Throughput Optimization Techniques for Heterogeneous Architectures

13th December 2023, Minatech

Nicolas Derumigny,
Colorado State University
Inria

Advisors:

Louis-Noël Pouchet,
Colorado State University

Fabrice Rastello,
Inria Grenoble

Colorado State University

Inria

Topic of the Ph.D

How to make the “best” hardware / software combination?

- **Inria Grenoble, France**
- **“How do I optimize for an architecture?”**
  - CPU
  - **Fixed** architecture
  - **Variable** application
  - **Performance** model
  - Automated resource **characterization**

Topic of the Ph.D

How to make the "best" hardware / software combination?

- Inria Grenoble, France
- "How do I optimize for an **architecture**?"
  - CPU
  - **Fixed** architecture
  - **Variable** application
  - **Performance** model
  - Automated resource **characterization**

- Colorado State University, USA
- "How do I optimize for an **application**?"
  - FPGA / ASIC
  - **Variable** architecture
  - **Fixed** application
  - **Resource** model
  - ≈ Automated resource **generation**

Outline

1) Background

2) Optimising for a fixed architecture: PALMED

3) Optimising for fixed applications: GA

Outline

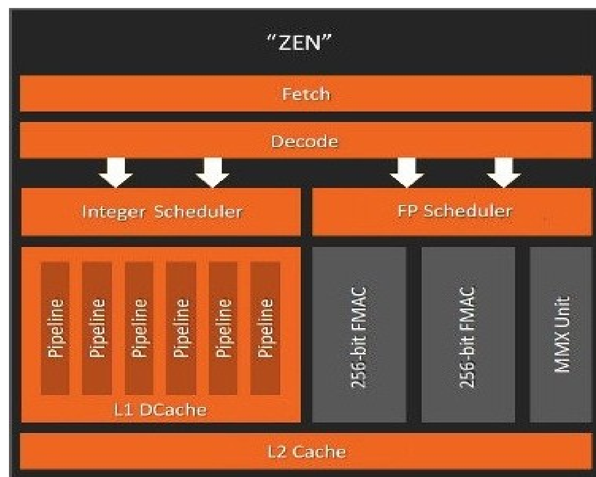
1) Background

2) Optimising for a fixed architecture: PALMED

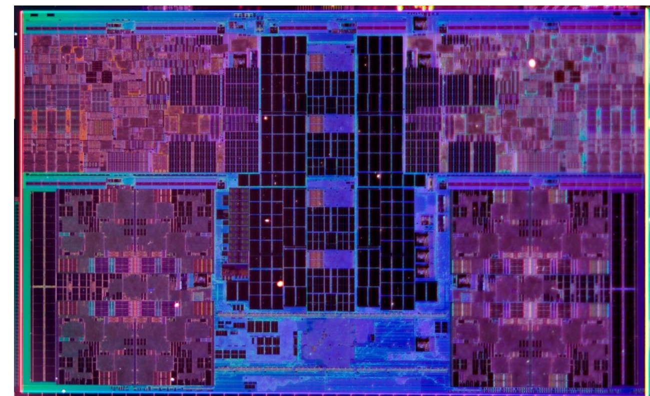
3) Optimising for fixed applications: GA

Hardware architectures: accelerators, instructions

- **CPU:**
  - **Fixed** (unknown) topology
  - A few **high performance** cores
  - **Variable** instructions



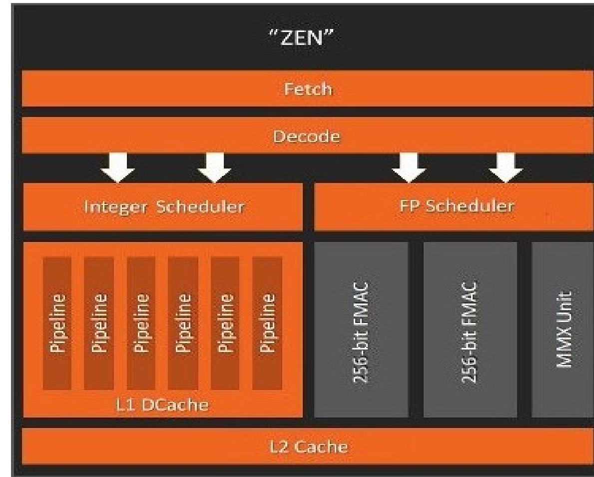
Zen core (Credit: AMD)



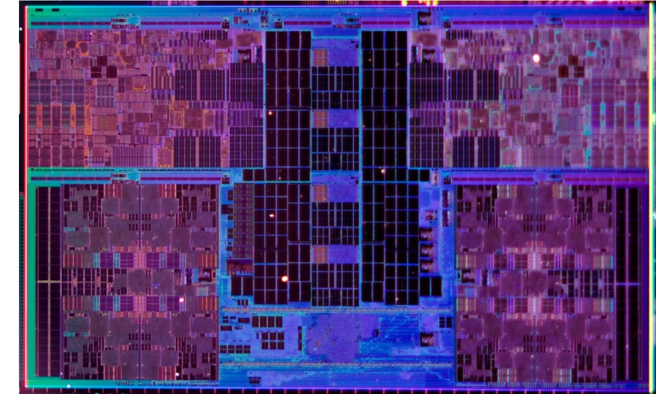
Meteor Lake die shot

Hardware architectures: accelerators, instructions

- **CPU:**
  - Fixed (unknown) topology
  - A few high performance cores
  - Variable instructions

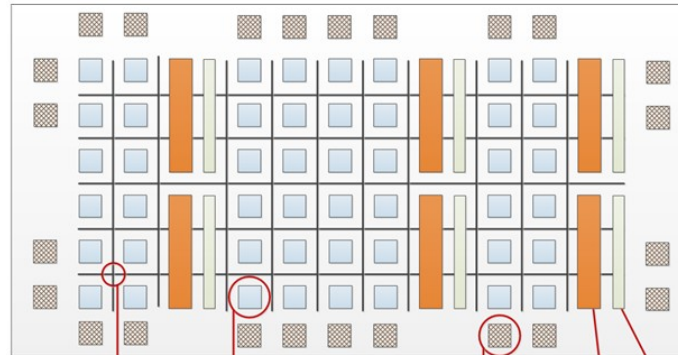


Zen core (Credit: AMD)

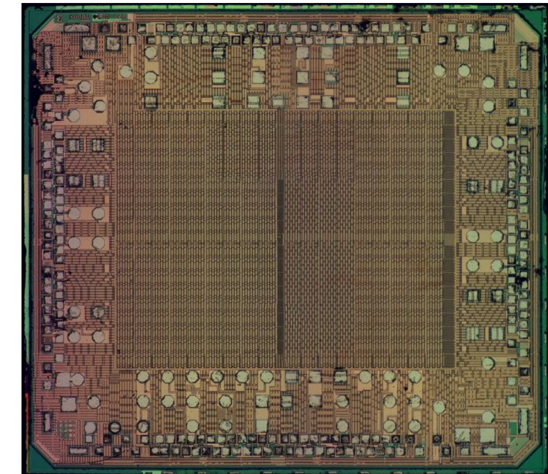


Meteor Lake die shot

- **FPGA/ASIC**
  - Dedicated accelerator logic
  - Variable topology
    - Decided by the designer
  - May have variable instructions

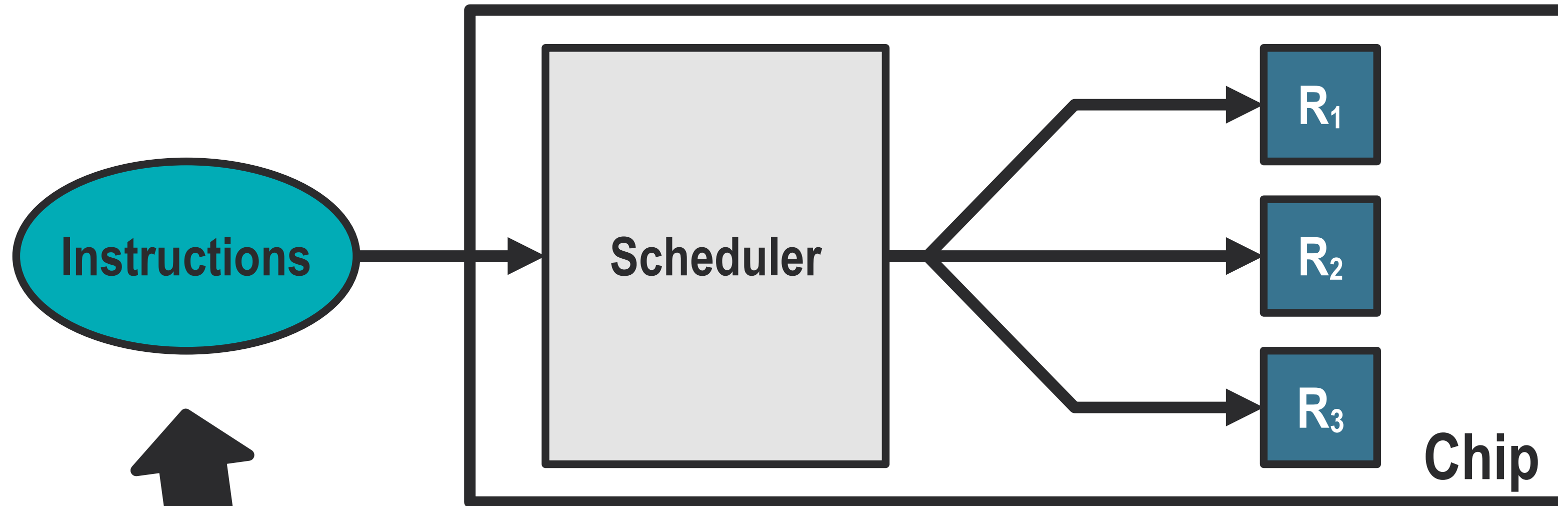


(Credit: Xilinx / AMD)



XC3S50A die shot
(Credit: Andrew Zonenberg)

