Dividing by Zero – How and Why?

Dr James A.D.W. Anderson

安德生

Agenda

- A word of comfort
- The story of complex numbers
- How to divide by zero
- Advantages for computing!
- Where do we go from here?

A Word of Comfort

- Dividing by zero is no more mysterious than finding the square root of a negative number
- Transreal arithmetic divides by zero using only accepted algorithms of arithmetic – so you already know how to divide by zero
- There is a machine proof that transreal arithmetic is consistent if real arithmetic is
- Every real result of mathematics stays the same, but there are some new, non-finite, results
- You can try out an implementation of transcomplex arithmetic

Can Calculators Divide by Zero?

- If you have an electronic calculator with you then turn it on and stand up
- Pick a number and divide it by zero on your calculator
- If your calculator shows an error or has crashed then sit down
- If your calculator is still working then multiply the current answer by zero
- If your calculator shows an error or has crashed then sit down
- Is there anyone left standing?

Can Computers Divide by Zero?

- My laptop can divide by zero. My FPGA model of a transreal computer can divide by zero. All of the PCs I have retrofitted with these arithmetics can divide by zero: trans-signed-integer, trans-two's-complement, transfloating-point, transreal, transcomplex

- Computers executing integer arithmetic cannot divide by zero

- Computers executing IEEE floating-point arithmetic cannot divide by zero. They produce a class of objects that are all Not a Number (NaN)

Can Computers Divide by Zero?

- The bridge of the missile cruiser, USS Yorktown, had networked computer control of navigation, engine monitoring, fuel control, machinery control, and damage control

Can Computers Divide by Zero?

- On September 21st, 1997, a sailor on the USS Yorktown entered a zero into a database field, causing a division by zero error which cascaded through the ship's network, crashing every computer on the network, and leaving the ship dead in the water for 2 hours 45 minutes
- The world would be a safer place if computers, calculators and people could divide numbers by zero, getting a number as an answer
- Coincidentally, I worked out how to do this in 1997

Complex Numbers

People used to believe that it is impossible to find the square root of a negative number

- $\sqrt{-4} = ?$
- $2 \times 2 = 4$
- $(-2) \times (-2) = 4$

Complex Numbers

- Invent a new number $i = j = \sqrt{-1}$
- Use only accepted algorithms of arithmetic
- BUT change the way the algorithms are applied by making addition non-absorptive, i.e. keep real and imaginary sums separate

Complex Numbers

- For example, complex multiplication is defined by:

$$\begin{aligned}(a+ib)(c+id) &= a(c+id)+ib(c+id)\\ &= ac+iad+ibc+i^2bd\\ &= ac+iad+ibc+(-1)bd\\ &= (ac-bd)+i(ad+bc)\\ &= k_1+ik_2\end{aligned}$$

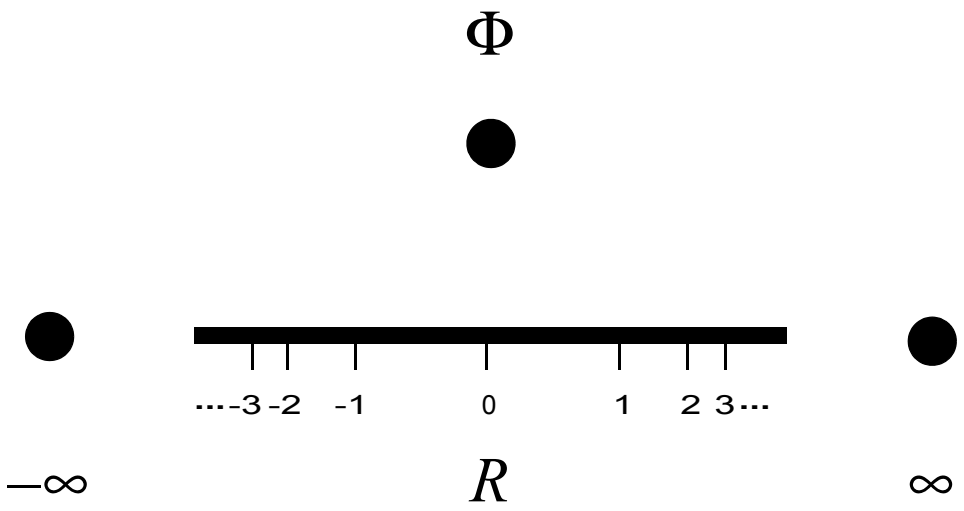
- Now $i2\times i2 = i^2 4 = -1\times 4 = -4$ so $\sqrt{-4} = i2$

Transreal Numbers

Invent some new numbers. For all $k > 0$ we define:

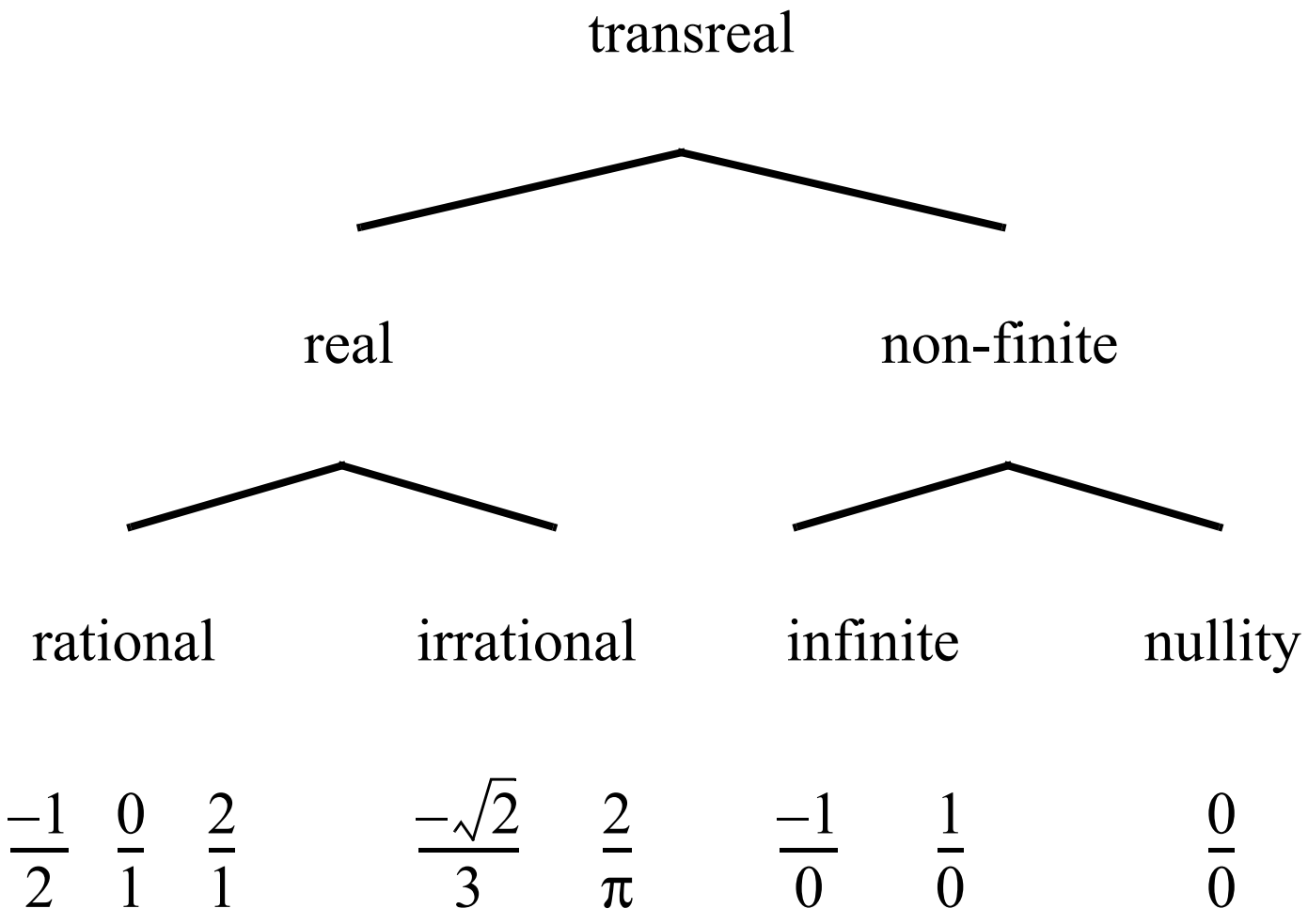
- $\infty = \frac{1}{0} \equiv \frac{k}{0}$
- $\Phi = \frac{0}{0}$
- $-\infty = \frac{-1}{0} \equiv \frac{-k}{0}$
- $0 = \frac{0}{1} \equiv \frac{0}{k} \equiv \frac{0}{-k}$

Transreal Numbers



- Positive infinity, ∞, is the biggest transreal number
- Negative infinity, −∞, is the smallest transreal number
- Nullity, Φ, is the only transreal number that is not negative, not zero, and not positive

Transreal Numbers



Transreal Fractions

A *transreal number* is a *transreal fraction* of the form $\frac{n}{d}$, where:

- n is the *numerator* of the fraction
- d is the *denominator* of the fraction
- n, d are *real numbers*
- $d \geq 0$
- Fractions with non-finite components simplify to the above form

Transreal Fractions

- An *improper transreal fraction,* $\frac{n}{-d}$, may have a negative denominator, $-d < 0$

- An improper transreal fraction is converted to a *proper transreal fraction* by multiplying both the numerator and denominator by minus one; or by negating both the numerator and the denominator, using subtraction; or it can be done, instrumentally, by moving the minus sign from the denominator to the numerator

$$\frac{n}{-d} = \frac{-1 \times n}{-1 \times (-d)} = \frac{-n}{-(-d)} = \frac{-n}{d}$$

Transreal Fractions

- Example: $\frac{2}{-3} = \frac{-1 \times 2}{-1 \times (-3)} = \frac{-2}{3}$

- Example: $\frac{0}{-1} = \frac{-1 \times 0}{-1 \times (-1)} = \frac{0}{1}$

Transreal Multiplication

Two *proper transreal fractions* are multiplied like this:

- $\frac{a}{b} \times \frac{c}{d} = \frac{a \times c}{b \times d}$

- Example: $3 \times \infty = \frac{3}{1} \times \frac{1}{0} = \frac{3 \times 1}{1 \times 0} = \frac{3}{0} = \infty$

- Example: $0 \times \infty = \frac{0}{1} \times \frac{1}{0} = \frac{0 \times 1}{1 \times 0} = \frac{0}{0} = \Phi$

- Example: $\frac{1}{2} \times \frac{3}{5} = \frac{1 \times 3}{2 \times 5} = \frac{3}{10}$

Transreal Division

Two *proper transreal fractions* are divided like this:

- $$\frac{a}{b} \div \frac{c}{d} = \frac{a}{b} \times \frac{d}{c}$$

- Example: $$\infty \div 3 = \frac{1}{0} \div \frac{3}{1} = \frac{1}{0} \times \frac{1}{3} = \frac{1 \times 1}{0 \times 3} = \frac{1}{0} = \infty$$

- Example:

$$\infty \div (-3) = \frac{1}{0} \div \frac{-3}{1} = \frac{1}{0} \times \frac{1}{-3} = \frac{1}{0} \times \frac{-1 \times 1}{-1 \times (-3)}$$

$$= \frac{1}{0} \times \frac{-1}{3} = \frac{1 \times (-1)}{0 \times 3} = \frac{-1}{0} = -\infty$$

Transreal division

- Example: $\frac{1}{2} \div \frac{5}{3} = \frac{1}{2} \times \frac{3}{5} = \frac{1 \times 3}{2 \times 5} = \frac{3}{10}$

