TRANSPIRE: An energy-efficient TRANSprecision floating-point Programmable archItectuRE

Rohit Prasad, S. Das, K. J. M. Martin, G. Tagliavini, P. Coussy, L. Benini, and D. Rossi

Lab-STICC

cnrs

Université Bretagne Sud
ubs

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

ETH zürich

Agenda

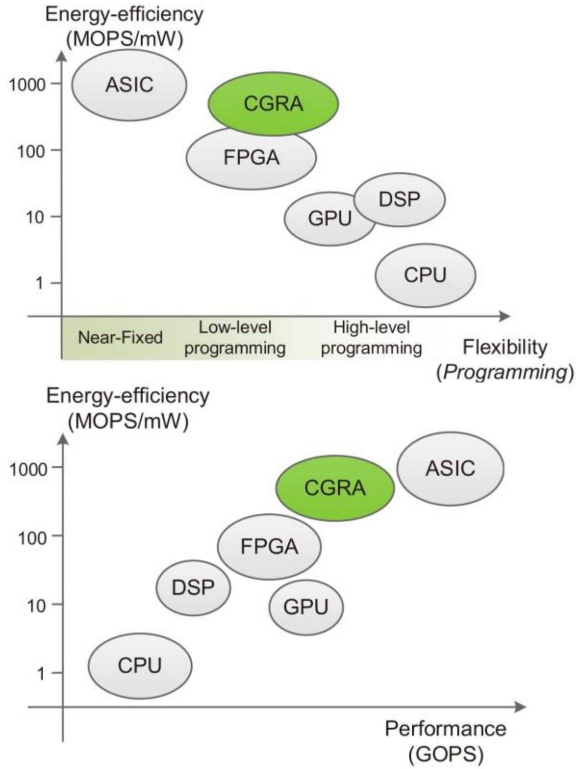
- **Introduction**
- **Background**
- **TRANSPIRE**
- **Experiments & Results**
- **Conclusion**

Agenda

- **Introduction**
- Background
- TRANSPIRE
- Experiments & Results
- Conclusion

Introduction: CGRA

- **CGRA can provide silicon approaching efficiency that of ASIC.**
- **CGRA can provide programmability typical of instruction processors.**
- **Recent CGRAs are competing for high-performance accelerators.**



CGRA : Coarse Grain Reconfigurable Architecture
ASIC : Application Specific Integrated Circuit

Reference: L. Liu et al., A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications, ACM Computing Surveys, October 2019

Introduction: CGRA and Floating-Point

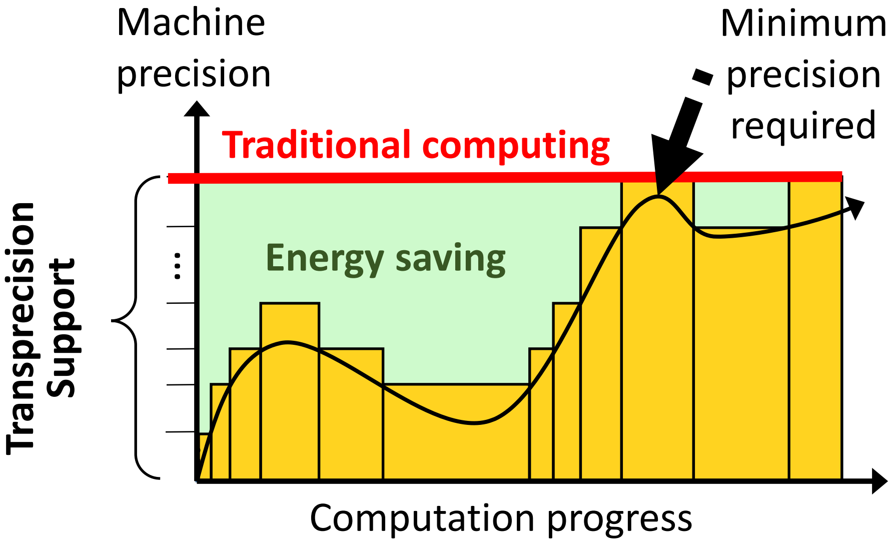
- **CGRAs are used in near-sensor processing and low-power wearable applications.**
- **CGRA architectures demonstrated leading energy-efficiency when executing fixed-point workloads.**
- **Floating-Point (FP) support is becoming a must for IoT end-nodes due to highly dynamic linear algebra algorithms [9].**
- **Native support for IEEE 754 FP data-types demands high-energy consumption.**

Introduction: Fixed-point Vs. Floating-point

Fixed-point Operations	Floating-point Operations
• Only a range of limited numbers can be represented, even normalization (-1 to +1) doesn’t help much, as accuracy drops drastically [8].	• Scaling is not an issue as much wider range of numbers can be represented using the same number of bits.
• Less energy per operation due to the simpler architecture of integer arithmetic units [10].	• Complex hardware leads to high energy consumption.
• Converting floating-point to fixed-points before computing output, demands extra overheads.	• Native support of FP brings no such overheads

Transprecision Computing

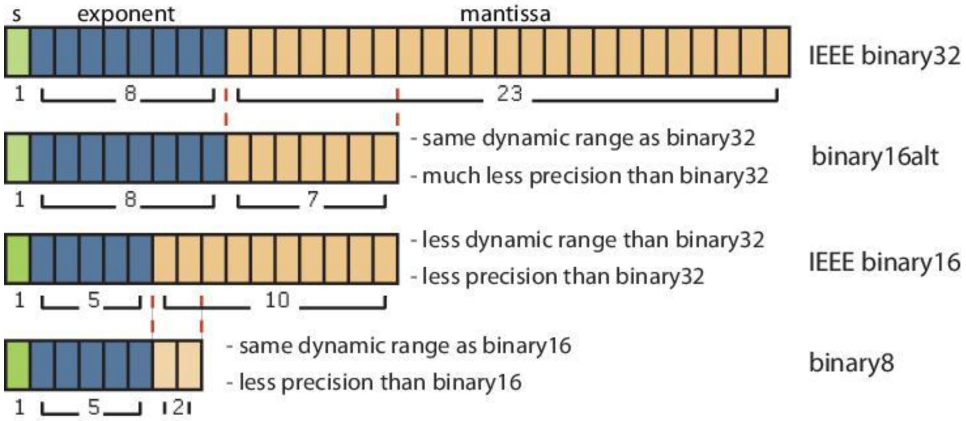
- **Transprecision computing → approximated computation does not automatically imply a quality loss**
  - **Accuracy requirements on the final results**
  - **Adaptive precision during computation for extra benefits (energy saving)**



Reference: G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini. A transprecision floating-point platform for ultra-low power computing. In DATE 2018, 2018.