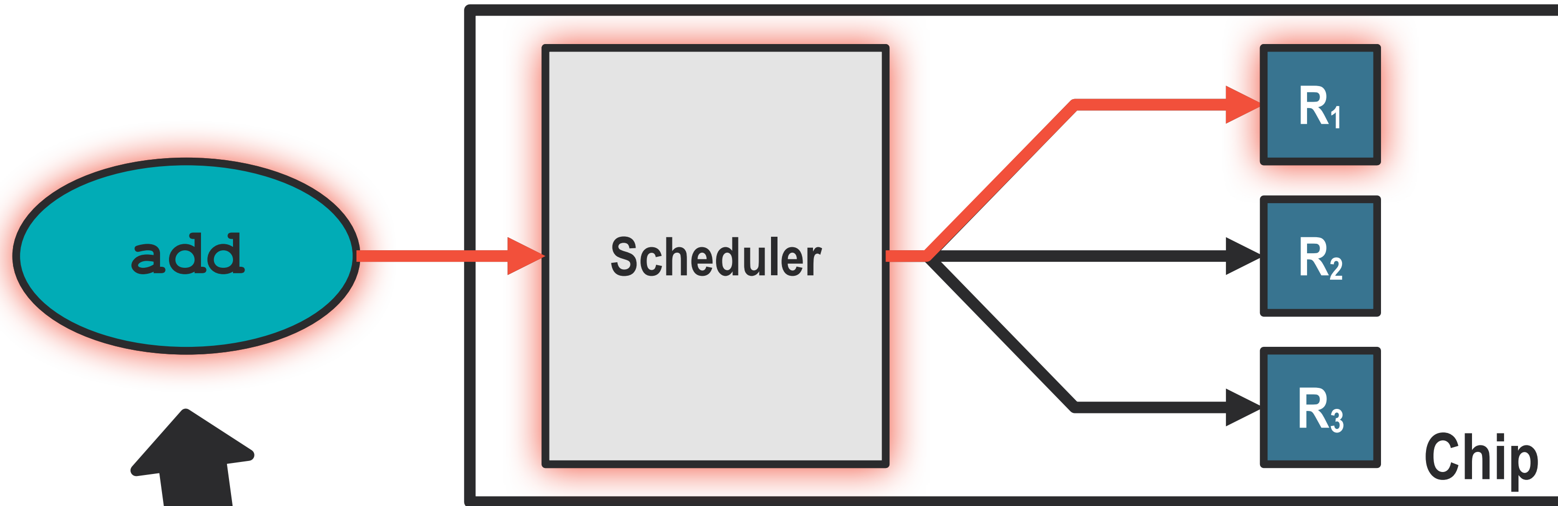
Mapping & Resources



input program

Mapping	
Instruction	Resource
`add`	R_1
`sub`	R_1
`mul`	R_1
`load`	R_2 or R_3

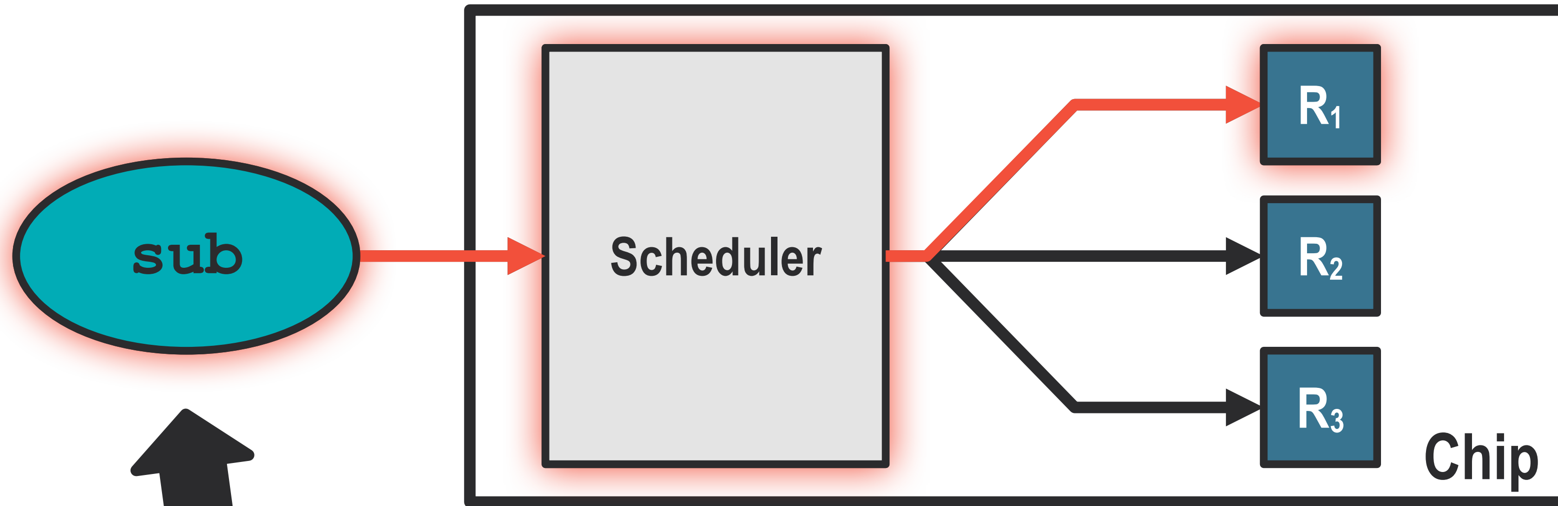
Mapping & Resources



input program

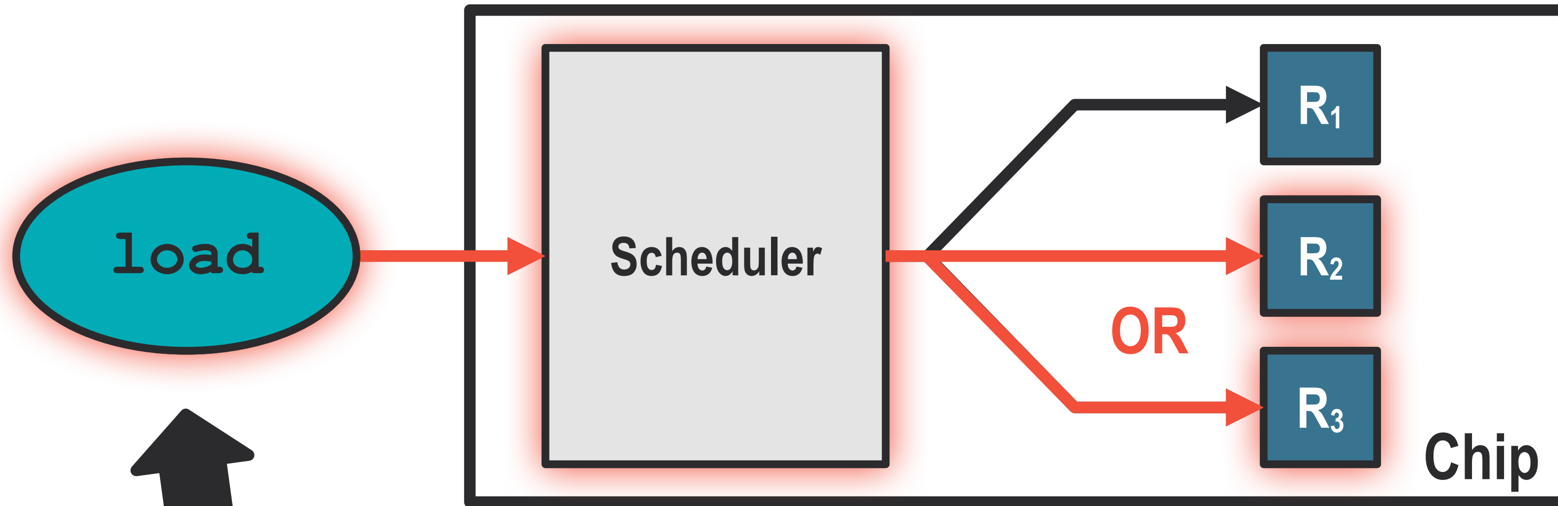
Mapping	
Instruction	Resource
add	R_1
sub	R_1
mul	R_1
load	R_2 or R_3

Mapping & Resources



Mapping	
Instruction	Resource
add	R_1
sub	R_1
mul	R_1
load	R_2 or R_3

Mapping & Resources

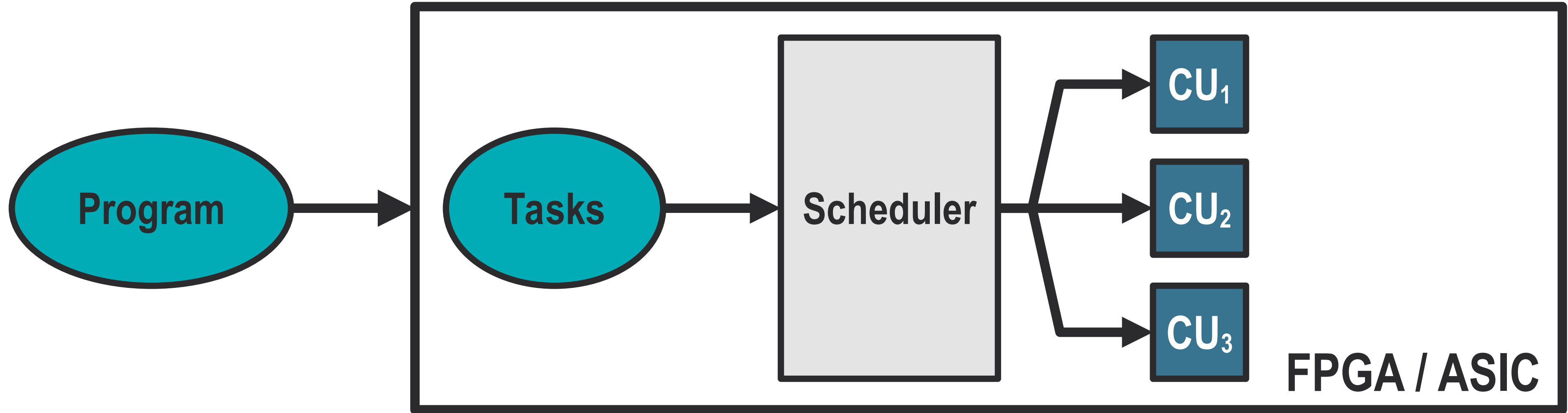



```
add
sub
load
```

input program

Mapping	
Instruction	**Resource**
`add`	R_1
`sub`	R_1
`mul`	R_1
`load`	R_2 or R_3

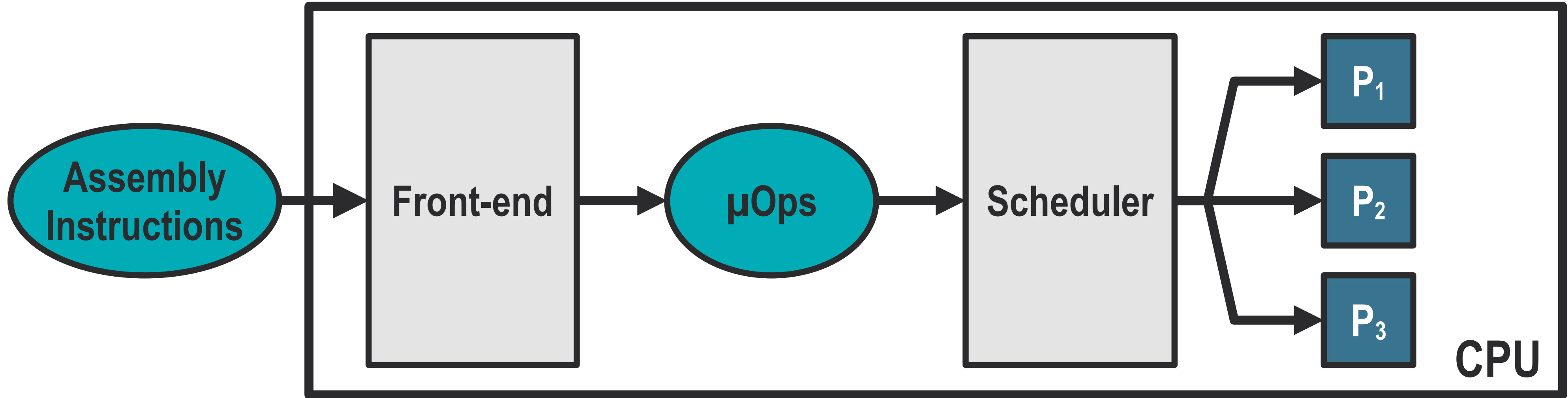
FPGA designs



- **Compute Units**
  - **Specialized** hardware units
  - Configurable number
  - Configurable capabilities
- **Interconnect**
  - **Fixed**, configurable routes
  - Between **CU**
  - Between **storage units**
  - **Fixed** Schedule

Mapping	
Program	Resource
Known, Fixed	**Unknown, Variable**

(Bits of) CPU architecture



- **Front End:**
  - Decoding
    - **Instructions -> µOPs**
  - Branch prediction
  - Various caches
- **Back-end:**
  - Execution pipeline
  - Functional units
    - **Execution ports**
  - Instruction life cycle:
    1) **Issued**
    2) **Scheded**
    3) **Retired**

Mapping	
Instruction	Resource
Known, Variable	**Unknown, Fixed**

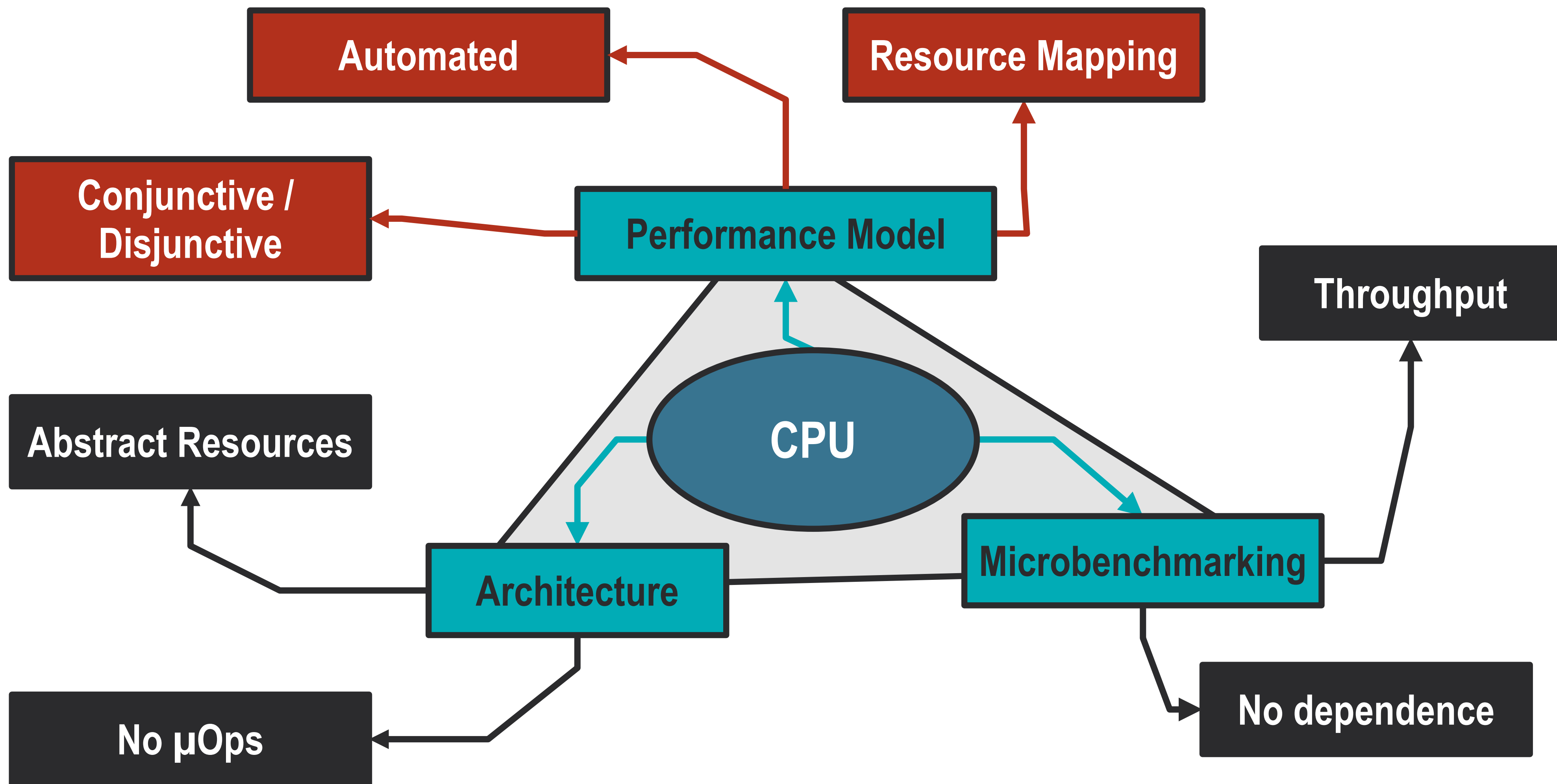
Outline

1) Background

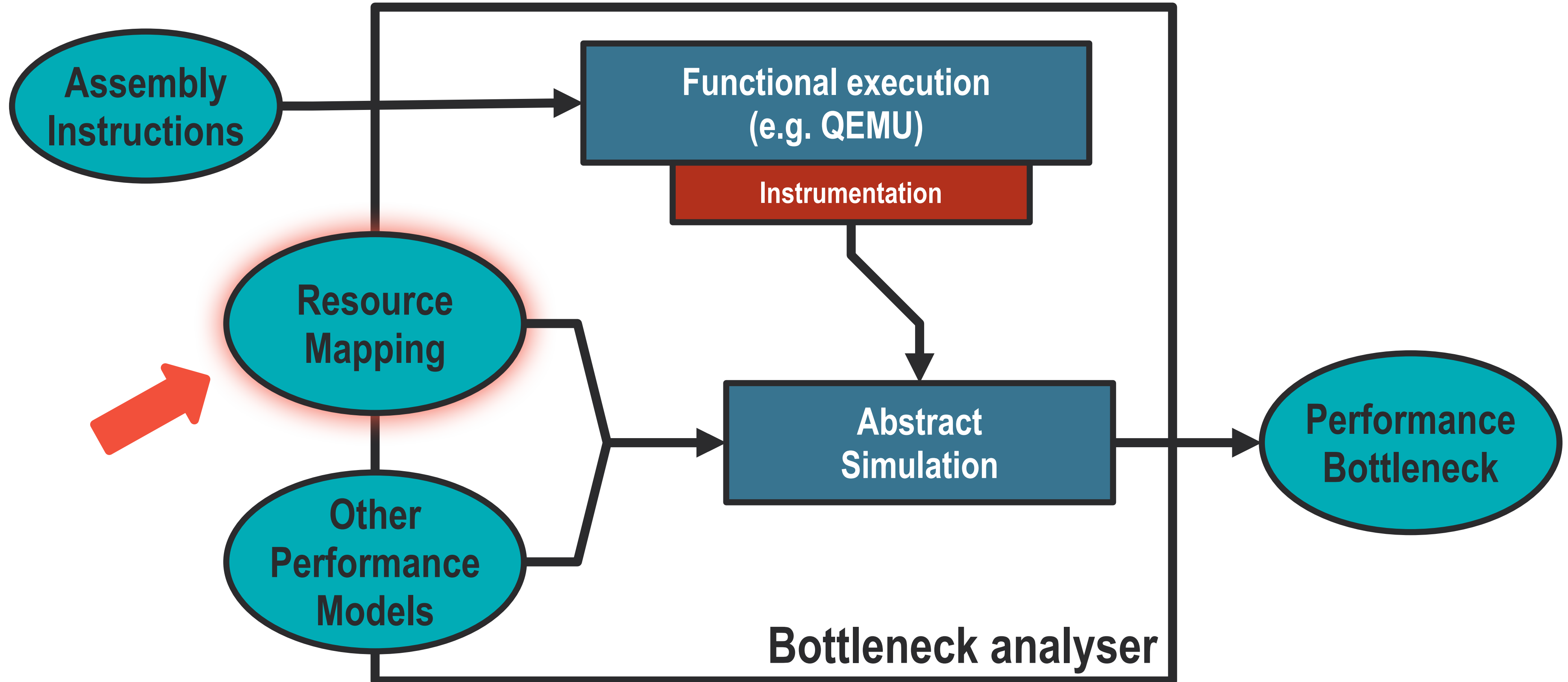
2) Optimising for a fixed architecture: PALMED

