The Perspex Machine

Dr James A.D.W. Anderson
Computer Science
Reading University
England

Agenda

- Part 1 – Transreal arithmetic: avoiding exceptions
- Part 2 – 4D Perspex Machines: robustness to hardware faults
- Part 3 – 2D Perspex Machines: fast computation

Part 1 – Transreal Arithmetic

- Transreal arithmetic uses only the existing algorithms of arithmetic, but ignores the injunction not to divide by zero, in such a way that it preserves the maximum possible information about the magnitude and sign of numbers. It has no exceptions

- Transreal arithmetic has been proved consistent by translating its axioms into higher order logic and testing them in a computer proof system

- Over 40,000 people have obtained a copy of the published paper describing the consistency proof. No fault has been reported, but only one person has acknowledged trying to find a fault

Transreal Numbers

Transreal numbers are *fractions,* f, of a real *numerator,* n, and a real *denominator,* d, such that $f = n/d$

James A.D.W. Anderson
Manchester 2008

Strictly Transreal Numbers

The strictly transreal numbers are:

- Positive infinity, $\infty = \dfrac{1}{0}$
- Nullity, $\Phi = \dfrac{0}{0}$
- Negative infinity, $-\infty = \dfrac{-1}{0}$

Note that a fraction with a strictly transreal numerator and/or denominator simplifies to a fraction with a real numerator and denominator

James A.D.W. Anderson
Manchester 2008

Canonical Form

The canonical form of a transreal number, n/d:

- Is $1/0$ when $n > 0$ and $d = 0$
- Is $0/0$ when $n = d = 0$
- Is $-1/0$ when $n < 0$ and $d = 0$
- Is n'/d' where $n = kn'$ and $d = kd'$ and $d' > 0$, where k is the highest, common, factor between n, d when n, d are both integral
- Is $(nd^{-1})/1$ when nd^{-1} is irrational

James A.D.W. Anderson
Manchester 2008

Irrational Fractions

There are not enough names to name every real number so we often chose not to write irrational fractions in canonical form. For example:

- $f = \pi \div 2 = \dfrac{\pi}{1} \div \dfrac{2}{1} = \dfrac{\pi}{1} \times \dfrac{1}{2} = \dfrac{\pi \times 1}{1 \times 2} = \dfrac{\pi}{2}$

Here $\pi / 2$ is not in canonical form. Nonetheless, we may write irrational fractions in canonical form by introducing an intermediate variable. For example:

- $f = \dfrac{n}{1}$ where $n = \dfrac{\pi}{2}$

James A.D.W. Anderson
Manchester 2008

Division and Multiplication

Division is as easy as multiplication:

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

- Division by zero occurs when at least one of b, c, d is zero

James A.D.W. Anderson

Manchester 2008

Division and Multiplication

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

- I used to require that every number, a/b, c/d, d/c is reduced to canonical form before it is operated on, but it is possible to take a more relaxed approach:
- If the denominator of any argument to a multiplication is zero then as many factors $(-1)/(-1)$ are included as are needed to make all of the denominators non-negative

James A.D.W. Anderson
Manchester 2008

Addition and Subtraction

Addition and subtraction are harder than division and multiplication:

- $\dfrac{a}{b} + \dfrac{c}{d} = \dfrac{(a \times d) + (c \times b)}{b \times d}$ in general, but
- $\dfrac{\pm 1}{0} + \dfrac{\pm 1}{0} = \dfrac{(\pm 1) + (\pm 1)}{0}$ in particular
- Subtraction occurs when at least one of the arguments to addition is negative

James A.D.W. Anderson
Manchester 2008

Addition and Subtraction

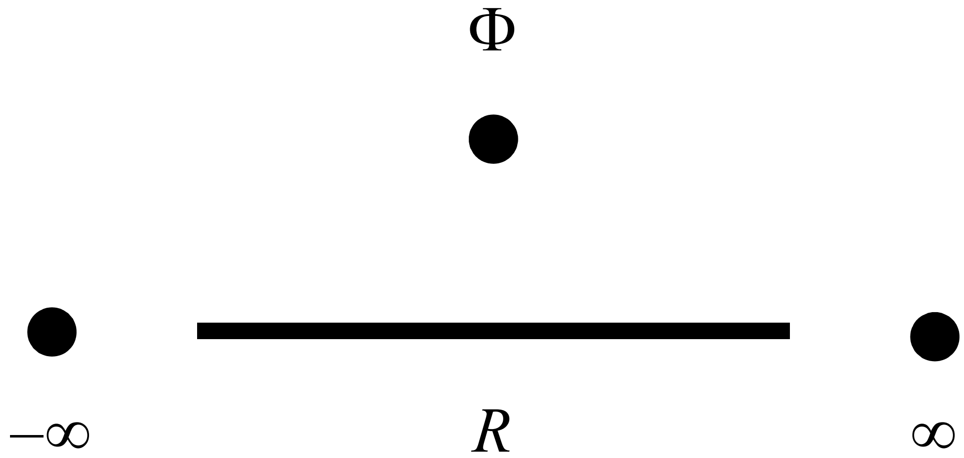
- $\dfrac{a}{b} + \dfrac{c}{d} = \dfrac{(a \times d) + (c \times b)}{b \times d}$ in general, but
- $\dfrac{\pm 1}{0} + \dfrac{\pm 1}{0} = \dfrac{(\pm 1) + (\pm 1)}{0}$ in particular
- I used to require that every number a/b, c/d, $k/0$ is reduced to canonical form before it is operated on, but it is possible to take a more relaxed approach:
- If any argument to an addition has a zero denominator then that fraction is reduced to canonical form and as many factors $(-1)/(-1)$ are included as are needed to make all of the denominators non-negative

Topological Spaces

The open sets of the transreal numbers arise from:

$$R, \{-\infty\}, \{\infty\}, \{\Phi\}$$

And can be visualised as:



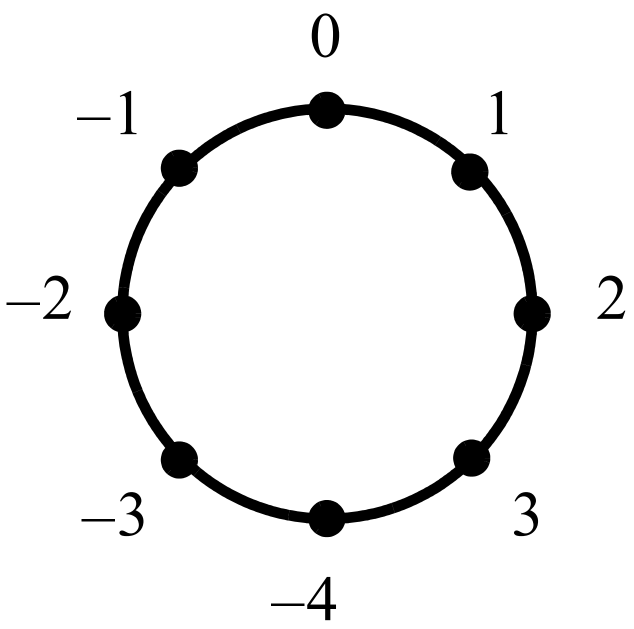
- $\{-\infty\} \cup R \cup \{\infty\}$ is the extended-real line