Small-Float Formats for Transprecision Computing

- **Preliminary experiments motivate *smaller-than-32-bit* FP types**
- **Several alternatives are possible. A few useful ones have been defined already.**

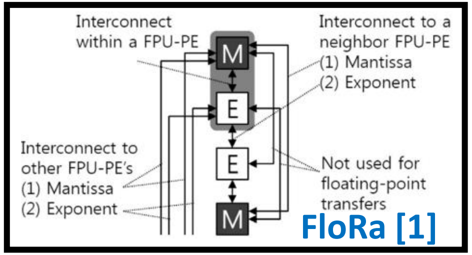

s
exponent
mantissa
1
8
23
IEEE binary32
- same dynamic range as binary32
- much less precision than binary32
1
8
7
binary16alt
- less dynamic range than binary32
- less precision than binary32
1
5
10
IEEE binary16
- same dynamic range as binary16
- less precision than binary16
1
5
2
binary8

Some applications require large dynamic range…

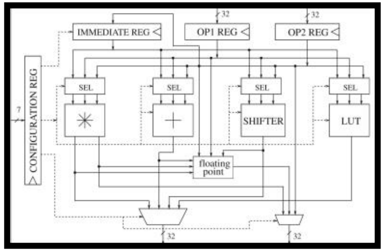
…some others require higher precision

Reference: G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini. A transprecision floating-point platform for ultra-low power computing. In DATE 2018, 2018.

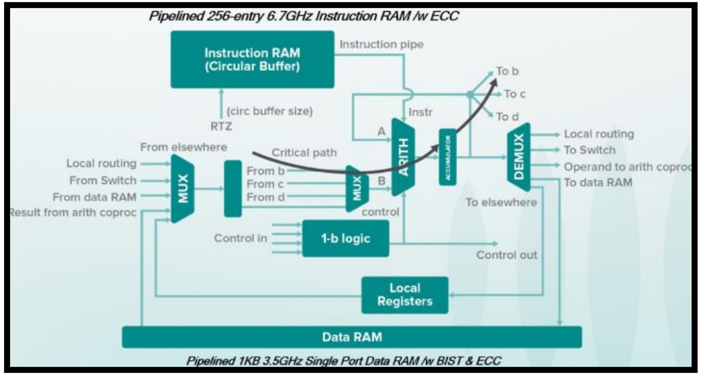
Introduction: CGRA and FP operations


Interconnect within a FPU-PE
Interconnect to a neighbor FPU-PE
(1) Mantissa
(2) Exponent
M
E
E
M
Interconnect to other FPU-PE's
(1) Mantissa
(2) Exponent
Not used for floating-point transfers

FloRa [1]



Butter array [3]



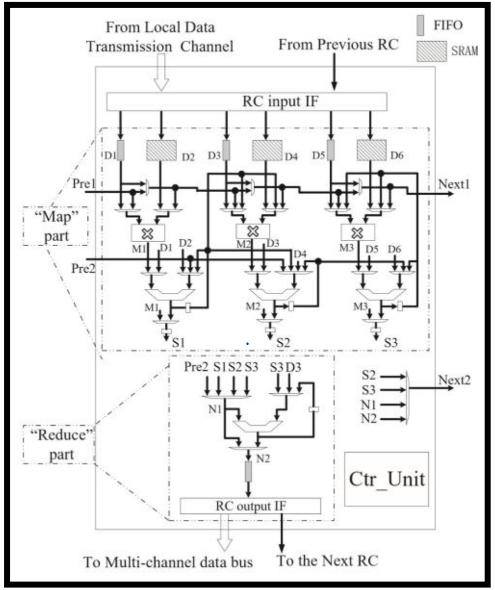
WAVE CGRA [2]

Integer operations are combined

Floating-Point

⬇

Fixed-Point



Stream Dual-Track-CGRA [4]

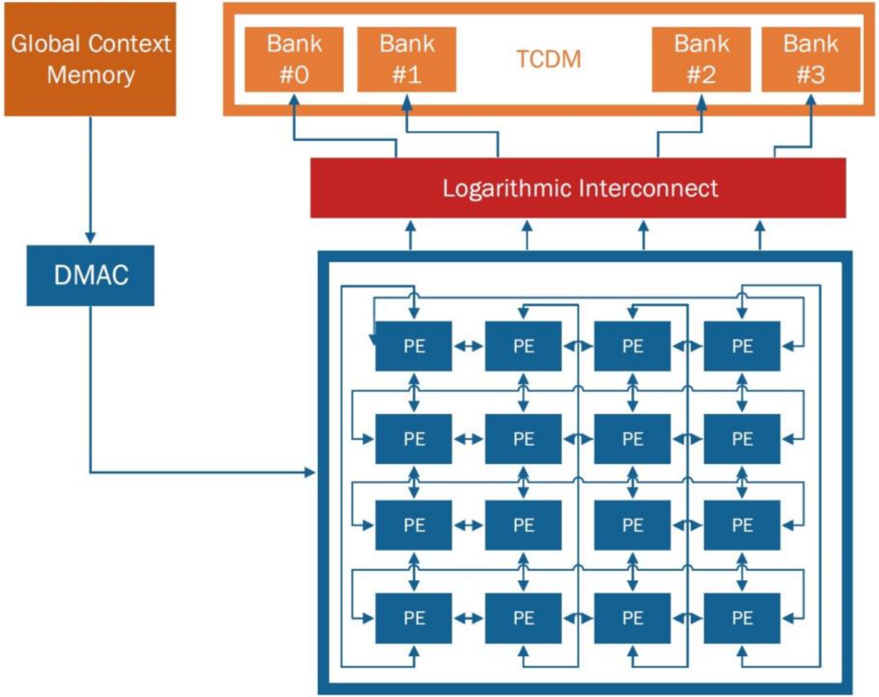
High energy consumption !!!!

