ui18 %c_03  , ui18 %c_04  ) par {
  call @f1( ui18 %y_01  , ui18 %a_01  , ui18 %b_01  , ui18 %c_01 ) seq
  call @f1( ui18 %y_02  , ui18 %a_02  , ui18 %b_02  , ui18 %c_02 ) seq
  call @f1( ui18 %y_03  , ui18 %a_03  , ui18 %b_03  , ui18 %c_03 ) seq
  call @f1( ui18 %y_04  , ui18 %a_04  , ui18 %b_04  , ui18 %c_04 ) seq
  ret void
}
; ---------------------------------------------------------------------------
; ** Main **
; ---------------------------------------------------------------------------
define void @main () {
  call @f2 ( ui18 @y_01  , ui18 @y_02  , ui18 @y_03  , ui18 @y_04  ,
             ui18 @a_01  , ui18 @a_02  , ui18 @a_03  , ui18 @a_04  ,
             ui18 @b_01  , ui18 @b_02  , ui18 @b_03  , ui18 @b_04  ,
             ui18 @c_01  , ui18 @c_02  , ui18 @c_03  , ui18 @c_04  ) par
}
```

Note the continued use of stream abstractions even through the PEs are Instruction Processors now