Faster floating-point square root for integer processors

Guillaume Revy\textsuperscript{a}, jointly with Claude-Pierre Jeannerod\textsuperscript{a}, Hervé Knochel\textsuperscript{b} and Christophe Monat\textsuperscript{b}

\textsuperscript{a}Aréna INRIA project-team / STMicroelectronics Compilation Expertise Centre\textsuperscript{b}

1. Introduction

- \textbf{Context & Motivation}
  - ST231 = integer processor for embedded media systems \cite{ST231} \rightarrow no FPU
  - Emulation of single precision floating-point arithmetic \cite{Emulation}
  - Audio/Video (HD-IPTV, cell phones, wireless terminals, PDAs)
    \rightarrow highly demanding on floating-point square root computations
  - Fast and accurate floating-point arithmetic software for mathematical functions

- \textbf{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

- \textbf{Architecture}
  - 4-way VLIW architecture
  - Efficient 32-bit immediate encoding (2 per cycle)
  - Select instruction to remove branch penalty
- \textbf{Compiler}
  - Open64 compiler technology
  - Instruction Level Parallelism (ILP) extractor and scheduler
  - Full ISA access through intrinsics

3. Square root implementation - General principle

- \textbf{Some properties of the square root function}
  - Input: normal single precision floating-point number $x = (\pm1)^e m \cdot 2^d$, with $e \in \{0, 1\}$, $m \in \mathbb{N} \cap [-128, 127]$ and $m \cdot f = 1.0, f, f_2, \ldots, f_4 \in [0, 1)$
  - Output: correct rounding-to-nearest of $\sqrt{x}$. $\delta(x)$ or exception
  - $\sqrt{x} = 2^d \cdot \sqrt{\frac{1}{2^d}}$ and $\delta(x) = \delta(e) \cdot 2^d$,
    with $d = \{\lfloor e/2 \rfloor, \lfloor e/4 \rfloor \} \in \{\lfloor 1/2 \rfloor \}$.
  - $x$ is normal number $\rightarrow \sqrt{x}$ is normal number
  - $\delta(e) \in [1, 2] \rightarrow$ no renormalization
- \textbf{Square root computation steps}
  1. Input $x = 32$-bit register $\rightarrow$ Unpack $= \text{masks} / \text{shifts}$
  2. Compute $d$ and $e$: $|e| - 1 \leq 2^\chi$
    $\rightarrow$ sufficient condition to get $\delta(e)$
  3. Round/Pack result
    $\rightarrow$ same format as for input $x$

4. Square root implementation - Methods to achieve

- \textbf{Existing methods}
  - Restoring/Nonrestoring algorithms: one result digit per iteration
  - Newton-Raphson/Goldschmidt iterations \cite{Goldschmidt}: refine approximations of $\sqrt{X}$ or $\frac{1}{\sqrt{X}}$
- \textbf{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

- \textbf{Pipeline}
  - $2X$ = same method with polynomials of lower degree
  - $3X$ = same method with polynomials of lower degree

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

- \textbf{Latencies for generic input values}
- \textbf{Pipeline}

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

- \textbf{Other rounding modes}
  - downward / to zero / upward / faithful
  - $x \in (0, 2^{-n})$ \rightarrow low extra cost $\approx 5$ cycles
  - $x \in (0, 2^{-n})$ \rightarrow high extra cost $\approx 5$ cycles

- \textbf{Subnormal numbers}
  - $x \in (0, 2^{-127}) = (0, 2^{-128})$
  - $x \in (0, 2^{-127}) = (0, 2^{-128})$
  - $x \in (0, 2^{-127}) = (0, 2^{-128})$

- \textbf{Medim precision / High precision}
  - High precision (24 bits)
  - Medium precision (16 bits)
  - $\rightarrow$ with no subnormals / 2 polynomial approximations

- \textbf{Special input values}
  - Graphics applications (OpenGL ES) / GPU (Nvidia/ATI)

Some references

\begin{thebibliography}{9}
\bibitem[1]{Eratovic} Mikol D. Erroovic and Toniang Lam, Digital Arithmetic, Morgan Kaufmann, 2003.
\end{thebibliography}