Software for squaring floats on ST231: a case study in bringing floating-point to VLIW integer processors

Jingyan Jourdan-Lu

Computer Arithmetic Team *Arénaire*
INRIA (Lyon)

Compilation Expertise Center
STMicroelectronics (Grenoble)

Joint work with C.-P. Jeannerod, C. Monat, G. Revy

Floating-point arithmetic on integer microprocessors

We aim at efficient software emulation of FP operators for integer-only microprocessors.

- *embedded systems everywhere, set top boxes, mobile phones,...*
- *fast FP emulation to avoid the cost of hardware FP units*

Our design is

- compliant with the main IEEE 754 features:
  - different *binaryk* formats
  - all rounding modes
  - gradual underflow
- portable to integer processors: *implementation in C*

Our hardware platform: ST231

The **ST231** is a 4-way integer-only VLIW processor from the ST200 family:

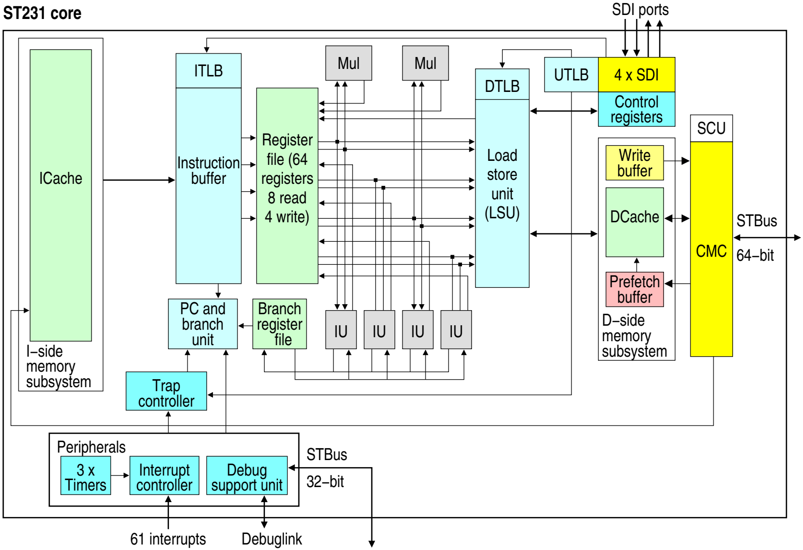
- Up to 4 instruction words can be grouped in one bundle.
- Up to 4 instructions can be executed in one cycle.

→ *Key to realize Instruction Level Parallelism (ILP).*

Typical applications:

- a *media processor* with an embedded OS
- a *host processor* running Linux and applications

Architecture of the ST231 core



Key instructions for FP designs

All arithmetic operations have a one-cycle latency except the multipliers which have a three-cycle latency.

- $\texttt{slct}\ \mathrm{R}_{\texttt{DEST}} = \texttt{B}, \texttt{Opnd1}, \texttt{Opnd2}.$
  returning $\mathrm{R}_{\texttt{DEST}} = \texttt{B}\ ?\ \texttt{Opnd1} : \texttt{Opnd2}$
  Transform branches to straight line code.
- $\texttt{clz}\ \mathrm{R}_{\texttt{DEST}} = \texttt{Opnd1}.$
  counting the leading zeros of `Opnd1`
  Key to subnormal numbers support.
- Up to two `mul`s can be issued in one cycle.
  Essential to polynomial evaluation.
- Others: (un)signed *min* and *max*, arbitrary length *shifts*.

Software libraries for FP emulation used on ST231

- SoftFloat [Hauser]
  - Robust. *It is a reference model with test vectors.*
  - Hardware independent. *It doesn't address performance.*
- FLIP [Arénaire team]
  - High ILP exposure.
  - Fast division, square root based on polynomial evaluation.

	SoftFloat cycles	**FLIP cycles**	**Speedup**
addition	48	26	1.9x
subtraction	49	26	1.9x
multiplication	31	21	1.5x
division	177	34	5.2x
square root	95	23	4.1x

Beyond FLIP

- Some typical applications may intensively use numerical blocks involving special operators.
- Discovering useful *fused* or *specialized* operators can improve the FP performance.