Transreal Addition

Two *proper transreal fractions* are added like this:

- $\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$, except that:

- $(\pm\infty) + (\pm\infty) = \frac{\pm 1}{0} + \frac{\pm 1}{0} = \frac{(\pm 1) + (\pm 1)}{0}$

Transreal Addition

- $(\pm\infty) + (\pm\infty) = \frac{\pm 1}{0} + \frac{\pm 1}{0} = \frac{(\pm 1) + (\pm 1)}{0}$

Examples:

- $\infty + \infty = \frac{1}{0} + \frac{1}{0} = \frac{1 + 1}{0} = \frac{2}{0} = \infty$

- $(-\infty) + (-\infty) = \frac{-1}{0} + \frac{-1}{0} = \frac{(-1) + (-1)}{0} = \frac{-2}{0} = -\infty$

- $\infty + (-\infty) = \frac{1}{0} + \frac{-1}{0} = \frac{1 + (-1)}{0} = \frac{0}{0} = \Phi$

Transreal Addition

- $\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$

Examples:

- $\frac{2}{3} + \infty = \frac{2}{3} + \frac{1}{0} = \frac{2 \times 0 + 3 \times 1}{3 \times 0} = \frac{3}{0} = \infty$
- $\frac{2}{3} + \Phi = \frac{2}{3} + \frac{0}{0} = \frac{2 \times 0 + 3 \times 0}{3 \times 0} = \frac{0}{0} = \Phi$
- $\frac{2}{3} + \frac{4}{5} = \frac{2 \times 5 + 3 \times 4}{3 \times 5} = \frac{22}{15}$

Transreal Subtraction

Two *proper transreal fractions* are subtracted like this:

- $$\frac{a}{b} - \frac{c}{d} = \frac{a}{b} + \frac{-c}{d}$$

Examples:

- $$\infty - \infty = \frac{1}{0} - \frac{1}{0} = \frac{1}{0} + \frac{-1}{0} = \frac{1 + (-1)}{0} = \frac{1 - 1}{0} = \frac{0}{0} = \Phi$$

- $$\frac{1}{2} - \frac{3}{5} = \frac{1}{2} + \frac{-3}{5} = \frac{(1 \times 5) + (2 \times (-3))}{2 \times 5} = \frac{5 + (-6)}{10} = \frac{-1}{10}$$

Transreal Arithmetic

- Transreal arithmetic is a superset of real arithmetic
- Transreal arithmetic is *total* – every operation of transreal arithmetic can be applied to any transreal numbers with the result being a transreal number
- Real arithmetic is *partial* – it fails on division by zero and on each of the infinitely many mathematical consequences of division by zero

Moral

- The arithmetic you have just seen has been taught to 12 year old children in England
- These children understand infinity and nullity
- These children use an arithmetic that never fails
- What do you want for your children?
- What do you want for your computers?
- What do you want for your self?

Advantages for Computing!

- All mathematical software can be extended to use transreal or transcomplex numbers

Advantages for Computing!

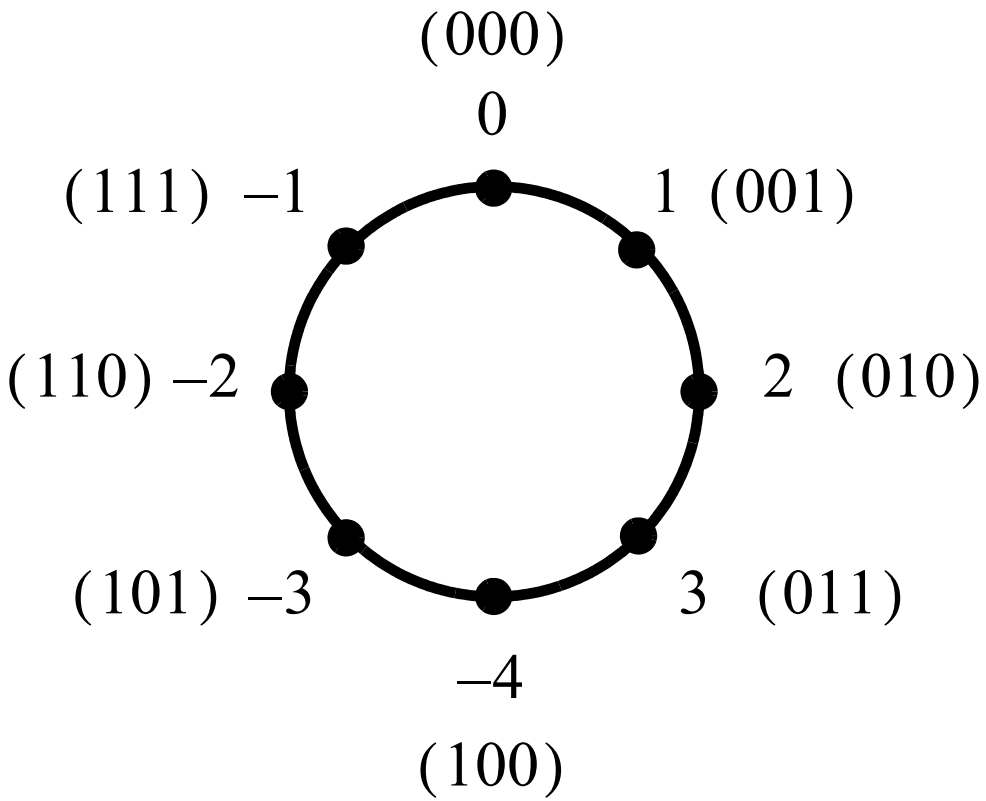
Every syntactically correct transarithmetical expression is semantically correct so:

- Compilers can perform full type checking
- There are no arithmetical run-time errors
- Any Turing program can be executed without any logical run-time errors
- Pipelines are Turing computable so, in a finite machine of sufficient size, pipelines never break and entire programs can be pipelined

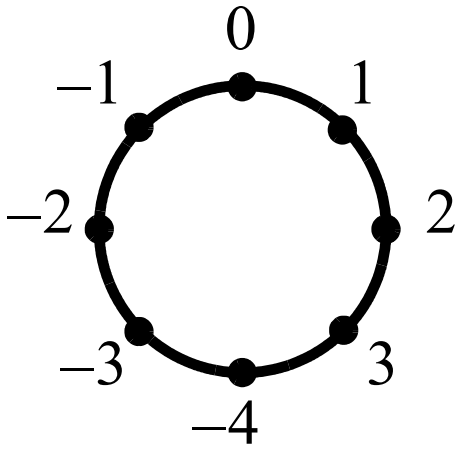
Advantages for Computing!

