July 18th 2018, NIPS Lab (Perugia, IT)
International Summer School on Energy Aware Transprecision Computing

Funded by the H2020 Framework Programme of the European Union

SW and TOOLS

A tool bag for transprecision computing

Giuseppe Tagliavini (*giuseppe.tagliavini@unibo.it*)
Andrea Marongiu (*a.marongiu@unibo.it*)

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

OPRECOMP
Open Transprecision Computing

© 2017 OPRECOMP – *http://oprecomp.eu*

IEEE 754 formats: Binary format

- 1 bit for sign, *e* bits for exponent, *m* bits for mantissa
- Binary representation: $b_{m+e}\ b_{m+e-1} \ldots b_m\ \ b_{m-1} \ldots b_0$
- Number encoded by a binary string:

$$(-1)^{b_{m+e}} * 1.(b_{m-1} b_{m-2} \ldots b_0)_2 * 2^{(b_{m+e} b_{m+1} \ldots b_m)_2 - BIAS}$$

- Exponent field is biased → BIAS = $2^{n-1}-1$
- Special values:
  - 0/1 00…0 00…0 → +/- zero
  - 0/1 11…1 00…0 → +/- infinity
  - x 11…1 xx…x → NaN (quiet, signaling)
  - x 00…0 xx…x → Denormal numbers → $(-1)^{b_{m+e}} * 0.(b_{m-1} b_{m-2} \ldots b_0)_2 * 2^{1-BIAS}$

IEEE 754 formats: Rounding modes and Exceptions

- Rounding modes:
  - DOWNWARD → rounding towards negative infinity
  - TO NEAREST → rounding to nearest representable value
  - TOWARD ZERO → rounding towards zero
  - UPWARD → rounding towards positive infinity

- Exceptions:
  - Invalid operation → mathematically undefined, returns qNaN by default
  - Division by zero → returns ±infinity by default
  - Overflow → Returns ±infinity by default for the round-to-nearest mode
  - Underflow → Returns a subnormal or zero by default
  - Inexact → the exact result is not representable exactly, returns the rounded result by default

MPFR

- MPFR = GNU Multiple Precision Floating-Point Reliably
- C library based on GNU Multi-Precision Library (GMP)
- Format of (normal) numbers → $(-1)^{sign} * (\,0.1 b_0 b_1 \ldots b_{prec})_2 * 2^{exp}$, $E_{min} < exp < E_{max}$

- Main features:
  - Each variable has its own precision
  - No denormal numbers (can be emulated with mpfr_subnormalize)
  - Support for special numbers → signed zeros (−0), infinities and NaN
  - Correct rounding (IEEE 754 rounding modes)
  - Exception handling

- Many software projects extensively use MPFR (e.g. GCC)

- Smaller-than-32-bits formats are not adherent to IEEE standard (exponent not bound)

MPFR resources

- Bibliography:
  L. Fousse et al., “*MPFR: A multiple-precision binary floating-point library with correct rounding*,” ACM Transactions on Mathematical Software (TOMS), vol. 33, no. 2, p. 13, 2007

- Website:
  https://www.mpfr.org/

SoftFloat

- A library of C functions implementing binary floating-point conforming to the IEEE-754 standard

- Five binary formats:
  - 16-bit half-precision → float16_t
  - 32-bit single-precision → float32_t
  - 64-bit double-precision → float64_t
  - 80-bit double-extended-precision → extFloat80_t
  - 128-bit quadruple-precision → float128_t

- Operations:
  - addition, subtraction, multiplication, division, fused multiply-add, square root
  - comparisons
  - round to integral value
  - conversions to/from other supported formats
  - conversions to/from 32-bit and 64-bit integers (signed and unsigned)

SoftFloat resources

- Bibliography:
  J. R. Hauser, "*Handling floating-point exceptions in numeric programs*," ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 18, no. 2, pp. 139–174, 1996

- Website:
  http://www.jhauser.us/arithmetic

FlexFloat

- **FlexFloat** emulates FP formats using float/double variables
  - FP operations are emulated using the backend format
  - values are sanitized after updates to be coherent with the target format

- Example:

10111111111000000110111101101001		← FLOAT →	01000001110000000000000000000000
1011111100000110	*(= -1.7529297)*	← TARGET →	0111110 0 *(= +Inf)*
10111111111000000110000000000000		← BACKEND →	01111111100000000000000000000000

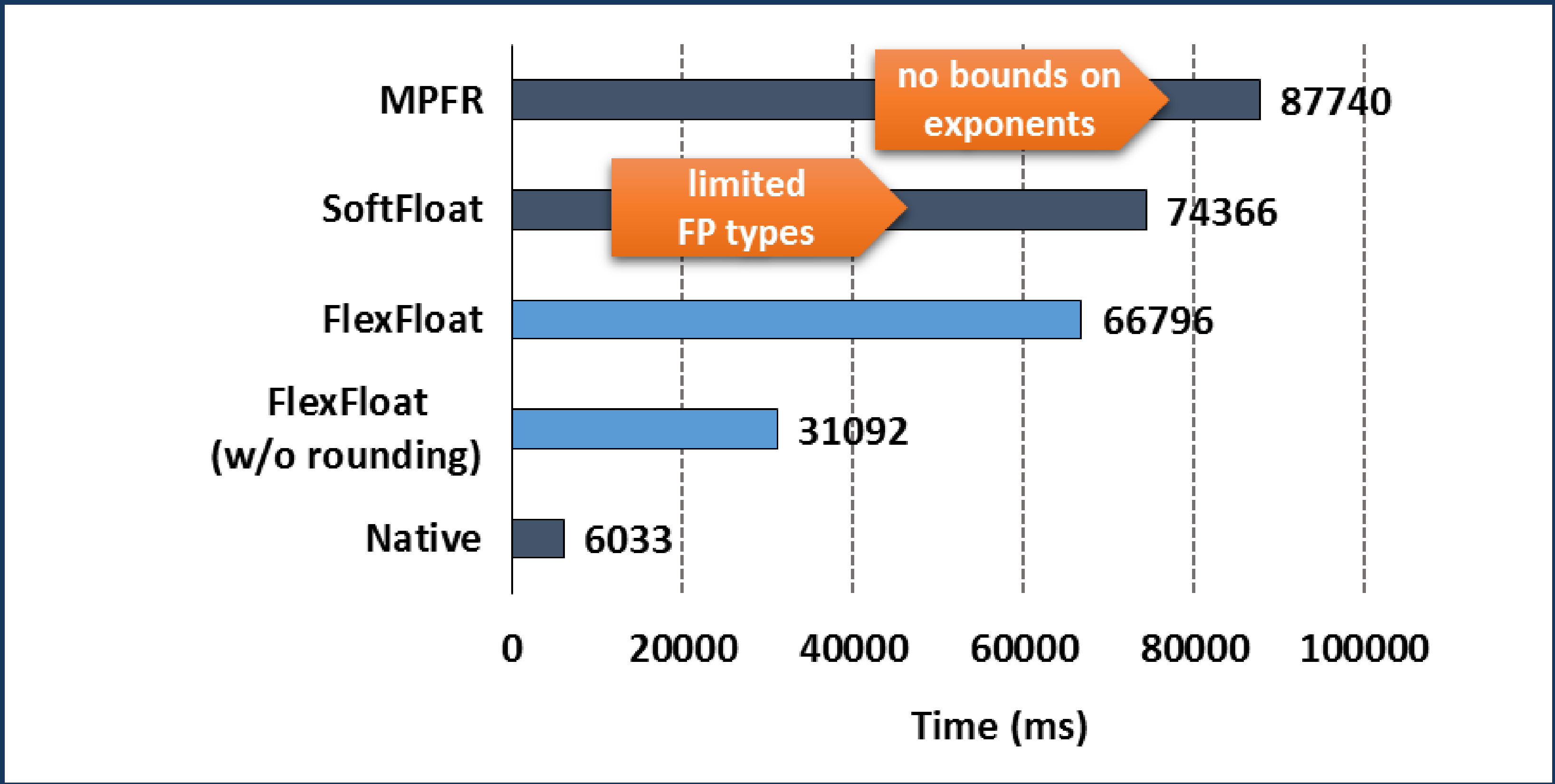
- Advanced features:
  - Type-based statistics
  - Error tracking for single variables
  - C++ wrapper

FlexFloat resources

- Bibliography:
  G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, L. Benini. "*A transprecision floating-point platform for ultra-low power computing*." In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, pp. 1051-1056. IEEE, 2018.