Agenda

- Introduction
- **Background**
- TRANSPIRE
- Experiments & Results
- Conclusion

Background: Integrate Programmable Array

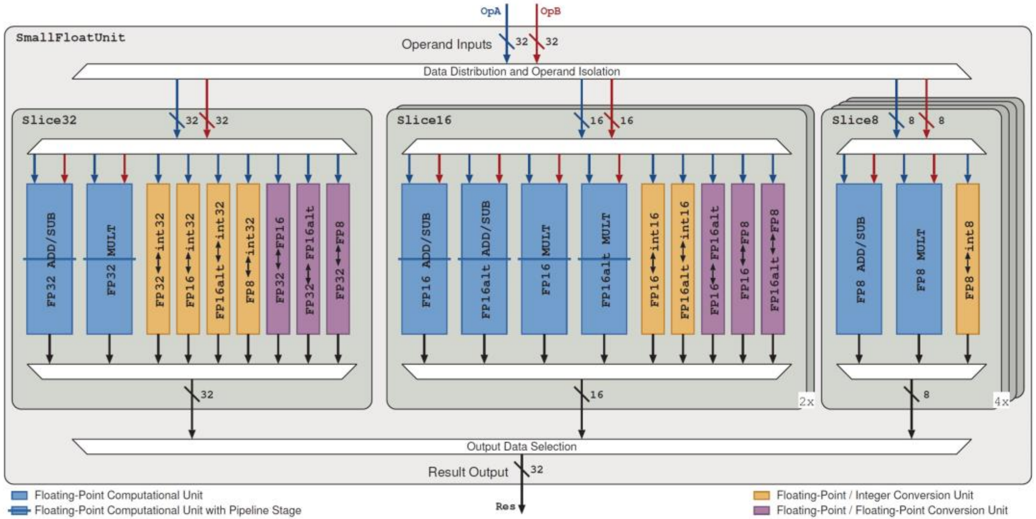
- **IPA supports integer based operations only.**
- **Compiler supports single-cycle operations.**
- **No SIMD support.**
- **Integrated system consists of 4x4 Processing Element (PE) array, a DMA controller, and a context memory.**



Reference: S. Das, D. Rossi, K. J. M. Martin, P. Coussy, L. Benini, A 142MOPS/mW integrated programmable array accelerator for smart visual processing, ISCAS, 2017

Background: smallFloat Unit (SFU)

- **SFU is a major step in realizing Transprecision Computing.**
- **1x IEEE-754 binary32,
2x IEEE-754 binary16,
2x binary16alt,
4x binary8**
- **5 rounding modes.**
- **SFU achieves 18% higher energy-efficiency w.r.t. IEEE 754 binary32.**



Reference: G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini. A transprecision floating-point platform for ultra-low power computing. In DATE 2018, 2018.

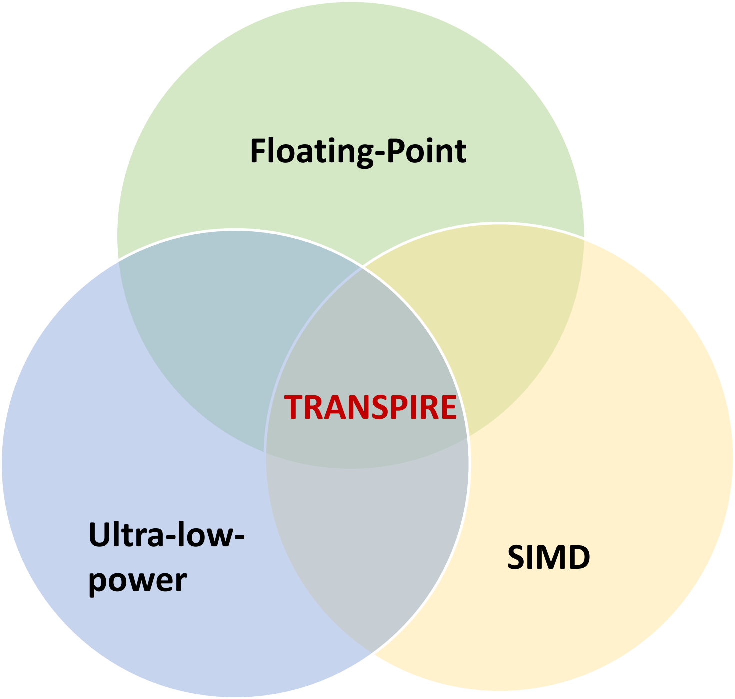
Reference: S. Mach, F. Schuiki, F. Zaruba, and L. Benini. A 0.80 pj/flop, 1.24 Tflop/sW 8-to-64 bit Transprecision Floating-Point Unit for a 64 bit RISC-V Processor in 22 nm FD-SOI. In VLSI-SOC, 2019

Agenda

- Introduction
- Background
- **TRANSPIRE**
- Experiments & Results
- Conclusion

Contribution: TRANSPIRE

- Energy consumption is always in ultra-low-power domain.
- Supports both integer and FP data-types.
- Features SIMD for FP operations.



Contribution: TRANSPIRE

Software

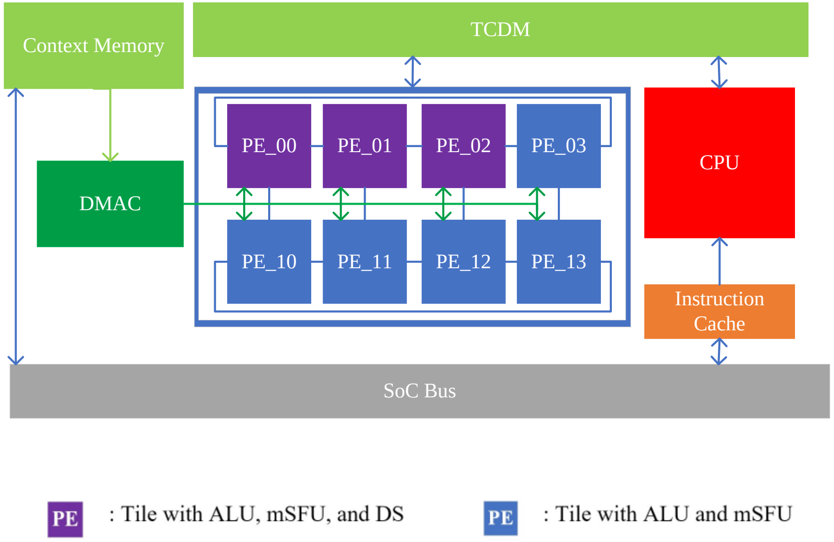
- Exploit a static mapping approach to natively support FP operations.
- Support for multi-cycle operations is introduced.

Hardware

- Use of transprecision computing to maintain energy consumption in ultra-low power domain.
- For optimal use of 32-bit wide datapath, SIMD is introduced.