3) Optimising for fixed applications: FPGAs

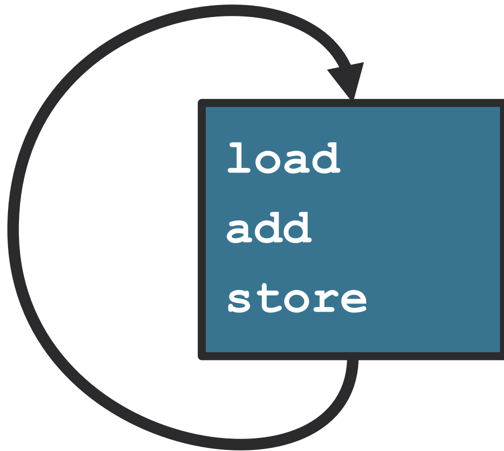
CPUs



How to build a CPU bottleneck analyser?



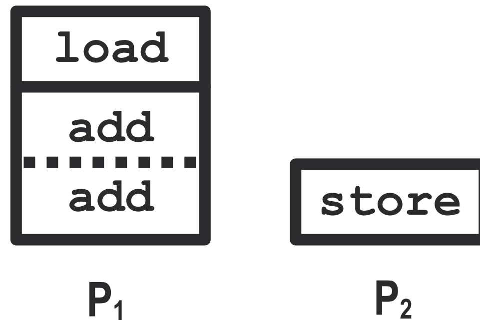
Microkernel & Execution Time



Repeated a high number of times

Ports	
P_1	P_2
1	
2	
	1

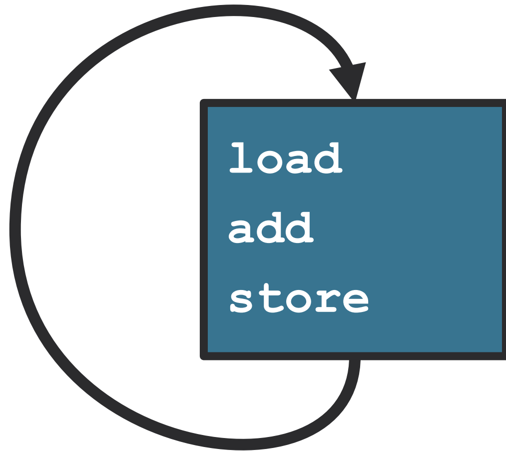
Mapping	
Instruction	Port
`load`	P_1 or P_2
`add`	$2*P_1$
`store`	P_2



- **Average execution time (▶ throughput):**

  3 cycles

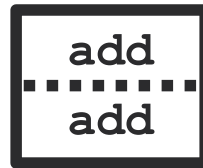
Microkernel & Execution Time



Repeated a high number of times

Ports	
P_1	P_2
	1
2	
	1

Mapping	
Instruction	Port
`load`	P_1 or P_2
`add`	$2{*}P_1$
`store`	P_2



P_1 **P_2**

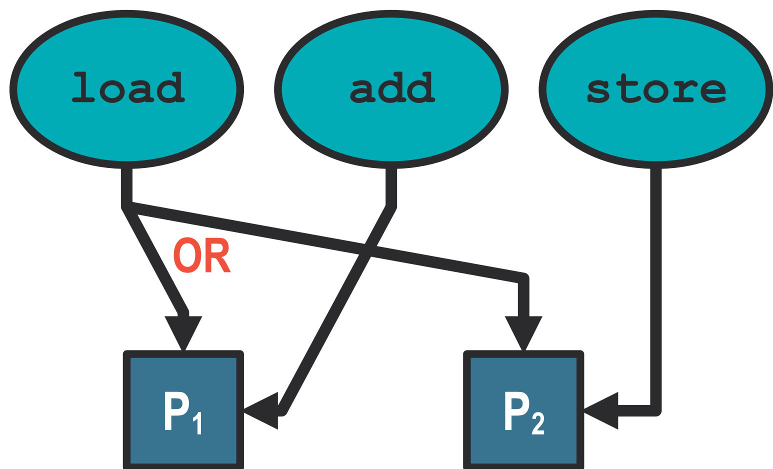
- **Average optimal execution time (▶ throughput):**

  2 cycles

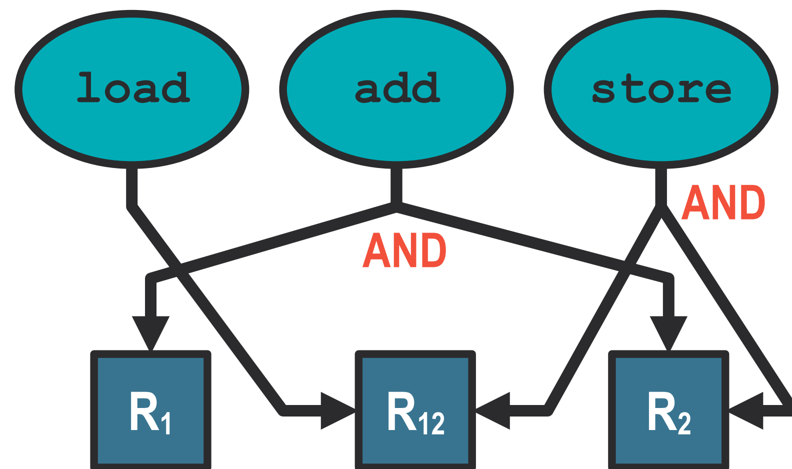
But....

Disjunctive Mapping	
Instruction	Port
load	P_1 or P_2
add	2^*P_1
store	P_2

Conjunctive Mapping	
Instruction	Resource
load	0.5^*R_{12}
add	2^*R_1 and R_{12}
store	R_2 and 0.5^*R_{12}

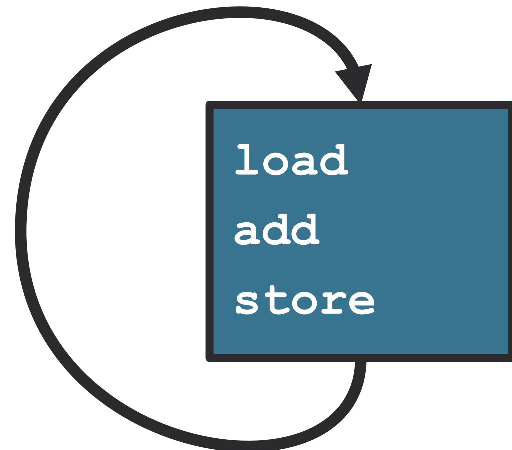


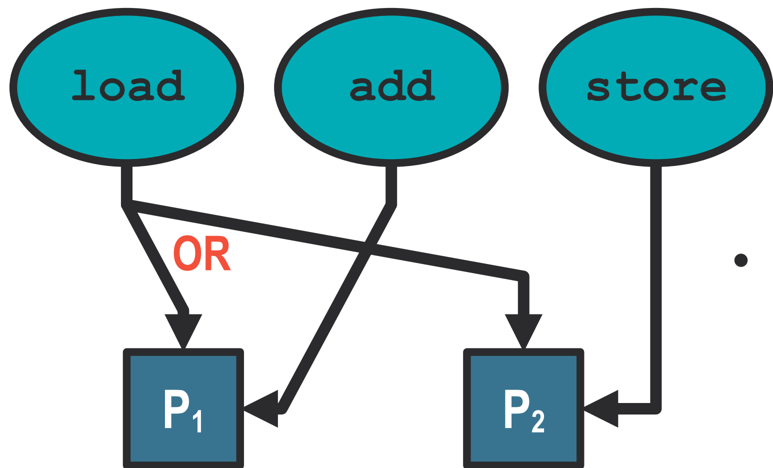
proved equivalent



But....

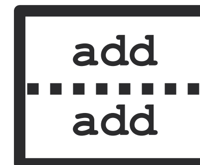
Disjunctive Mapping	
Instruction	Port
`load`	P_1 or P_2
`add`	$2{*}P_1$
`store`	P_2





- Average optimal execution time (▶ throughput):

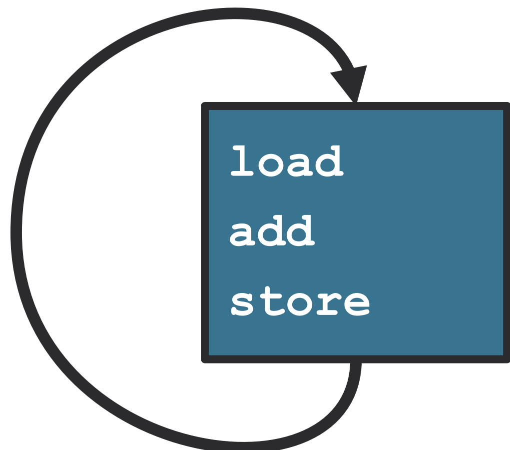
  2 cycles



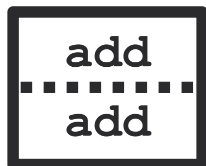
P1

P2

But....



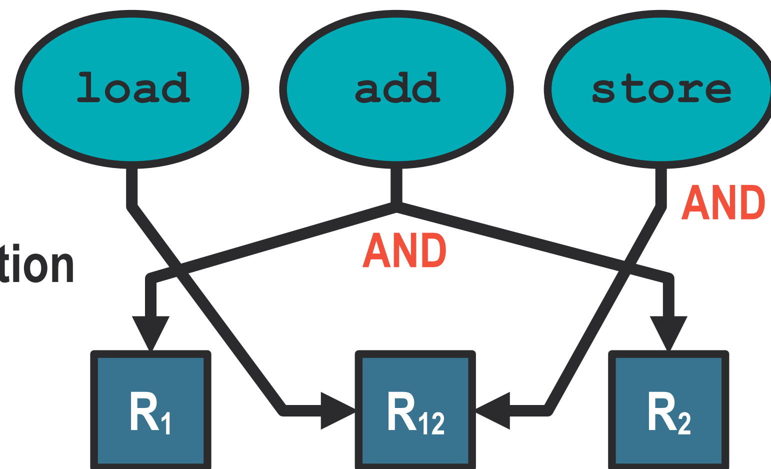
Conjunctive Mapping	
Instruction	Resource
load	$0.5*R_{12}$
add	$2*R_1$ and R_{12}
store	R_2 and $0.5*R_{12}$



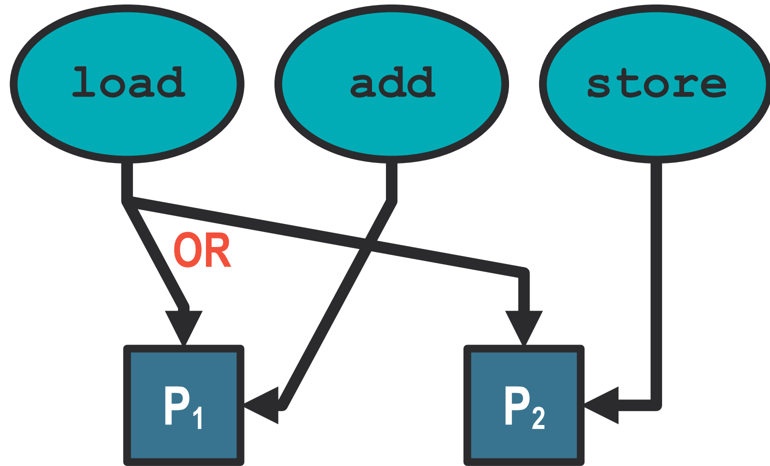
R_1 R_{12} R_2

- Average optimal execution time (▶ throughput):

2 cycles

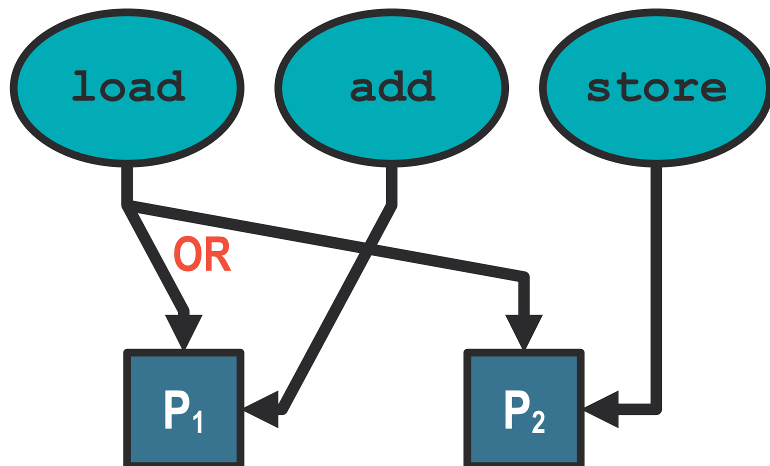


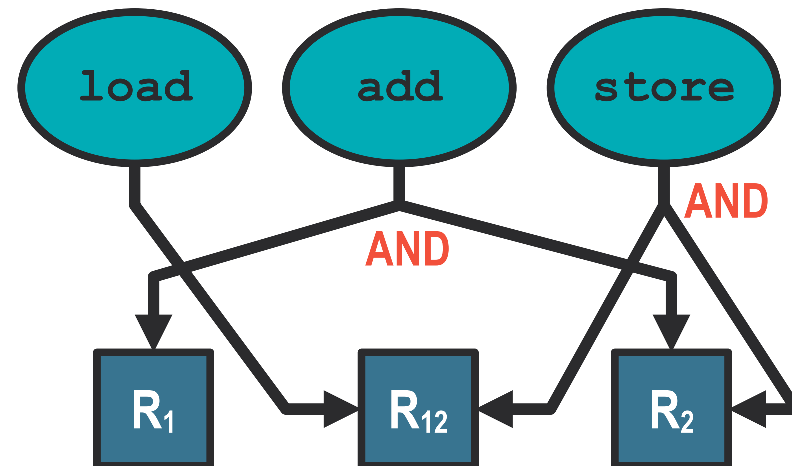
Trick of the dual formulation



- **Disjunctive form:**
  - Derived from hardware
  - Instruction **may** be executed on several ports
  - Optimal excecution time is an **optimization problem**

