Bridging High-Level Synthesis and Application-Specific Arithmetic: The Case Study of Floating-Point Summations

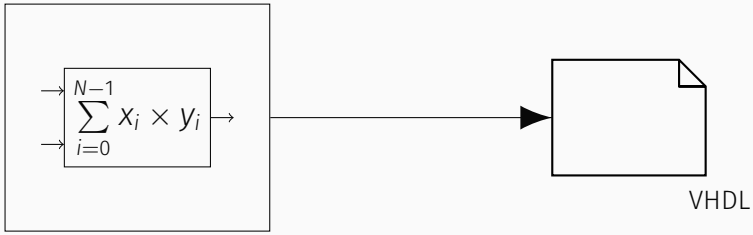
Yohann Uguen **Florent de Dinechin** **Steven Derrien**

September, FPL2017

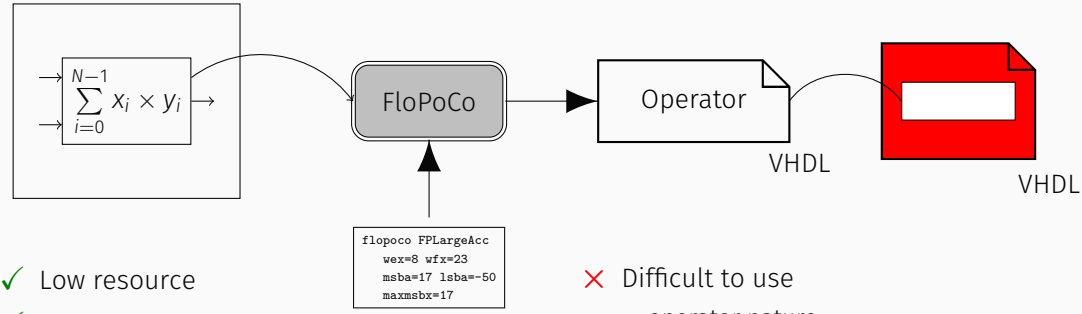
University of Lyon, INSA Lyon, Inria, CITI
F-69621 Villeurbanne, France
{Yohann.Uguen, Florent.de-Dinechin}@insa-lyon.fr

University Rennes 1, IRISA
Rennes, France
Steven.Derrien@univ-rennes1.fr

Circuit: from specification to implementation?



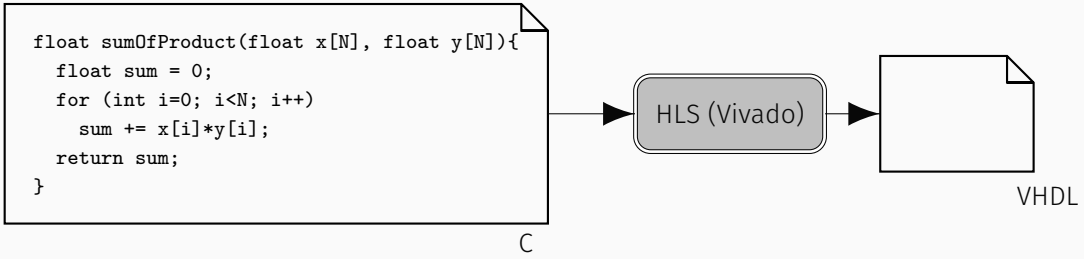
Arithmeticians point of view: highly tuned hardware, custom datatypes



- ✓ Low resource
- ✓ Low latency
- ✓ Accuracy control

- ✗ Difficult to use
  - operator nature
  - FloPoCo's interface
  - integration of the operator

