September 14th 2018, Siena, Italy

***3rd Italian Workshop on Embedded* Systems (IWES)**

Funded by the H2020 Framework Programme of the European Union

Hardware and Software Support for Transprecision Computing on Ultra-Low-Power Embedded Systems

Giuseppe Tagliavini
Michela Milano
Luca Benini

DEI – Department of Electronic Engineering
DISI - Department of Computer Science and Engineering
University of Bologna
Bologna, Italy

OPRECOMP
Open Transprecision Computing

Agenda

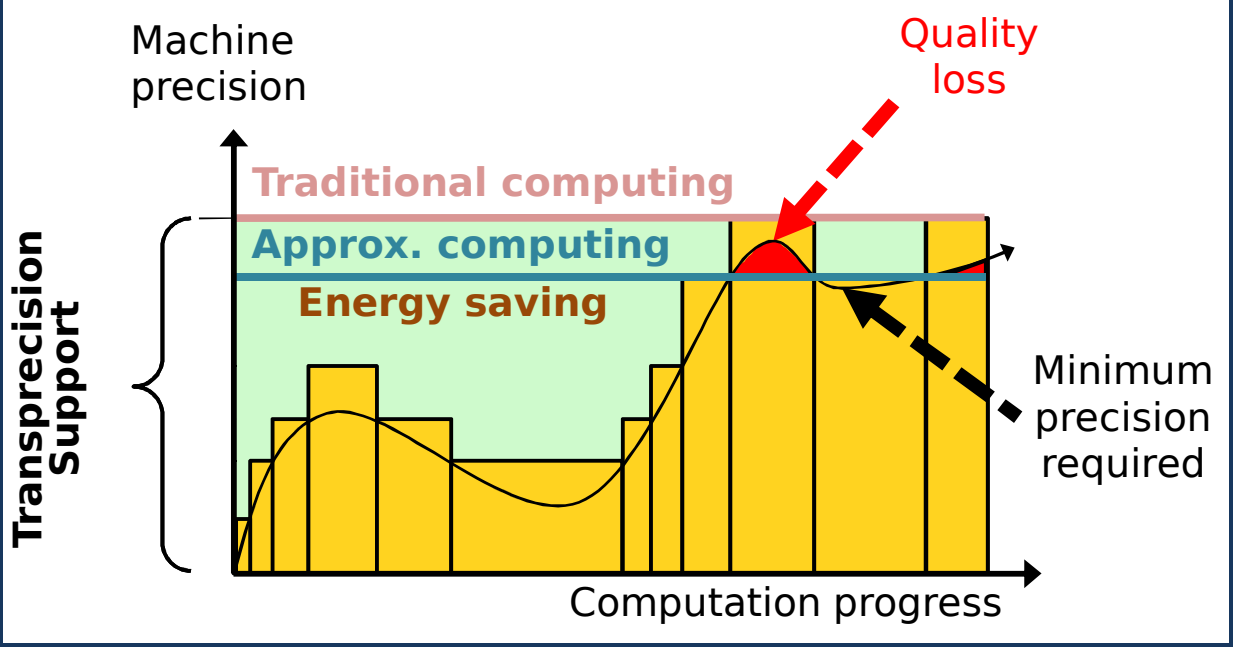
- **Introduction – Transprecision Computing**
- *Smaller-than-32-bit* floating point types
- Implementing the *smallFloat* extension
  - HW support
  - Compiler support
- Simplifying the deployment of *SmallFloat-based* applications
- Conclusion

Towards a new computing paradigm: **Transprecision Computing**

Beyond approximate computing!
A transprecision computing framework:

- controls approximation in space and time (when and where) at a fine grain though multiple hardware and software feedback control loops.
- does not imply reduced precision at the application level
  - it is still possible to soften precision requirements for extra benefits.
- defines computing architectures that operate with a smooth and wide range of precision vs. cost trade-off curve.

OPRECOMP
Open Transprecision Computing [5]



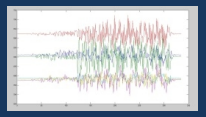
[5]: Malossi et al.: The Transprecision Computing Paradigm: Concept, Design, and Applications. DATE 2018, 2018.