- Transreal arithmetic removes an intrinsic bug from two's complement arithmetic, making both hardware and software safer
- Floating-point hardware can have all wasted states re-allocated to transreal numbers, thereby improving arithmetical range or precision
- Floating-point hardware and software can have simplified ordering operations and exceptions
- It is possible to remove all exceptions from floating-point hardware by reserving an inexact flag in the number representation and by knowing the rounding mode

Two's Complement

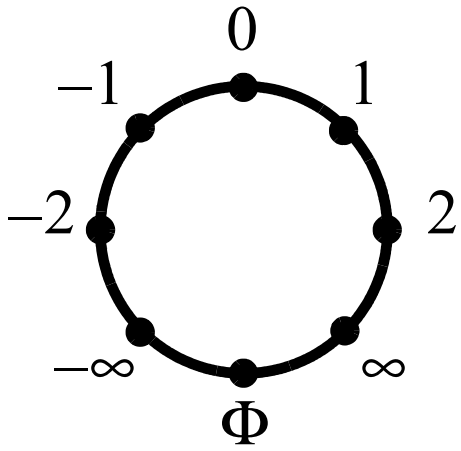


Two's Complement



- The complement of the most negative number is not its negation $-(-4) = -4$
- Almost every computer suffers this weird-number fault

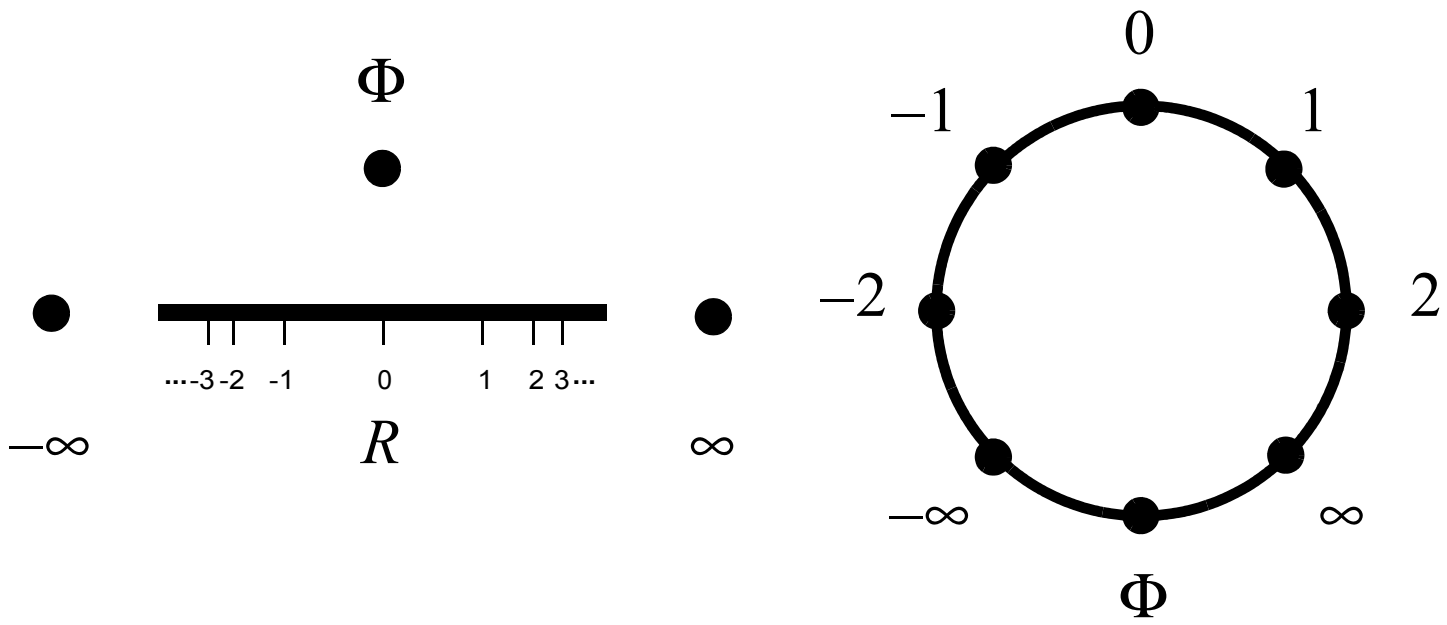
Trans Two's Complement



- The complement of the most negative number is now its negation $-(-\infty) = \infty$
- And the complement of nullity is its negation $-\Phi = \Phi$

Trans Two's Complement

- Trans two's complement removes the weird-number fault and preserves the topology of the transreal numbers



Trans Two's Complement

- Trans two's complement gives transinteger and transfixed-point programming superior exception handling to ordinary floating-point arithmetic

- The topology of transreal numbers extends to floating-point arithmetic so that it can match, and exceed, the exception handling of transinteger and transfixed arithmetic

Floating-Point

- IEEE 754 defines floating point numbers in terms of three of bit fields that encode the Sign (S), Exponent (E) and Mantissa (M)

- In general, a floating point number $n = -1^{S}2^{E}M$, but bit patterns are reserved for –0, –∞, ∞, NaN_i where $i = 2^{m+1} - 2$ with m being the number of bits explicitly represented in the mantissa. The "+1" arises from the sign bit and the "–2" from –∞ and +∞