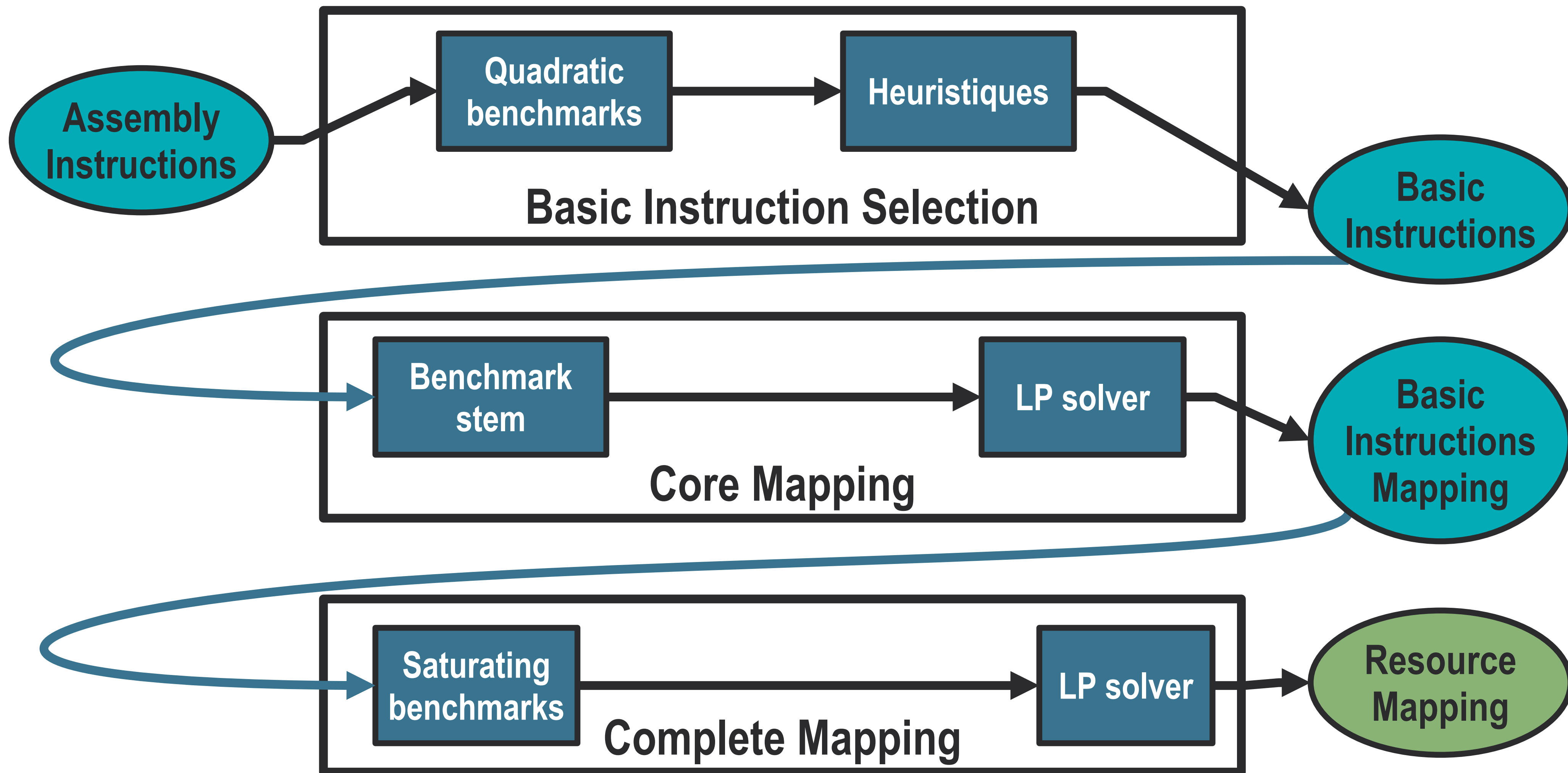
Trick of the dual formulation



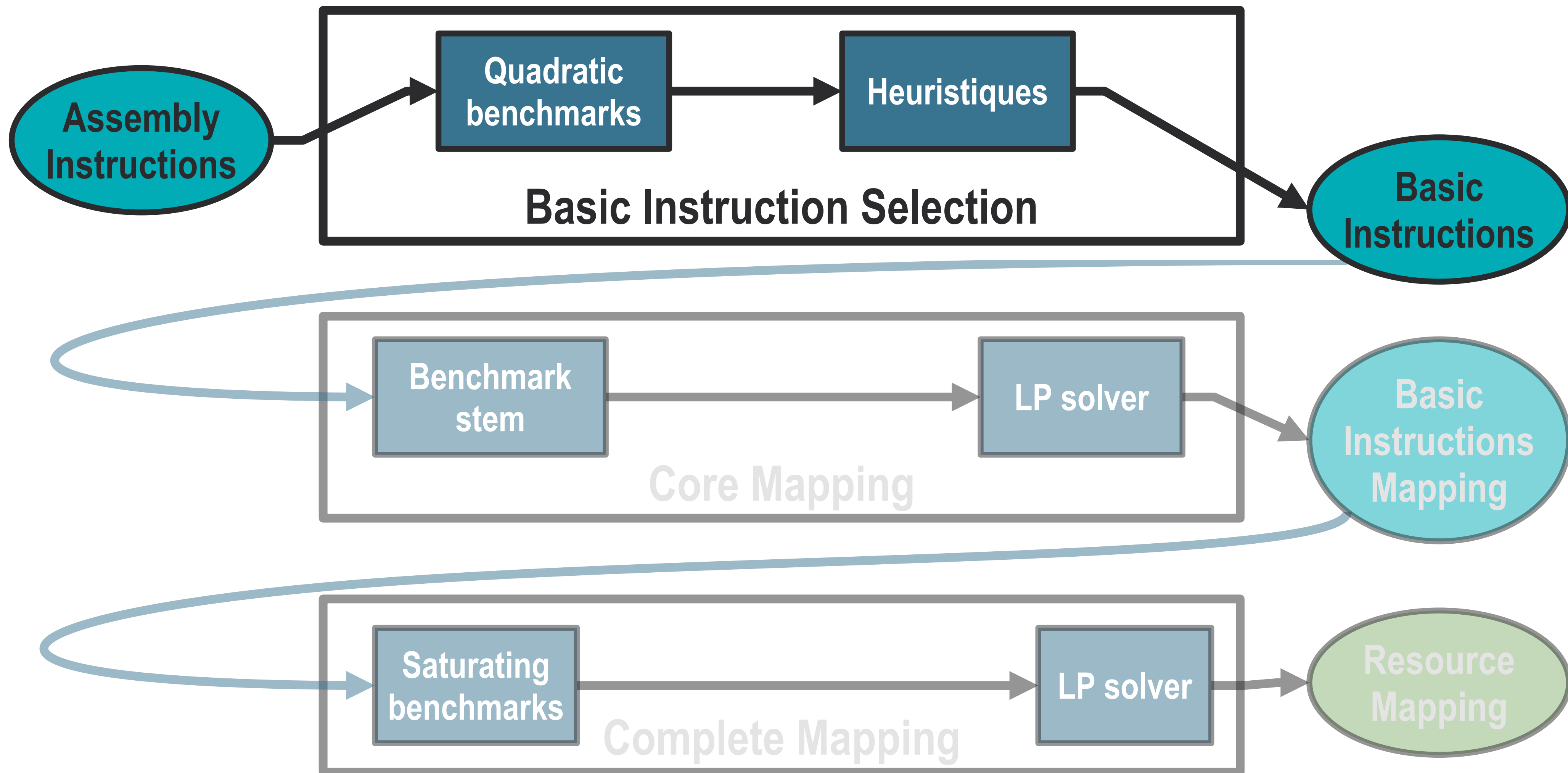


- **Disjunctive form:**
  - Derived from hardware
  - Instruction **may** be executed on several ports
  - Optimal excecution time is an **optimization problem**
- **Conjunctive form:**
  - Simpler representations
  - More resources
  - No µOps
  - Optimal execution time is **a maximum of a sum**
  - Decomposition of **always used** resources
  - For all disjunctive mapping, there exists an equivalent **dual** conjunctive mapping

PALMED: overview

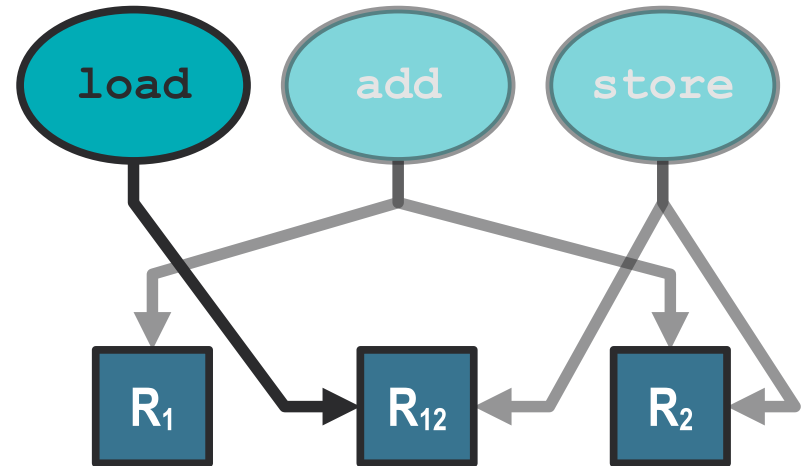


PALMED: overview

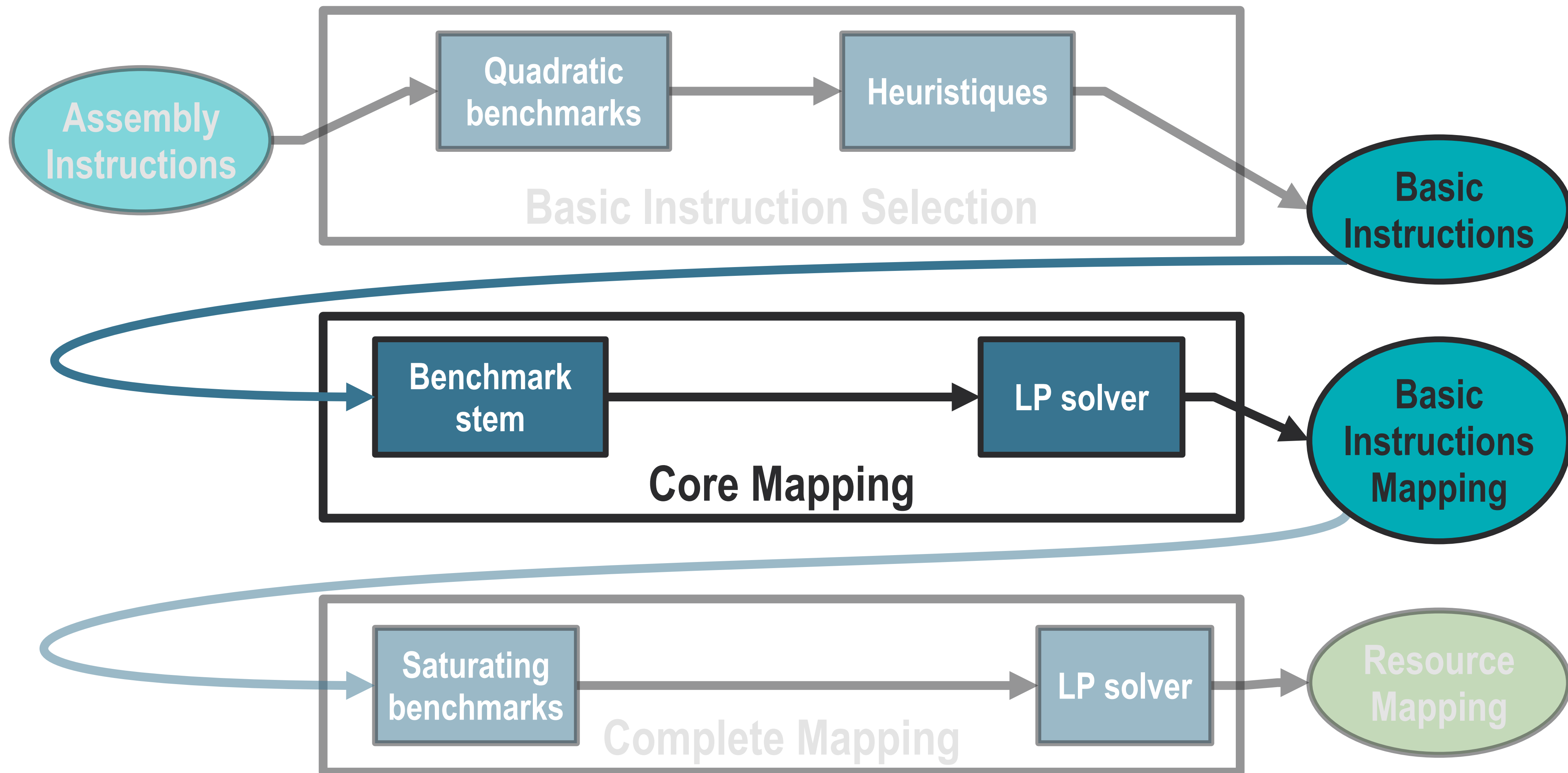


Basic Instruction Selection

- **Input: ISA with syntax rules**
- **Output: Reduced set of Basic Instructions**
  - ~10-20 instructions
  - Use preferrentially **one** resource
- **Based on three selection filters**
  - Equivalence classes of instructions
    - **Quadratic benchmarks**
  - Independant instructions
  - Instruction using resource of high throughput