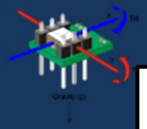
Towards a new computing paradigm: **Transprecision Computing**

Context: Distributed Embedded Computing

Sense

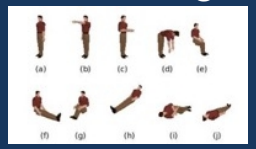
MEMS IMU



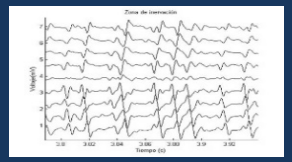


MEMS Microphone

ULP Imager



EMG/ECG/EIT



100 µW ÷ 2 mW

Analyze and Classify

Low Power, High Performance

- Data processing usually requires FP support
- HW support needed for performance (speed)
- Up to 50% of processor power for FP-related operations. [1]

→ Make processing more **energy efficient** on a **system level**

IOs

1 ÷ 2000 MOPS
1 ÷ 10 mW

Transmit

Short range, medium BW

LoRa SEMTECH

Long range, low BW

Low rate (periodic) data

Idle: ~1µW
Active: ~ 50mW

[1]: Tagliavini et al.: A Transprecision Floating-Point Platform for Ultra-Low Power Computing. DATE 2018, 2018.

Agenda

- Introduction – Transprecision Computing
- ***Smaller-than-32-bit* floating point types**
- Implementing the *smallFloat* extension
  - HW support
  - Compiler support
- Simplifying the deployment of *SmallFloat-based* applications
- Conclusion

Floating point formats

- Floating-point (FP) formats are widely adopted to design applications characterized by a **large dynamic range**

- IEEE 754 specification defines an encoding format that breaks a FP number into 3 parts:
  a ***sign***, a ***mantissa***, and an ***exponent***
  - **exponent** ⇔ ***dynamic range***
  - **mantissa** ⇔ ***precision***

The Need for Floating-Point Arithmetic

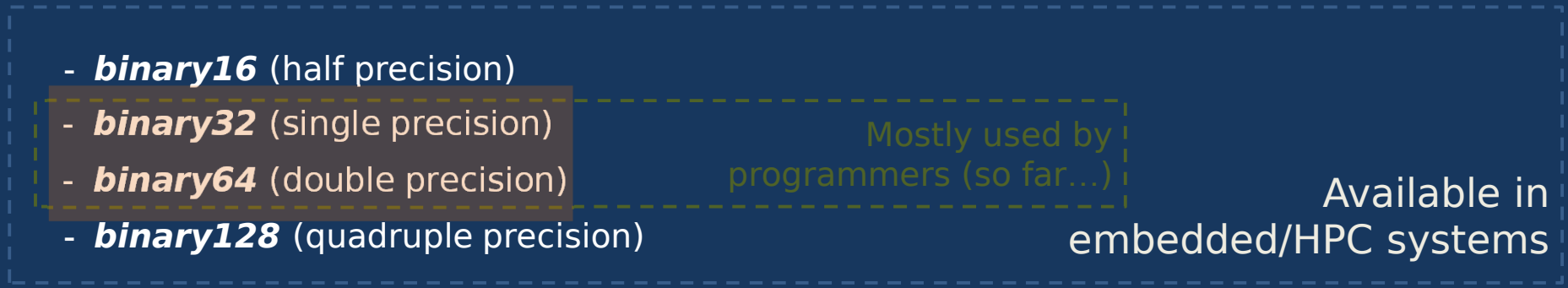
Do we need **floating-point** at all?

- **Fixed-Point?**
  - Not enough flexibility (dynamic range)
- **Logarithmic Number Systems (LNS)?**
  - Add/Subtract very expensive [1]
- **UNUM?**
  - Unwieldy for LP HW implementation [2]

[1] Gautschi et al.: An Extended Shared Logarithmic Unit for Nonlinear Function Kernel Acceleration in a 65-nm CMOS Multicore Cluster. IEEE Journal of Solid-State Circuits, 52(1):98–112, 2017.
[2] Glaser et al.: An 826 MOPS, 210 uW/MHz Unum ALU in 65 nm. ISCAS 2018

The Need for Floating-Point Arithmetic

- **IEEE 754-2008 standard types**



Smaller-than-32bit floating point types one step further

1) How much precision do we actually need?