- Website:
  https://github.com/oprecomp/flexfloat

FlexFloat: Comparison with other libraries



FloatX

- Header-only C++ library for low precision floating point type emulation
  - Heavy inlining, resulting in relatively high performance
  - Native Template notation: floatx<exp_bits, sig_bits, backend_float>
  - Change precision of the type at runtime: floatxr<backend_float>
  - arithmetic operations between different types with implicit type promotion → a supertype is automatically derived

- Website:
  https://github.com/oprecomp/FloatX

fpPrecisionTuning

- ε → required accuracy on the program results
- **N** → number of floating-point variables
- **L / U** → lower/upper bound of precision
- **p_i** → precision assigned to variable x_i
- **P** → vector of precisions
- **δ** → error running with precision P
- **Influence group of x_i** → list of variables along some program path from variable x_i to the last variable that is affected by the x_i

- **GOAL** → find the smallest precision for each variable while keeping $\delta \leq \varepsilon$

```
procedure ITERATIVE SEARCH                                   ▷ main procedure
    MWL_0 ← {L_1, L_2, ..., L_N}                             ▷ initialize
    P_0 ← {U_1, U_2, ..., U_N}                               ▷ initialize
    repeat
        MWL_k ← ISOLATED DOWNWARD(MWL_{k-1}, P_{k-1})
        P_k ← GROUPED UPWARD(MWL_k)
    until Converged
    return P_k
end procedure
procedure ISOLATED DOWNWARD(MWL, P)
    P_temp ← P
    for i ← 1, N do                                          ▷ MPI_Parallel
        P_temp[i] ← BINARYSEARCH(MWL[i], P[i], P, i)
    end for
    return P_temp
end procedure
procedure GROUPED UPWARD(P)
    δ_min ← F(P)
    Δ ← {0, 0, ..., 0}                                       ▷ results from parallel threads
    P_min ← P
    repeat
        for i ← 1, N do                                      ▷ MPI_Parallel
            P_temp ← INCGROUPPREC(P_min, i)
            Δ[i] ← F(P_temp)
        end for
        δ_min, I_min ← min value and its index in Δ
        P_min ← INCGROUPPREC(P_min, I_min)
    until δ_min ≤ ε
    return P_min
end procedure
```

fpPrecisionTuning + FlexFloat: Basic usage

- [*Prerequisites*] Install Python 2.7 and its MPI support. On Ubuntu 14.04 systems (or later versions) this can be accomplished with:
  `sudo apt-get install python python-mpi4py`

- Download the fpPrecisionTuning toolchain from its official git repo:
  `git clone https://github.com/minhhn2910/fpPrecisionTuning.git`

- Enter the folder of a specific benchmark, for instance:
  `cd flexfloat-benchmarks/kmeans`

- Execute the precision tuning tool:
  `mpirun -np 8 <fpPrecisionTuning path>/PrecisionAnalysis/greedy_search_mpi.py 1 kmeans_flex`

© 2018 OPRECOMP – *http://oprecomp.eu*

fpPrecisionTuning + FlexFloat: More insights

- ❑ The accuracy required for the results can be tuned changing the error_rate parameter in the "greedy_search_mpi.py" script
- ❑ The available floating-point types can be modified updating the set_coefficient_bits function in the "compile.py" script
- ❑ The output provides the minimum bit-width of the mantissa for each input variable → a total order among program variables is defined in "compile.py"

© 2018 OPRECOMP – *http://oprecomp.eu*

Other tools

- **PRECiSA** (Program Round-off Error Certifier via Static Analysis) → fully automatic analyzer for the estimation of round-off errors of floating-point valued functional expressions
  Website: https://github.com/nasa/PRECiSA

- **FPTuner** → automatic precision-tuning of real expressions ((single, double, or quadruple precision)
  Website: https://github.com/soarlab/FPTuner

- **Precimonious/Blame analysis** → dynamic code analysis on LLVM IR
  Website: https://github.com/corvette-berkeley

- … and many others!