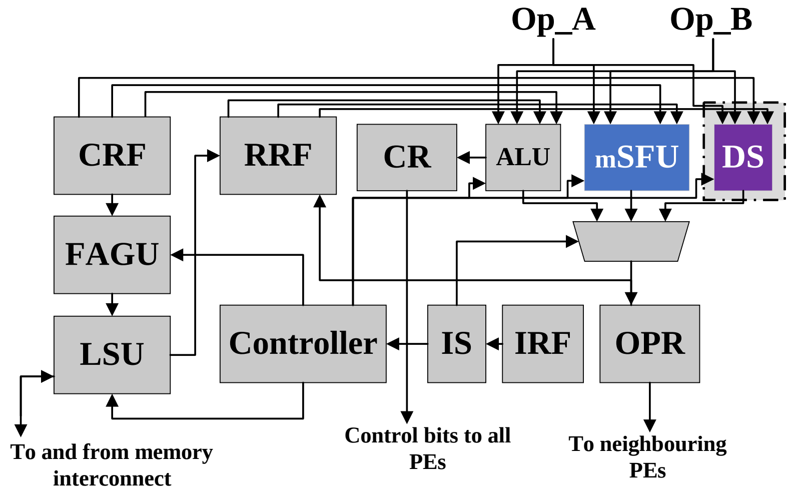
TRANSPIRE: Integrated System

- **TRANSPIRE is a programmable accelerator loosely coupled to a host CPU and shares data through a Tightly Coupled Data Memory (TCDM).**
- **The integrated system consists of 4x2 heterogenous PE array, a DMA controller, and a context memory.**
- **PEs are connected through a mesh torus network for sharing data with adjacent PEs and a bus network for context broadcast.**



TRANSPIRE: Architecture

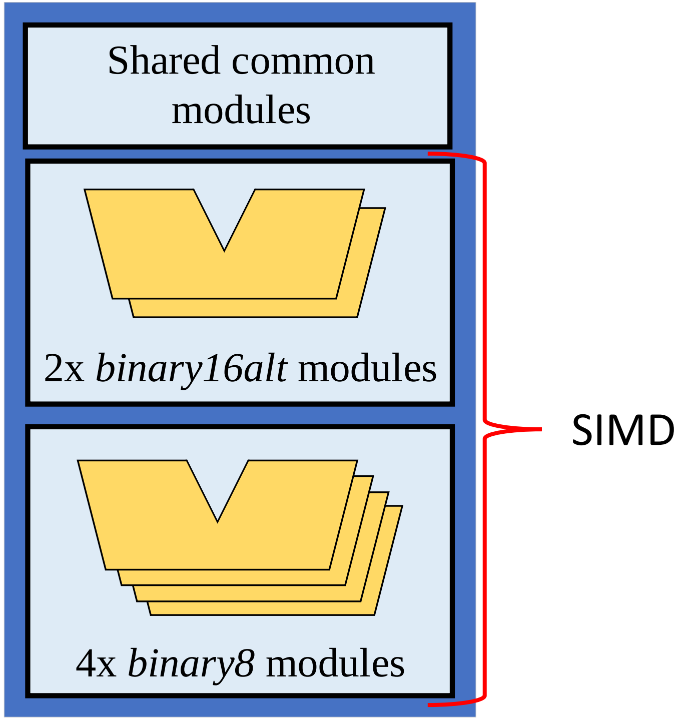
- **TRANSPIRE supports both data and control flow including loops.**
- **TRANSPIRE has its own Instruction Set Architecture (ISA).**
- **Each PE supports both integer and FP datatype operations**
- **Hardware based Flexible Address Generator Unit (FAGU) is also introduced to further accelerate the address computations.**



CRF	: Constant Register File	**CR**	: Condition Register
FAGU	: Flexible Address Generation Unit	**OPR**	: OutPut Register
LSU	: Load Store unit	**ALU**	: Arithmetic Logic Unit
RRF	: Regular Register File	**mSFU**	: mini-smallFLoat Unit
IRF	: Instruction Register File	**DS**	: Divide-Square-root Unit
IS	: Instruction Synchronizer	**PE**	: Processing Element

TRANSPIRE: Architecture (mini-SFU)

- **An mini-SFU (mSFU) includes 2 slices of binary16alt units and 4 slices of binary8 units.**
- **Divide-Square-root (DS) unit includes binary16alt based divide and square-root operators.**
- **Only 1 rounding mode i.e., truncation is used for FP operations.**

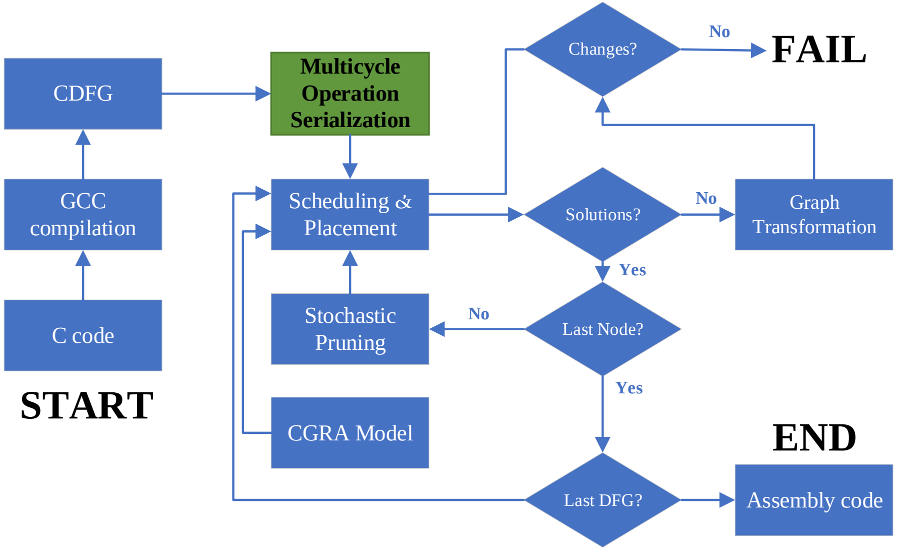


SFU Vs. mSFU

SFU	mSFU
• **1x IEEE-754 binary32, 2x IEEE-754 binary16, 2x binary16alt, 4x binary8**	• **2x binary16alt, 4x binary8**
• **5 rounding mode**	• **1 rounding mode** • **Truncation**
• **15 Operators**	• **7 Operators** • *f_add, f_sub, f_mul, f_div, f_sqrt, f_LT, f_abs*
• **Total cell area: 81,306 µm^2**	• **Total cell area: 3,335 µm^2**

TRANSPIRE: Compilation Flow

- The compilation flow exploits the GCC front-end to get the intermediate representation of the application code.
- TRANSPIRE is modeled as a bipartite directed graph with operator and register nodes.
- The homomorphism between TRANSPIRE model and DFG makes the mapping of CDFG onto TRANSPIRE a sub-graph finding problem.