Compiler designers point of view: follow language semantics and datatypes



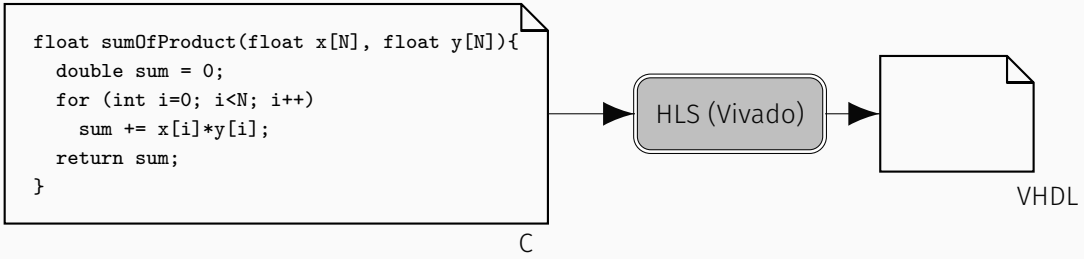
✓ Ease of use

✗ Moderate resource

✗ Moderate latency

✗ No accuracy control

Compiler designers point of view: follow language semantics and datatypes



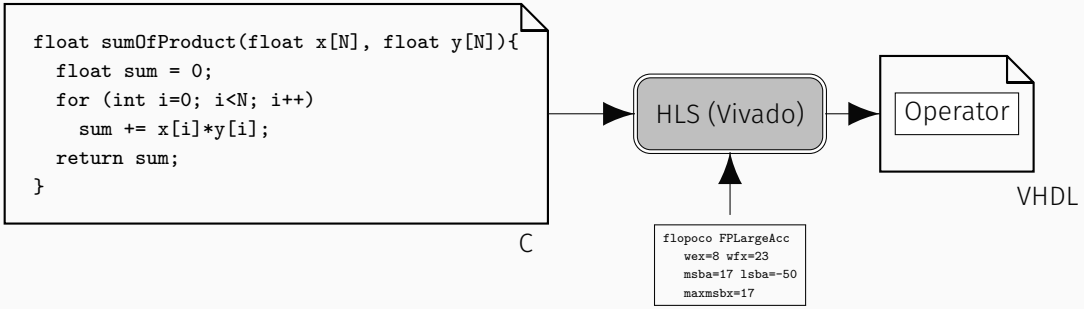
✓ Ease of use

✗ Moderate resource

✗ Moderate latency

✗ No accuracy control

Naive solution : interface FloPoCo operators with HLS



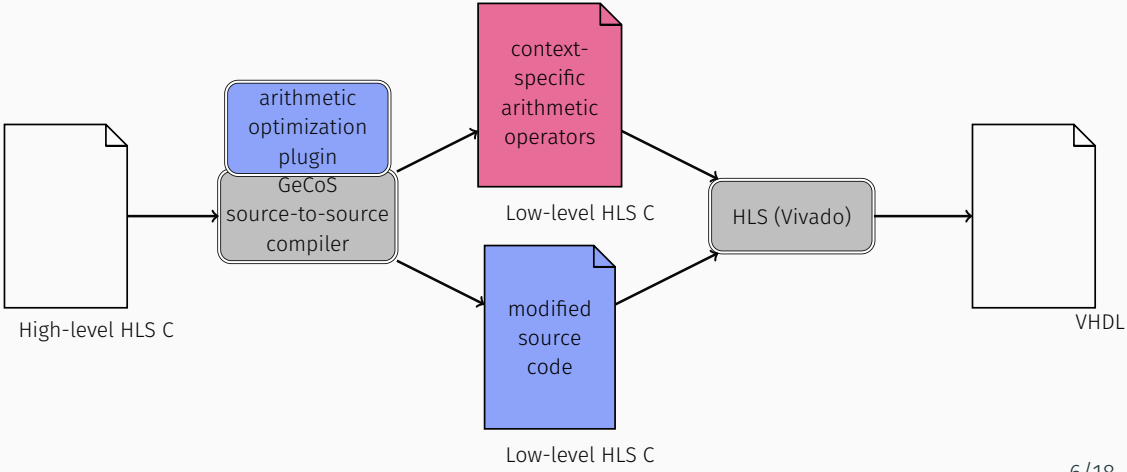
- ✓ Custom datatypes
- ✓ Better resource/accuracy control
- ✗ Vivado HLS strongly discourages it
- ✗ Sub-optimal compiler optimizations
- ✗ Can't simulate circuit in C

Proposed method : embed FloPoCo's spirit in HLS compilers

- ✓ Ease of use
- ✓ Low resource
- ✓ Low latency
- ✓ Accuracy control

- Transform variable datatypes when specified
- Better than custom datatypes: a high-level pragma
- Vivado HLS compliant