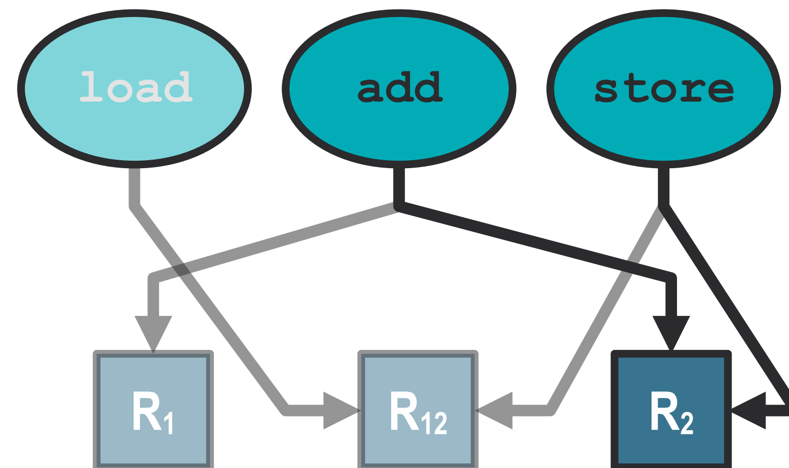


PALMED: overview



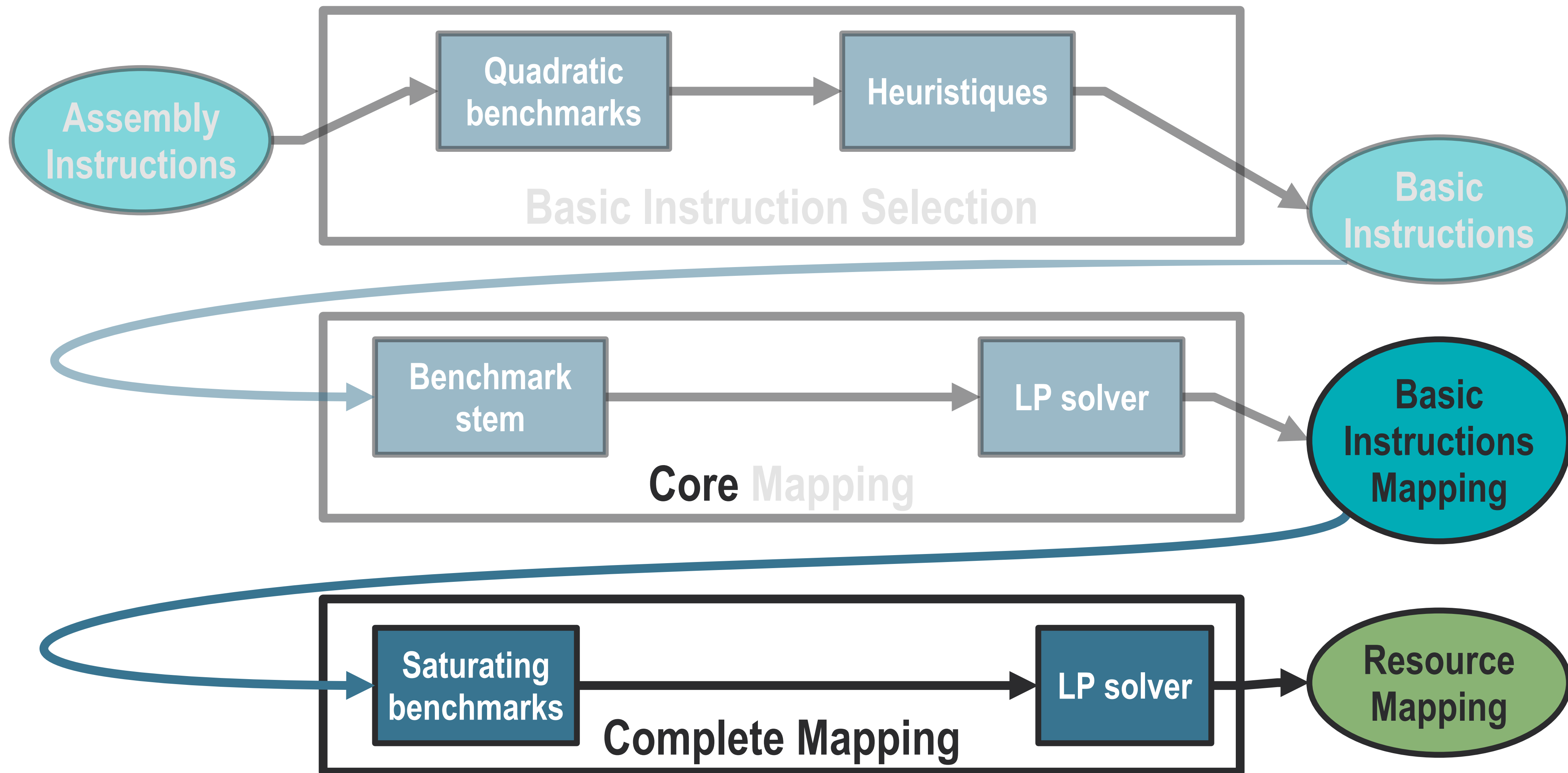
Core Mapping

- **Input: Set of Basic Instructions**
- **Outputs:**
  - **Mapping** of the Basic Instruction
  - **Saturating benchmarks**
- **Multi-step solving**
  1) Determine the **shape** of the mapping
     - number of resources
     - possible edges
     - Iterative process
  2) Determine the **value** of the edges



▶ (add, store) saturates R_2

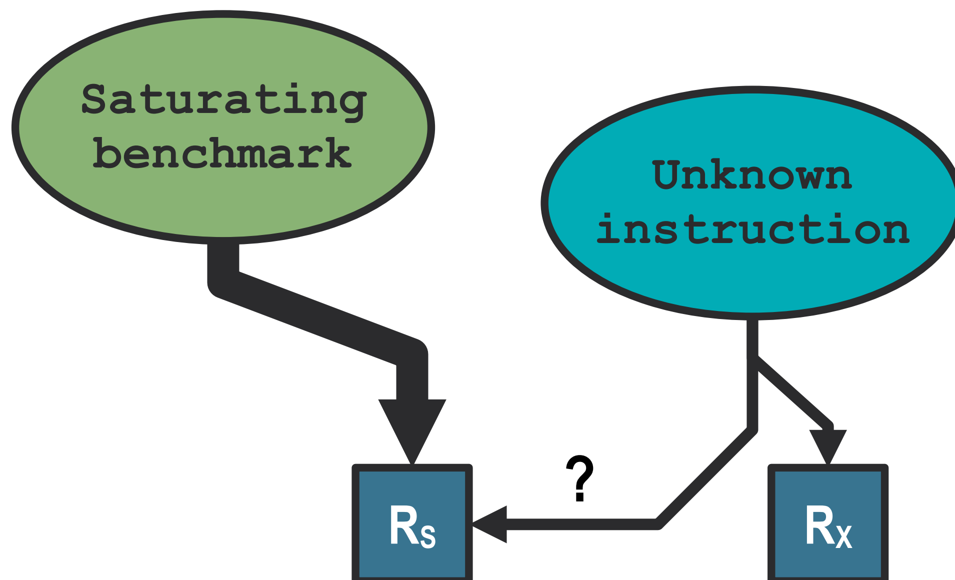
PALMED: overview



Complete Mapping

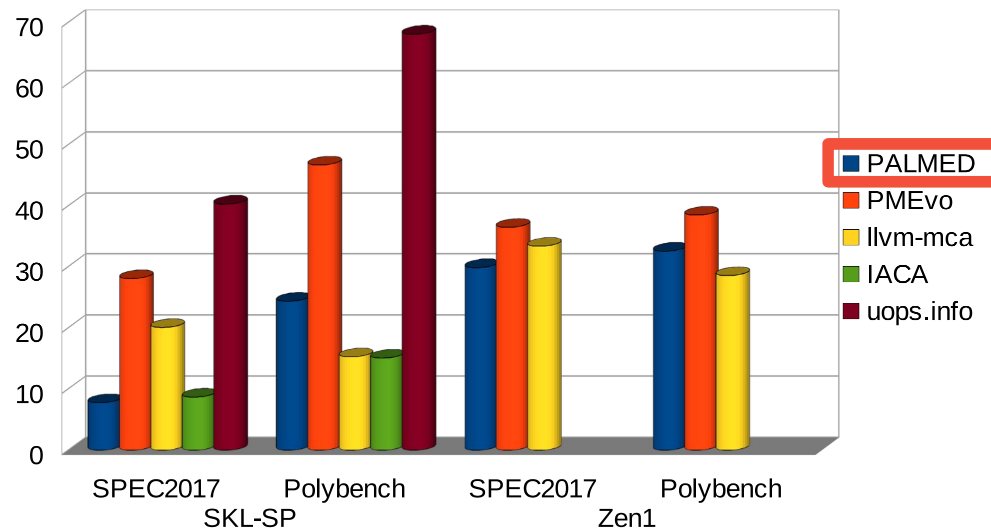
- **Input: Saturating benchmarks**
- **Output: Complete mapping**

- **Use saturating benchmarks as resource indicators**
  - Force a resource to be saturated...
  - ... even if the unknown instruction does not use it!
- **Proved**



PALMED Accuracy

- **Solving time: 2h**
  - Two times faster than PMEvo[1] on Skylake
  - Eight time faster than PMEvo[1] on Zen
- **Supports ~2500 instructions**
- **PMEvo: ~300 instructions**



Mean Square Error (exec. time prediction) of basic blocs with no dependencies

	Unit	PMD Cov. (%)	PMD Err. (%)	PMD τ_K (1)	uops.info Cov. (%)	uops.info Err. (%)	uops.info τ_K (1)	PMEvo Cov. (%)	PMEvo Err. (%)	PMEvo τ_K (1)	IACA Cov. (%)	IACA Err. (%)	IACA τ_K (1)	llvm-mca Cov. (%)	llvm-mca Err. (%)	llvm-mca τ_K (1)
SKL-SP	**SPEC2017**	N/A	7.8	0.90	99.9	40.3	0.71	71.3	28.1	0.47	100.0	8.7	0.80	96.8	20.1	0.73
	Polybench	N/A	24.4	0.78	100.0	68.1	0.29	66.8	46.7	0.14	100.0	15.1	0.67	99.5	15.3	0.65
ZEN1	**SPEC2017**	N/A	29.9	0.68	N/A	N/A	N/A	71.3	36.5	0.43	N/A	N/A	N/A	96.8	33.4	0.75
	Polybench	N/A	32.6	0.46	N/A	N/A	N/A	66.8	38.5	0.11	N/A	N/A	N/A	99.5	28.6	0.40

PALMED: limitations

- **Limited to x86 architecture**
  - Armv8 port in progress
- **Limited to port-bound assembly code**
  - Transient effects?
  - Instruction cache?
- **No dependencies**
- **Not an optimisation tool *as it***

PALMED[3]: Main contributions

Automated and scalable reverse engineering of port mapping

- **Based on a novel conjunctive resource mapping**
  - Key design point for **scalability**
- **Only rely on timing measurement**
- **Microbenchmark-driven**
  - Only measures asymptotic **throughput** of list of instructions
  - **Dynamic generation of microbenchmarks** depending of the target architecture
- **Architecture-agnostic**
  - Tested on Intel and AMD CPUs
  - WIP: Adaptation on ARM CPUs

[3] Nicolas Derumigny, Théophile Bastian, Fabian Gruber, Guillaume Iooss, Christophe Guillon, Louis-Noël Pouchet, and Fabrice Rastello. 2022. PALMED: Throughput Characterization for Superscalar Architectures. In IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2022