Reference: S. Das, K. J. M. Martin, P. Coussy, D. Rossi, and L. Benini. Efficient mapping of cdfg onto coarse-grained reconfigurable array architectures. In ASP-DAC 2017, 2017.

TRANSPIRE: Compiler

- Previous version of compiler supported operations with single cycle latency only.
- Code modification is required for identification of FP operations.
- It took only few hours to re-write all of the kernels used in this paper.

Original Code

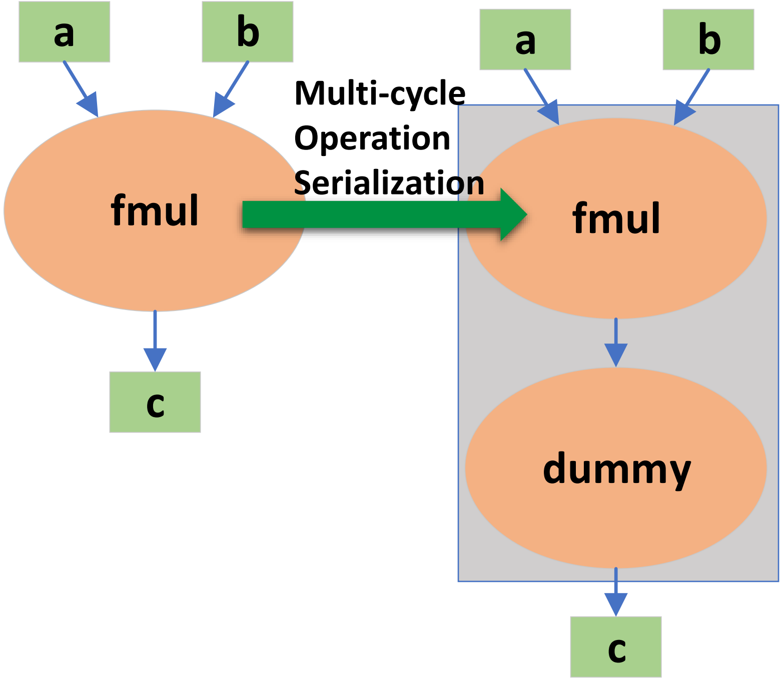
```
int i, j, k; float a[30], b[30], c[256][40], out;
for (i=30; i>0; i--){
   for (j=0; j<40; j++){
      for (k=0; k<256; k++){
         out = a[i] + ( b[i] * c[k][j] );
}}}
```

Modified Code (explicit SIMD)

```
int i, j, k; float a[15], b[15], c[256][20], out;
for (i=15; i>0; i--){
   for (j=0; j<20; j++){
      for (k=0; k<256; k++){
         out = fadd16alt( a[i] , fmul16alt( b[i] , c[k][j] ) );
}}}
```

TRANSPIRE: Static Mapping

- DFG is analyzed and multi-cycle operations are detected.
- Each multi-cycle operation node is then transformed by adding dummy nodes.
- Only first node is sent for mapping and binding.
    - PE is locked for multi-cycle operation
- Obtained mapping is then copied onto dummy nodes with updated (incremented) timestamp.

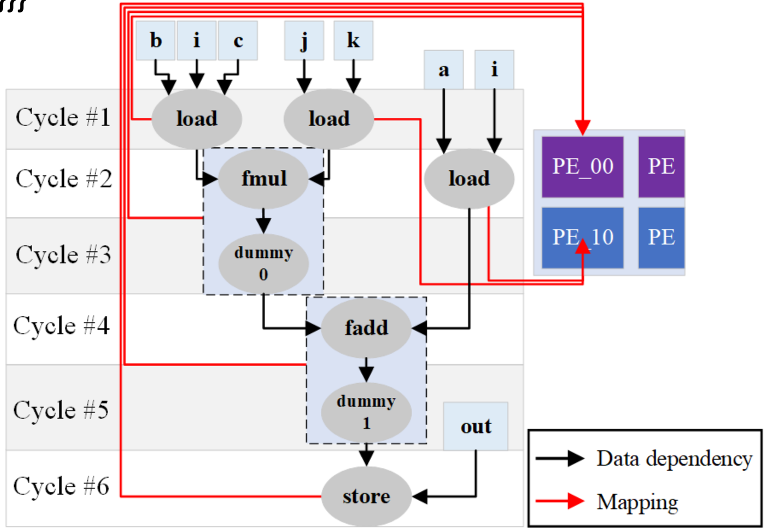


TRANSPIRE: Compiler Challenges

- While mapping multi-cycle operations on TRANSPIRE, there are 2 main challenges to address.

1. All consecutive multi-cycle operations should be mapped onto the same PE.
2. Impose minimum restriction on the algorithm in terms of resource availability.

```
int i, j, k; float a[15], b[15], c[256][20], out;
for (i=15; i>0; i--){
   for (j=0; j<20; j++){
      for (k=0; k<256; k++){
         out = fadd16alt( a[i] , fmul16alt( b[i] , c[k][j] ) );
}}}
```



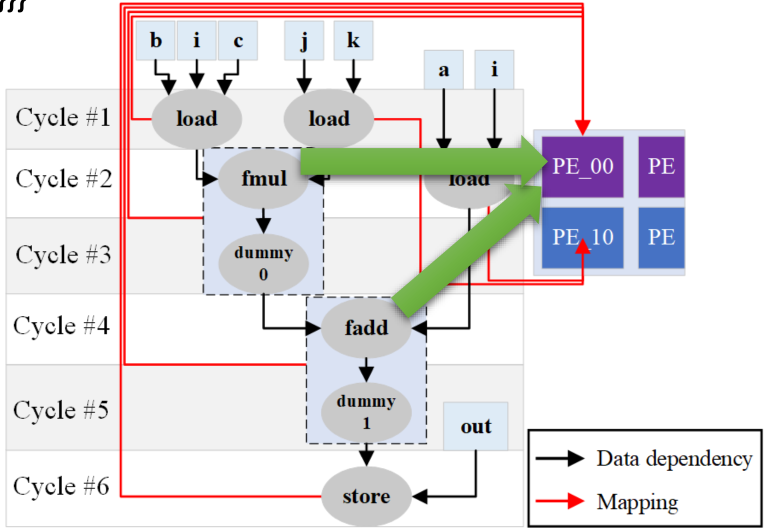
TRANSPIRE: Compiler Challenge #1

- **All consecutive multi-cycle operations should be mapped onto the same PE, to eliminate the chances of undesirable *MOVE* operations.**