- IEEE arithmetic encodes –0, but –0 does not occur in transreal arithmetic so transfloating-point arithmetic reallocates the binary code for –0 to Φ

Floating-Point

- Nullity now lies in the middle of the lexical collation range of floating-point numbers so sorting routines must handle the unique nullity as a special case (IEEE sorting routines must handle all NaNs as special cases)
- Notice that IEEE arithmetic collates numbers in reverse order because S is the most significant bit and $S = 1$ encodes negative numbers
- IEEE arithmetic uses reverse collation so that the binary codes for integer zero and floating-point positive-zero are identical. As many computers have an integer test zero instruction this makes positive-zero comparisons quick

Floating-Point

- Transfloating-point arithmetic uses the most positive binary code for ∞ and the most negative binary code for $-\infty$

- IEEE arithmetic has $2^{m+1} - 2$ NaNs, but transreal arithmetic has no NaNs so all of these states can be reallocated to real numbers

- Taking the reallocated codes as signed numbers means that there are $2^m - 1$ new mantissas. This is almost one binade: $2^m - 1$ bit patterns in a near binade as against 2^m bit patterns in an entire binade

Floating-Point

- Leaving the exponent bias unchanged takes this near binade with a positive exponent so as to almost double the arithmetical range of the real floating-point numbers
- Incrementing the bias takes this near binade with a positive exponent, but frees up an entire binade with a negative exponent, thereby increasing precision by delaying underflow to denormal numbers
- Only one of these options can be taken in one thread: either increase the range or else increase the precision

Floating-Point

Name	Common Name	m	Wasted NaN States
binary16	half precision	10	2 046
binary32	single precision	23	16 777 214
binary64	double precision	52	9 007 199 254 740 990
binary128	quadruple precision	112	10 384 593 717 069 655 257 060 992 658 440 190

Floating-Point

- The IEEE standard defines four relational operations: *less-than* (<), *equal* (=), *greater-than* (>), *unordered* (?)

- Transreal arithmetic defines three relational operations: *less-than* (<), *equal* (=), *greater-than* (>)

Floating-Point

- The IEEE standard defines 14 composite relations:
  =, ?<>, >, >=, <, <=, ?, <>, <=>, ?>, ?>=, ?<, ?<=, ?=

- The IEEE standard defines negations of 12 out of 14 of the composite relations:
  NOT(>), NOT(>=), NOT(<), NOT(<=), NOT(?),
  NOT(<>), NOT(<=>), NOT(?>), NOT(?>=),
  NOT(?<), NOT(?<=), NOT(?=)

- I have never seen a computer language that supports all of these 26 composite relations with 26 relational operators

Floating-Point

- Transreal arithmetic supports 7 composite relations:
  =, >, >=, <, <=, <>, <=>

- Transreal arithmetic supports negations of all of the composite relations:
  !=, !>, !>=, !<, !<=, !<>, !<=>

- Transreal arithmetic preserves symmetry (orthogonality) of negation that the IEEE standard breaks, this makes it easier to program with transreal numbers

Floating-Point

- 12 of the IEEE relations can raise an exception: >, >=, <, <=, <>, <=>, NOT(>), NOT(>=), NOT(<), NOT(<=), NOT(<>), NOT(<=>)
- Specifically, all of the relations that do not contain the predicate *unordered* can raise an exception on NaN
- None of the transreal relations can raise an exception so it is easier and safer to program with transreal numbers

Floating-Point

- The IEEE standard defines that $\mathrm{NaN}_i \neq \mathrm{NaN}_j$ so that it is false that $x = x$ for some floating-point objects, x
- The above breaks a cultural stereotype that everything is equal to itself and destroys equality in mathematical physics so that mathematical physics does not work with NaN

Floating-Point

- Transreal arithmetic has $x = x$ for all transreal numbers, x

- The above preserves a cultural stereotype that everything is equal to itself and maintains equality in mathematical physics

High Performance Computing

I have designed a new computer architecture and I have built a team to design, manufacture, test, and sell it!

- Pass tokens on a pipeline so that all cores can simultaneously receive and transmit tokens in every direction with zero Input/Output (I/O) latency
- Hold all working memory in on-chip cache, thereby accessing memory at processor speeds
- Run I/O from every edge of the chip with zero latency so that there is no memory wall

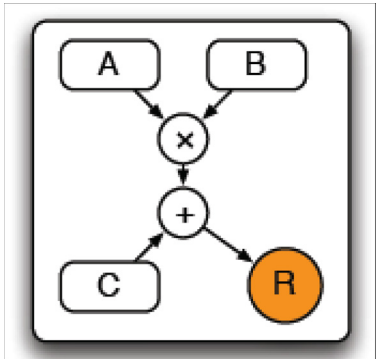
High Performance Computing

Hardware layouts say we can achieve:

- order 10 k double-precision floating-point cores on a chip
- order 10 kW per PFLOP of double-precision floating-point arithmetic

High Performance Computing

- The only arithmetical operation is $A \times B + C \rightarrow R$:



High Performance Computing

- All other arithmetical operations are synthesised from this one, because fabricating them would waste space in most of the processors on a chip

- The floating-point chip also has one non-arithmetical operation

High Performance Computing

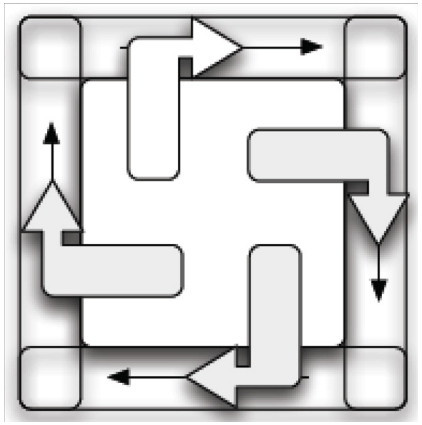
Fetchless architecture:

- Minimises circuitry
- Reduces on-chip fetch-latency to zero

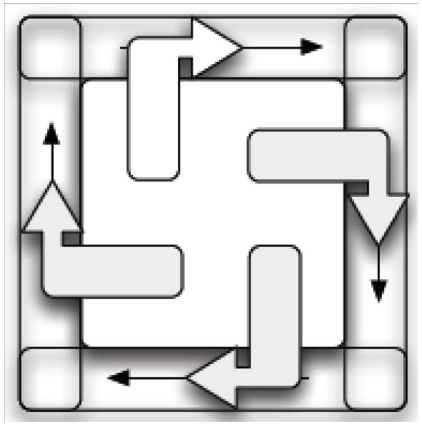
Achieved by token passing

High Performance Computing

- Tokens transmitted, conditionally on the sign of $A \times B + C \rightarrow R$, through every edge of the square processor and internally (Grey arrow blocked, white transmitted)

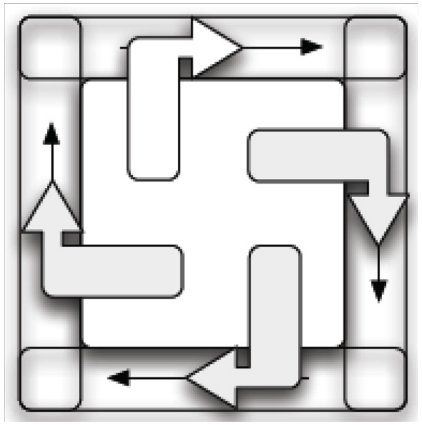


High Performance Computing



- There are four signs – negative, zero, positive, nullity – encoded in two sign bits so all selectors are maximally efficient

High Performance Computing



- If desired, each of the four separate signs can transmit a token in a different direction

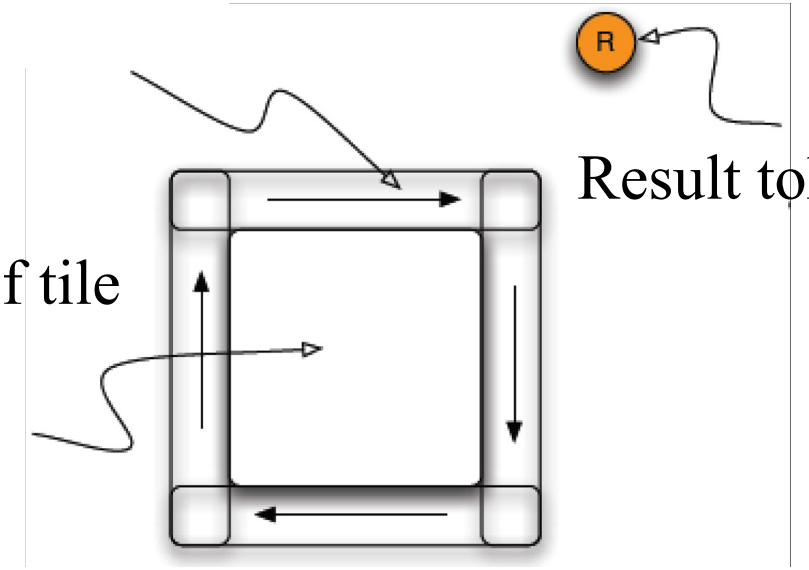
High Performance Computing

There are no arithmetical error states so:

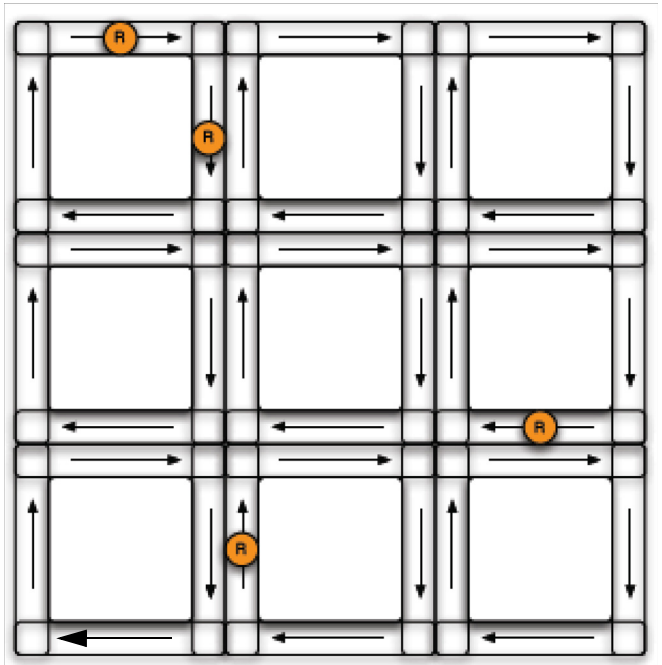
- There is no arithmetical error handling circuitry on a processor
- Program execution is more secure