HLS tools can perform IR optimizations


arithmetic optimization plugin
GeCoS source-to-source compiler
High-level HLS C
context-specific arithmetic operators
Low-level HLS C
modified source code
Low-level HLS C
HLS (Vivado)
VHDL

SUMMARY

The arithmetic side:
Exact accumulator in Vivado HLS

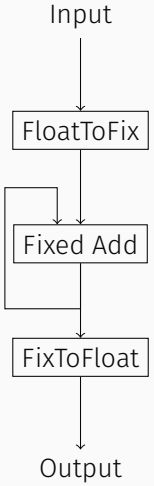
The compiler side:
Source-to-source transformations using GeCoS

Evaluation

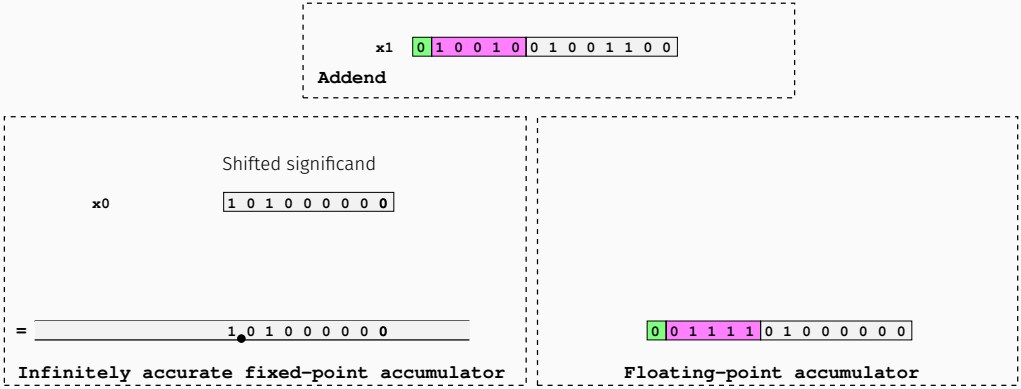
Conclusion

THE ARITHMETIC SIDE: EXACT ACCUMULATOR IN VIVADO HLS

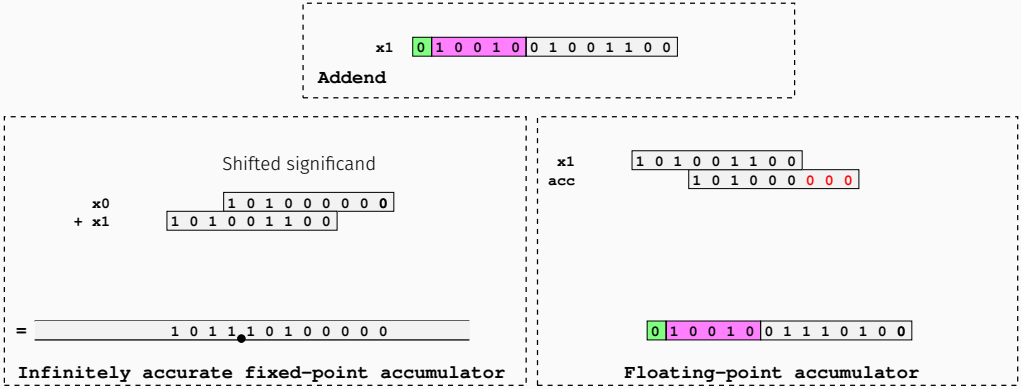
- Fixed-point accumulator:
  custom application specific format
- Based on [An FPGA-specific approach to floating-point accumulation and sum-of-products, 2008]
- Minor improvements
  - support for sub-normals
  - increased modularity
- Each box tailored to application requirements