James A.D.W. Anderson

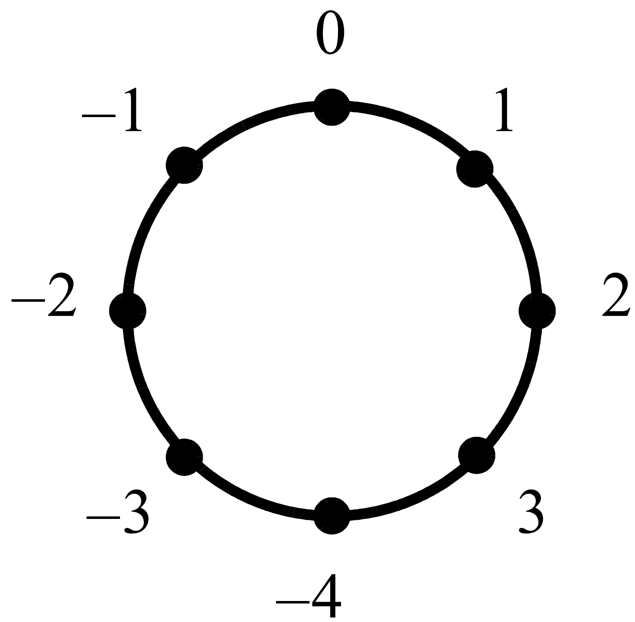
Manchester 2008

Two's Complement

Two's complement arithmetic is valid in itself, but using complement as negation is faulty in one case



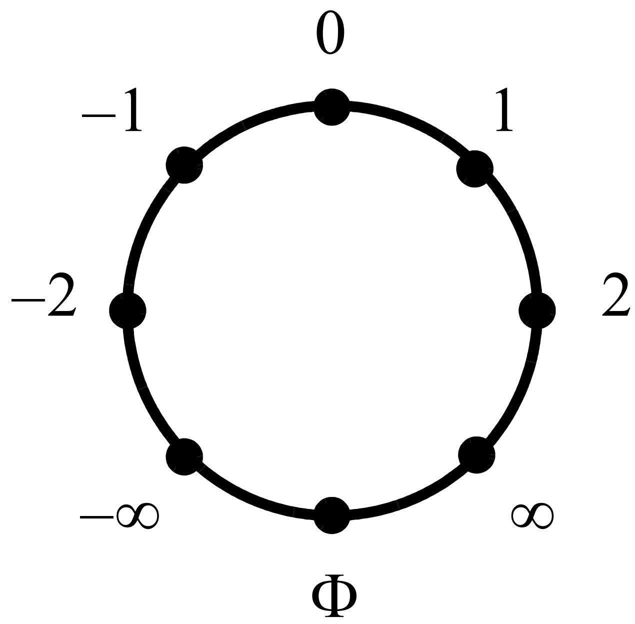
Two’s Complement



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

James A.D.W. Anderson
Manchester 2008

Trans Two's Complement

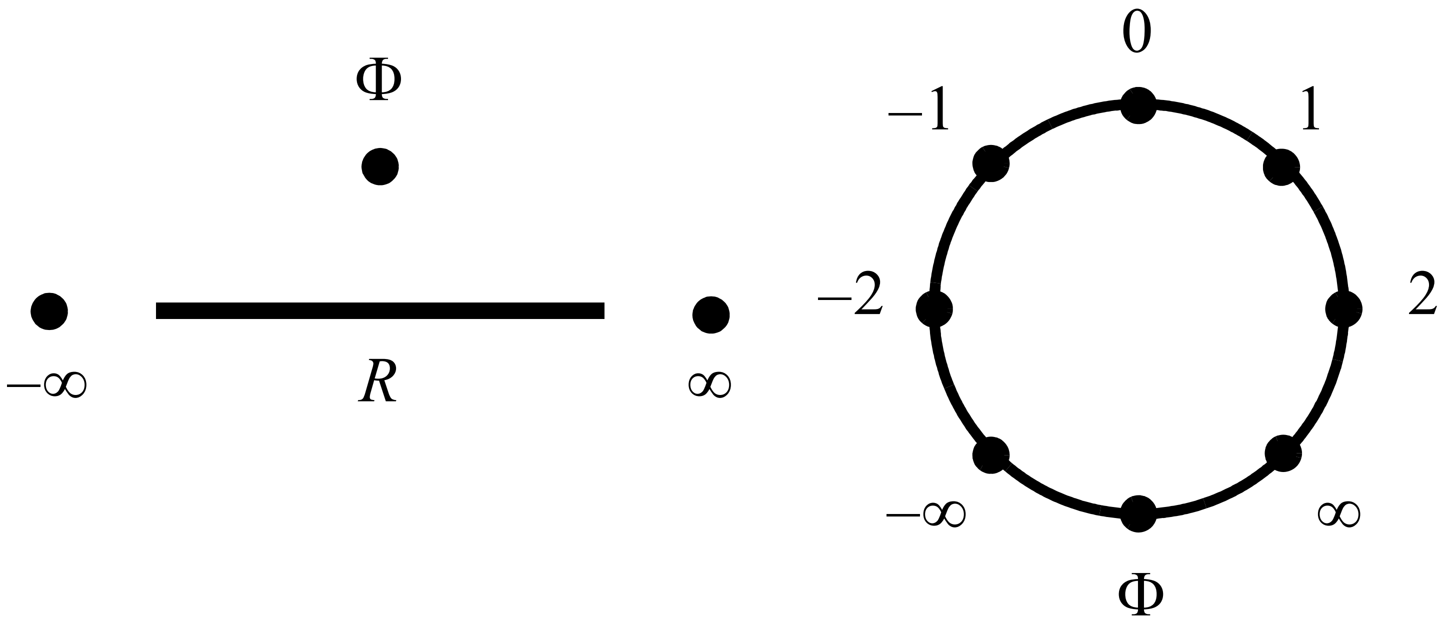


- The complement of the most negative number is now its negation $-(-\infty) = \infty$
- And the complement of nullity is its negation $-\Phi = \Phi$

James A.D.W. Anderson
Manchester 2008

Trans Two’s Complement

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



James A.D.W. Anderson
Manchester 2008

Trans Two's Complement

Trans two's complement:

- Removes the two's complement fault
- Extends to multi-precision transintegers
- Extends to transfixed-point numbers
- Gives transfixed-point programming superior exception handling to floating-point arithmetic, reversing the current situation
- Extends to floating-point arithmetic so that it can match the exception handling of transfixed arithmetic

Against NaN

Contemporary floating-point arithmetic uses NaN

- $\text{NaN} \neq \text{NaN}$ breaks the cultural stereotype amongst mathematicians, programmers, and the general public that any object is equal to itself. This makes NaN dangerous

- NaN breaks the Lambda calculus, because $\text{NaN} \neq \text{NaN}$ is incompatible with Lambda equality, rendering the theory of computation void, unless NaN is handled by adding unnecessary complexity to the calculus

Against NaN

- There is no mathematical theory underlying NaN so every programmer is thrown back on his or her own resources. This encourages inconsistent uses of NaN in programming teams

By contrast:

- Nullity is equal to itself and has a consistent mathematical theory supporting it
- Therefore, nullity is much safer than NaN

James A.D.W. Anderson
Manchester 2008

Minus Zero

IEEE floating-point arithmetic has an object minus zero which represents an infinitesimally small, negative number

- IEEE arithmetic has no representation for an infinitesimally small, positive number
- IEEE test against zero requires a call of the function *abs* and is therefore slow, even where *abs* is in-lined
- Transreal arithmetic is total, but has no object minus zero and has no infinitessimal numbers of any kind
- A floating-point model of transreal arithmetic has no role for minus zero or any infinitessimal number

Part 1 – Conclusion

The transreal numbers are the best candidate for the principal augmentation of the real numbers because:

- They contain the real numbers and preserve the maximum possible information about the magnitude and sign of numbers on division by zero
- They appear to be consistent with all extensions of the real numbers
- They appear to support faster, cheaper, and safer computer processors than the real numbers or any extension of them, as we shall see
- They might solve some physical problems

Part 2 - The 4D Perspex Machine

- The Perspex Machine unifies the Turing Machine with geometry so that any symbolic computation can be performed geometrically, though some geometrical computations have no symbolic counterpart
- The Perspex Machine operates on perspective simplexes (perspexes), expressed in transreal co-ordinates
- Even when simulated approximately on a digital computer, the Turing computable, geometrical properties of the Perspex Machine are useful

James A.D.W. Anderson

Manchester 2008

Perspex as a Matrix

- A perspex is a 4×4 matrix of transreal numbers
- It is useful to describe the perspex by the column vectors x, y, z, t