Core Tile

Pipeline
N, S, E, W



Pipeline

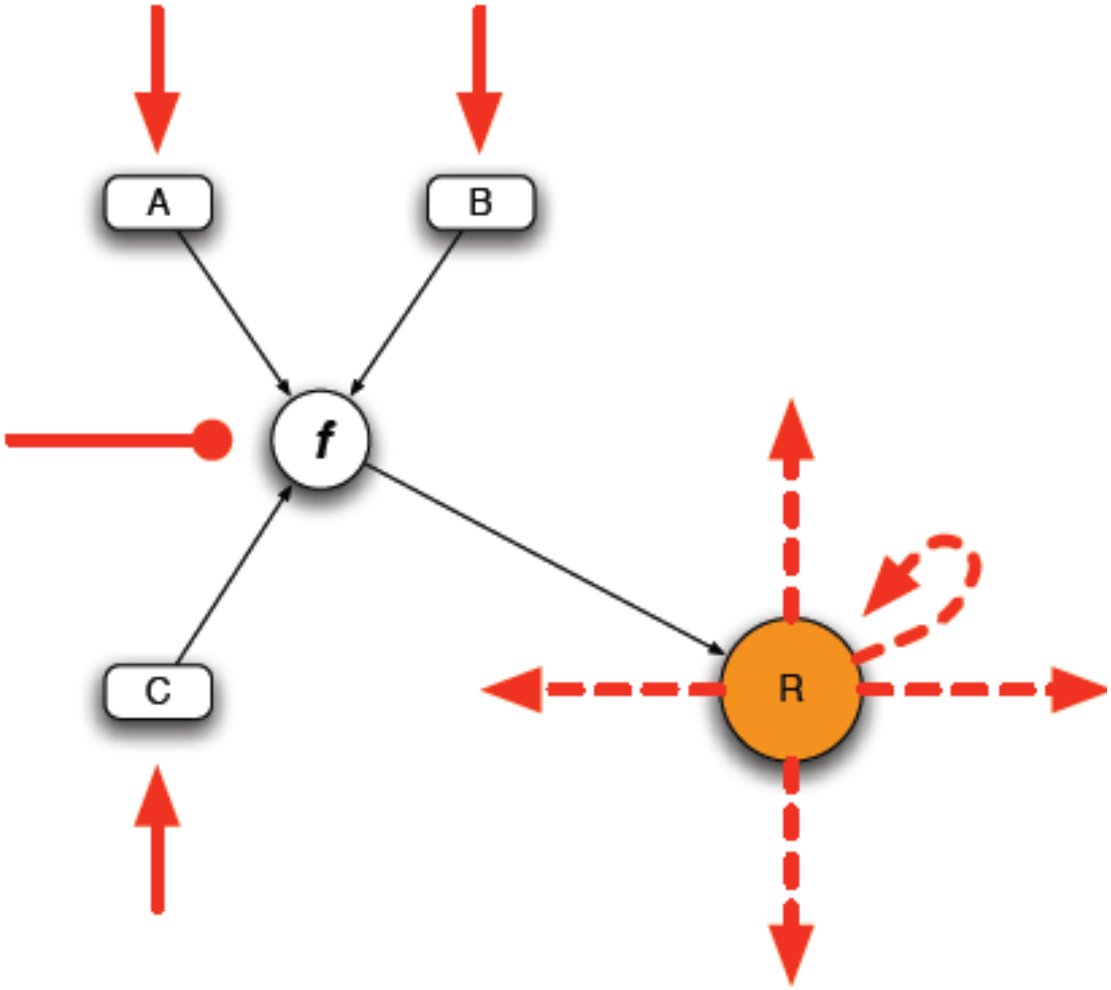


- Pipeline emerges from the processor tiles

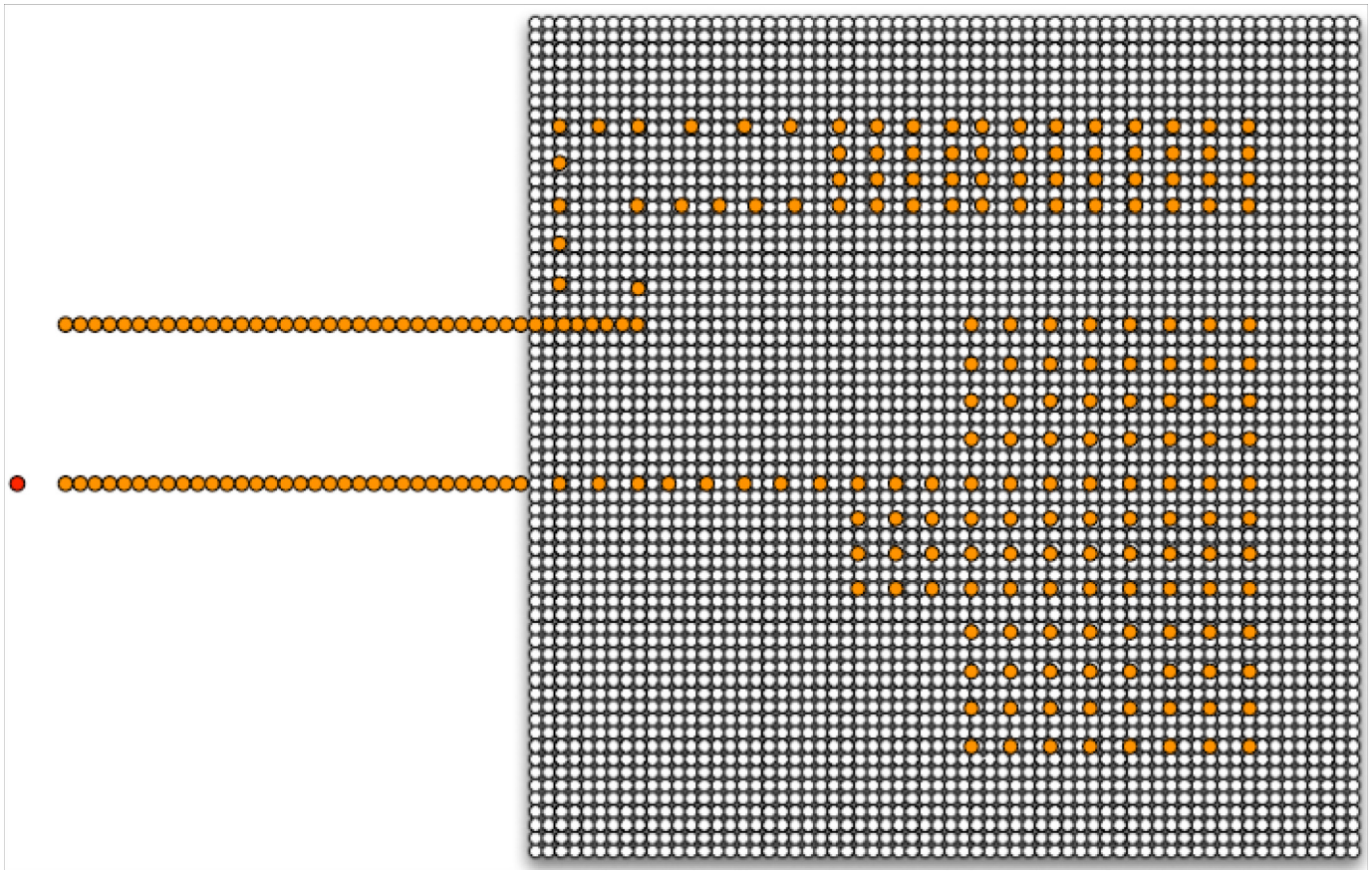
Pipeline

- Every core can simultaneously read from and write to the pipeline on every processor clock cycle with zero latency

Programming: an Army of Ants



Programming: Initial Orders



Programming: Is it Possible?

We have programmed these machines:

- Emulated a Turing complete machine
- Eliminated race conditions by using a single thread
- Eliminated race conditions by travel-time inequalities
- Implemented fully pipelined, mathematical functions with a throughput of one result per clock tick, and a latency down to half that of the Itanium 2 on: reciprocal, reciprocal square root, square root, exponential

Programming: Is it Possible?

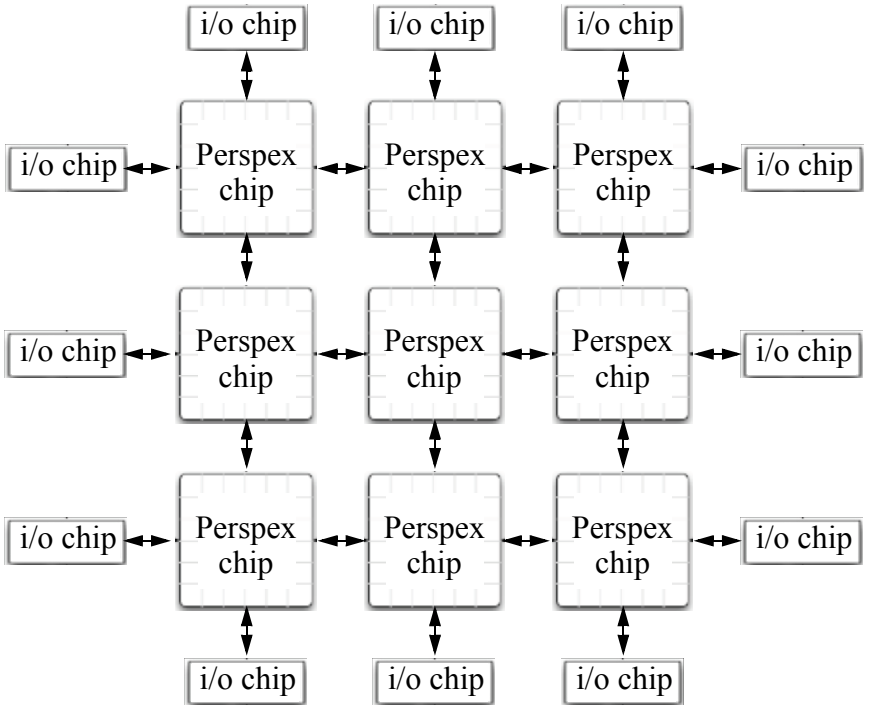
- Pipelines do not break on branches, they just tee-off in some city-block direction within the 2D surface of a chip
- Non-recursive subroutines have call and return implemented by branching so when they are in-lined they do not break pipelines
- Multiple calling points for a single subroutine introduce bubbles in each pipeline, and may break the pipeline – but code replication prevents this
- Loops force pipeline data into blocks of a size that will fit inside the loop, this breaks the pipeline unless the machine is huge or the problem is small

Compilation

- General compilation is straightforward, but uses our high performance computer as a co-processor
- Compilation via single assignment (functional programming) is attractive because of pipelining
- Systolic programming is highly applicable, but with a different I/O model
- Pipeline programming is highly applicable

I/O

- Chips can be tiled together with one pipeline input and one pipeline output per edge of the chip, each running at 250 M Tokens Per Second



I/O

- Chips have an address horizon, not an address space, so arbitrarily many chips can be tiled together