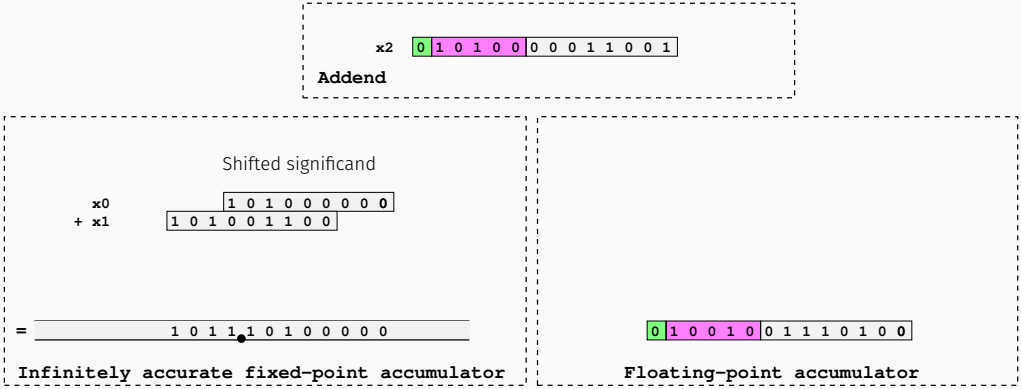
FIXED-POINT VS FLOATING POINT ACCUMULATOR



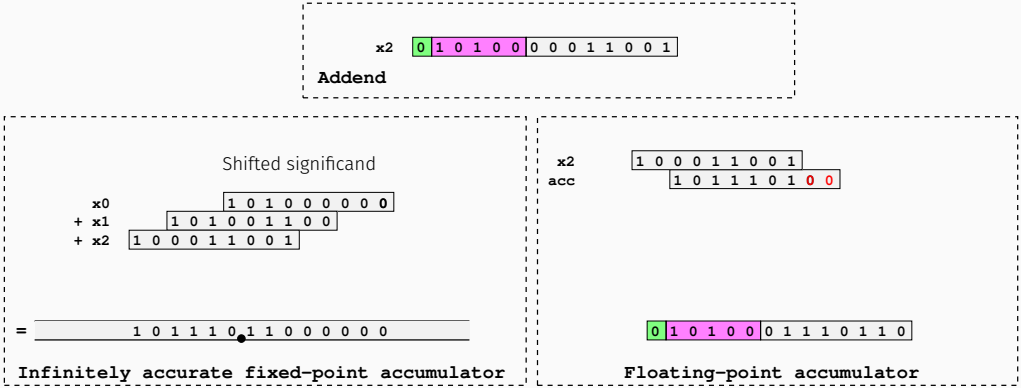
FIXED-POINT VS FLOATING POINT ACCUMULATOR



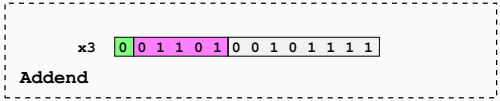
FIXED-POINT VS FLOATING POINT ACCUMULATOR

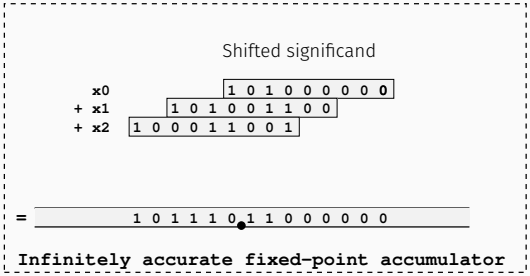


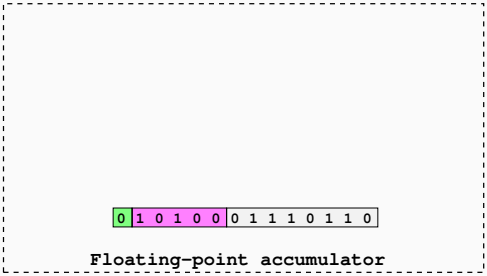
FIXED-POINT VS FLOATING POINT ACCUMULATOR



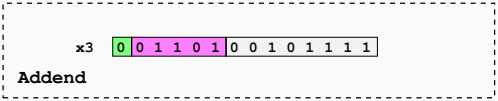
FIXED-POINT VS FLOATING POINT ACCUMULATOR

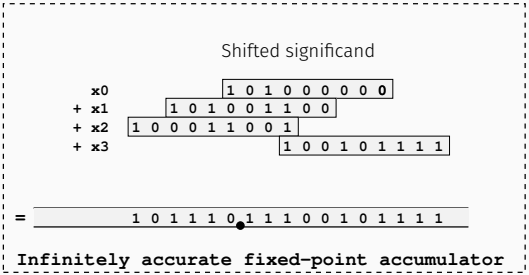


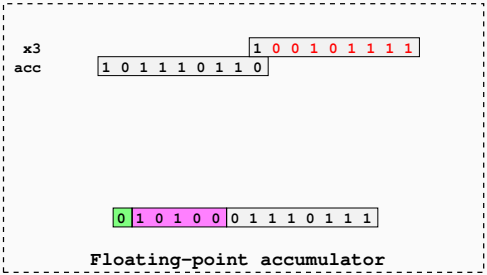




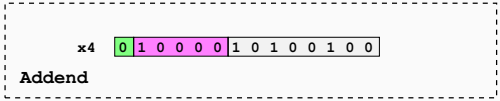
FIXED-POINT VS FLOATING POINT ACCUMULATOR

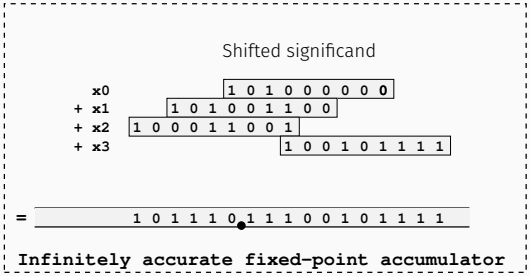


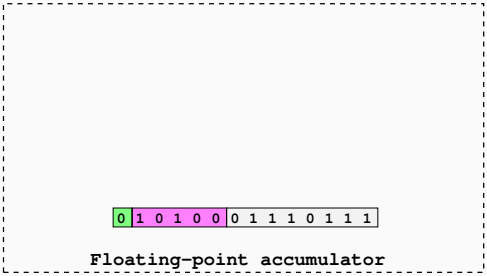




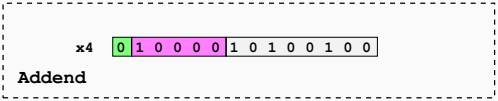
FIXED-POINT VS FLOATING POINT ACCUMULATOR