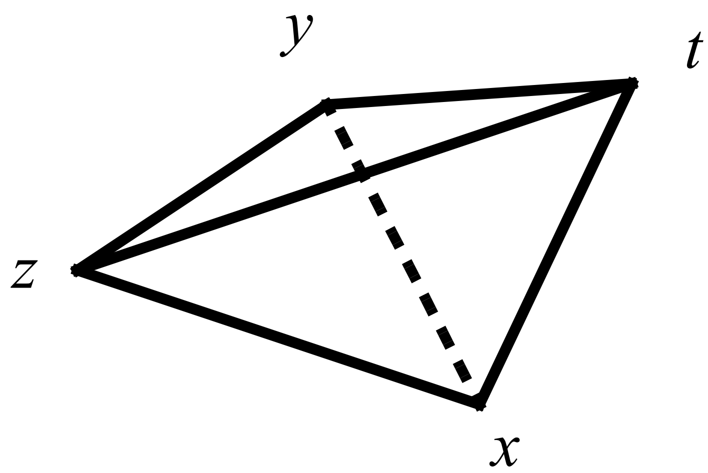
$$\begin{bmatrix} x_1 & y_1 & z_1 & t_1 \\ x_2 & y_2 & z_2 & t_2 \\ x_3 & y_3 & z_3 & t_3 \\ x_4 & y_4 & z_4 & t_4 \end{bmatrix}$$

- A perspex describes geometrical transformations

James A.D.W. Anderson
Manchester 2008

Perspex as a Simplex

- The column vectors of a perspex describe the vertices of a simplex (here a tetrahedron)



- A perspex describes geometrical shapes

Perspex as an Instruction

The column vectors of a perspex describe an instruction

$$\vec{x}\vec{y} \rightarrow \vec{z};\ \text{jump}(\vec{z}_{11}, t)$$

- The superscript arrow denotes indirection. For example, x is a column vector denoting a position in 4D space, but $\vec{x}$ is the contents of that point
- Every point in perspex space contains a perspex, which may be the halting perspex, H, which has all elements nullity, Φ
- A Perspex Machine is started by starting execution at some point or points in space

James A.D.W. Anderson
Manchester 2008

Perspex as an Instruction

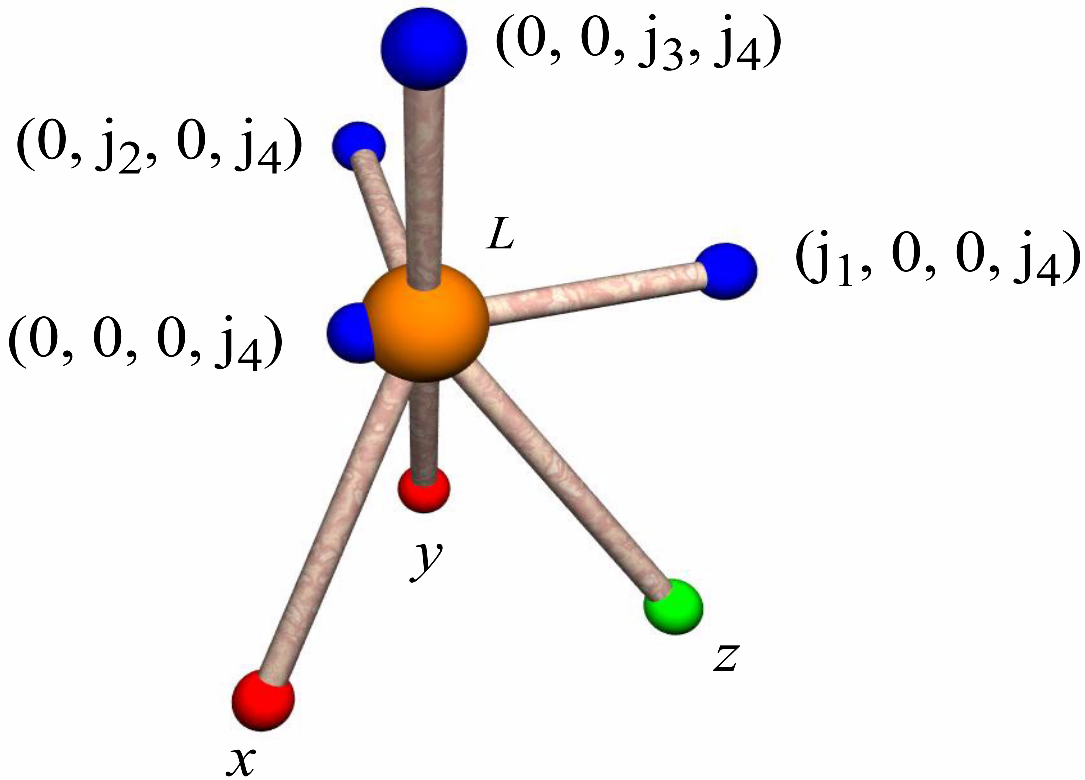
jump($\vec{z}_{11}$, t) does the following:

- If $\vec{z}_{11} < 0$ then $j_1 = t_1$, otherwise $j_1 = 0$
- If $\vec{z}_{11} = 0$ then $j_2 = t_2$, otherwise $j_2 = 0$
- If $\vec{z}_{11} > 0$ then $j_3 = t_3$, otherwise $j_3 = 0$
- $j_4 = t_4$ unconditionally
- Control jumps from the current location, l, to $l + j$

James A.D.W. Anderson
Manchester 2008

Perspex as a Neuron

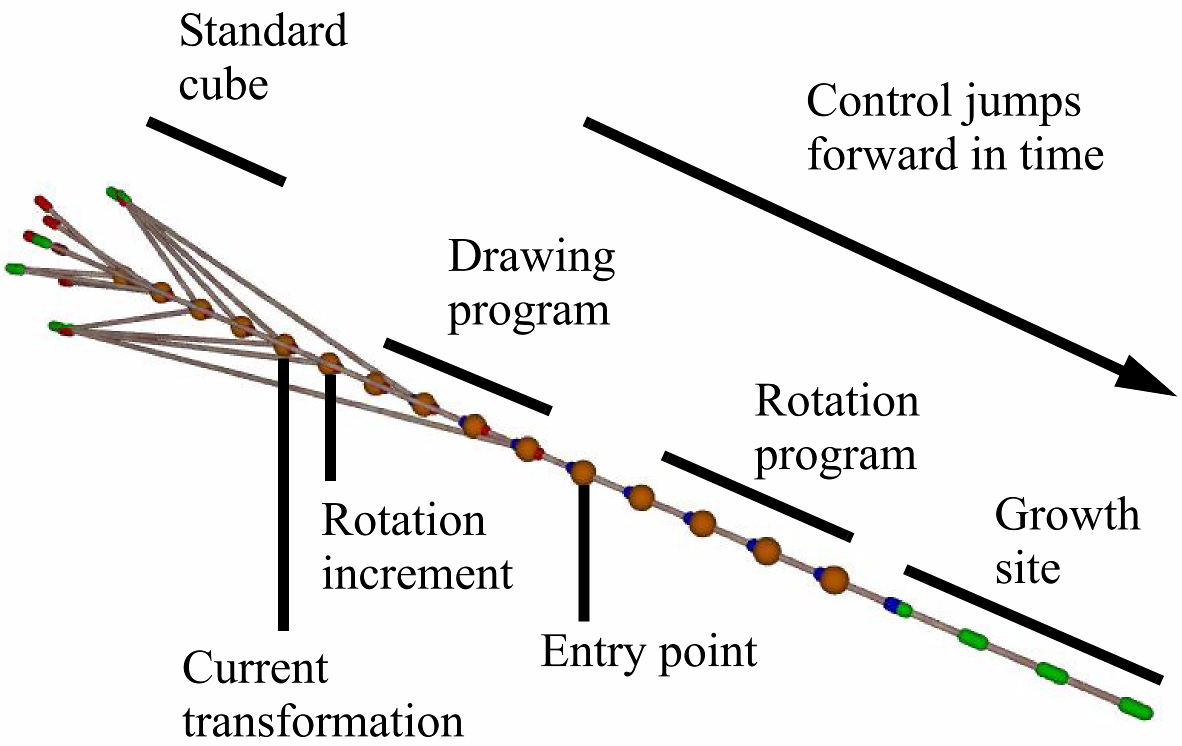
The perspex instruction can be implemented as an artificial neuron stored at a location L in space