- **Actual levels of precision are quite limited**
  - Why stop there?
  - Which ones are useful? [3]

2) How to simplify deployment of applications with *smaller-than-32-bit* floats?

[3]: Tagliavini et al.: A Transprecision Floating-Point Platform for Ultra-Low Power Computing. DATE 2018, 2018.

Smaller-than-32bit floating point types one step further

SmallFloat formats for transprecision computing

- *Smaller-than-32-bit* FP formats (**smallFloats** can reduce execution time and energy consumption
  - **Simpler logic in arithmetic units**
  - **Vectorization**
  - **Bandwidth reduction**

***SmallFloat* extension of a standard FP type system**

- Need architecture support
- Need compiler support (language frontend, machine backend)

How to address the two key goals?

1. Supporting the *SmallFloat* data type extension
   - Hardware Support
   - Compiler Support

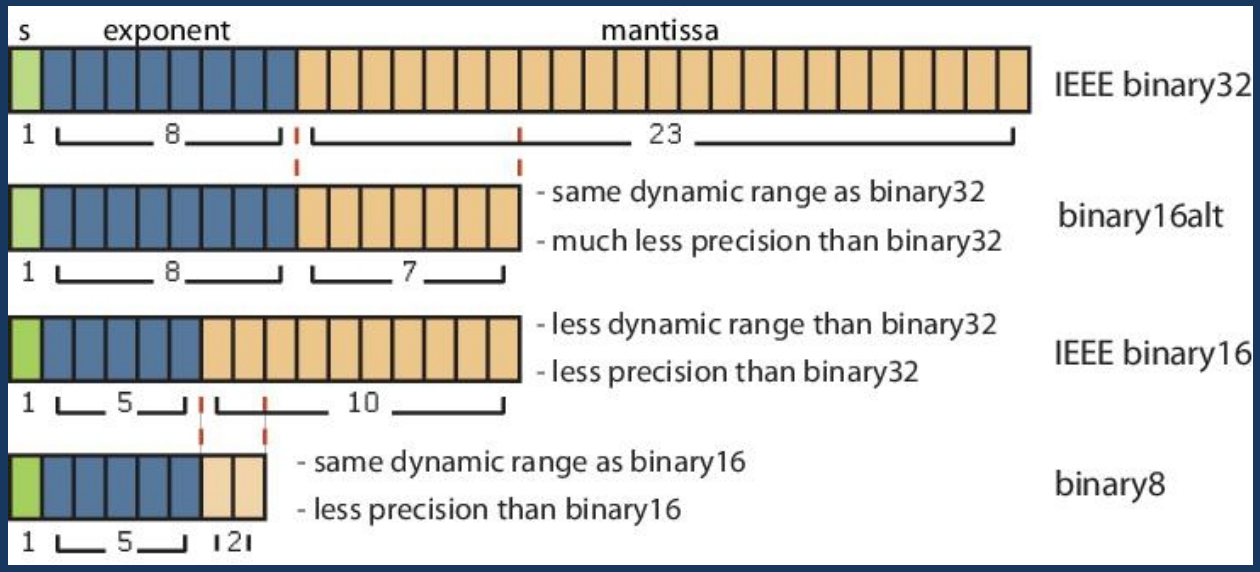
2. Simplifying the deployment of *SmallFloat-based* applications
   - SmallFloat emulation
   - Precision Tuning
   - Automation (compiler support)

Agenda

- Introduction – Transprecision Computing
- *Smaller-than-32-bit* floating point types
- **Implementing the *smallFloat* extension**
  - HW support
  - Compiler support
- Simplifying the deployment of *SmallFloat-based* applications
- Conclusion

smallFloat type system

- ❑ Preliminary experiments [1] motivate *smaller-than-32-bit* FP types
- ❑ Several alternatives are possible. A few useful ones have been defined already.



Some applications require large dynamic range…

…some others require higher precision

[1] Giuseppe Tagliavini, Stefan Mach, Andrea Marongiu, Davide Rossi, Luca Benini
A Transprecision Floating-Point Platform for Ultra-Low Power Computing
In Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1051-1056. IEEE, 2018.