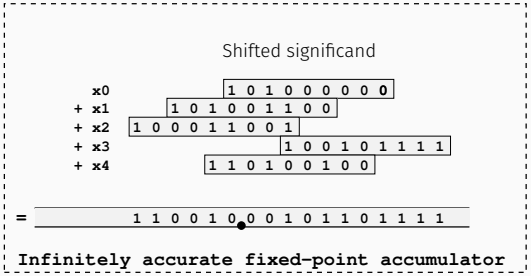


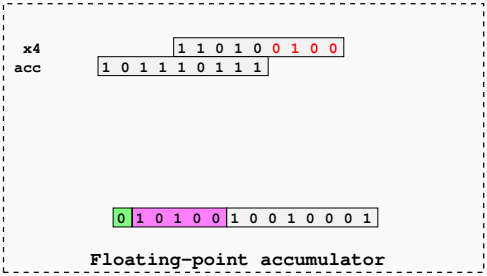




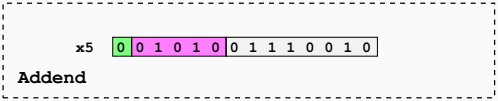
FIXED-POINT VS FLOATING POINT ACCUMULATOR

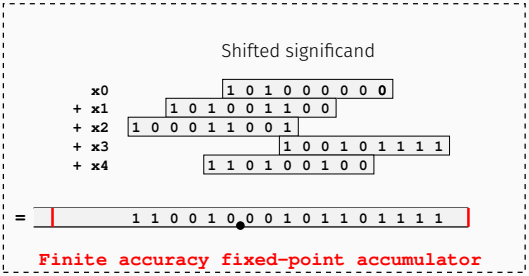


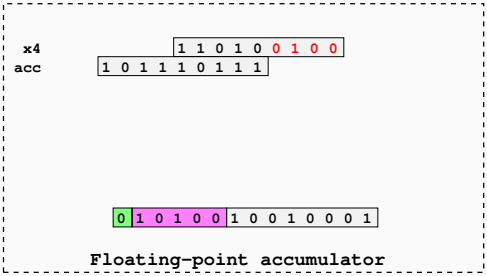




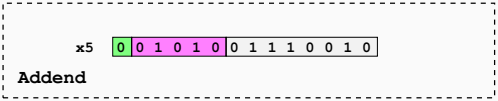
FIXED-POINT VS FLOATING POINT ACCUMULATOR

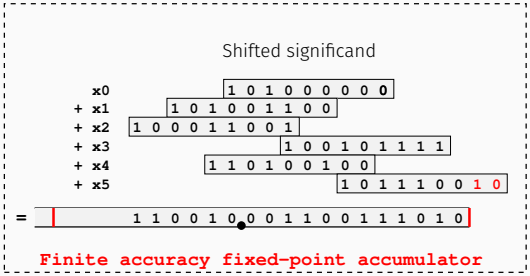


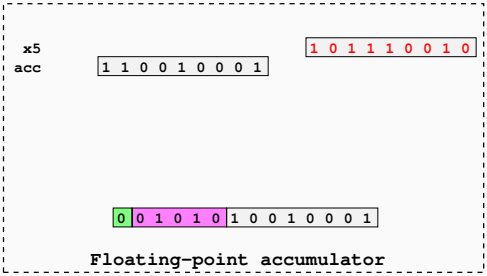




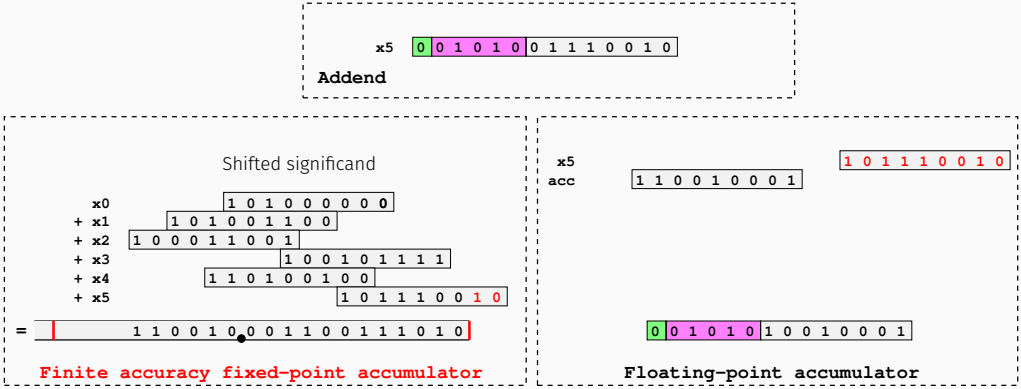
FIXED-POINT VS FLOATING POINT ACCUMULATOR







FIXED-POINT VS FLOATING POINT ACCUMULATOR



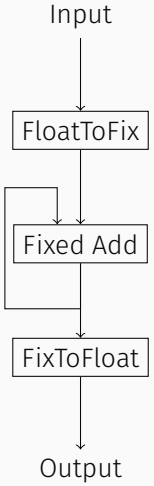
Accuracy:

Exact Result	=	50.2017822265625
FP Acc	=	50.125
Fixed-Point Acc	=	50.20166015625

Vivado HLS compliant C

Same results as FloPoCo operator in $\sim$ 50 lines of code in terms of:

- LUTs
- Registers
- DSPs



THE COMPILER SIDE:
SOURCE-TO-SOURCE TRANSFORMATIONS USING GECOS

Use of a pragma

```c
#define N 100000
float computeSum(float in1[N], float in2[N]){
        float sum = 0;
        for (int i=1; i<N-1; i++){
                sum += in1[i]*in2[i-1];
                sum += in1[i];
                sum += in2[i+1];
        }
        return sum;
}
```

C

Use of a pragma

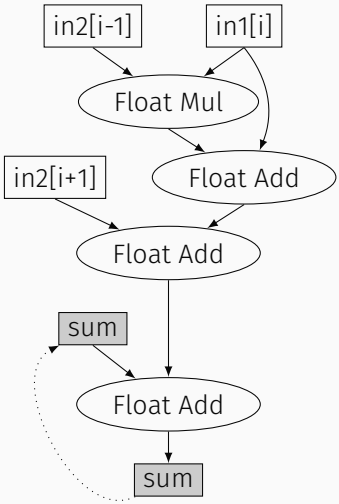
```c
#define N 100000
float computeSum(float in1[N], float in2[N]){
        float sum = 0;
        #pragma FPacc VAR=sum MaxAcc=300000.0 epsilon=1e-15
        for (int i=1; i<N-1; i++){
                sum += in1[i]*in2[i-1];
                sum += in1[i];
                sum += in2[i+1];
        }
        return sum;
}
```

