Optimizing correctly-rounded reciprocal square roots
for embedded VLIW cores
Claude-Pierre Jeannerod‡ \textsuperscript{1, 2} and Guillaume Revy\textsuperscript{2, 1}
\textsuperscript{1}INRIA Grenoble - Rhône-Alpes (Arénaire project-team, LIP, ENS Lyon) \textsuperscript{2}Université de Lyon

1. Introduction

\begin{itemize}
  \item Context & Motivation
    \begin{itemize}
      \item Implementation of an efficient software support for IEEE 754 floating-point arithmetic on integer processors
      \item set of correctly-rounded mathematical operators, handling of subnormal numbers, and handling of special inputs
      \item development of the FLIP library \cite{1} for the binary32 floating-point format
    \end{itemize}
  \item Purpose of our work
    \begin{itemize}
      \item Software implementation of correctly-rounded reciprocal square root \((x^{-1/2})\) → 20 cycles
      \item frequently-used in digital signal processing \cite{2, 3, 4}
      \item correctly-rounded implementation recommended by the latest revision of the IEEE 754 standard \cite{5, 6, 7}
      \item Optimized for the binary32 format and the ST231 core
      \item Correctly-rounded RoundToEven (rounding to nearest)
      \item Extension of our innovative polynomial evaluation-based method introduced in \cite{4} for square root
      \item Efficiency achieved by exploiting at best the instruction-level parallelism (ILP) of the ST231
    \end{itemize}
  \item Example of application
    \begin{itemize}
      \item Typical use of reciprocal square root = 3D vector normalization
        \[
        \mathbf{v} = \mathbf{v}/\|\mathbf{v}\|
        \]
      \item with \(\mathbf{v} = (x, y, z)\)
      \item Without reciprocal square root:
        \[
        1 \rightarrow \text{sqrt}(23 \text{ cycles}) \rightarrow 3 \text{ div}(3 \times 32 \text{ cycles}) = 119 \text{ cycles}
        \]
      \item With reciprocal square root:
        \[
        1 \rightarrow \text{r sqrt}(29 \text{ cycles}) \rightarrow 3 \text{ mul}(3 \times 21 \text{ cycles}) = 92 \text{ cycles}
        \]
      \item Latency reduction by over 21 \%
    \end{itemize}
\end{itemize}

2. ST231 architecture and compiler

\begin{itemize}
  \item ST231, a 4-issue VLIW 32-bit embedded integer architecture
    \begin{itemize}
      \item 4 parallel ALUs / 2 parallel pipelines \(32 \times 32 \rightarrow 32\)-bit multipliers
      \item 1 leading zero counter
      \item Predicate execution --- select instruction to remove branch penalty
      \item 64 general purpose 32-bit registers / 8 1-bit branch (condition) registers
    \end{itemize}
  \item ST231 Compiler
    \begin{itemize}
      \item Open64 compiler technology
      \item Instruction level parallelism extractor and scheduler
      \item Select-based d-conversion --- straight-line assembly code
      \item Sequences of select instructions instead of costly control flow
      \item Linear Assembler Optimizer (LAO): generates schedule very close to the optimal
    \end{itemize}
\end{itemize}

3. Some properties of reciprocal square root

\begin{itemize}
  \item Handling of special operands \(x \in (-\infty, 0, \infty, \pm \infty, \pm \text{NaN})\)
    \begin{itemize}
      \item Filter out special operands using the standard binary interchange encoding format
      \item Compute special results required by \cite{2} in parallel with the generic case
    \end{itemize}
  \item Positive finite operand \(x\) (precision \(p \geq 2\))
    \begin{itemize}
      \item Input: binary32-floating-point number \(x = m \cdot 2^e\), with \(m \cdot 2^{-p} \leq x < 2^p\)
      \item \(e' \in \mathbb{N}\) with \(|e' - p + 1| < \ell\) and \(m' = m \cdot 2^{-p} \cdots m_{p-1}\)
      \item Output: \(RN(x^{-1/2}) = \text{correct rounding-to-nearest of } x^{-1/2}\)
      \item \(\varepsilon = 2^p\) and \(RN(x^{-1/2}) = RN(x)^{-1/2}\), and \(\ell \in \{0, 2\}\) and \(\varepsilon \leq d \leq \varepsilon\)
      \item Two useful properties
        \begin{itemize}
          \item \(x^{-1/2}\) falls in the range of normal floating-point numbers
          \item \(x^{-1/2}\) cannot be halfway between two consecutive floating-point numbers
        \end{itemize}
    \end{itemize}
\end{itemize}

4. How to approximate the exact value \(\ell\)?

\begin{itemize}
  \item Existing method in FLIP 0.3: multiplicative method
    \begin{itemize}
      \item Initial approximation: degree-3 univariate polynomial
      \item Refinement by Goldschmidt’s iteration \cite{8}
    \end{itemize}
  \item Our approach: one-sided truncated approximation \cite{4}
    \begin{itemize}
      \item Approximation of \(\ell\) from above by \(v = 2^{-2p} - \ell\) \(\ell < 2^{-2p} - 1\)
      \item Computation of \(u\) = truncation of \(v\) after \(p\) fraction bits
      \item Approximation of \(\ell\) = result of the evaluation of a single bivariate polynomial
        \(P(u, \ell) = 2^{-p} + v \cdot \ell\)
        \(v \in \mathbb{Q}\) a degree-9 truncated Remez approximant computed with Sollya
      \item How to evaluate \(P(u, \ell)\) efficiently?
        \begin{itemize}
          \item Horner’s rule: 38 cycles, no ILP exposure
        \end{itemize}
    \end{itemize}
  \item Efficient and certified parenthesisation automatically generated using CGPE \cite{5}
    \begin{itemize}
      \item Reduction of evaluation latency
        \[
        \rightarrow 13 \text{ cycles on unbounded parallelism, 14 cycles on ST231}
        \]
      \item Evaluation error checked with Gappa
        \[
        \rightarrow \text{ensure correct rounding}
        \]
  \item How to deduce \(RN(\ell)\) from the approximation \(\ell\)?
    \begin{itemize}
      \item Deduce \(RN(\ell)\): decide whether \(u > \ell\), which is equivalent to
        \[
        u^{-1} > 2^{-2p} - 1 \quad \text{with} \quad 2^{-2p} > 1
        \]
        \[
        u, \ell \quad \text{and} \quad 2^{-2p} \quad \text{exactly representable with 32 bits}
        \]
      \item Computing the first 64 bits of the exact product are enough
      \item Test done on the first 32 bits of the exact product
    \end{itemize}
  \item CGPE generation flowchart
    \begin{itemize}
      \item Problems
        \begin{itemize}
          \item evaluate \(P(u, \ell)\)
          \item \[
          \sum_{i=0}^{3} a_i \ell^i
          \]
          \item evaluation error no larger than \(u\)
          \item exploit at best the ILP of the ST231
        \end{itemize}
      \item Gappa certificate
      \item ST231 features
      \item \(P(u, \ell)\) evaluation code
    \end{itemize}
\end{itemize}

6. Validation and performances

\begin{itemize}
  \item Exhaustive comparison with Glibc and MPFR
  \item Performances on ST231
    \begin{itemize}
      \item 32-bit VLIW cores (with Hardware Floating-Point Unit)
    \end{itemize}
  \item Interest of the specialization of the reciprocal square root operator
    \begin{itemize}
      \item Code sequence used for computing \(x^{-1/2}\)
      \item Number of floating-point operations
      \item Control flow instructions
      \item latency
      \item Speedup
    \end{itemize}
  \item Some references
    \begin{itemize}
    \end{itemize}
\end{itemize}