Example of 2-norm:

```
float two_norm(float a[], int n){
  int i;  float s = 0.0f;
  for (i=0; i<n; i++)    s = s + a[i]*a[i];
  return sqrtf(s);}
```

Essential part of radix-2 FFT:

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

An extension of general operators

- A *specialized* operator replaces a *generic* operator when the compiler can prove properties about the arguments.
- A *fused* operator replaces a set of 2 or more FP operators by a single one.

The operators we aim at implementing in software include:

- $\circ(x \cdot y) \rightarrow \circ(x^2)$ square
- $\circ(x \cdot y + z) \rightarrow \circ(x^2 + z)$ fused square-add
- $\circ(\circ(x \cdot y) + \circ(z \cdot t)) \rightarrow \circ(x \cdot y + z \cdot t)$ 2D dot product
- fused add-sub, a unit to compute the pair $[\circ(x+y), \circ(x-y)]$.

  $\hookrightarrow$ for *hardware* designs, see Saleh and Swartzlander (2008), Saleh (2009)

We work at two levels:

- Detection of such operators during compilation
- Design and software implementation of arithmetic algorithms

Detecting square during compilation

st200cc:

- based on Open64 technology
- further developed by STMicroelectronics

WHIRL: the intermediate representation for Open64 compilers.

- supporting different front-end languages, *C*, *C++*, ...
- independent of target processor architectures

At WHIRL level, we can detect square by checking the identity of the WHIRL tree of the two operands of each multiplication.

For instance:

- $\mathtt{x} \cdot \mathtt{x} \rightarrow \mathtt{x}^2$
- $(\mathtt{x} + \mathtt{1.0f}) \cdot (\mathtt{x} + \mathtt{1.0f}) \rightarrow (\mathtt{x} + \mathtt{1.0f})^2$

Designing a fast FP square operator $x \mapsto r = x^2$

Goals

- From the integer encoding $X = [\text{sign}|\text{biased exp}|\text{fraction}]$ of x, get the integer encoding R of the IEEE 754 result r.
- On ST231, don't save just a few cycles (compared to general multiply), but divide the latency by ≈ 2.

Design principles

- Define generic vs. special input very carefully.
- Maximize ILP exposure in the generic path:
  - In parallel: biased exp, truncated fraction L, sticky bit t.
  - Fast parallel expressions for L and t.
- Reuse previous work to optimize the special path.
- Do all this "symbolically" ($=$ parameterized by the format).

Generic vs. special input

For the binary$k$ format with rounding $\circ$, specializing the IEEE-754 specification of multiplication $x \times y$ to the case $x = y$ gives:

$$r = \begin{cases} +0 & \text{if } |x| = 0, \\ \min_\circ & \text{if } |x| \in [\alpha, \alpha'), \\ \circ(x^2) & \text{if } |x| \in [\alpha', \Omega'), \\ \max_\circ & \text{if } |x| \in [\Omega', \Omega], \\ +\infty & \text{if } |x| = \infty, \\ \text{qNaN} & \text{if } x \text{ is NaN.} \end{cases}$$

with

- α smallest positive number
- Ω largest finite number
- $\alpha' = 2^{\lfloor (e_{\mathsf{min}} - p)/2 \rfloor} \approx \sqrt{\alpha}$
- $\Omega' = 2^{(e_{\mathsf{max}}+1)/2} \approx \sqrt{\Omega}$
- $\min_\circ$, $\max_\circ$ depending only on rounding mode $\circ$

$\hookrightarrow$ Input x is generic if $\alpha' \leq |x| < \Omega'$, and special otherwise.

Generic path

Here, the fraction of R is of course the hardest part and we get it as $L + b$ with L the truncated fraction of x^2 and b the round bit.

For rounding "to nearest even" b depends on the sticky bit t.