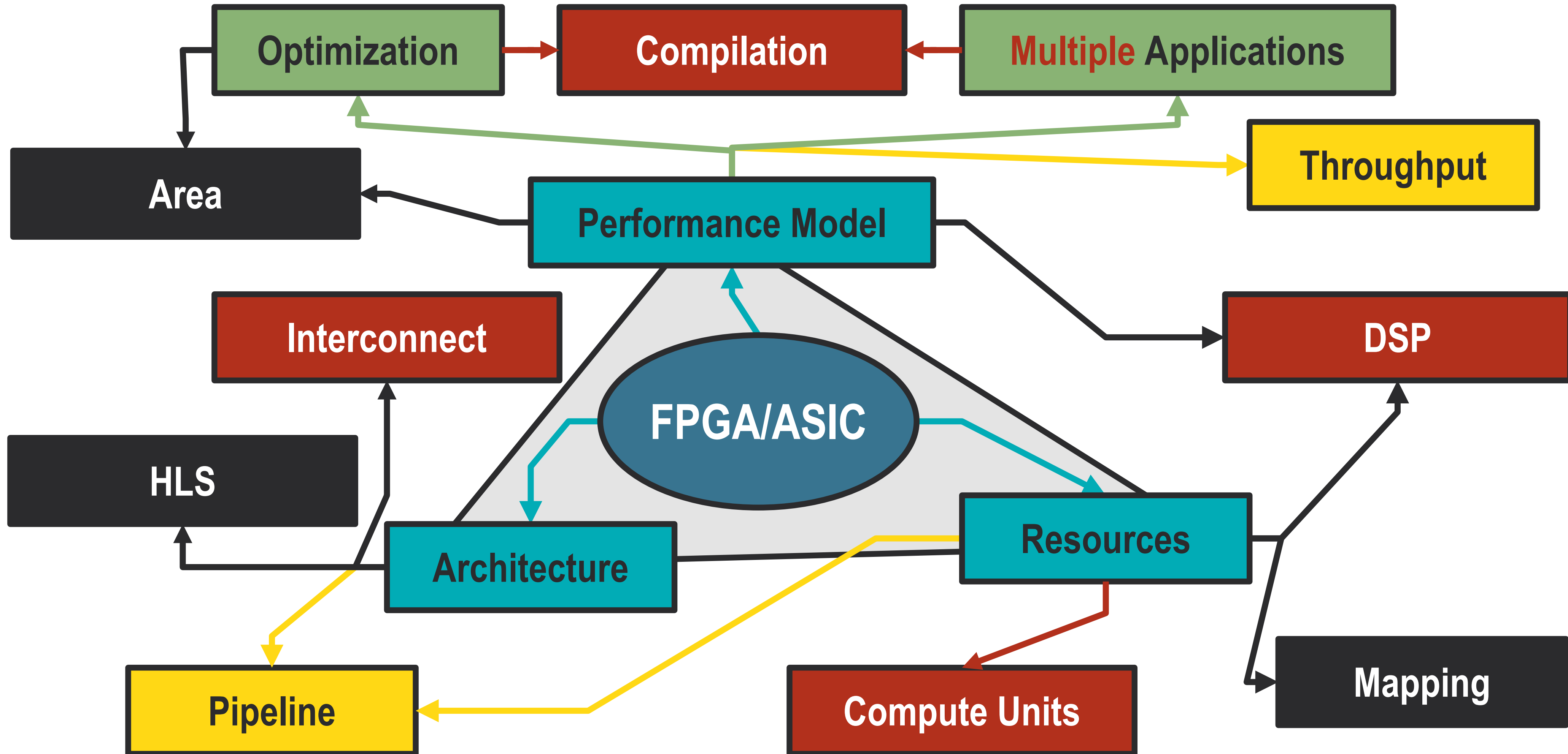
Outline

1) Background

2) Optimising for a fixed architecture: PALMED

3) Optimising for fixed applications: GA

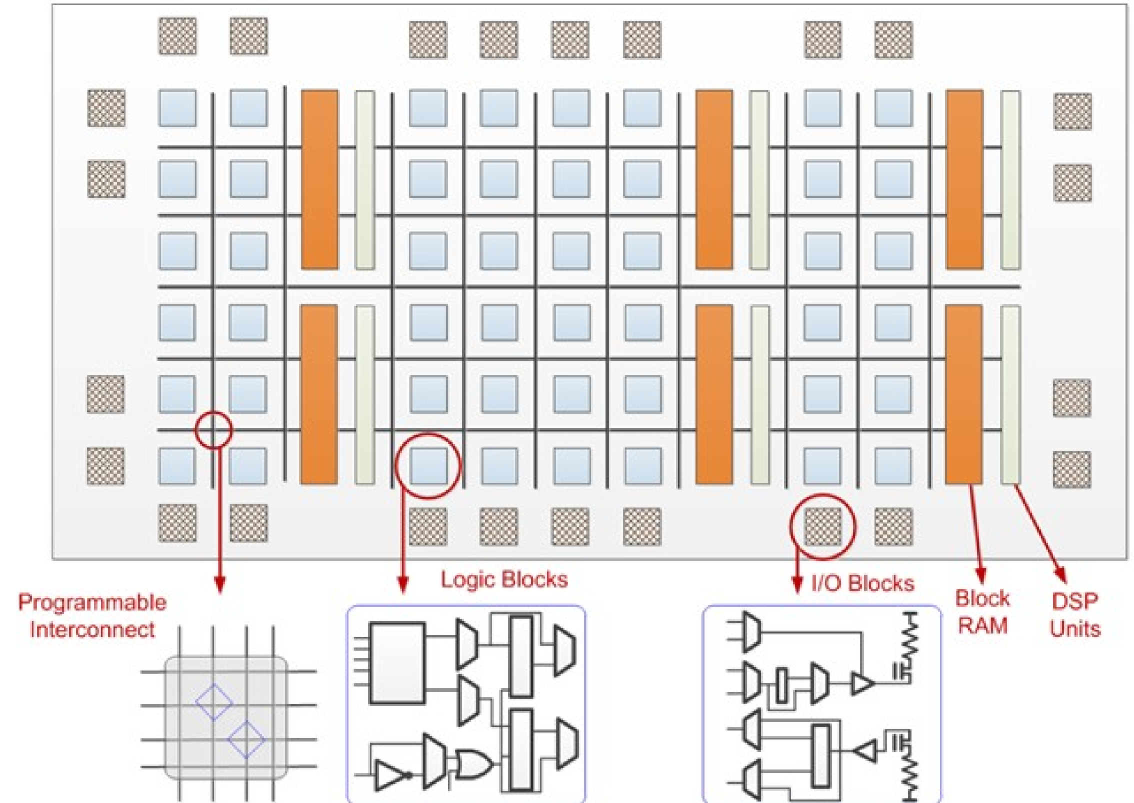
FPGA/ASIC



FPGAs

Field-programmable gate array

- **Customizable hardware**
  - Grid of atomic **gate** element
- **Integrates:**
  - Routing logic
  - **LUT**: Elementary computation unit
  - **FF**: Elementary Storage units
  - **DSP**: Embedded accelerators
  - **Block-RAM**: On chip, denser memory
- **Used for:**
  - High Throughput signal processing
  - Deep Learning acceleration
  - Design Prototyping

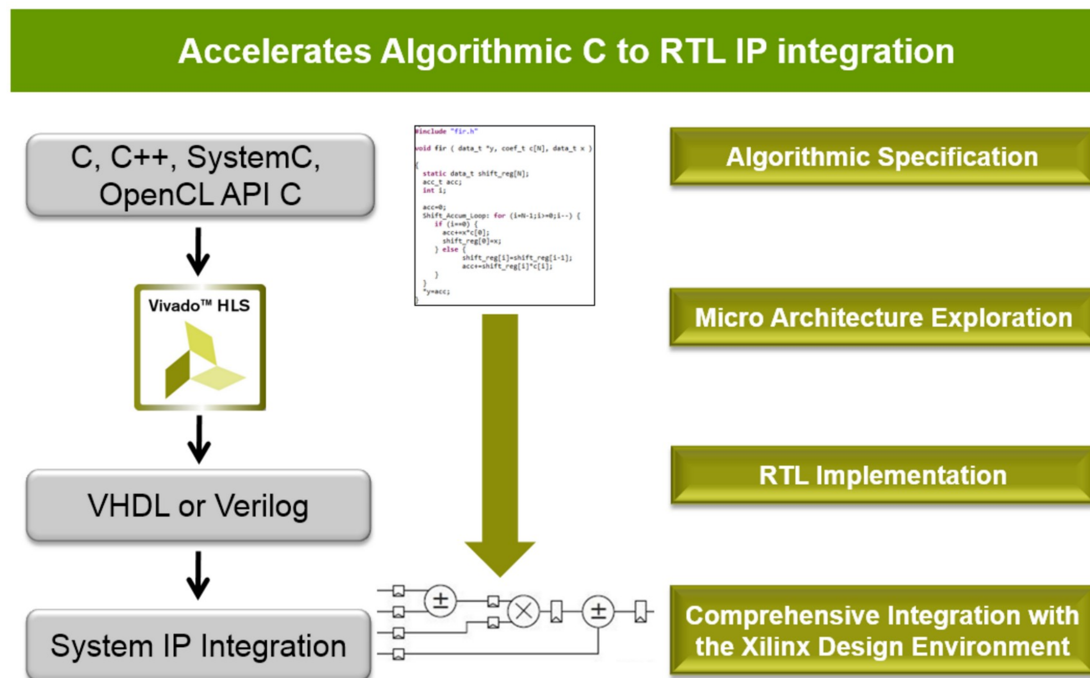


(Credit: Xilinx)

High Level Synthesis: principle, tools

Compilers for hardware design

- **Switch from HDL languages to C/C++**
  - Semantic definition of the computation
  - Semi-automated definition of the hardware topology
    - Extensive use of pragmas
    - High expertise required
- **Faster design time**
  - Annotated C is still less verbose than VHDL
- **Faster verification time**
  - Semantic is embedded in the design
- **Faster TTM / Cost-effective solution**
  - Used in industry



High Level Synthesis: examples

- **Several designs can execute the same program**
  - Compiler optimise for **efficiency**
  - **Paretto Front** of optimal design
- **Sensitive to program syntax**
  - Function and loop bodies forms compute units
  - Two equivalent codes may lead to different designs
- **Relies on annotations:**
  - #pragma pipeline
  - #pragma unroll

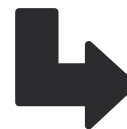
- **Example:**

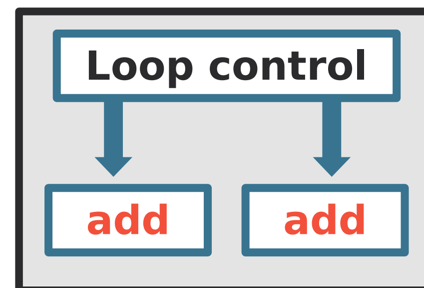
High Level Synthesis: examples

- **Several designs can execute the same program**
  - Compiler optimise for **efficiency**
  - **Paretto Front** of optimal design
- **Sensitive to program syntax**
  - Function and loop bodies forms compute units
  - Two equivalent codes may lead to different designs
- **Relies on annotations:**
  - #pragma pipeline
  - #pragma unroll

- **Example:**

```
fun vect_add():
    for i in [0,N]:
        #pragma HLS pipeline
        #pragma HLS unroll factor=2
        a[i] = b[i] ⊕ c [i]
    return a
```





Accelerator topology

HLS: Resource Sharing

- **We consider Compute Unit sharing**
- **More resources *may* translate in...**
  - More **performances**
  - More **area** / **power consumption**
- **Idea: Mutualise part of resources**
  - Often trade **part of the performance** for efficiency
  - Reduce area
  - Requires additional control logic
- **Uses two types of floating-point operations:**
  - **Additions**
  - **Multiplications**

HLS: Resource Sharing

- We consider **Compute Unit** sharing
- More resources *may* translate in...
  - More **performances**
  - More **area** / **power consumption**
- Idea: Mutualise part of resources
  - Often trade **part of the performance** for efficiency
  - Reduce area
  - Requires additional control logic
- Uses two types of floating-point operations:
  - **Additions**
  - **Multiplications**

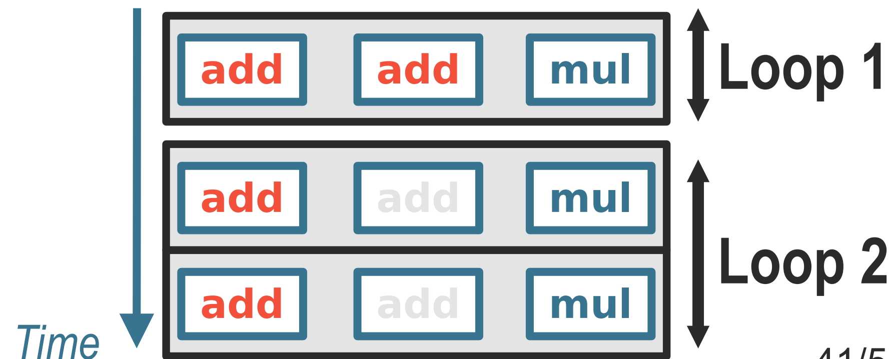
- Example: extract of Discrete Wavelet Transform

```
for j=1 to m-3 step 2 do
    tmp[i][j] ⊕= a1 ⊗ (tmp[i][j-1] ⊕ tmp[i][j+1])

for j=1 to m-3 step 2 do
    tmp[i][j] ⊕= a3 ⊗ (tmp[i][j -1] ⊕ tmp[i][j+1])
    img[j/2+m/2][i] = k2 ⊗ tmp[i][j]
```

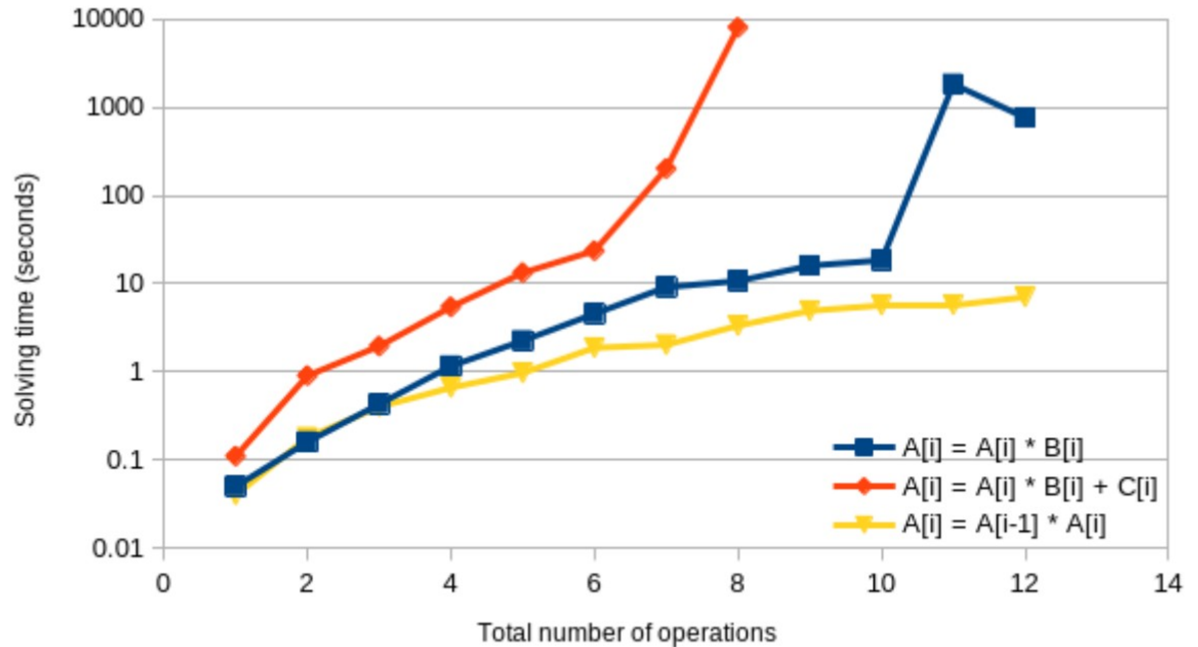
- A valid accelerator design is:

- Corresponding execution:



Optimal resource sharing with an LP

- **Convex, naive encoding (ILP)**
  - **Objective:** Find the fastest accelerator
  - Constraints:
    - **Maximal resource budget**
    - **Dependencies must be satisfied**
    - **Number of Compute Units**



Scaling w.r.t. the type and number of operations

- **Log y scale!**
- **There is no compromise here...**

Optimal resource sharing with an LP

- **Convex**
  - Obje
  - Cons

10000

B[i]

B[i] + C[i]