Solution:

- **Carefully removing unwanted nodes between 2 consecutive multi-cycle operations which might cause *MOVE* operations.**
- **Updating the algorithm for resource availability after a PE has been locked for performing multi-cycle operation.**

```
int i, j, k; float a[15], b[15], c[256][20], out;
for (i=15; i>0; i--){
   for (j=0; j<20; j++){
      for (k=0; k<256; k++){
         out = fadd16alt( a[i] , fmul16alt( b[i] , c[k][j] ) );
}}}
```



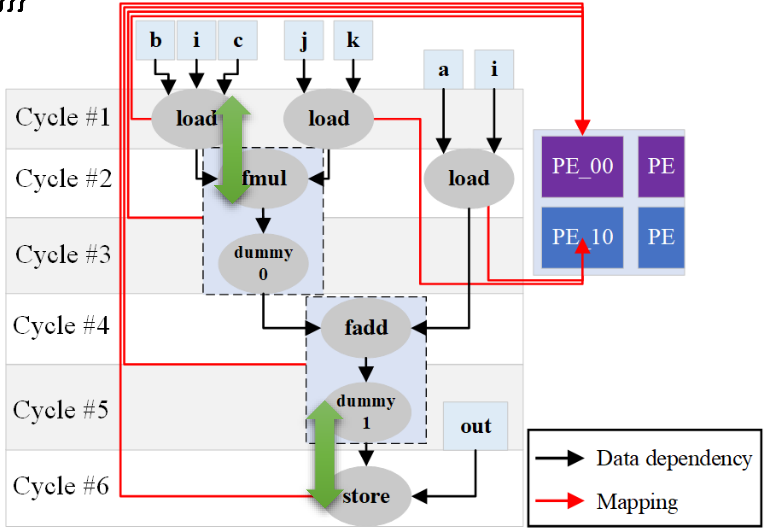
TRANSPIRE: Compiler Challenge #2

- **Impose minimum restriction on the algorithm in terms of resource availability (i.e., minimizing data routing by mapping nodes which share data onto adjacent PEs).**

Solution:

- **Immediately unlock that PE for mapping of the next operation, without consuming any extra cycles.**
- **If not done then, next FP operations is mapped onto other tile resulting in extra *MOVE* operations or decreased PE utilization.**

```
int i, j, k; float a[15], b[15], c[256][20], out;
for (i=15; i>0; i--){
   for (j=0; j<20; j++){
      for (k=0; k<256; k++){
         out = fadd16alt( a[i] , fmul16alt( b[i] , c[k][j] ) );
}}}
```



Agenda

- Introduction
- Background
- TRANSPIRE
- **Experiments & Results**
- Conclusion

Experiments: Kernels

- **These applications implement the fundamental algorithms used in two domains relevant for ultra-low-power systems, near-sensor computing and embedded machine learning.**

	Kernel	Operations Executed	Highest loop iteration	Input Data size (bits)
PCA	*mean_covariance*	397,348	47,104	94,208
	Householder	35,632	1,360	9,216
	Accumulate	106,298	1,240	8,704
	Diagonalize	74,987	2,368	9,216
	PC	168,738	11,776	102,400
	CONV	766,728	4,096	131,072
	DWT	39,456	448	16,384
	SVM	15,630	896	72,000

Accuracy Performance

- **Accuracy Deviation is always less than 10% w.r.t. IEEE-754 FP formats.**

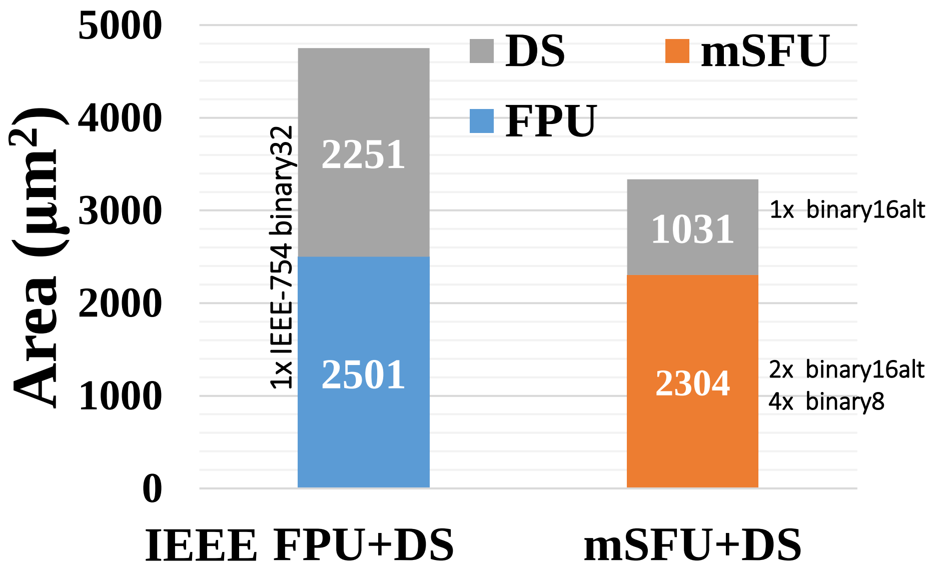
Kernel	Average Deviation (%)	Data-type
mean_covariance	4.80	binary16alt
Householder	0.33	binary16alt
Accumulate	9.03	binary16alt
Diagonalize	5.49	binary16alt
PC	1.54	binary16alt
CONV	2.32	binary8
DWT	6.98	binary8
SVM	7.11	binary8

Experimental Setup

- **All experiments have been performed on a post-synthesis netlist.**
- **28nm UTBB – FDSOI process node**
- **50 MHz frequency, 0.6V operating voltage**
- **Worst-case analysis corner (slow NMOS, slow PMOS, 125°C , low power low *Vt* transistors)**