- Hence the performance of the architecture scales linearly

Performance

The current specification of the chip has:

- order 10 k cores
- cores clocked at 250 MHz
- 4 input channels, and 4 output channels, each running at 250 M Tokens Per Second
- power consumption of order 10 kW per PFLOP

Performance

- We have implemented conventional programs
- We have implemented systolic programs
- We have implemented pipelined programs
- We get better pipeline latencies than other architectures
- We get better pipeline throughput than other architectures
- We can sometimes match, but can never beat, systolic architectures

Performance

- It is not practical to implement *programmable* systolic arrays in ASIC, but our chip is a viable alternative
- Compiling by travel-time inequalities builds in some robustness to asynchronous operation of the array of processors
- Asynchronicity can be built into the array to smooth power usage
- Our power consumption is of order 1% of competing architectures

Offer

- We plan to sell high performance computers at 5 M US Dollars (USD) per PFLOP, three years after funding a start-up company
- We offer a discount of two, 1 PFLOP machines for 5 M USD for anyone willing to make staged payments of one third on contract, one third on tape out, and one third on delivery
- We plan to accept discounted orders for 5 to 10 PFLOPs
- If the project fails, we will return all unspent monies – this spreads the risk between customers

The Future

- Transmathematics and its application to transphysics will develop slowly
- Transcomputation can deliver benefits now
- The first company to sell a trans-floating-point chip will kill its competition on marketing strengths: more precision for the same bits and astronomically fewer error states
- The first company to deliver massively pipelined transreal computers will transform the market for high performance computing, signal processing, cryptanalysis, and so on …