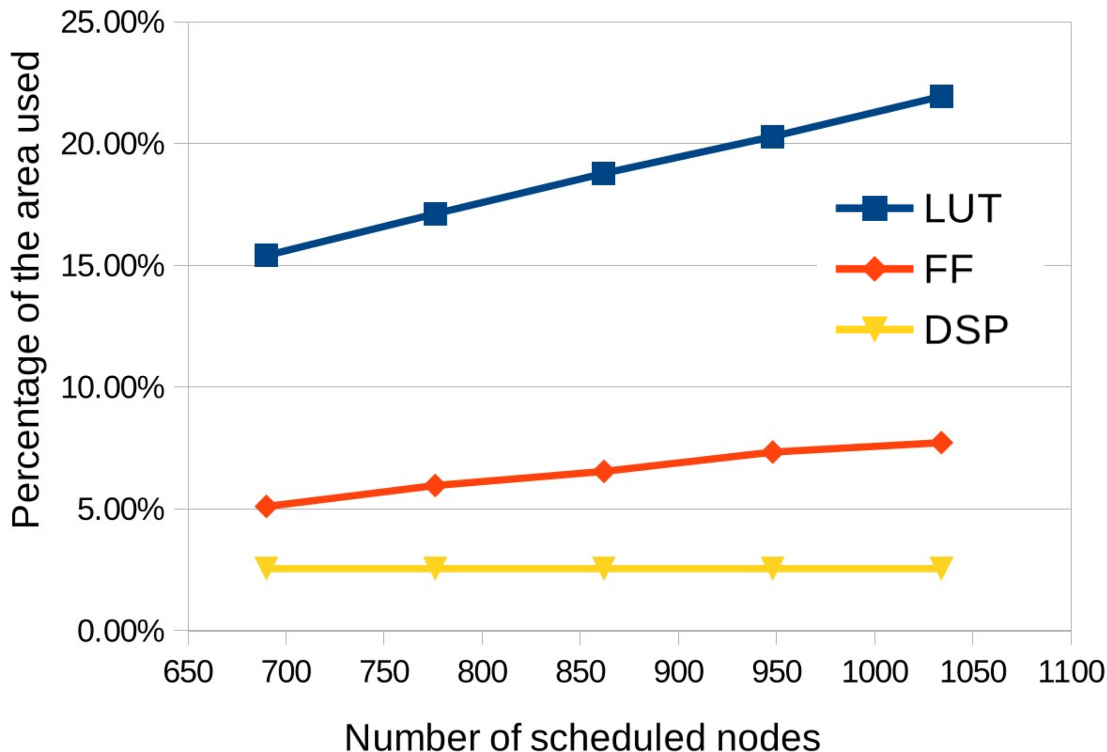
] * A[i]

14

tions

▶ Optimal sharing **is not the right approach !**

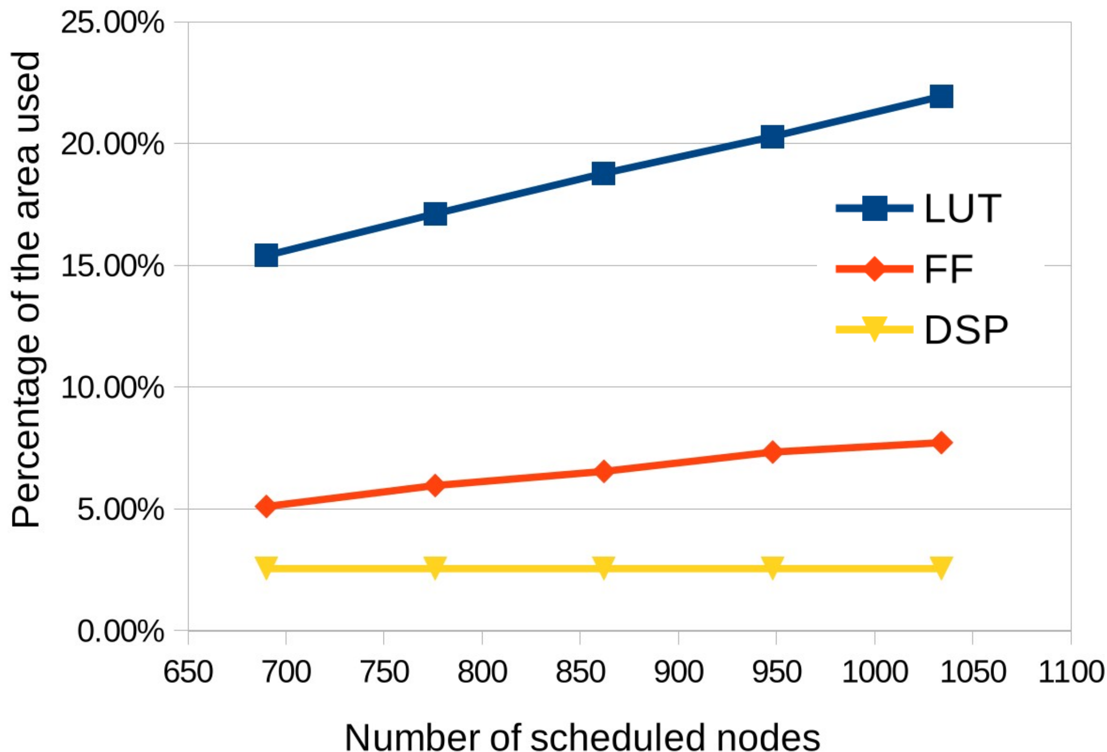
Inexact solving: dominance of the interconnect



TODO EXPLAIN HEURISTIC

- **Routing elements (LUTs) are over-used**
- **Regularity of the compute pattern lost**
  - High-fanout multiplexers

Inexact solving: dominance of the interconnect

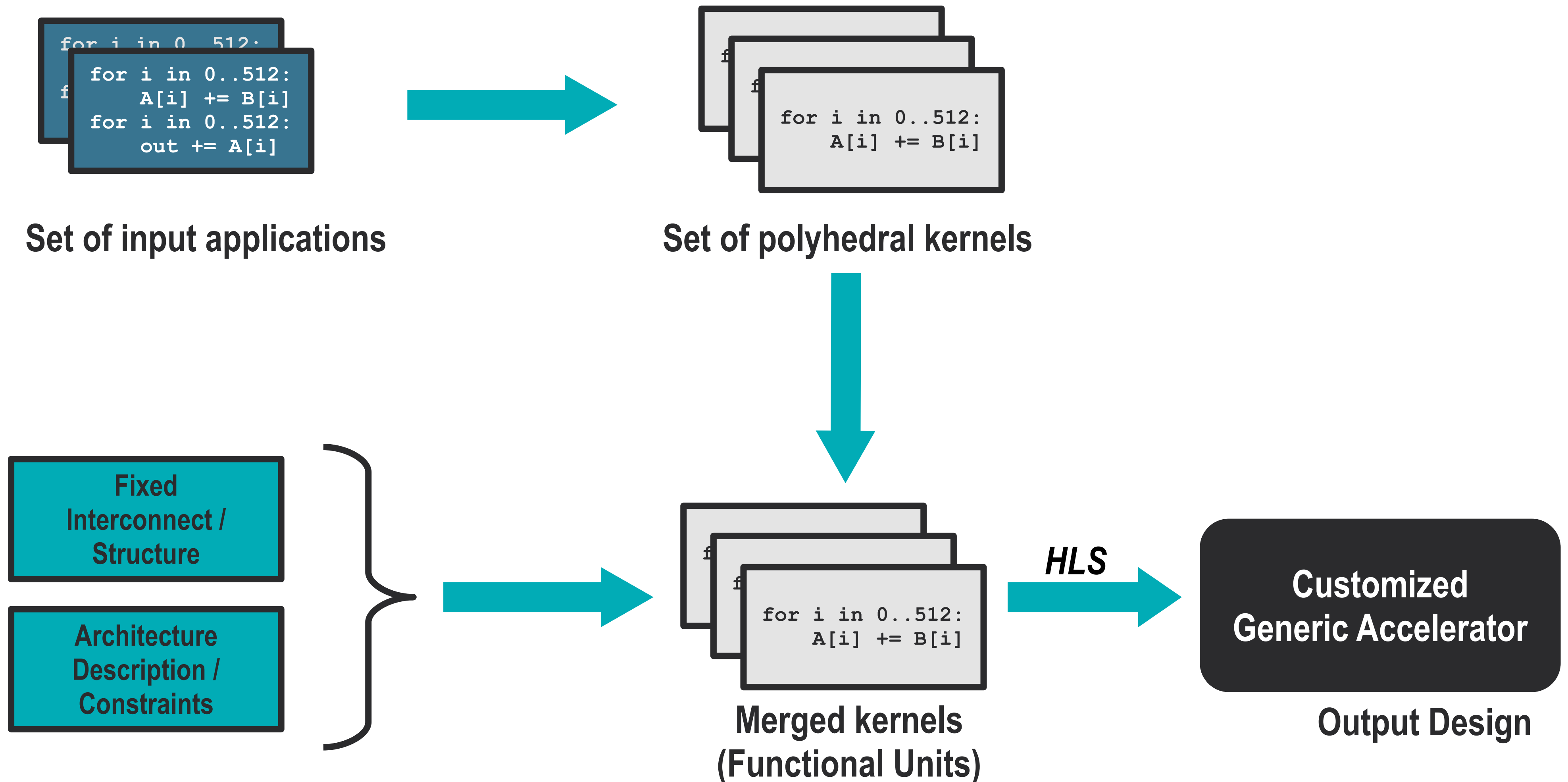


TODO EXPLAIN HEURISTIC

- **Routing elements (LUTs) are over-used**
- **Regularity of the compute pattern lost**
  - High-fanout multiplexers

Can we use these routing logic more intelligently?

Generic Accelerator: Flow of the work



Generic Accelerator: User perspective

```
for i in 0..512:
    A[i] += B[i]
for i in 0..512:
     out += A[i]
```

Input application

```
for i in 0..512:
    A[i] += B[i]
```

Polyhedral kernels (used by the application)