Theorem 1 (formula for L): $L = H \gg (\mu + w - 1)$ with

- H the higher half of square of input significand m_x
- $\mu = \max(c, F)$ with c and F functions of m_x and $2E_x$
- w the exponent width of the format

Advantages:

- covers normal and subnormal cases
- H and μ in parallel
- on ST231 and for binary32,
  - only 1 multiply instruction is used
  - we proved the shift value is C99 compliant: $\mu + 8 - 1 < 32$

Generic path (cont'd)

Similarly, we have proved a parallel formula for the sticky bit t:

Theorem 2 (sticky bit formula): $t = [T_1 \neq 0] \vee [T_2 \neq 0]$ with

- $T_1 = H \ll (p + 2 - \mu)$
- $T_2 = X \bmod 2^{p - \lfloor k/2 \rfloor}$
- H and μ as before, and p the precision of the binary$k$ format

$\hookrightarrow$ very fast in practice: 7 cycles on ST231 for binary32.

Special path

Recall that special x is either NaN or such that $|x|$ is "small" ($|x| < \alpha'$) or "large" ($|x| \geq \Omega'$).

Hence any special input is filtered out via the condition

$$C_{\text{spec}} = C_{\text{nan}} \vee C_{\text{small}} \vee C_{\text{large}}.$$

Theorem 3: One can reuse $2E_x$ from the generic path:

- $C_{\text{small}} = [2E_x \leq e_{\max} - p - 1]$
- $C_{\text{large}} \vee C_{\text{nan}} = [2E_x \geq 3e_{\max} + 1]$

$\hookrightarrow$ once $2E_x$ is available, 3 instructions suffice to get C_{spec}.

Special path (cont'd)

Besides C_{spec}, C_{small}, and $C_{\text{large}} \vee C_{\text{nan}}$, we need C_{nan} in order to handle special input.

- For rounding to nearest, we thus have:

```
if (Cspec) {
  if (Cnan) return qNaN;
  else {
    if (Csmall) return 0;
    else return +oo;}
}else {...//generic case}
```

$\xrightarrow[\text{ST231}]{\textit{binary32}}$

```
slct $r1 = Csmall, 0, +oo
slct $r2 = Cnan, qNaN, $r1
slct $rDEST = Cspec, $r2, $rx
```

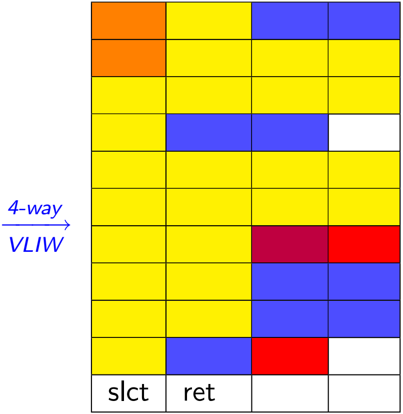
`$rx` *holds the result from the generic path.*

- For other roundings, easy adaptation by using `min` and `max`:
  - for RD and RZ, $+\infty$ is replaced by $\max(|x|, \Omega)$,
  - for RU, 0 is replaced by $\min(|x|, \alpha)$.

Summary of ILP exposure

Three independent tasks: detect special inputs (T1), handle them (T2), and handle generic inputs (T3):

```
uint square(uint X) {
   Cspec = ... ; // (T1)
   if (Cspec)
     { ... } // (T2)
   else
     { ... } // (T3)
}
```



Performance of our FP square operator on ST231

- Performance for various rounding modes:

$\circ$	FLIP multiply	square	speedup
RN	21	12	1.75x
RD	21	9	2.3x
RU	21	11	1.9x
RZ	18	9	2x

  - Speed up from 1.75x to 2.3x.
  - On average, 3.4 instructions per cycle (IPC), so that all bundles are almost full.

- Application example: 1.15x speedup when computing 2-norms.

- Our symbolic approach made it immediate to produce C code for binary64 as well:
  - already a 1.74x speedup

Conclusions

In summary:

- High ILP can be exposed on ST231 for FP operators.
- Specialized or fused operators really improve FP applications.
- Selecting special operators requires sophisticated compiler optimizations.

On-going work:

- Detecting and implementing fused square-add (FSA), 2D dot product, fused add-sub
  $\hookrightarrow$ thanks to FSA, the speedup for 2-norms is 1.3x instead of 1.15x.
- Addressing operators like *sincos* [Markstein (2003)].