1) Supporting the SmallFloat data type extension

Hardware Support (1): The **PULP** Platform

pulp-platform.org

- Open-source *ultra-low-power* computing platform by **ETH Zürich** and **University of Bologna**
- Based on the open-source **RISC-V** instruction set architecture
  - extensible without breaking official RISC-V support

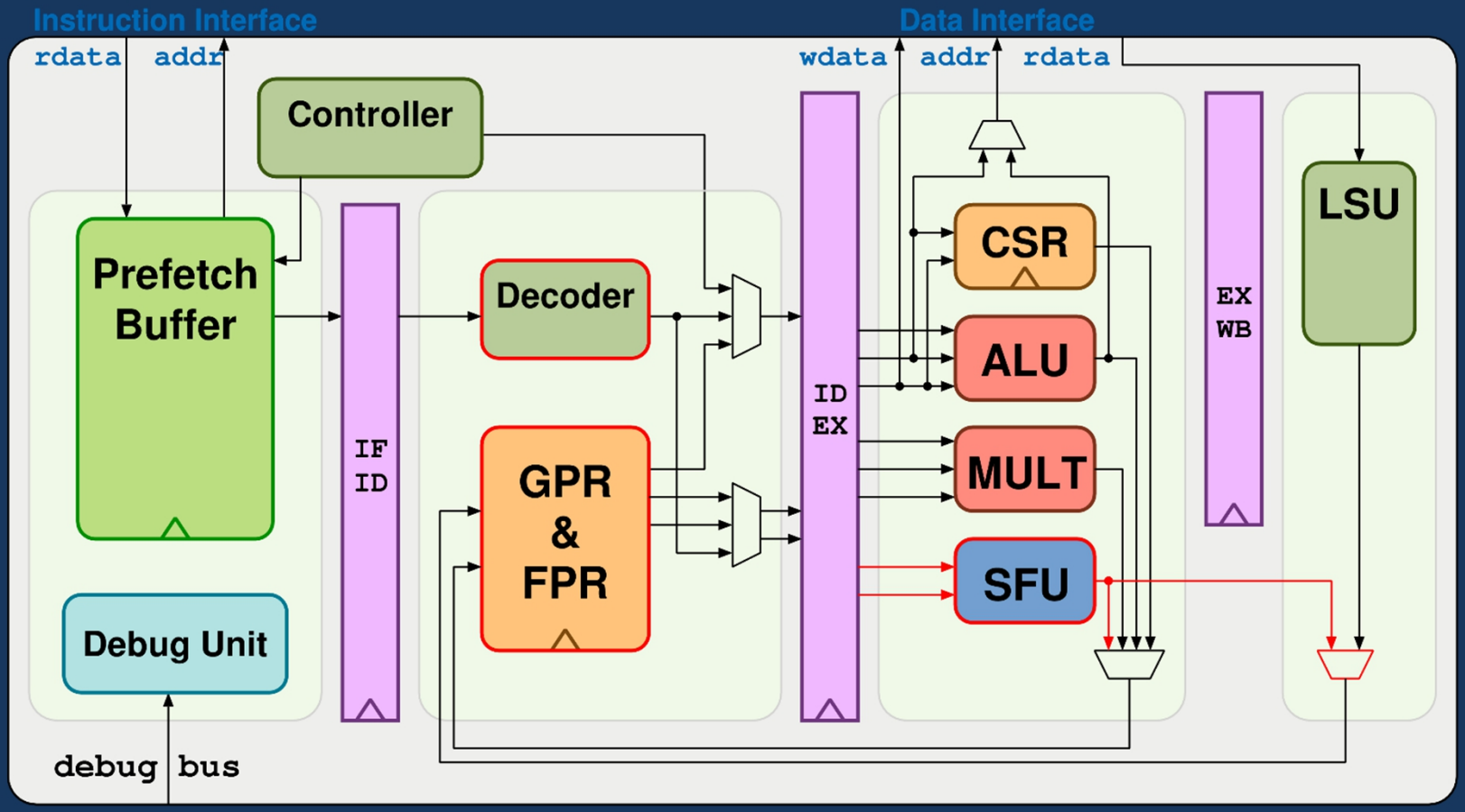
1) Supporting the SmallFloat data type extension