Customized Generic Accelerator

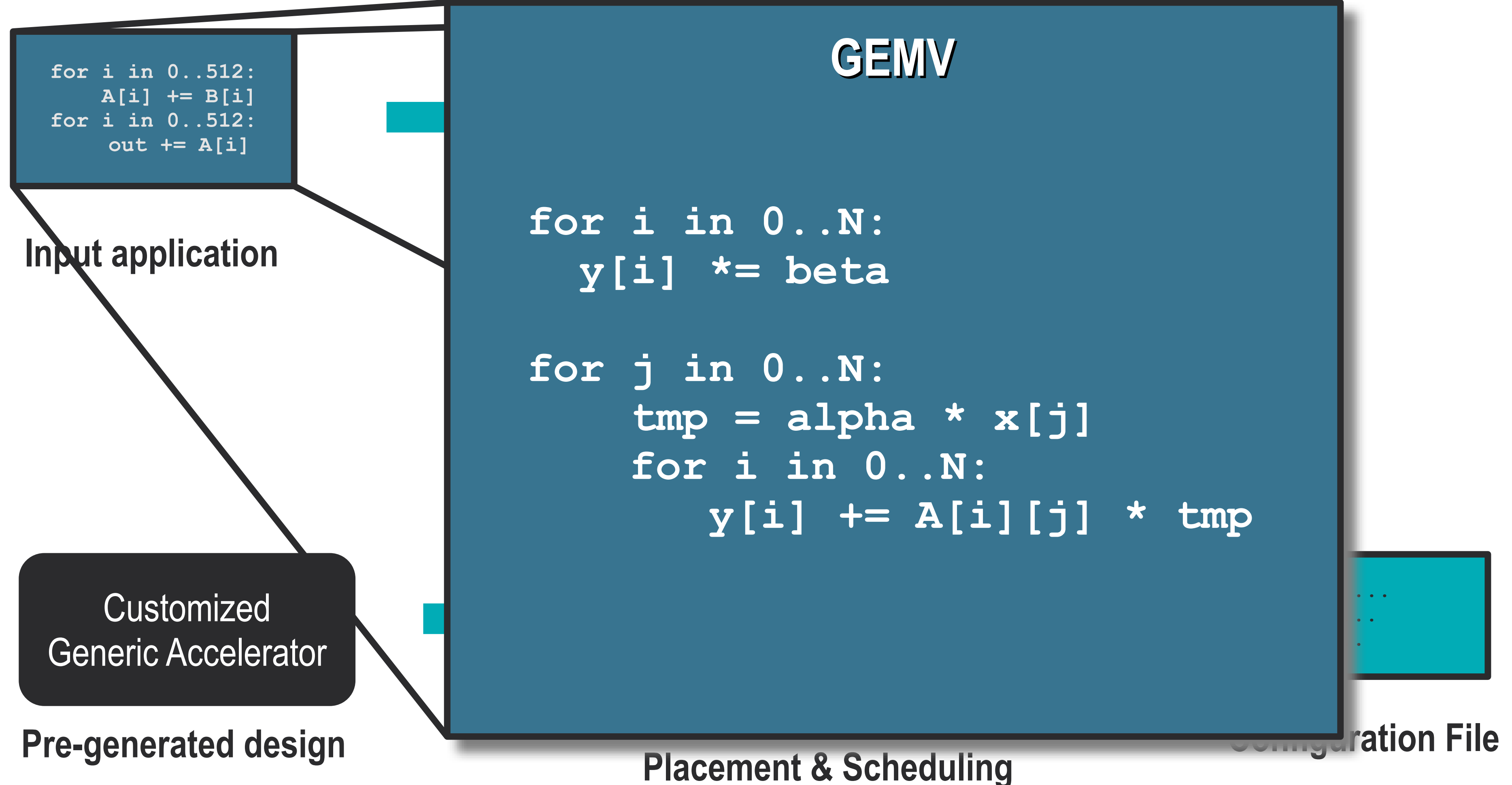
Pre-generated design

Placement & Scheduling

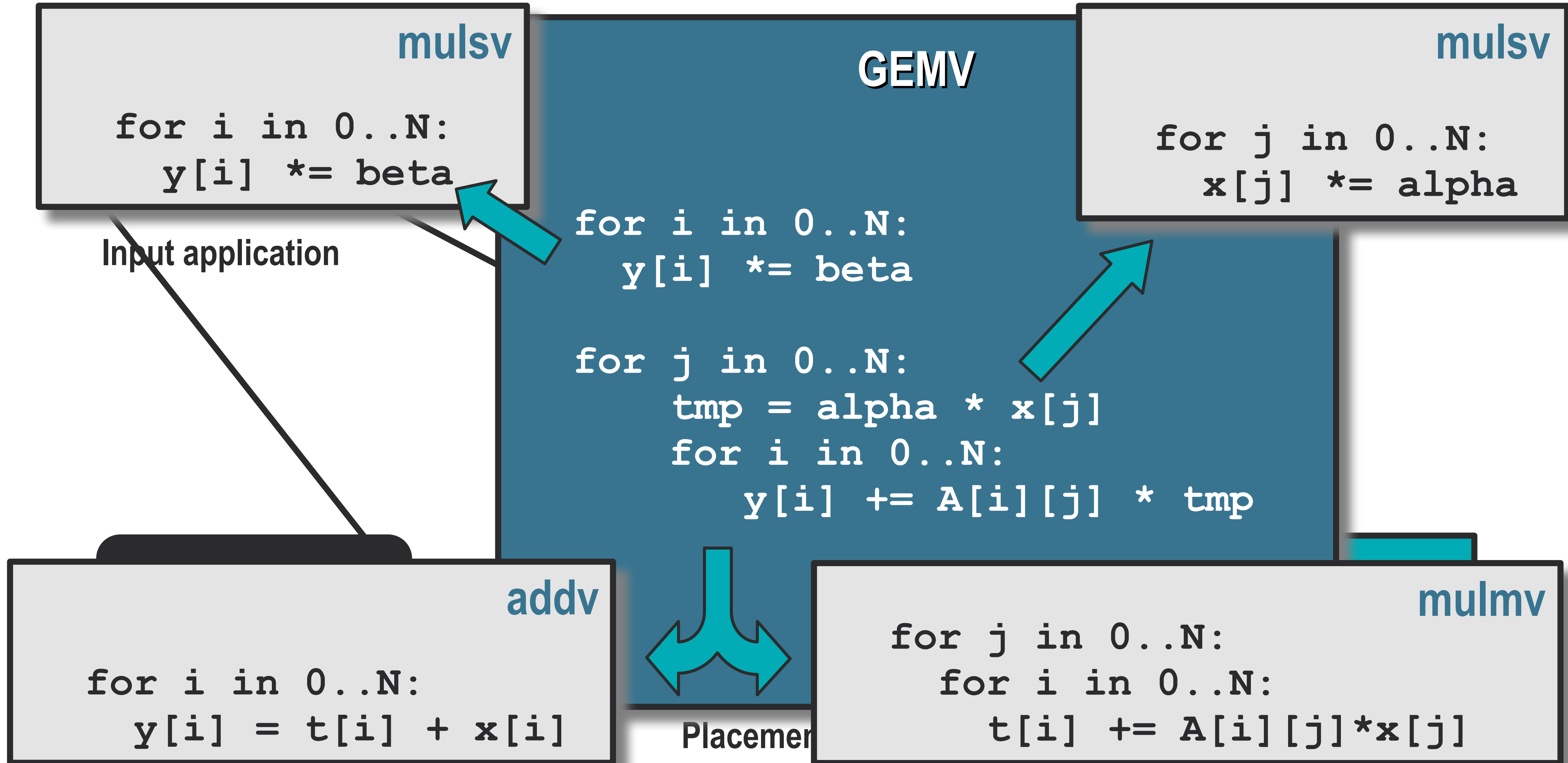
```
addcmv ...
mulmm ...
addm ...
```

Configuration File

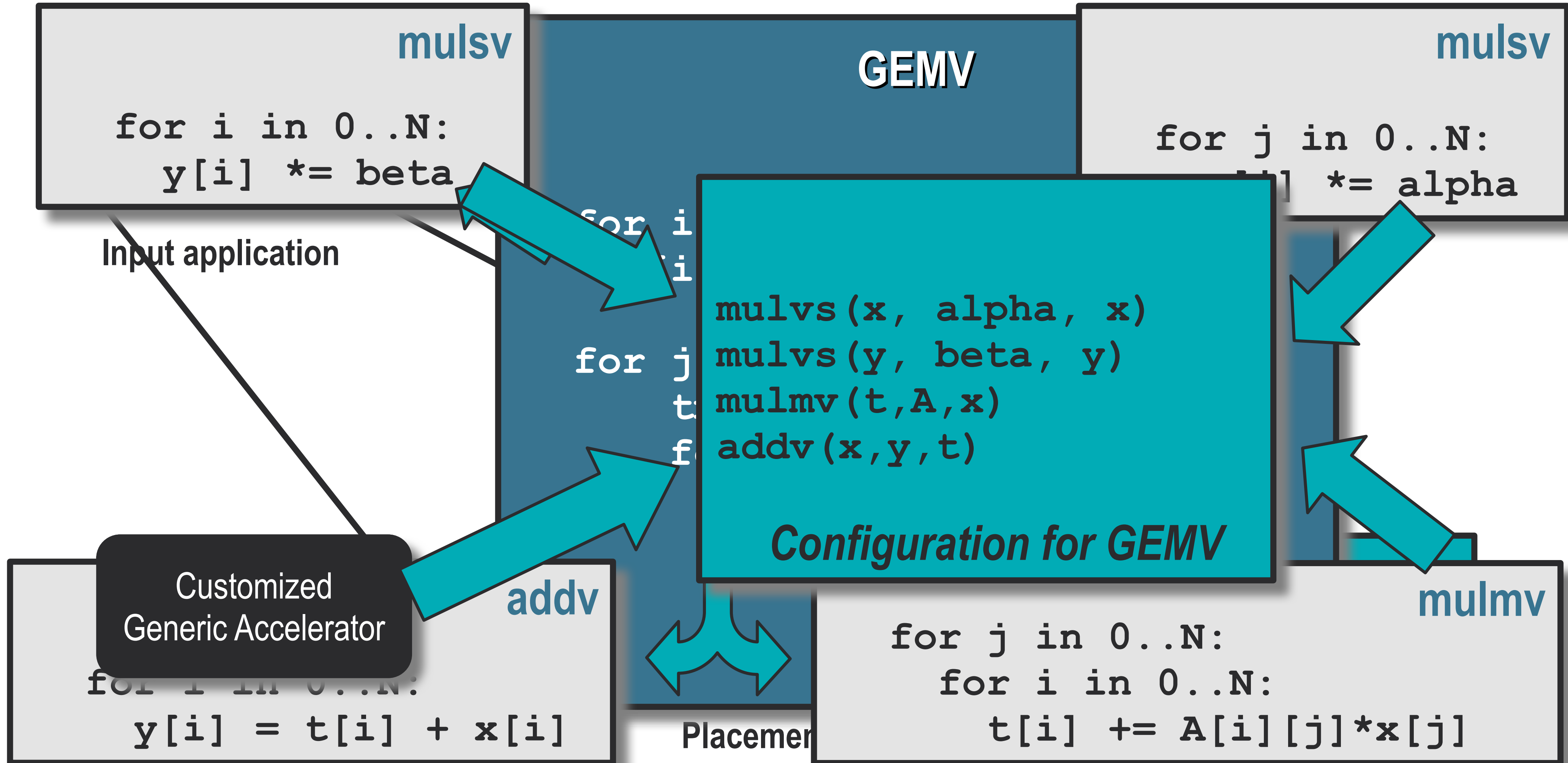
Generic Accelerator: User perspective



Generic Accelerator: User perspective



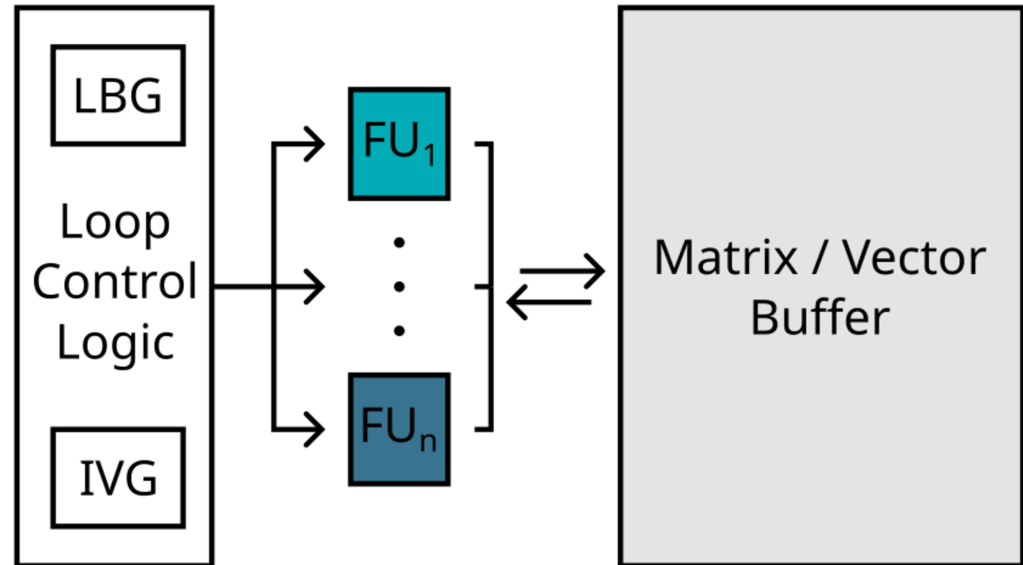
Generic Accelerator: User perspective



Generic Accelerator: template architecture & FUs

Create an accelerator for a family of tasks

- **Loop-based detection of kernels**
- **Execution of Kernels on Functional Units (FU)**
  - Composed of simple operations:
  - Corresponds to merged CU
- **Shared elements:**
  - Loop Control Logic
    - **Loop Bound Generator**
    - **Iteration Vector Generator**
  - Unified buffer



Supported Functionnalities

Kernel	Description	Op.	LA-GA	CORR-GA
noop	Do nothing	None	✓	✓
mulmm	Matrix-matrix multiplication	$\pm$ and $*$	✓	✓
mulmv	Matrix-vector multiplication	$\pm$ and $*$	✓	✓
multrmm	Triangular matrix-matrix multiplication	$\pm$ and $*$	✓	
multrmv	Triangular matrix-vector multiplication	$\pm$ and $*$	✓	
mulsm	Scalar-matrix multiplication	$*$	✓	✓
multrsm	Scalar-triangular matrix multiplication	$*$	✓	
mulsv	Scalar-vector multiplication	$*$	✓	✓
muls	Scalar-scalar multiplication	$*$	✓	✓
trm	Matrix transposition	None	✓	✓
addm	Matrix addition	$\pm$	✓	✓
addv	Vector addition	$\pm$	✓	✓
adds	Scalar addition	$\pm$	✓	✓
addtrm	Triangular matrix addition	$\pm$	✓	
subm	Matrix subtraction	$\pm$	✓	✓
subcmv	Column-wise matrix subtraction	$\pm$	✓	✓
subv	Vector subtraction	$\pm$	✓	✓
subs	Scalar subtraction	$\pm$	✓	✓
pmulm	Point-wise matrix multiplication	$*$	✓	✓
pmulv	Point-wise vector multiplication	$*$	✓	✓
oprodv	Outer (vector) product	$*$	✓	✓
sqrtv	Point-wise vector square root	$\sqrt{\cdot}$		✓
sqrts	Scalar square root	$\sqrt{\cdot}$		✓
accsumcm	Columns-wise accumulation of a matrix	$\pm$	✓	✓
cutminv	Vector round to 1 low values	None	✓	✓
divms	Pointwise division of matrices	$/$		✓
divvs	Pointwise division of vectors	$/$		✓
divcmv	Point-wise division with column-wise value	$/$		✓
set0m	Initialisation of a matrix to 0	None	✓	✓
setidm	Initialisation of a matrix to Id	None	✓	✓
setd1	Initialisation of the diagonal of a matrix to 1	None	✓	✓

	Number of operators				Nb. of FU
	$a \pm b$	$a * b$	a/b	$\sqrt{a}$	
BLAS	2	1	0	0	2
CORR	3	3	1	1	4

- **Two accelerators have been tested**
  - LA-GA: Linear algebra
  - CORR-GA: Correlation computation
- **Hardware primitives:**
  - Add, mul, div, sqrt
  - Different routing/iteration spaces combination creates 31 kernels

Performances

Bench name	Arithmetic expression	Execution Time (cycles)			FLOP/C/DSP			FLOP/C/10kFF			FLOP/C/10KLUT		
		MS	MT	LA-GA	MS	MT	LA-GA	MS	MT	LA-GA	MS	MT	LA-GA
SCALE	$A = \alpha \cdot A + B$	5572	2059	8258	0.368	0.497	0.165	5.080	13.593	2.193	7.722	20.466	1.053
GEMV	$y = \alpha \cdot A \cdot x + \beta \cdot y$	4553	2126	4396	0.457	0.391	0.315	3.960	12.950	4.184	5.686	18.752	2.010
TRMV	$y = \alpha \cdot A \cdot x + \beta \cdot y$	2339	2435	2380	0.458	0.293	0.300	6.177	5.894	3.982	9.311	8.735	1.913
GER	$A = \alpha \cdot x \cdot y^t + A$	4738	2057	8343	0.436	0.401	0.165	6.093	13.528	2.187	9.348	20.058	1.051
GEMM	$C = \alpha \cdot A \cdot B + \beta \cdot C$	307586	134018	274540	0.433	0.397	0.323	5.759	12.934	4.287	8.860	19.011	2.059
TRMM	$C = \alpha \cdot A \cdot B + \beta \cdot C$	149696	155840	145516	0.458	0.293	0.314	5.964	5.816	4.169	8.991	8.688	2.002

Bench name	Arithmetic expression	Execution Time (cycles)			FLOP/C/DSP			FLOP/C/FF			FLOP/C/LUT		
		MS	MT	CORR-GA	MS	MT	CORR-GA	MS	MT	CORR-GA	MS	MT	CORR-GA
CENTER	$X^C_{ij} = X_{ij} - (\sum_{i'} X_{i'j})/n$	8343	4166	12480	0.495	0.495	0.055	10.362	19.448	1.090	9.570	15.374	0.425
STDDEV	$\sigma^X_j = \sqrt{\sum_i (X^C_i)^2/n}$	16691	8370	29053	0.247	0.247	0.047	7.796	13.991	0.936	6.148	10.148	0.365
CENTER-REDUCE-DIV	$X^{CR}_{ij} = (X_{ij} - \sum_{i'} X_{i'j})/(\sigma^X_j \cdot \sqrt{n})$	20935	10486	33352	0.247	0.164	0.052	6.579	9.707	1.021	5.579	7.761	0.398
CORR	$(X^{CR})^t \cdot X^{CR}$	291221	144614	303763	0.468	0.314	0.150	10.905	17.962	2.955	9.119	13.834	1.152
CORRx3	3×CORR	873663	433842	320603	0.468	0.314	0.425	10.905	17.962	8.400	9.119	13.834	3.275

- **GA is efficient when batching independant computation**
  - Sharing low-usage operations
- **GA performs similarily to Most Sharing dedicated designs on 6 out of 10 benchmarks**
- **Low-performance of the GA on 4 out 10 benchmarks is due to loop merging**
  - GA performs in **several macro-instructions** what is compiled into **a single pipeline** on dedicated hardware

Generic Accelerator: Limitations

- **FUs are limited in their variety**
  - No vector FU
- **Limited automation (w.r.t. the program input)**
- **Memory subsystem is resource-dominant**
  - HLS limitation
- **Only optimised for throughput-per-DSP**
  - No tradeoff at all on latency

Conclusion

- **We present an overview of throughput optimisation techniques for heterogeneous architectures:**
  - Automated detection of resources for **superscalar architectures** (PALMED)
    - **CPU-oriented**
    - **Proved**
    - **Porting on other archs (Arm) in progress**
  - Generation of throughput-efficient **FPGA/ASIC** designs (GA)
    - **Suports a set of application as input**
    - **Automated selection of kernels**
    - **Limited expressivity of the FUs**