Faster floating-point square root for integer processors

Guillaume Revy[a], jointly with Claude-Pierre Jeannerod[a], Hervé Knochel[b] and Christophe Monat[b]

Arénaire INRIA project-team[a] / STMicroelectronics Compilation Expertise Centre[b]

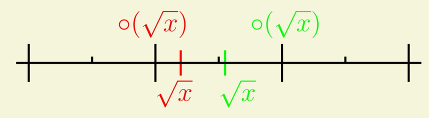
1. Introduction

▶ Context & Motivation

- ST231 = integer processor for embedded media systems [2] → no FPU
- Emulation of single precision floating-point arithmetic [3]
- Audio/Video (HD-IPTV, cell phones, wireless terminals, PDAs)
  → highly demanding on floating-point square root computations
- Fast and accurate floating-point arithmetic software for mathematical functions

▶ Goals

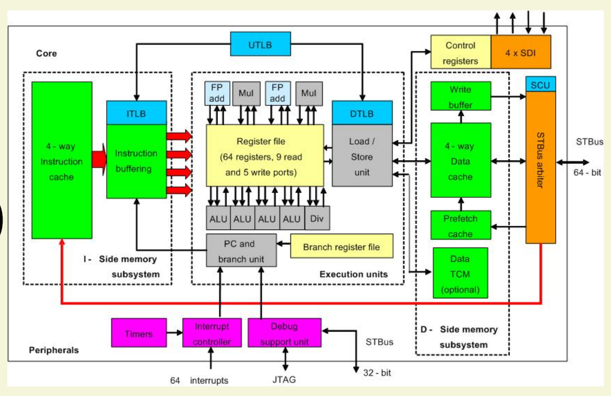
- Exploit at best the ST231 architecture: ILP/32-bit registers
- Achieve correct rounding-to-nearest (even) of $\sqrt{x}$, for x a normal number



- Simpler algorithms/implementations using polynomial approximants evaluation

2. ST231 architecture and compiler

▶ Architecture



- 4-way VLIW architecture
- 32 x 32 bit multipliers
- Efficient 32-bit immediate encoding (2 per cycle)
- Select instruction to remove branch penalty

▶ Compiler

- Open64 compiler technology
- Instruction Level Parallelism (ILP) extractor and scheduler
- Select-based if-conversion
- Full ISA access through intrinsics

3. Square root implementation - General principle

▶ Some properties of the square root function

Input normal single precision floating-point number $x = (-1)^s \cdot m \cdot 2^e$, with $s \in \{0,1\}$, $e \in \mathbb{N} \cap [-126, 127]$ and $m = 1.f$ with $f = 0.f_1 f_2 \cdots f_{23} \in [0,1)$

Output correct rounding-to-nearest of $\sqrt{x}$: $\circ(\sqrt{x})$ or exception

$$\sqrt{x} = \ell \cdot 2^d \text{ and } \circ(\sqrt{x}) = \circ(\ell) \cdot 2^d,$$

with $d = \lfloor e/2 \rfloor$, $\ell = t\sqrt{m}$ and $t \in \{1, \sqrt{2}\}$.

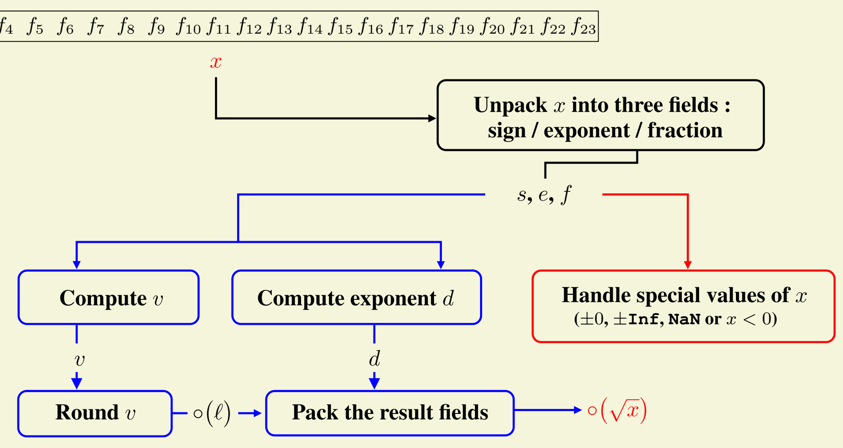
- x = normal number → $\sqrt{x}$ = normal number
- $\circ(\ell) \in [1,2)$ → no renormalization

▶ Square root computation steps

1. Input x = 32-bit register → Unpack = masks / shifts

s	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}

2. Compute d and v: $|v - \ell| \le 2^{-25}$
   → sufficient condition to get $\circ(\ell)$



- $E = e + 127 \Rightarrow D = d + 127 = \lfloor (E+127)/2 \rfloor \Rightarrow$ 1 bit right shift
- $v \in [1,2)$ = 32-bit register

1	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}	v_{21}	v_{22}	v_{23}	v_{24}	v_{25}	v_{26}	v_{27}	v_{28}	v_{29}	v_{30}	v_{31}

3. Round/Pack result
   → same format as for input x

4. Square root implementation - Methods to achieve v

▶ Existing methods

- Restoring/Nonrestoring algorithms: one result digit per iteration
- Newton-Raphson/Goldschmidt iterations [1]: refine approximations of $\sqrt{m}$ or $\frac{1}{\sqrt{m}}$