Hardware Support (2): **Goals for SmallFloat HW**

- Provide ***smallFloat*** formats in RISCV core
  - Computational operations (ADD, SUB, MUL)
  - Conversions between integers and FP formats, and among FP formats
- **Vectorize** reduced-precision operations – 2x 16bit or 4x 8bit
- smallFloat operations (16bit, 8bit) and conversions in **single cycle**
- RISC-V **ISA extensions** to handle new formats/instructions

1) Supporting the SmallFloat data type extension

smallFloat Unit – Core integration



Energy consumption of SmallFloat operations

Format	Operation	Instruction (smallFloat ISA extension)	Energy
	Idle Cycle	nop	62.2 pJ
int32	Data movement	lw,sw	94.4 pJ
	Arithmetic	add,mul	106.4 pJ
float32	Arithmetic	f{add,mul}.s	106.8 pJ
	Conversions	fcvt.s.X	79.7 pJ
float16	Arithmetic	f{add,mul}.h	98.8 pJ
	Conversions	fcvt.h.X	74.7 pJ
	Vector Arithmetic	vf{add,mul}.h	132.6 pJ
	Vector Conversions	vfcvt.h.X	86.4 pJ
float16alt	Arithmetic	f{add,mul}.ah	87.2 pJ
	Conversions	fcvt.ah.x	73.5 pJ
	Vector Arithmetic	vf{add,mul}.ah	108.9 pJ
	Vector Conversions	vfcvt.ah.X	79.5 pJ
float8	Arithmetic	f{add,mul}.b	74.0 pJ
	Conversions	fcvt.b.x	72.5 pJ
	Vector Arithmetic	vf{add,mul}.b	95.2 pJ
	Vector Conversions	vfcvt.b.X	77.8 pJ

Idle System Energy per Cycle

Almost Identical

Energy decreases with fewer mantissa bits

95.2 pJ / 4 = 23.8 pJ

Average energy per operation (from post-layout simulations)

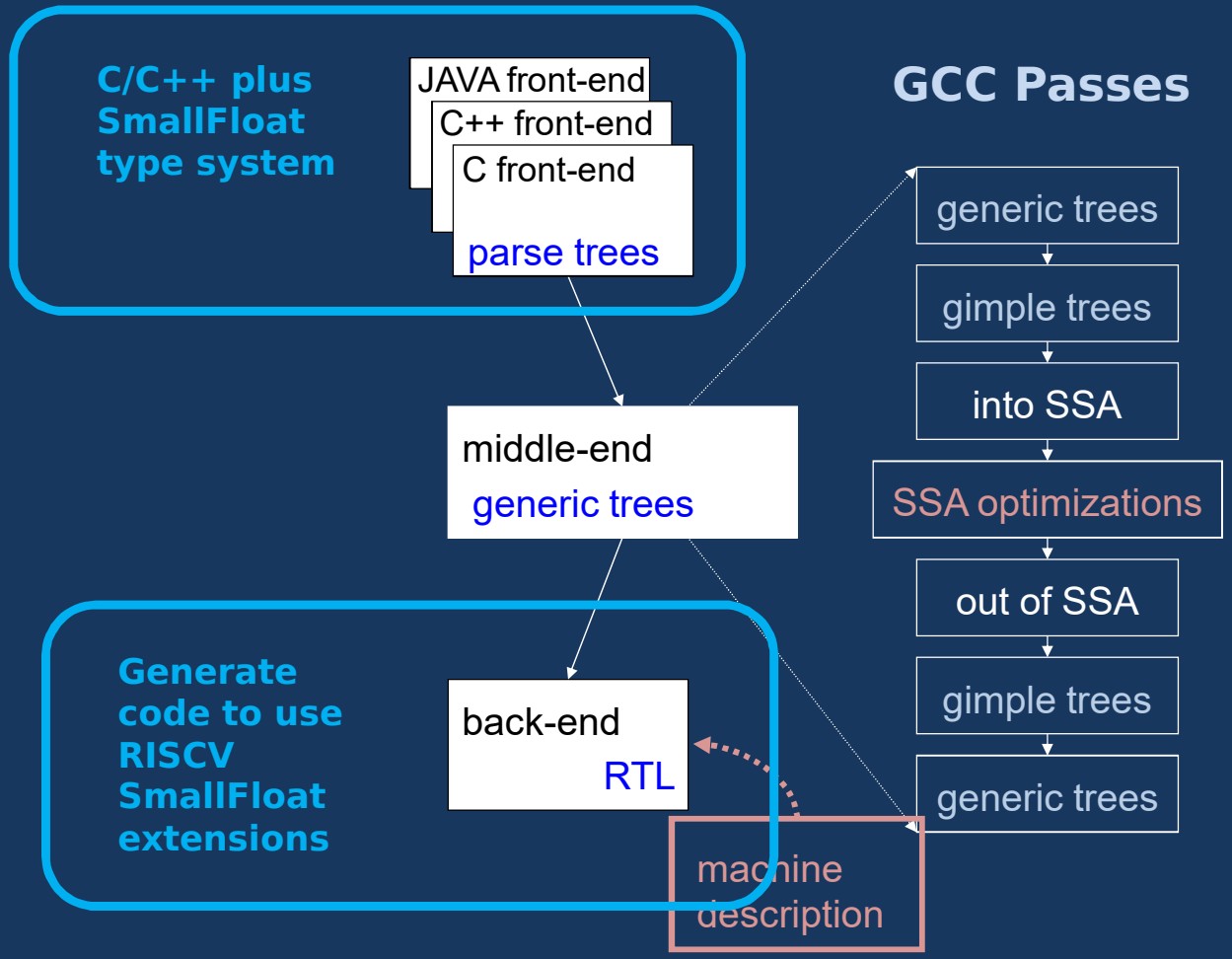
UMC 65nm, target @350MHz
Worst-case libraries (1.08V, 125°C)