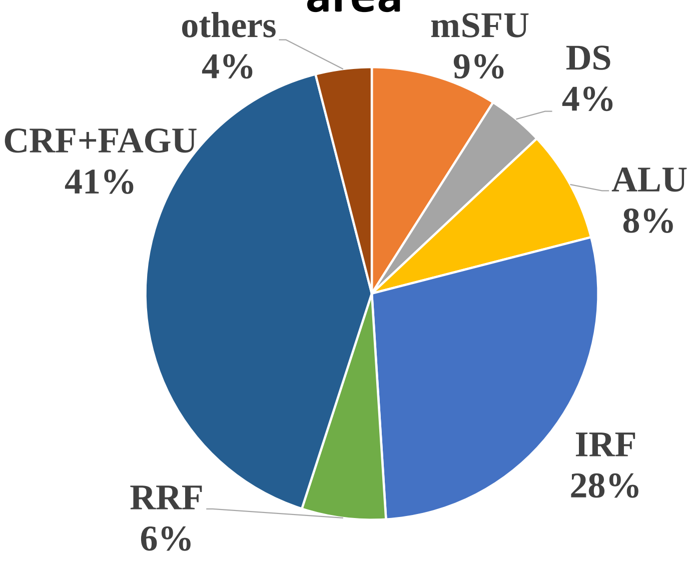
Area Results: PE, mSFU, and DS

mSFU+DS is 1.42x smaller than FPU+DS



🟦 IEEE-754 binary32 FP Unit ⬜ Divide-Square_root Unit

mSFU+DS takes 13% of PE‘s total cell area



Area breakdown of single PE

Area Comparison

- **TRANSPIRE's area overhead is 1.25x only w.r.t. RI5CY_SFU**

	TRANSPIRE (µm^2)	RI5CY_SFU (µm^2)	TRANSPIRE_FPU (µm^2)	RI5CY_FPU (µm^2)
DMA Controller	593	+ 4 KiB Instruction Cache	593	+ 4 KiB Instruction Cache
Interconnect	6,273		6,273	
Context Memory	9,345		9,345	
TCDM	65,164		64,164	
PE Array	186,407		174,230	
Total Cell Area	**267,784**	**213,371**	**255,605**	**185,812**

TRANSPIRE Architecture presented in this paper

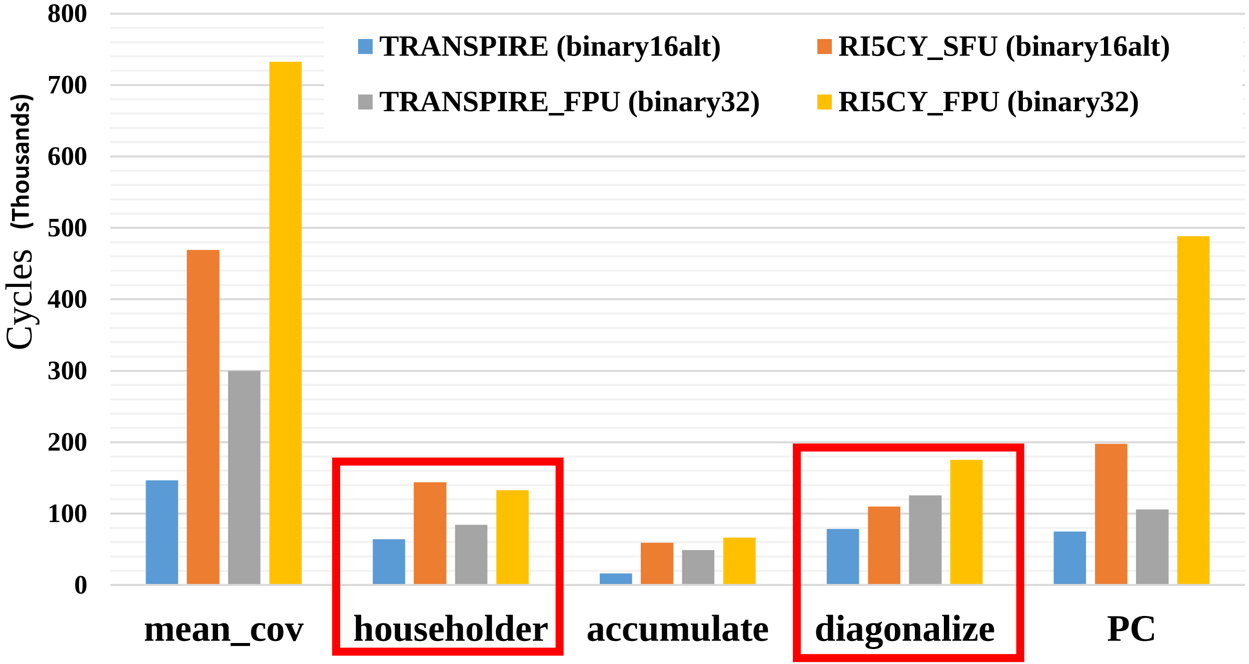
TRANSPIRE_FPU *TRANSPIRE* featuring IEEE FPU

RI5CY_SFU *RISC-V* based CPU featuring SFU

RI5CY_FPU *RISC-V* based CPU featuring IEEE FPU

Performance Results

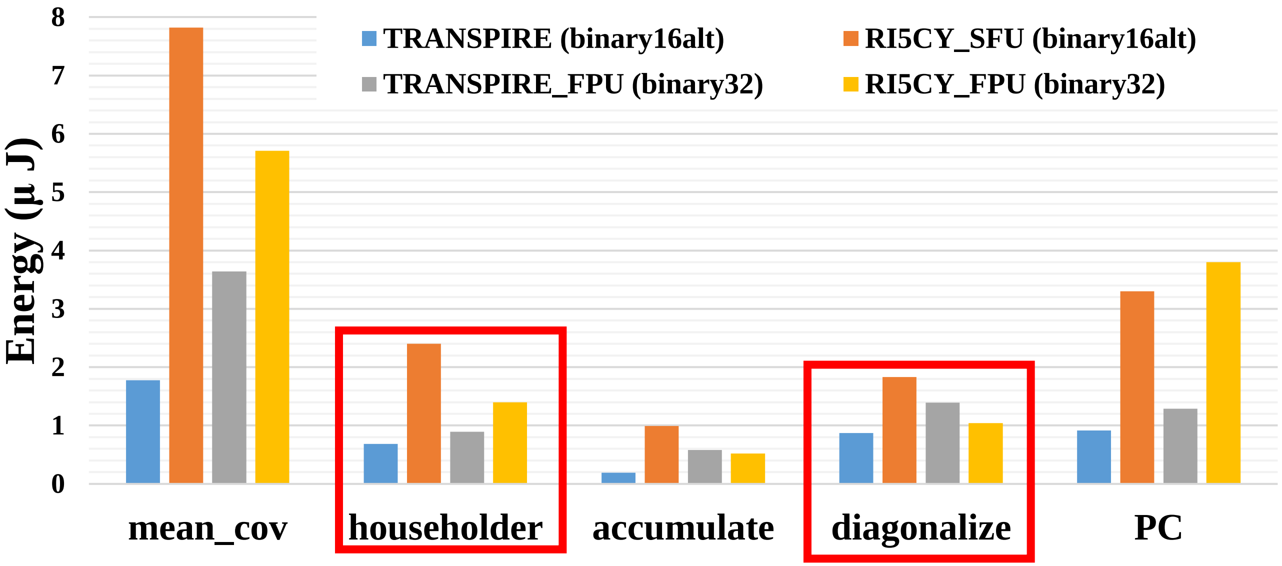
- **TRANSPIRE achieves up to 10.06x better performance w.r.t. RI5CY_SFU**