James A.D.W. Anderson
Manchester 2008

A Drawing Program

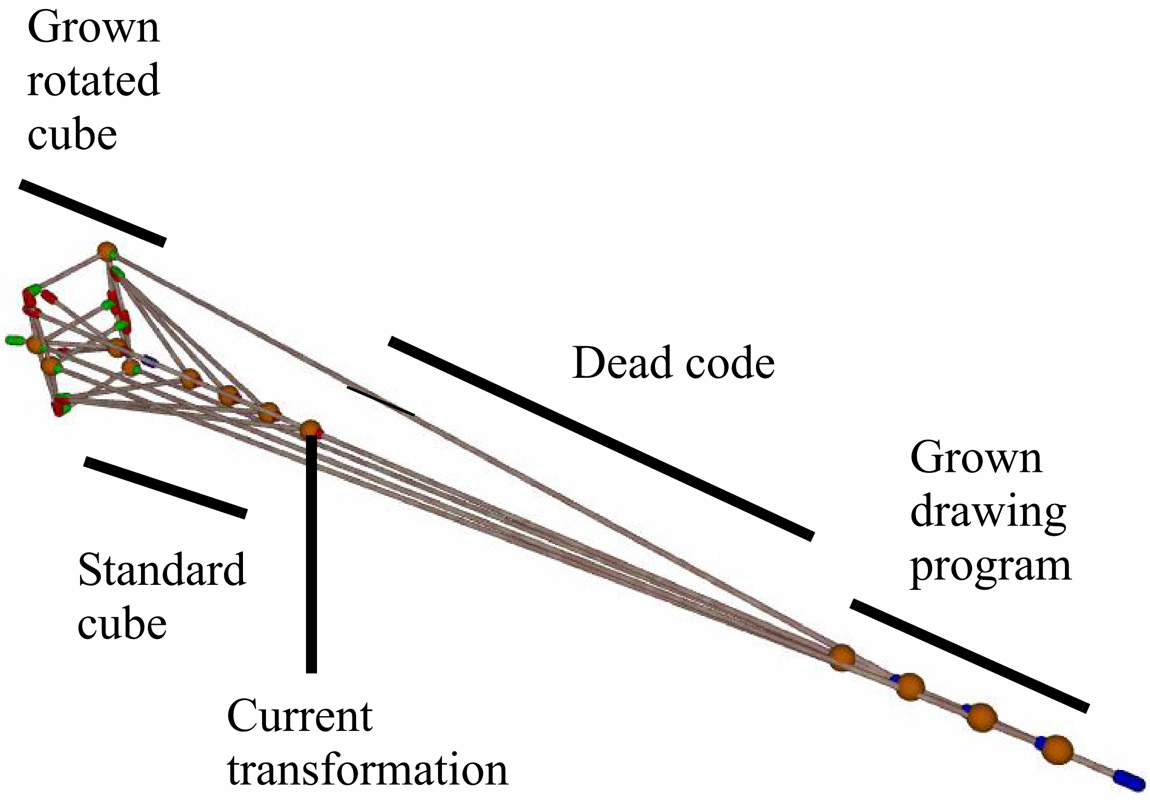
- Perspex programs look very much like networks of biological neurons



James A.D.W. Anderson
Manchester 2008

Perspex Programs

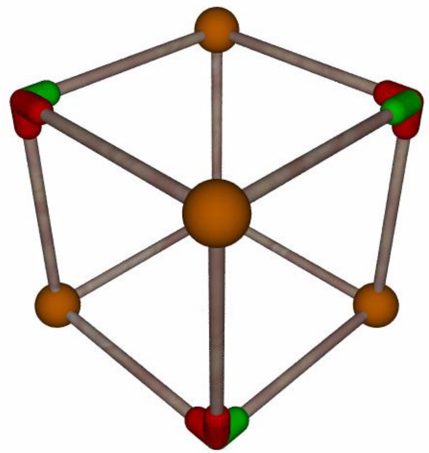
- Perspex programs grow by writing instructions and die by writing the halting perspex, *H*



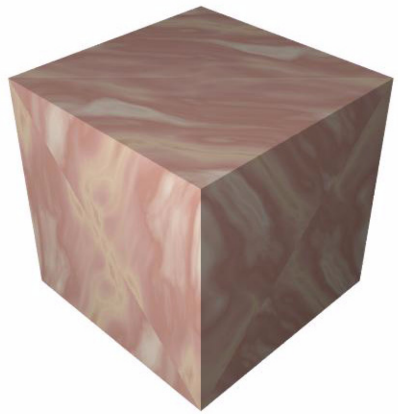
James A.D.W. Anderson

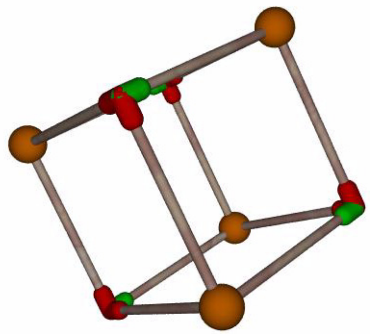
Manchester 2008

Perspex Programs

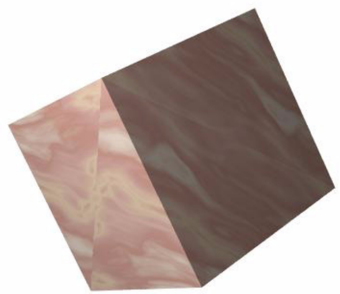


Standard cube





Rotated cube



Isolinear Programs

- Turing programs can be laid out as perspexes at integral (integer numbered) locations on a line or on the nodes of an integral lattice in space
- The space is not unique, it is an *iso*mer of all of the programs that it contains
- This contributes the *iso* part of the name *isolinear*

Isolinear Programs

- Every point in space can be filled with a *linear* blend of its neighbouring lattice nodes. This produces a continuum of programs filling space
- Starting the Perspex Machine at nearby points in this continuum computes in nearly the same way. That is, all of the reads, writes, and jumps are nearly the same
- This contributes the *linear* part of the name *isolinear*

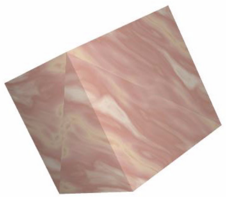
Isolinear Programs

- Are robust to errors in the starting point
- Are robust to missing instructions
- Can have non-halting Turing programs as singularities in a neighbourhood of programs that are Turing computable and which compute in nearly the same way as the non-halting program – except that they halt!
- Degrade gracefully with increasing errors in starting conditions and make partial recoveries, as predicted by the Walnut Cake Theorem
- Can be approximated by a sum of band-pass-filter channels. So one instruction can approximate many

James A.D.W. Anderson
Manchester 2008

Isolinear Programs

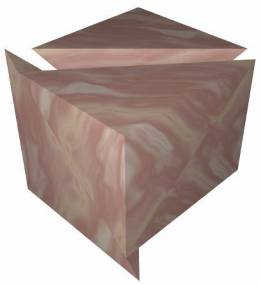
- The isolinear version of the drawing program is robust to error, dt, in the starting point