1) Supporting the SmallFloat data type extension

Compiler Support

- Language type system extension (front-end)
- ISA extension (back-end)
- The role of vectorization

Compiler support to the SmallFloat data types



Compiler support to the SmallFloat data types

- Ok, now our compiler understands and handles smallFloat types.
- Is this sufficient to enable the expected energy savings?

The role of vectorization

```
.L3:
   flw       fa5,0(s0)
   flw       fa3,0(s2)
   flw       fa4,0(s3)
   add       s0,s0,4
   add       s4,s4,4
   add       s2,s2,4
   add       s3,s3,4
   fadd.s    fa5,fa5,fa3
   fadd.s    fa4,fa4,fa5
   fsw       fa5,-4(s0)
   fsw       fa5,-4(s4)
```

1111.2 pJ (iter) * 1024 iters = 1138 nJ

```
.L3:
   flw       fa5,0(s2)
   flh       a3,0(s1)
   flh       a2,0(s3)
   add       s1,s1,2
   add       s3,s3,2
   add       s2,s2,4
   add       s4,s4,2
   fcvt.h.s  a4,fa5
   fadd.h    a3,a3,a2
   fadd.h    a4,a4,a3
   sh        a3,-2(s1)
   sh        a4,0(s4)
```

1169.9 pJ (iter) * 1024 iters = 1198 nJ

```
.L3:
   lw        a0,0(s4)
   lw        a4,0(s6)
   flw       fa4,8(s5!)
   flw       fa5,8(a1!)
   add       s6,s6,4
   add       a3,a3,4
   add       s4,s4,4
   vfcpka.h.s a5,fa4,fa5
   vfadd.h   a4,a4,a0
   vfadd.h   a5,a5,a4
   sw        a4,-4(s4)
   sw        a5,0(a3)
```