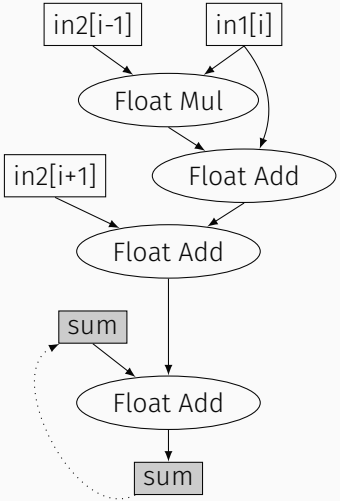
C

Build a GeCoS plug-in

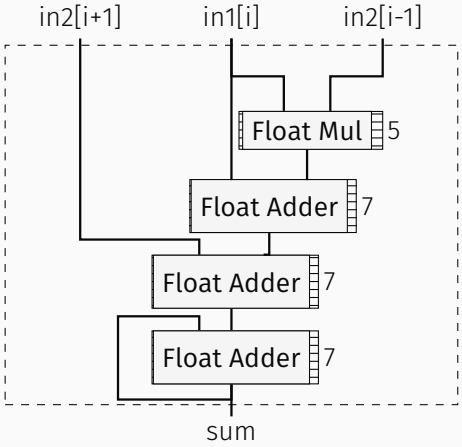
- Read C code
- Transform to compiler IR
- Retrieve DFG associated to pragma
- Apply transformations
- Regenerate C code

```
sum += in1[i]*in2[i-1];
sum += in1[i];
sum += in2[i+1];
```

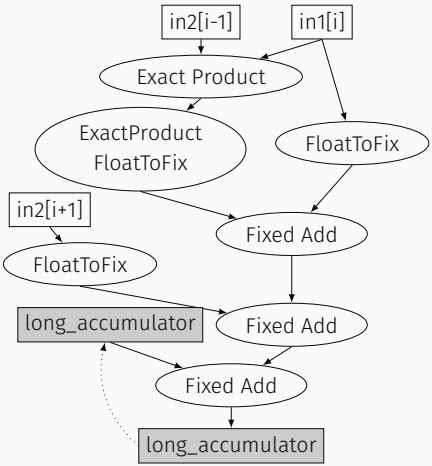



in2[i-1]
in1[i]
Float Mul
in2[i+1]
Float Add
Float Add
sum
Float Add
sum

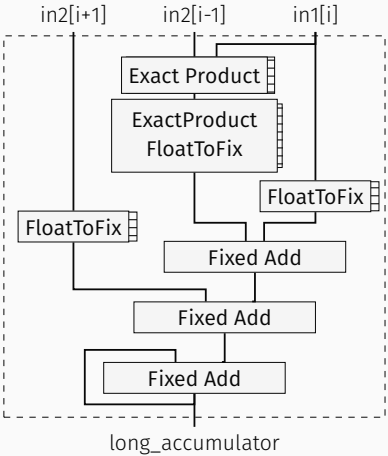
DFG


in2[i+1]
in1[i]
in2[i-1]
Float Mul
5
Float Adder
7
Float Adder
7
Float Adder
7
sum

Circuit

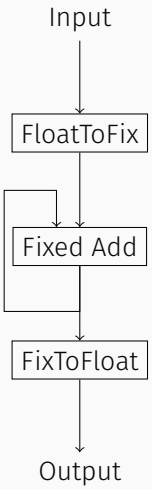


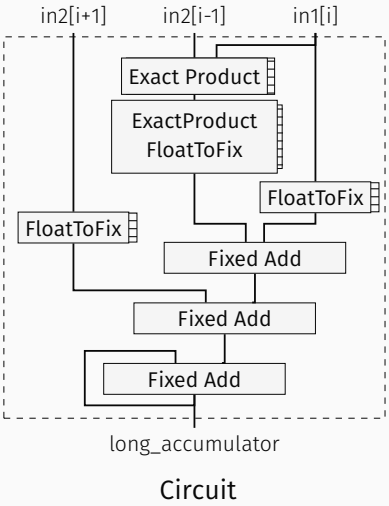
DFG



Circuit

This CANNOT be obtained using FloPoCo





Circuit

This CANNOT be obtained using FloPoCo

EVALUATION

Evaluation of the previous example

Single cycle accumulation improves latency

- After place-and-route
- For 100MHz
- On Xilinx Kintex 7
- Using Vivado HLS 2016.3
- Using Vivado 2016.3

