Non-generic floating-point software support for embedded media processing

Jingyan Jourdan-Lu

Computer Arithmetic group *ARIC*
INRIA and ENS Lyon, France

Compilation Expertise Center
STMicroelectronics Grenoble, France

Joint work with Claude-Pierre Jeannerod and Christophe Monat

Context & Motivation

- Implementation of an efficient software support for IEEE 754 floating-point arithmetic on integer processors
  - A Floating-point Library for Integer Processors: FLIP
    - *correctly-rounded basic operators for the binary32 format*
    - *handling of subnormal numbers, and handling of special inputs*
    - *aiming at high instruction-level parallelism (ILP)*
  - Optimized for the ST231 processor
    - *4-way VLIW 32-bit embedded integer architecture from STMicroelectronics*
    - *integer processor for embedded media systems, such as HD-IPTV, PDAs,...*
- Non-generic operators: an extension of basic operators
  - Media processing applications typically involve numerical blocks having regular floating-point computation patterns.
  - For integer processors, these patterns can be profitably turned into non-generic operators to achieve better performance.

Purpose of our work

- Design of non-generic operators to enhance floating-point support for integer processors
- Development of compiler optimizations to select such operators in application codes

Motivating example: essential part of radix-2 FFT computation

```
for  (k=j; k<n; k=k+n2 ){  /* float t1, t2, x[], y[], s, c; */
     t1 = c*x[k+n1] - s*y[k+n1]; t2 = s*x[k+n1] + c*y[k+n1];
     x[k+n1] = x[k] - t1; x[k] = x[k] + t1;
     y[k+n1] = y[k] - t2; y[k] = y[k] + t2;}
```

- `t1` and `t2` selected as dot product in dimension two (DP2): $xy + zt$
- (`x[k+n1]`,`x[k]`), (`y[k+n1]`,`y[k]`) selected as addsub: computing the pair $(x + y, x - y)$

Three categories of non-generic operators implemented

- Specialized operator replaces a generic operator when the compiler can prove properties about its arguments.
  - **mul2** (multiplication by two): $2x$.
  - **div2** (multiplication by one half): $\frac{1}{2}x$.
  - **scalb** (multiplication by an integer power of two): $2^n x$ with n a 32-bit signed integer.
  - **square** (squaring): x^2.
  - **addnn** (addition of non-negative terms): $x + y$ with $x \geq 0$ and $y \geq 0$.
- Fused operator replaces a set of two or more floating-point operators by a single one.
  - **FMA** (fused multiply-add): $xy + z$.
  - **FSA** (fused square-add): $x^2 + z$ with $z \geq 0$.
  - **DP2** (dot product in dimension two): $xy + zt$.
  - **SOS** (sum of two squares): $x^2 + y^2$.
- Paired operator simultaneously evaluates two operators.
  - **addsub** (simultaneous addition and subtraction): $(x + y, x - y)$.
  - **sincos** (simultaneous sine and cosine): $(\sin x, \cos x)$.

Speedups and code size reduction on the ST231

- Speedup: latency of naive implementation / latency of non-generic operator
- Code Reduction Ratio (CRR): size of non-generic operator / size of naive implementation

$\circ$ = RN	Speedup	CRR
mul2	4.2	0.15
div2	4.86	0.17
scalb	1.4	0.70
square	1.75	0.49
addnn	1.73	0.54
FSA	2.14	0.46
FMA	1.12	1.02
SOS	2.62	0.35
DP2	1.33	0.84
addsub	1.86	0.56
sincos	1.95	0.82

- Speedups > 1 denote an acceleration
- CRRs < 1 indicate a code size reduction
- Small CRRs may be a good indication of power reduction

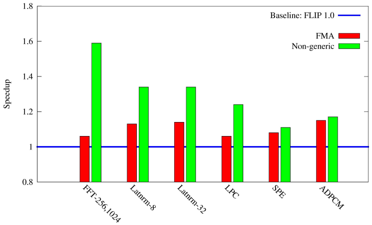
Speedups range from 1.12 to 4.86

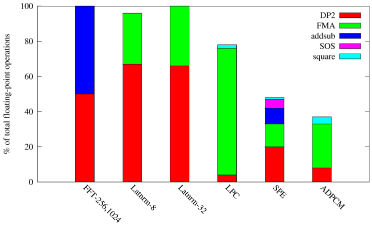
CRRs can be as low as 0.15

FMA's adverse CRR due to bigger alignment logic in the addition stage, which is necessary for correct rounding

Performances on the UTDSP benchmark

- UTDSP benchmark
  - Assessing C compilers efficiency on typical DSP code
  - Good predictor of improvements obtained at a larger scale
  - Divided into two classes: kernels (FFTs, Latnrm,...) and applications (LPC, SPE, ADPCM,...)
- Speedups and usage of non-generic operators





  - Speedups can be up to 1.6
  - 100% usage of non-generic operators in FFT test suites
  - Fused operators (such as DP2, FMA) greatly improve floating-point performance