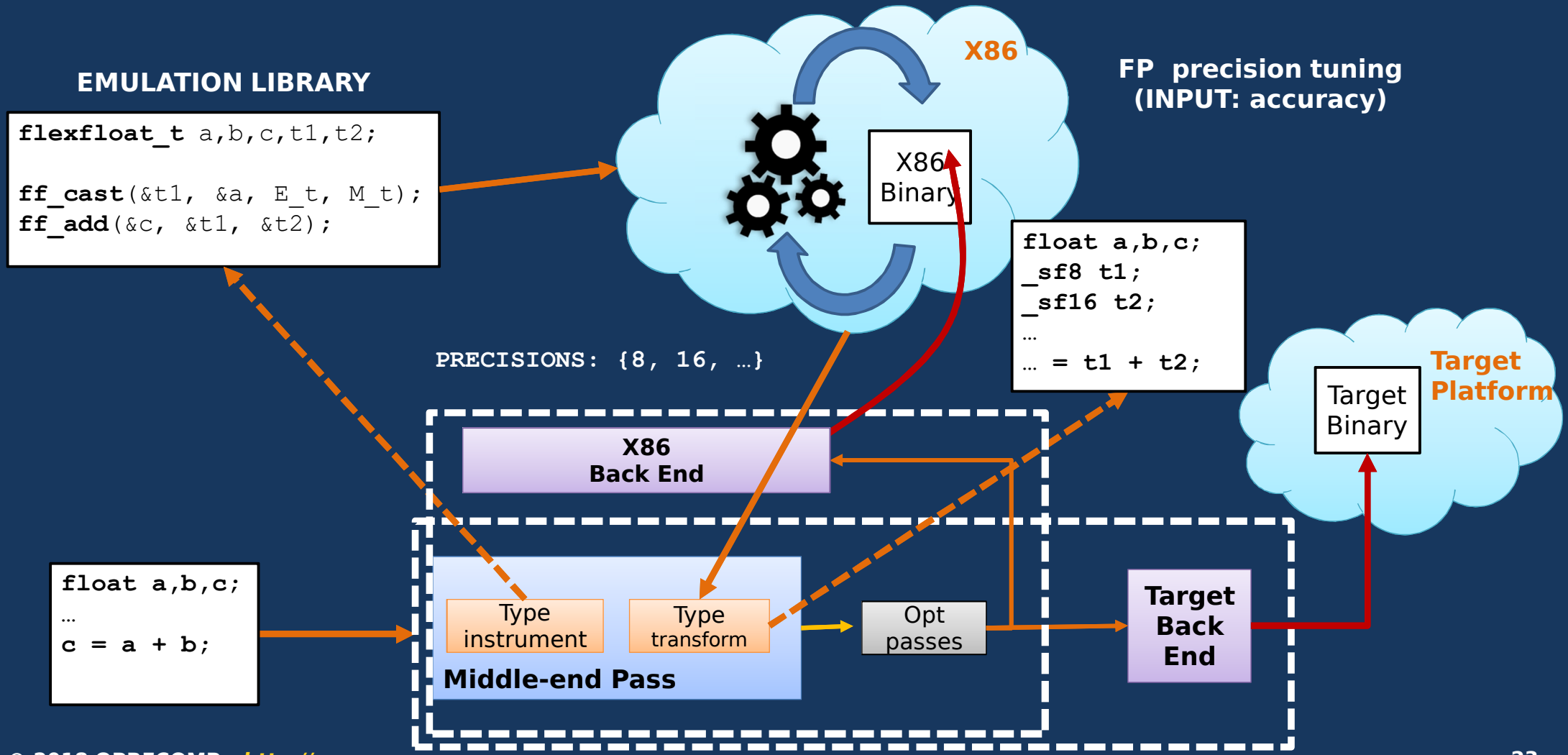
566.4 pJ LOAD/STORE
319.2 pJ ADD (integer)
265.2 pJ vADD (float16)
86.4 pJ CONV

1237.2 pJ (iteration) * 512 iterations = 633.5 nJ

Agenda

- Introduction – Transprecision Computing
- *Smaller-than-32-bit* floating point types
- Implementing the *smallFloat* extension
  - HW support
  - Compiler support
- **Simplifying the deployment of *SmallFloat-based* applications**
- Conclusion

Automation: integration with compilation toolchain



Agenda

- Introduction – Transprecision Computing
- *Smaller-than-32-bit* floating point types
- Implementing the *smallFloat* extension
  - HW support
  - Compiler support
- Simplifying the deployment of *SmallFloat-based* applications
- **Conclusion**

Conclusion

- *Less-than-32-bit* floating point types are beneficial to reduce execution time/energy consumption
- Support is required at HW level and compiler level to implement SmallFloat types
- A compilation toolchain can provide automatic tuning
  - In the best case, programmers use float/double variables as usual and do not care about auxiliary FP types