	Original no pragma	Transformed epsilon=1e-7	Transformed epsilon=1e-10	Transformed epsilon=1e-19
LUTs	538	693	824	1400
DSPs	5	2	2	2
Latency (N=100K)	2000K	100K	100K	100K

FP benchmark suite

- 10 kernels
- 5 contains accumulations

Methodology

- Implemented a golden reference (MPFR) for accuracy
- Synthetized the 5 kernels w/ and w/o transformations
- Accumulator tailored to be just more accurate than original benchmarks

Three types of results

1. One kernel gets +33% LUT usage, -60% DSPs for a 400% speed improvement

2. Two kernels different loop sizes: between +24% and +26% LUT usage, -60% DSPs for a speed improvement between 480% and 840%

3. Two kernels have less than 1% difference for resource usage but unpredictable latency
   - Better latency observed by simulation
   - Benchmarks should be completely re-written to be suitable for HLS

CONCLUSION

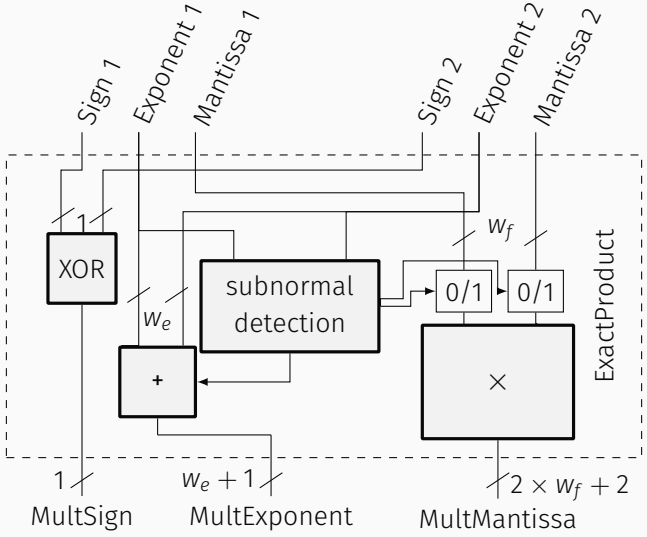
Arithmetic optimizations and HLS are compatible

- At least for sums-of-products
- Tool distributed with GeCoS
  - `http://gecos.gforge.inria.fr/doku/doku.php?id=subprojects:arithmetic_optimizations_for_hls`

Future work

- Support more operators
- Make more use of compiler knowledge such as loop counts when possible
- Integration to a HLS tool

Thank you


Sign 1
Exponent 1
Mantissa 1
Sign 2
Exponent 2
Mantissa 2
1
XOR
w_e
subnormal detection
w_f
0/1
0/1
+
×
ExactProduct
1
MultSign
$w_e + 1$
MultExponent
$2 \times w_f + 2$
MultMantissa

Benchmark	Type	LUTs	DSPs	Latency	Accuracy
Livermore	Original	384	5	80K	11 bits
	Transformed	576 (+33%)	2 (-60%)	20K (/4)	13 bits
LU-8	Original	809	5	82	8-23 bits
	Transformed	1007 (+24%)	2 (-60%)	17 (/4.8)	23 bits
LU-45	Original	819	5	452	8-23 bits
	Transformed	1034 (+26%)	2 (-60%)	54 (/8.4)	23 bits
Scholes	Original	15640	175	N/A	19 bits
	Transformed	15923 (+1%)	175 (+0%)	N/A	23 bits
Fourier	Original	34596	64	N/A	6 bits
	Transformed	34681 (+1%)	59 (-8%)	N/A	11 bits

ACCUMULATOR - ARCHITECTURE