Software Implementation of Decimal Floating-Point

John Harrison

johnh@ichips.intel.com

JNAO, 1st June 2006

Decimal and binary arithmetic

Some early computers like ENIAC used decimal. But nowadays:

- Most computers use binary exclusively, for both integer and floating-point
- Integer and floating-point are separate datatypes, with mainly separate register sets and functional units.

Some exceptions:

- Limited support like IA-32's DAA instruction
- IBM z900 mainframe has hardware for decimal *integer* arithmetic

Commercial applications

Binary works very well in scientific applications, but some financial operations need decimal. (So COBOL requires support for decimal fixpoint.)

- They often follow earlier methods of hand calculation
- Specific rules of calculation and rounding are legally required
- These rules are specified assuming decimal arithmetic.

Without direct support for decimal floating-point, correct rounding needs to be simulated with manual scaling and rounding in software.

Example

The following example is given by Mike Cowlishaw

> For example, consider a calculation involving a 5% sales tax on an item (such as a $\$0.70$ telephone call), rounded to the nearest cent.
>
> Using double-precision binary floating-point, the result of multiplying 0.70 by 1.05 is a little under 0.7349999999999999 whereas a calculation using decimal fractions would yield exactly $\$0.735$.
>
> The latter would be rounded up to $\$0.74$, but using the binary fraction the result returned would be the incorrect $\$0.73$.

This is a typical case where currently one would need to do correct scaling and rounding in software.

The revised IEEE-754 Standard

The IEEE-754 Standard has recent undergone a revision process

- Various improvements and clarifications, standardization of `fma`
- Merging in of the IEEE-854 radix-independent standard
- Addition of decimal floating-point datatypes

Incorporation of decimal led by enthusiasm of Mike Cowlishaw from IBM.

IEEE-754 Highlights

Notable features of the decimal formats in the Standard:

- Three formats: 32 bits (7 digits), 64 bits (16 digits) and 128 bits (34 digits).
- Numbers are represented in a redundant unnormalized format to preserve “scale” (e.g. $2.00 \times 2.00 = 4.0000$).
- One of the possible memory encodings uses an intricate ‘densely packed decimal’ (DPD).

The DPD encoding represents base-1000 digits using a Huffman-like code, to ease translation to and from individual decimal digits.

Need for hardware?

IBM in general, and Mike Cowlishaw in particular, have been making quite strong claims for the advantages of hardware support for decimal arithmetic:

> *Initial benchmarks indicate that some applications spend 50% to 90% of their time in decimal processing, because software decimal arithmetic suffers a 100 to 1000 performance penalty over hardware. The need for decimal floating-point in hardware is urgent.*

Based on our results, both claims are dubious.

Real applications

Study of customer profiles for various apps involving decimal.

- Database benchmarks
- Real financial industry application
- Profiles from real database apps
- SPECjbb Java benchmark

In no case has the decimal runtime been more than about 5%, and it's often much lower.

Even though the software implementations used are *very* slow!

Unreal applications: Telco

In support of his claims, Cowlishaw gives a small and simple synthetic benchmark called “Telco”.

`http://www2.hursley.ibm.com/decimal/telco.html`

This is intended to be representative of telephone billing, with a representative balance of computation and I/O.

Telephone calls are charged according to several rates, and several taxes are applied, all with decimal rounding rules.

Telco profiling

Cowlishaw runs Telco using his own well-written but *generic* library for decimal arithmetic.

Depending on system details, he finds that decimal arithmetic accounts for between 72.2% and 93.2% of total runtime.

83.4% on our test system.

Our main thesis

Telco is avowedly an extreme case of domination by decimal computation, and unlikely to be exceeded by any real application.

If we can speed up the decimal processing so it no longer dominates the runtime, further gains from an even faster (e.g. hardware) implementation cannot be dramatic.

We probably *can* do this with a careful software implementation.

Our initial experiments targeted the Intel Itanium architecture.

The binary approach

We chose to represent numbers using a *binary* integer for the coefficient.

- Core arithmetic operations are very quick and direct using standard binary primitives like addition and multiplication.
- Decimal rounding and conversion to/from strings are less easy, but can be made efficient with careful attention to algorithms.

Fast rounding

To perform decimal rounding, we need to perform integer rounding of $x/10^k$ for integers k.

Since there are only a few possible values of k, we can exploit standard techniques of reciprocal multiplication $(w \cdot x) \gg m$.

An extension of this technique allows us to check divisibility properties by looking at another bitfield of the product $w \cdot x$.

Fast conversion to decimal digits (1)

This is potentially a very costly operation if done naively. For example a truncating division:

$$x \mapsto \lfloor x/10 \rfloor$$

within the inner loop would be *very* slow.

We use a nice technique to obtain the various digits independently and in parallel.

They can then be passed to the integer register and output as characters one per cycle.

Fast conversion to decimal digits (2)

Form the various quotients $Q_k = \lfloor x/10^k \rfloor$ using reciprocal multiplication.

Now get the digits by $d_k = Q_k - 10 \cdot Q_{k+1}$.

On a pipelined machine, these can be streamed through at one digit per cycle.

Experiments

We tried Telco with our prototype implementation and with Cowlishaw's `decNumber` to measure cycle times for individual operations:

Operation	Estimated	Actual	`decNumber`
From packed	38	63	126
To string	45 + C	119	90
Addition	21	25	83
Multiplication	16	18	198
Rescaling	52	56	317

Certainly no "100 to 1000 performance penalty over hardware", even for Cowlishaw's `decNumber`.

Decimal proportions

Global effect on Telco's performance:

	Ours	`decNumber`	Ratio
Total time (seconds)	0.950	2.831	2.98
Decimal time (seconds)	0.425	2.363	5.56
Decimal % of overall time	44.7%	83.4%	

We get a 3x overall improvement in application performance.

Decimal computation is less than half of overall runtime, rather than the dominant component.

And this is by no means highly tuned.

Conclusions

- Decimal arithmetic is almost certainly going to be in the new Standard, and many customers want it.
- IBM has been making considerable claims for the benefits of a hardware implementation.
- Our experiments suggest that a good software implementation may be entirely satisfactory.
- A binary representation seems better for software, and perhaps for hardware too (better sharing/evolution path).
- We don’t want to have the overhead of conversion to and from DPD, hence the need for a binary-friendly encoding.