# How Fast, Exact Computing Works

**James Douglas Boyd**
**Founder and CEO/CTO, SciSci Research**

## Beyond Floating-Point

SciSci is building its exact processing units (EPUs) based on an architecture involving an alternative number representation and logic design to floating-point, enabling faster, perfect-precision compute. I'll explain here, as much as I can, what this alternative's promise is and how it works.

The floating-point (FP) format makes rounding errors because the it can't encode numbers exactly, no matter how many bits it uses. This is due to the mathematical properties of floating-point numbers. From the point of view of number theory, floating-point errors are inevitable because floating-point numbers are real numbers (i.e., in $\mathbb{R}$) approximated by rationals (i.e., in $\mathbb{Q}$), and these are unavoidably prone to rounding errors. Fortunately, from a number theory point of view, there's an exact alternative to FP.

## The Origin of Rounding Errors

Every real number is given as some (infinite) sequence, called a Cauchy sequence. (This can just be thought of as the decimal expansion of a number, such as $5.1621\ldots$) The trouble with $\mathbb{R}$ is that these Cauchy sequences are non-unique; they belong to equivalence classes. Take the example of the number 1. It can be encoded as $1.000\ldots$ or $0.999\ldots$ Both converge to 1.

Why is non-uniqueness a problem? Let's cut off each of these Cauchy sequences at two significant figures. One is 1.0, and the other is 0.99; the rounding error, 0.010, is the difference between these non-unique sequences under finite cutoff. One might say, "let's just use 1.0 then; forget 0.99", but if we want to encode, say, $\frac{1}{3}$, we have to use 0.99 and divide by 3, giving us 0.33; that's why the decimal expansion for $\frac{1}{3}$ always admits some error. So, floating-point rounding error, technically speaking, is just a consequence of the fact that $\mathbb{R}$ doesn't support unique encodings.

## Exact Arithmetic

On the other hand, if Cauchy sequences are unique, this problem doesn't arise; one can do exact arithmetic. The only way to get Cauchy sequences, however, is to give a "completion" of $\mathbb{Q}$ (i.e., a larger field that can support infinite sequences), and $\mathbb{R}$ is easily the best-known completion. From Ostrowski's theorem, however, we know that there is one alternative to $\mathbb{R}$, the $p$-adic fields $\mathbb{Q}_p$. The $p$-adic fields have unique Cauchy sequences, allowing exact arithmetic. SciSci's EPU number representation and architecture is based on $p$-adic arithmetic.

## Why Exact Arithmetic is Fast

Although I can't discuss SciSci's instruction set architecture (ISA) in detail, I will say that it works with exact number representations in a way that enables major parallelization unlocks for arithmetic, lending the ISA to the extremely multi-core implementation of the EPU. Such parallelization makes the ISA suitable for an AI accelerator chip, and also contributes to speed advantages over floating-point.

Another speed advantage, which I can more liberally discuss, has to do with a per-operation efficiency gain from exact arithmetic. Floating-point arithmetic follows the IEEE standard, which encodes numbers using an exponent/mantissa format. Adding these numbers is actually rather laborious, requiring separate steps for exponents, mantissas, rounding, etc. By contrast, the EPU addition involves only one single step.

All told, the EPU should enjoy a $\sim 15\times$ efficiency gain *per operation per ALU* over floating-point. Now, consider how that scales up for an extremely multi-core EPU... For an EPU with as many ALUs as a Blackwell Ultra, that's a gain of roughly a million operations.[1]

[1] The Blackwell Ultra has 160 SMs, each with 128 CUDA cores. $160 \times 128 \times 15 = 921600$.