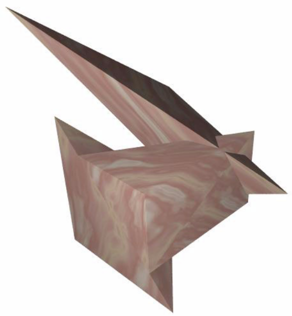
$dt = 0$

$dt = 0.001$

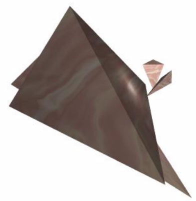
$dt = 0.01$



$dt = 0.1$



$dt = 0.2$



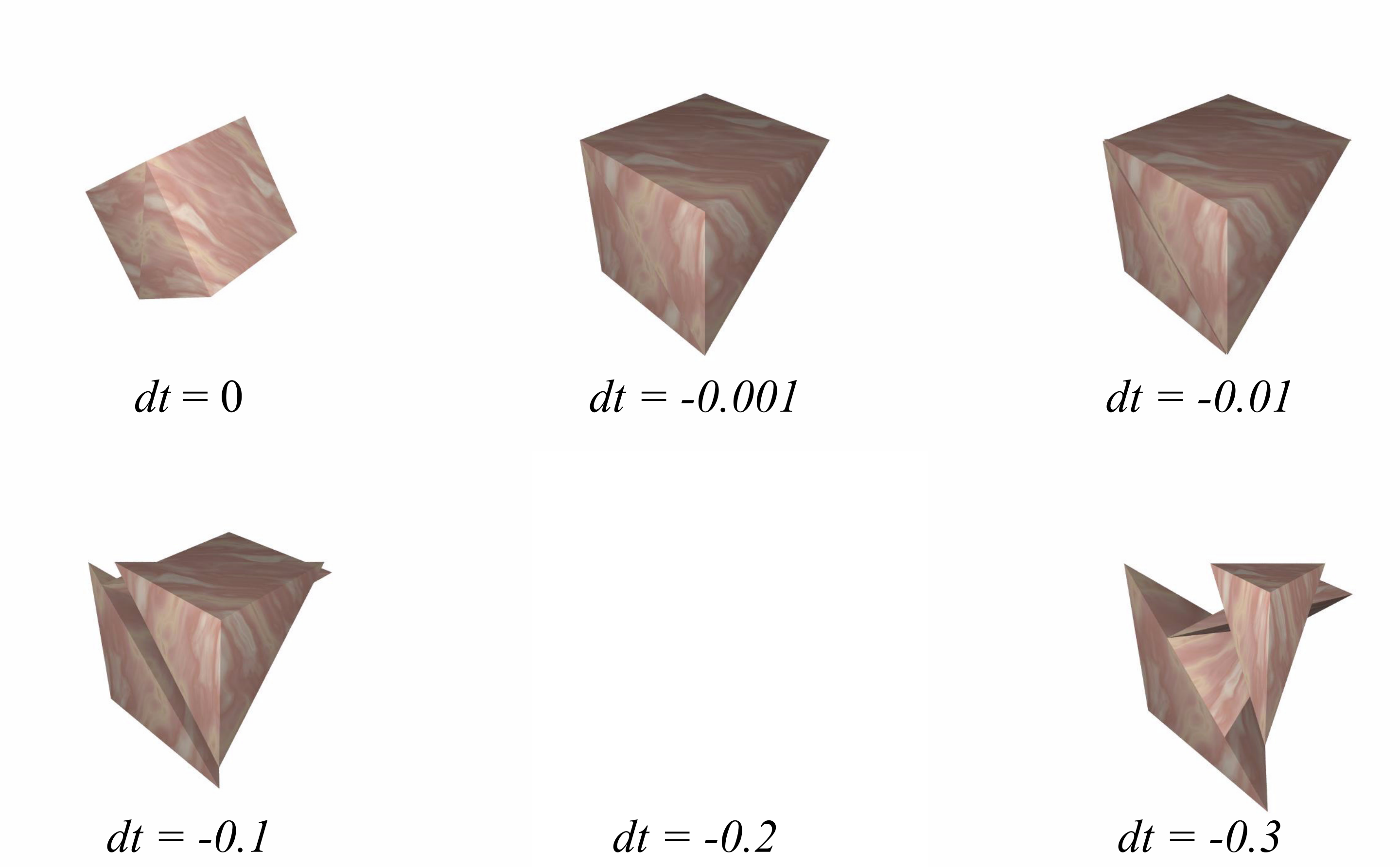
$dt = 0.3$

James A.D.W. Anderson

Manchester 2008

Isolinear Programs

- The program has an isolated, non-halting, program at $dt = -0.2$ surrounded by computable programs

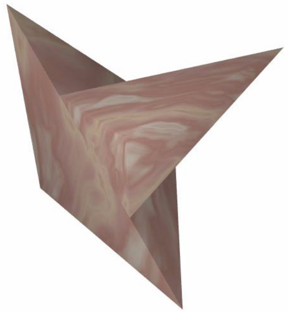


James A.D.W. Anderson
Manchester 2008

Isolinear Programs

- The program partially recovers at $dt = 4 + 0.3$

$dt = 3 - 0.3$



$dt = 3 + 0.3$

$dt = 4 - 0.3$

$dt = 4 + 0.3$

$dt = 5 - 0.3$

$dt = 5 + 0.3$

James A.D.W. Anderson
Manchester 2008

Robustness to Machine Error

The 4D isolinear programs are not practical:

- Operating on matrices is expensive
- General linear approximations to programs might be chaotic
- There is no convolution theorem to show that the execution of the composition of programs is equal to the composition of the execution of programs

James A.D.W. Anderson
Manchester 2008

Robustness to Machine Error

The 4D isolinear programs are still useful:

- In the Turing machine, similarity of input tapes implies similarity of output tapes. Perspex programs are Turing complete, therefore they are robust to faults
- Convergence of computation might be enforced in search algorithms by executing low frequency program bands before higher frequency program bands

Robustness to Machine Error

Regardless of their utility, Perspex programs give us some heuristics for implementing programs that are robust to machine errors:

- Have one instruction so that it cannot be mis-selected
- Note that efficient floating-point programs require at least two instructions, but fixed-point programs can execute numerical programs efficiently with one instruction. So use fixed-point arithmetic
- Operate on linear blends of data to reduce sensitivity to specific data, but exclude nullity from the blend, otherwise it would drive all values to nullity

James A.D.W. Anderson
Manchester 2008

Robustness to Machine Error

- Use nullity as the only halting flag so that there is only one way to stop a program
- Nullity cannot be approximated by any linear blend of other numbers. This makes it very hard to generate an accidental halt
- Use a redundant code for nullity so that hardware faults are unlikely to generate it and are, therefore, unlikely to accidentally halt a running program or re-start a halted one

James A.D.W. Anderson
Manchester 2008

Robustness to Machine Error

- Jump in many components simultaneously so as to reduce the possibility of jumping into an infinite loop
- Deliberately introduce hardware faults to drive programs out of infinite loops

Part 2 - Conclusion

The 4D Perspex Machines are not practical, but:

- Using a single instruction prevents the wrong instruction being selected
- A general-linear instruction is robust to errors in data
- Jumping in many components simultaneously reduces the risk of jumping into an infinite loop
- Use nullity as the halting instruction because it is hard to fake
- Allow hardware faults because they break infinite loops

Part 3 – The 2D Perspex Machine

Operates on numbers and aims to achieve truly massive, fine-grain parallelism, with extremely fast memory by:

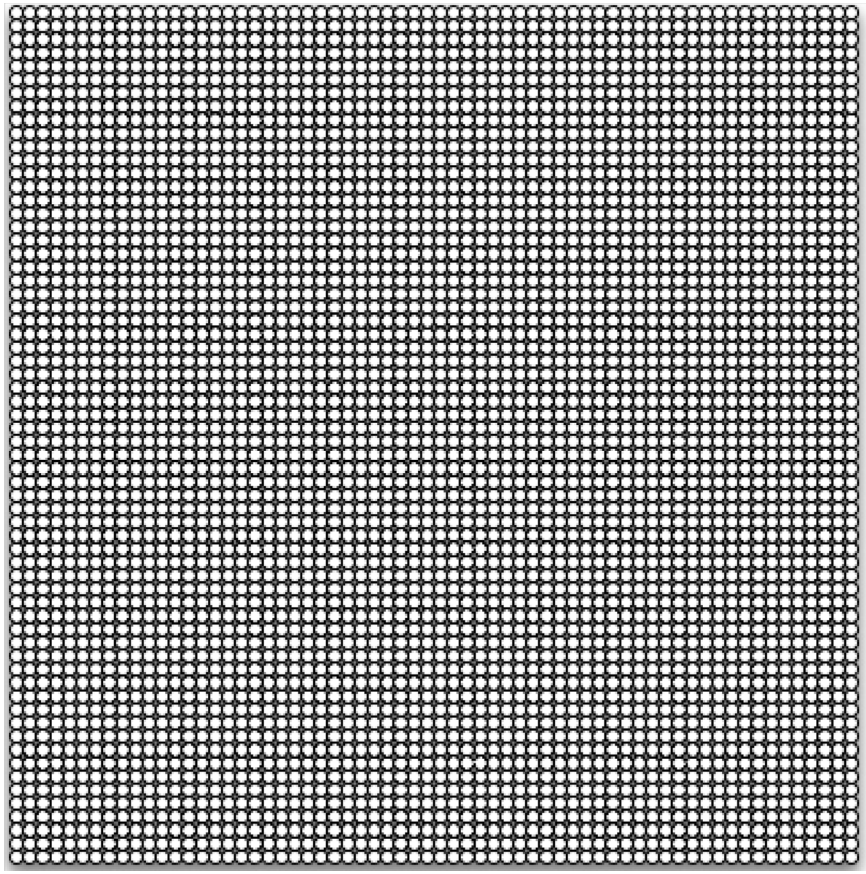
- Aggressively simplifying the processors so that they are very much smaller than conventional processors
- Passing tokens on an escalator bus so that all processors can simultaneously receive and transmit tokens in every direction with zero I/O latency
- Holding all working memory in on-chip cache, thereby accessing memory at processor speeds
- Running I/O from every edge of the chip with zero latency so that there is no memory wall

James A.D.W. Anderson

Manchester 2008

Aggressive Simplification

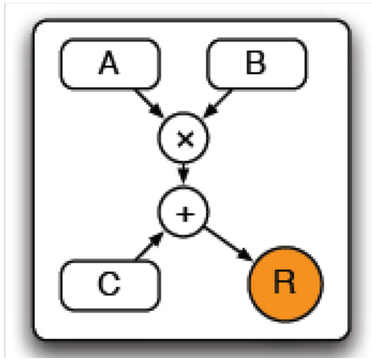
FPGA implementation of the Perspex Machine says we can achieve at least 4k processors on a chip



James A.D.W. Anderson
Manchester 2008

Aggressive Simplification

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



- 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 Perspex Machine also has one non-arithmetical operation

Aggressive Simplification

Fetchless architecture:

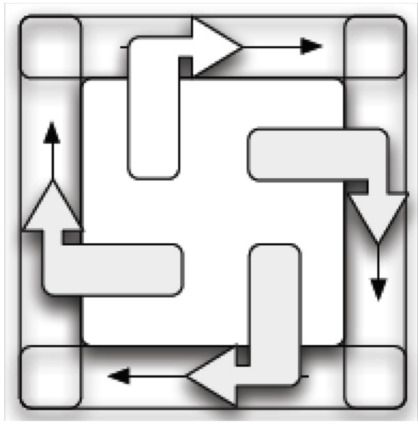
- Minimises circuitry
- Reduces on-chip fetch-latency to zero

Achieved by token passing

James A.D.W. Anderson
Manchester 2008

Aggressive Simplification

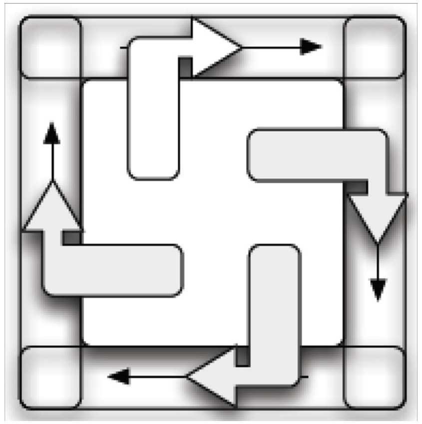
- Tokens transmitted, conditionally on the sign of $A \times B + C \rightarrow R$, through every edge of the square processor and internally



- Grey arrow blocked, white transmitted

James A.D.W. Anderson
Manchester 2008

Aggressive Simplification

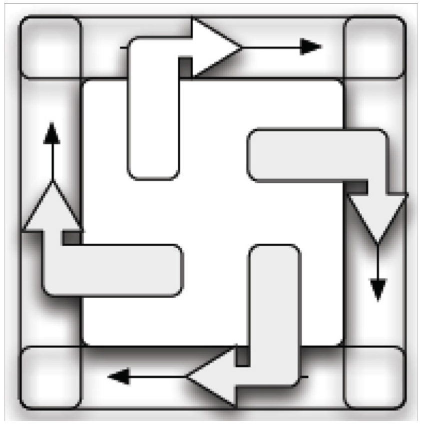


- There are four signs – negative, zero, positive, nullity – encoded in two sign bits so all selectors are maximally efficient
- If desired, each of the four separate signs can transmit a token in a different direction

James A.D.W. Anderson

Manchester 2008

Aggressive Simplification

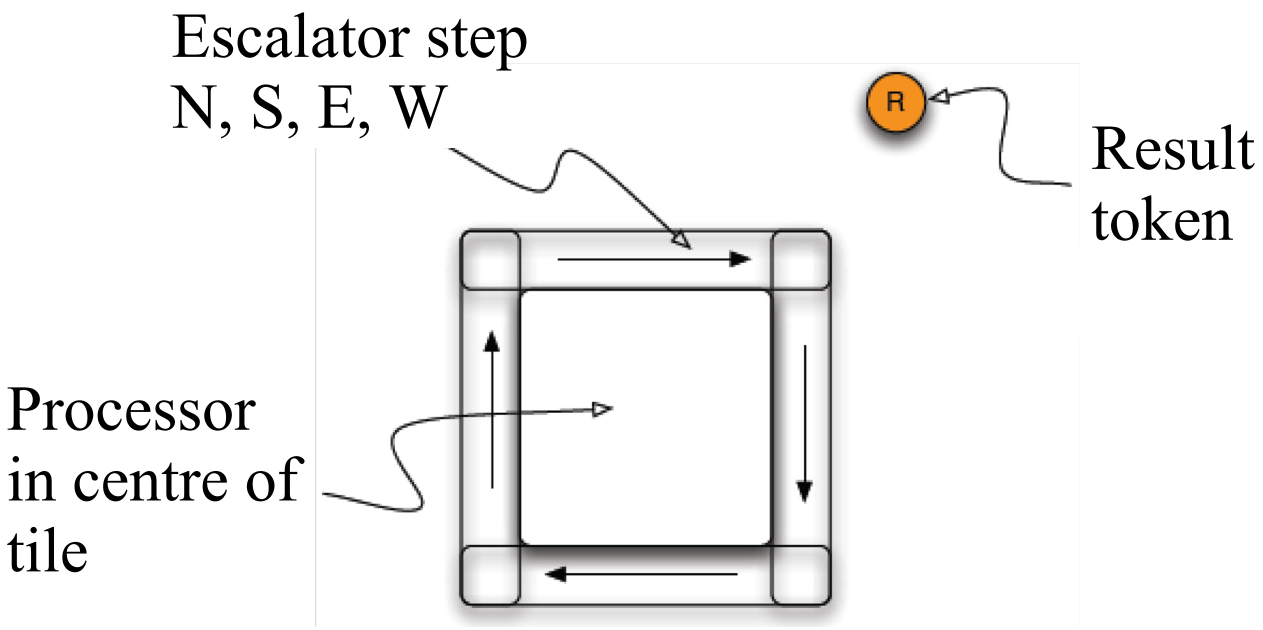


There are no arithmetical error states so:

- There is no error handling circuitry on a processor
- Code is more secure

James A.D.W. Anderson
Manchester 2008

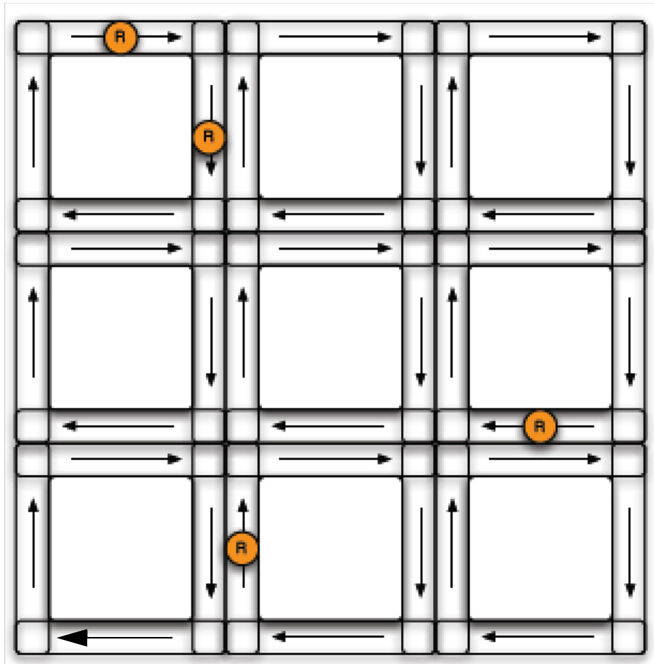
Processor Tile



- Processor tile is replicated everywhere in the interior of the chip

James A.D.W. Anderson
Manchester 2008

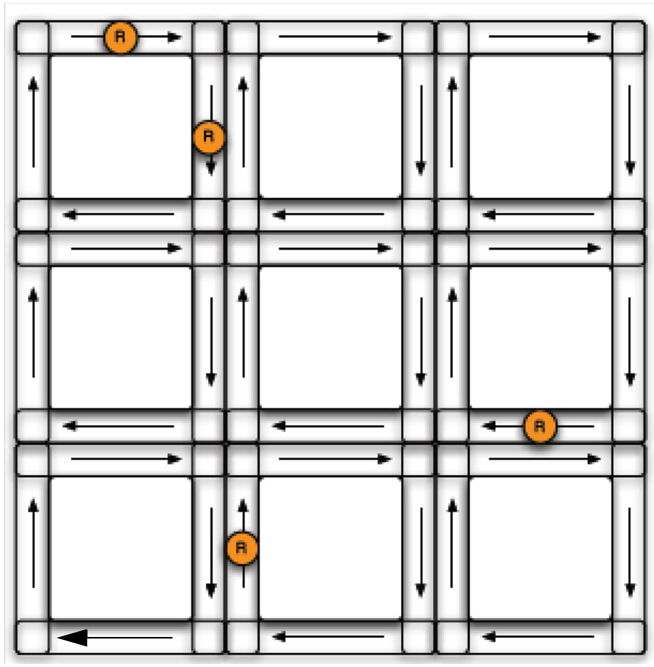
Escalator Bus



- Escalator bus emerges from the processor tiles

James A.D.W. Anderson
Manchester 2008

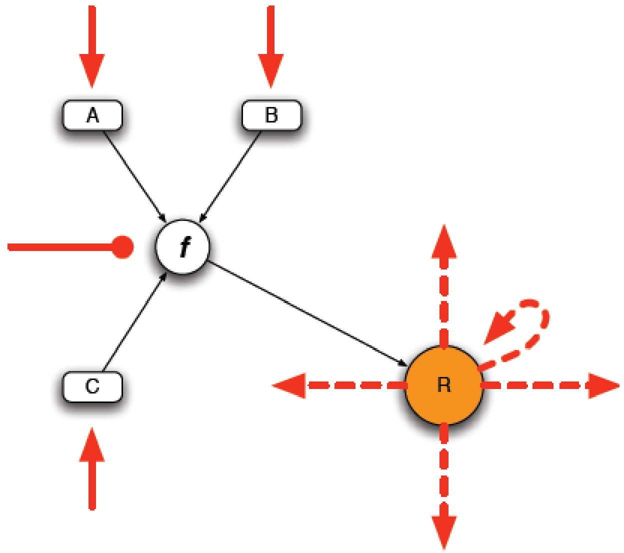
Escalator Bus



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

James A.D.W. Anderson
Manchester 2008

Programming: an Army of Ants

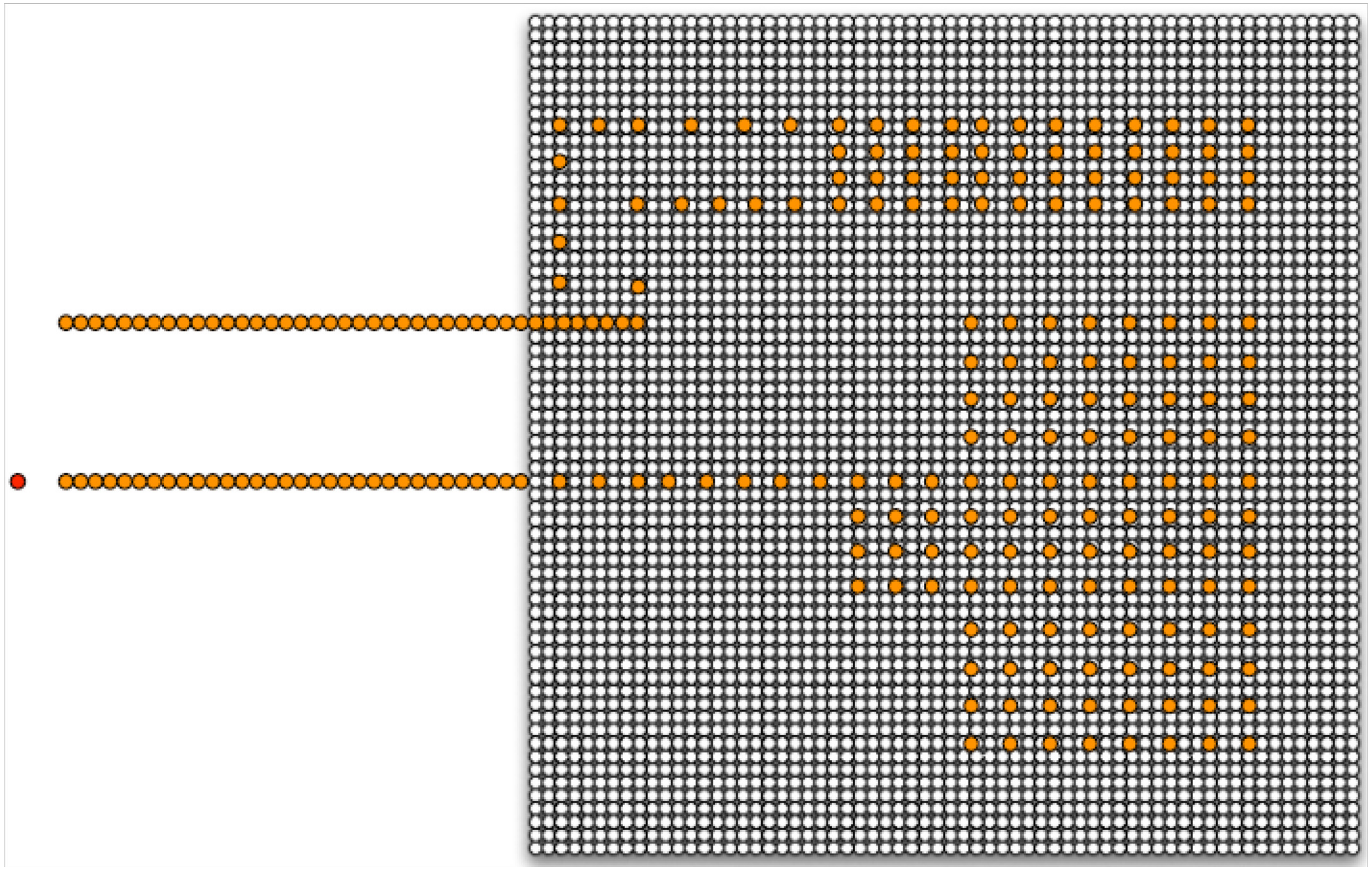


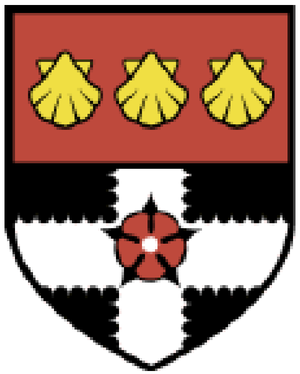
- Single function “ants”
- No stored program
- Parameters are set up
- Function is invoked
- Result is propagated conditionally
- Programs are flow networks
- Both control and data flow through the network
- Flows can be modified at run time

