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
IEEE 754 formats: Binary format

- 1 bit for **sign**, e bits for **exponent**, m bits for **mantissa**

- Binary representation: \( b_{m+e} b_{m+e-1} b_m b_{m-1} ... b_0 \)

- Number encoded by a binary string:
  \((-1)^{b_{m+e}} \times 1. (b_{m-1} b_{m-2} ... b_0)_2 \times 2^{(b_{m+e} b_{m+1} ... b_m)_2 - BIAS}\)

- Exponent field is biased \(\rightarrow BIAS = 2^{n-1} - 1\)

- Special values:
  - 0/1 00...0 00...0 \(\rightarrow +/- \) zero
  - 0/1 11...1 00...0 \(\rightarrow +/- \) infinity
  - x 11...1 xx...x \(\rightarrow\) NaN (quiet, signaling)
  - x 00...0 xx...x \(\rightarrow\) Denormal numbers \(\rightarrow (-1)^{b_{m+e}} \times 0. (b_{m-1} b_{m-2} ... b_0)_2 \times 2^{1- BIAS}\)
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 → $(-1)^{sign} \times (0.1b_0b_1 ... b_{prec})_2 \times 2^{exp}$, $E_{min} < exp < E_{max}$

- 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:

- 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:

\[
\begin{align*}
101111111111000000110111101101001 & \quad \text{FLOAT} \quad \rightarrow \quad 0100001110000000000000000000000 \\
101111110000000110 & \quad (= -1.7529297) \quad \text{TARGET} \quad \rightarrow \quad 01111100 \quad ( = +\text{Inf}) \\
10111111111000000110000000000000000 & \quad \text{BACKEND} \quad \rightarrow \quad 011111111000000000000000000000000
\end{align*}
\]

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

© 2018 OPRECOMP - [http://oprecomp.eu](http://oprecomp.eu)
FlexFloat resources

- Bibliography:

- Website:
  https://github.com/oprecomp/flexfloat
FlexFloat: Comparison with other libraries

- **MPFR**: 87740 ms (no bounds on exponents)
- **SoftFloat**: 74366 ms (limited FP types)
- **FlexFloat**: 66796 ms
- **FlexFloat (w/o rounding)**: 31092 ms
- **Native**: 6033 ms

Time (ms)
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

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

- **GOAL** \( \rightarrow \) find the smallest precision for each variable while keeping \( \delta \leq \varepsilon \)

```plaintext
1: procedure ITERATIVE SEARCH  ▷ main procedure
2: MWL_0 ← \{L_1, L_2, \ldots, L_N\}  ▷ initialize
3: P_0 ← \{U_1, U_2, \ldots, U_N\}  ▷ initialize
4: repeat
5:    MWL_k ← ISOLATED DOWNWARD(MWL_{k-1}, P_{k-1})
6:    P_k ← GROUPED UPWARD(MWL_k)
7:    until Converged
8: return P_k
9: end procedure
10: procedure ISOLATED DOWNWARD(MWL, P)
11:    P_temp ← P
12:    for i ← 1, N do ▷ MPLParallel
13:        P_{temp}[i] ← BINARY SEARCH(MWL[i], P[i], P, \delta)
14:    end for
15:    return P_{temp}
16: end procedure
17: procedure GROUPED UPWARD(P)
18:    \( \delta_{\text{min}} \leftarrow F(P) \)
19:    \( \Delta \leftarrow \{0, 0, \ldots, 0\} \) ▷ results from parallel threads
20:    \( P_{\text{min}} \leftarrow P \)
21: repeat
22:    for i ← 1, N do ▷ MPLParallel
23:        P_{temp} ← INCGROUPPREC(P_{\text{min}}, P_{\text{temp}})
24:        \( \Delta[i] \leftarrow F(P_{\text{temp}}) \)
25:    end for
26:    \( \delta_{\text{min}}, I_{\text{min}} \leftarrow \min \) value and its index in \( \Delta \)
27:    \( P_{\text{min}} \leftarrow \) INCGROUPPREC(P_{\text{min}}, I_{\text{min}})
28:    until \( \delta_{\text{min}} \leq \varepsilon \)
29: return P_{\text{min}}
30: 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
  ```
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".
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!