Kernel	TRANSPIRE (binary8) (cycles)	RI5CY_SFU (binary8) (cycles)
CONV	268,179	1 455,097
DWT	11,140	16,912
SVM	11,408	114,747

Energy Results

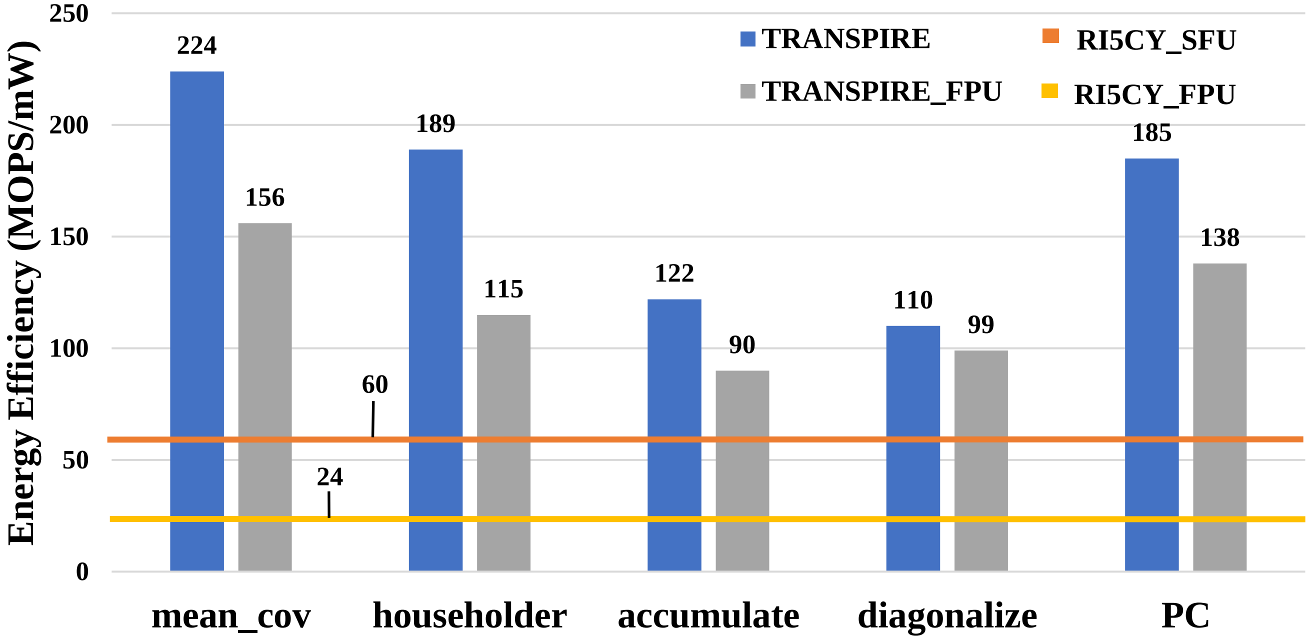
- **Energy consumption of TRANSPIRE is up to 12.91x less w.r.t. RI5CY_SFU**



Kernel	TRANSPIRE (binary8) (uJ)	RI5CY_SFU (binary8) (uJ)
CONV	3.036	21.506
DWT	0.124	0.256
SVM	0.123	1.588

Energy-efficiency Results

- **TRANSPIRE reaches a maximum of 224 MOPS/mW**



Agenda

- Introduction
- Background
- TRANSPIRE
- Experiments & Results
- **Conclusion**

Conclusion

- Exploit a static mapping approach to natively support FP operations in CGRA together with transprecision computing to maintain energy consumption in ultra-low power domain.
- TRANSPIRE achieves a maximum of 10.06x performance gain w.r.t RI5CY_SFU
- TRANSPIRE consumes up to 12.91x less energy w.r.t RI5CY_SFU
- Area overhead is 1.25x only w.r.t RI5CY_SFU

THANK YOU

References

1. M. Jo, D. Lee, K. Han, and K. Choi. Design of a coarse-grained reconfigurable architecture with floating-point support and comparative study. Integration, the VLSI Journal, 47, Jan. 2013
2. C. Nicol. A Coarse Grain Reconfigurable Array (CGRA) for statically scheduled data flow computing. WAVE Computing, 2016.
3. C. Brunelli, F. Garzia, D. Rossi, and J. Nurmi. A Coarse-grain Reconfigurable Architecture for Multimedia Applications Supporting Subword and Floating-point Calculations. J. Syst. Archit., 56(1), 2010
4. X. Fan, D. Wu, W. Cao, W. Luk, and L. Wang. Stream Processing Dual-Track CGRA for Object Inference. IEEE Transactions on VLSI Systems, 26(6), 2018.
5. G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini. A transprecision floating-point platform for ultra-low power computing. In DATE 2018, 2018.
6. S. Mach, F. Schuiki, F. Zaruba, and L. Benini. A 0.80 pj/flop, 1.24 Tflop/sW 8-to-64 bit Transprecision Floating-Point Unit for a 64 bit RISC-V Processor in 22 nm FD-SOI. In VLSI-SOC, 2019
7. S. Das, K. J. M. Martin, P. Coussy, D. Rossi, and L. Benini. Efficient mapping of cdfg onto coarse-grained reconfigurable array architectures. In ASP-DAC 2017, 2017.
8. Chapter 8: Fixed Point vs. Floating Point , *DSP System Design: Using the TMS320C6000 by Nasser Kehtarnavaz , Mansour Keramat*
9. STM32L4 MCU series: Excellence in ultra-low-power with performance. STM32 Ultra Low Power MCUs, 2018.
10. S. Mach, D. Rossi, G. Tagliavini, A. Marongiu, and L. Benini. A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing. In ISCAS, pages 1–5, May 2018.
11. S. Das, D. Rossi, K. J. M. Martin, P. Coussy, L. Benini, A 142MOPS/mW integrated programmable array accelerator for smart visual processing, ISCAS, 2017.