▶ Our approach: evaluation of polynomial approximants

- Approximate $\sqrt{1+X}$ for $X \in [0,1)$ by one or several minimax polynomials
- Evaluate such polynomials with fast, parallel schemes similar to Estrin's

$$a(X) = \underbrace{(a_5X + a_4)}_{r_3} X^4 + \underbrace{(\underbrace{(a_3X + a_2)}_{r_2} X^2 + \underbrace{(a_1X + a_0)}_{r_1})}_{r_{21}}$$

	– I1 –	– I2 –	– I3 –	– I4 –
cycle 1	X^2	a_3X	-	-
cycle 2	a_5X	a_1X	-	-
cycle 3	-	-	-	-
cycle 4	X^4	r_2	-	-
cycle 5	r_2X^2	r_3	-	-
cycle 6	r_1	-	-	-
cycle 7	r_3X^4	-	-	-
cycle 8	r_{21}	-	-	-
cycle 9	-	-	-	-
cycle 10	$a(X)$	-	-	-

⇒ expected scheduling

⇒ exploit ILP/32-bit registers

- Solutions: Poly-5 (3 polynomials/degree 5) / Poly-6 (2 polynomials/degree 6)

5. Results for rounding-to-nearest and normal numbers

▶ Latencies for generic input values

	2	3	4
Restoring	170	152	147
Nonrestoring	235	181	132
Newton-2	53	49	45
Goldschmidt-2	50	46	42
Goldschmidt-1	45	42	36
Poly-5	53	45	33
Poly-6	42	35	25

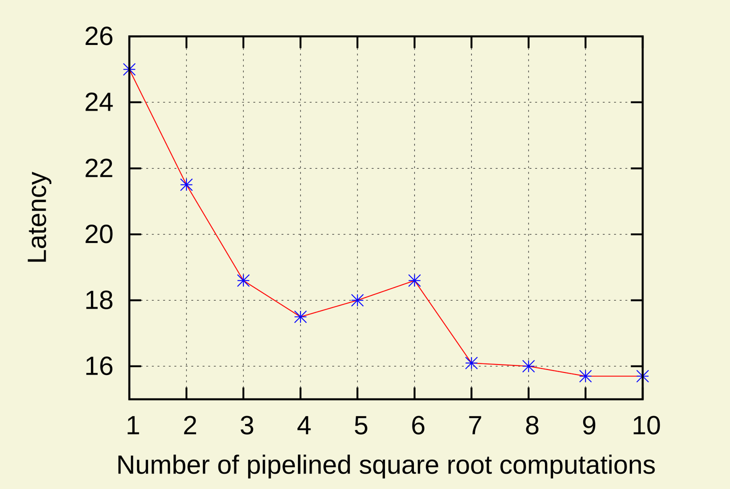
→ 25 cycles (previous version: 48 cycles)

→ speedup of ≈ 48 %

→ IPB/IPC: 2.58

→ impact of issue width (2,3,4)

▶ Pipelined square root computations



→ bound ≈ 15 cycles/execution

▶ Latencies for special input values

x	± 0	$\pm$Inf	NaN	< 0
cycles	16	17	17	17

6. Some preliminary results for other rounding modes, numbers and formats

▶ Other rounding modes

= downward / to zero / upward / faithful

	RN	RD/RZ	RU	FR
1 polynomial	29	31	30	26
2 polynomials	25	27	27	22

(Poly-6)

- More expensive tests for directed rounding modes
  → low extra cost ≈ 2 cycles (6.8% - 8%)
- No test for faithful rounding

▶ Subnormal numbers

= $x \in (0, 2^{-e_{min}}) = (0, 2^{-126})$

	RN	RD/RZ	RU	FR
1 polynomial	33	35	35	30
2 polynomials	30	32	32	27

(Poly-6)

- Costly test to decide whether x is subnormal
  → extra cost ≈ 5 cycles (17.2 % - 20 %)

▶ Medim precision / High precision

- High precision (24 bits)

s	E_1	E_2	E_3	E_4	E_5	E_6	E_7	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}

- Medium precision (16 bits)

s	E_1	E_2	E_3	E_4	E_5	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}

= same method with polynomials of lower degree

→ with no subnormals / 2 polynomial approximants

Generic input values

	RN	RD/RZ	RU	FR
Medium precision	21	21	24	18
High precision	24	26	26	21

Special input values

x	± 0	$\pm$Inf	NaN	< 0
High precision	15	17	17	17
Medium precision	13	13	13	13

⇒ Graphics applications (OpenGL ES) / GPU (Nvidia/ATI)

7. Conclusions

- ▶ Software implementation: 25 cycles (speedup of ≈ 48 %)
- ▶ Exploit at best the ILP / 32-bit registers
- ▶ Low extra cost due to the handling of other rounding modes and subnormal numbers
- ▶ Impact of the precision and the number of polynomial approximants on latencies
- ▶ Possible extension to other functions: $1/\sqrt{x}$, $1/x$, x/y,...
- ▶ Automating the generation of fast and accurate mathematical functions

Some references

[1] Miloš. D. Ercegovac and Tomás Lang. *Digital Arithmetic.* Morgan Kaufmann, 2003.

[2] Joseph A. Fisher, Paolo Faraboschi, and Cliff Young. *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools.* Morgan Kaufmann, 2005.

[3] Saurabh-Kumar Raina. *FLIP: a floating-point library for integer processors.* PhD thesis, ENS Lyon, 2006.