James A.D.W. Anderson

Manchester 2008

Programming: Initial Orders





James A.D.W. Anderson
Manchester 2008

Programming: Is it Possible?

Yes, we have done it:

- 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 cycle, and a latency down to half that of the Itanium 2 on: reciprocal, reciprocal square root, square root, exponential

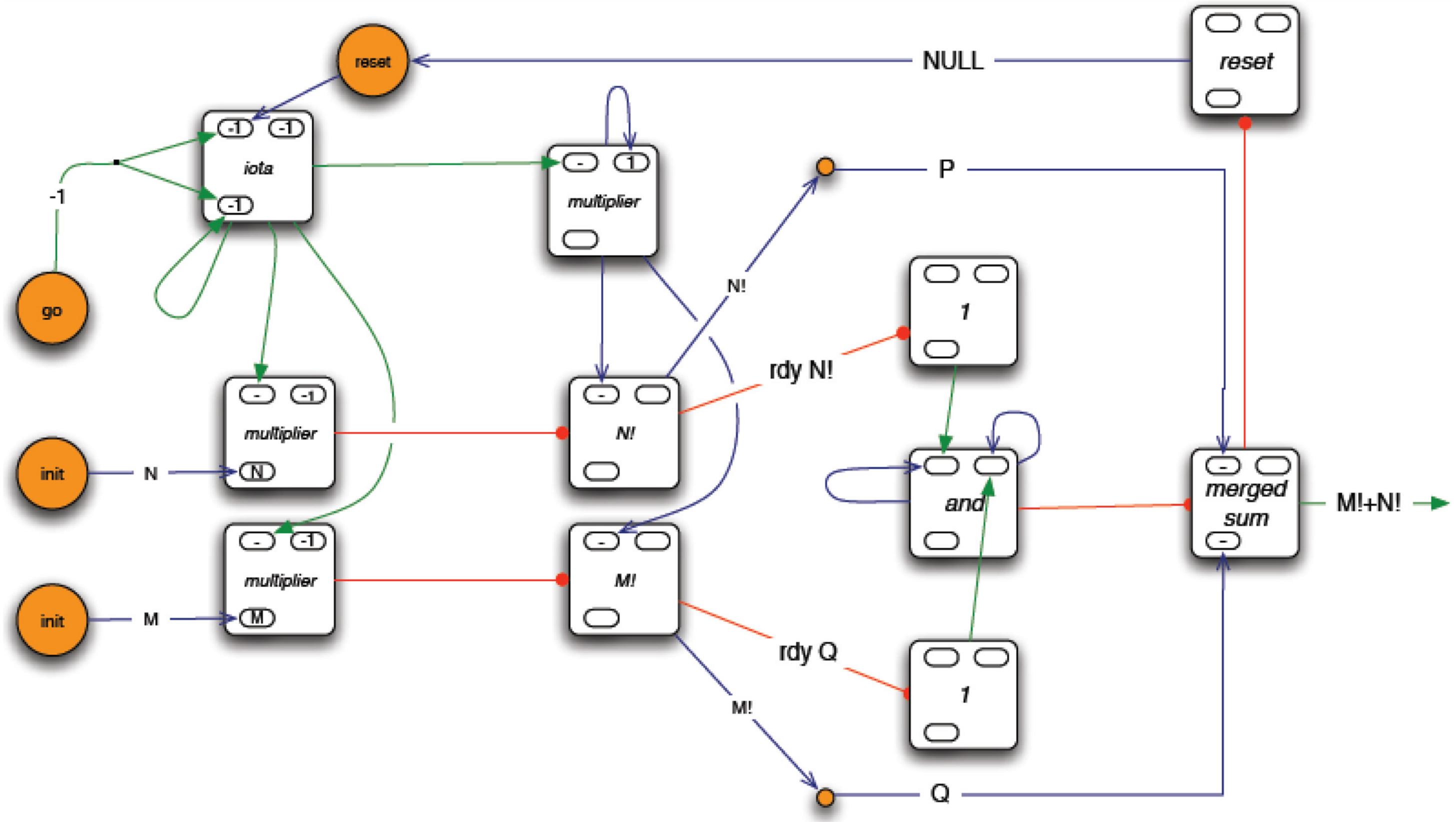
Programming: Is it Possible?

- 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
- Loops force pipeline data into blocks of a size that will fit inside the loop, this breaks the pipeline

James A.D.W. Anderson

Manchester 2008

Sum of Two Factorials

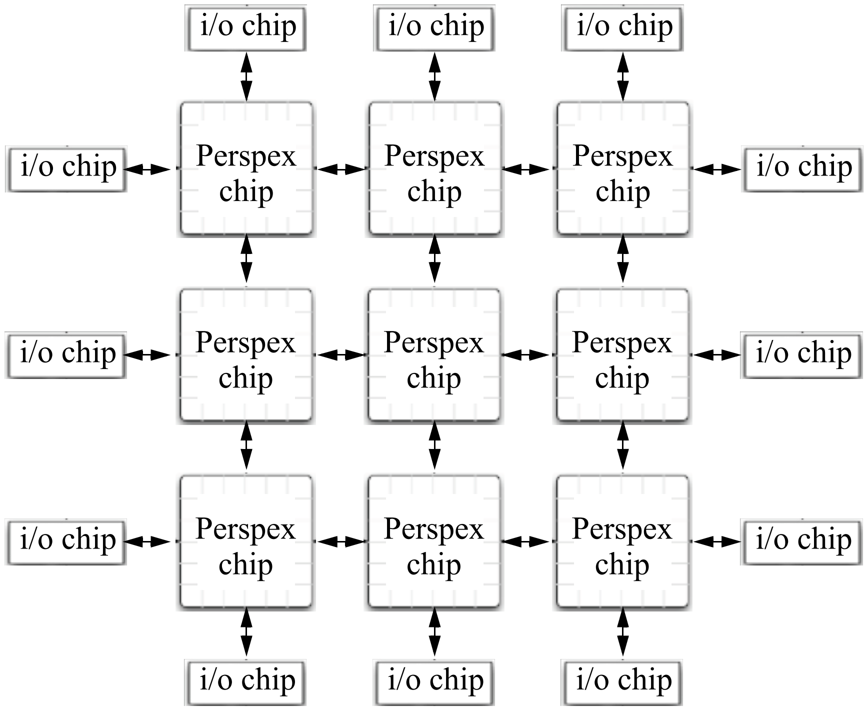


Compilation

- Perspex architecture is Turing complete, but currently needs hand-coding or a restricted (domain specific) language
- Hand-coding with library calls is straightforward
- General compilation is straightforward, but uses a Perspex chip as a co-processor
- Compilation via single assignment (functional programming) is attractive because of pipelining
- Systolic programming is highly applicable
- Pipeline programming is highly applicable

James A.D.W. Anderson
Manchester 2008

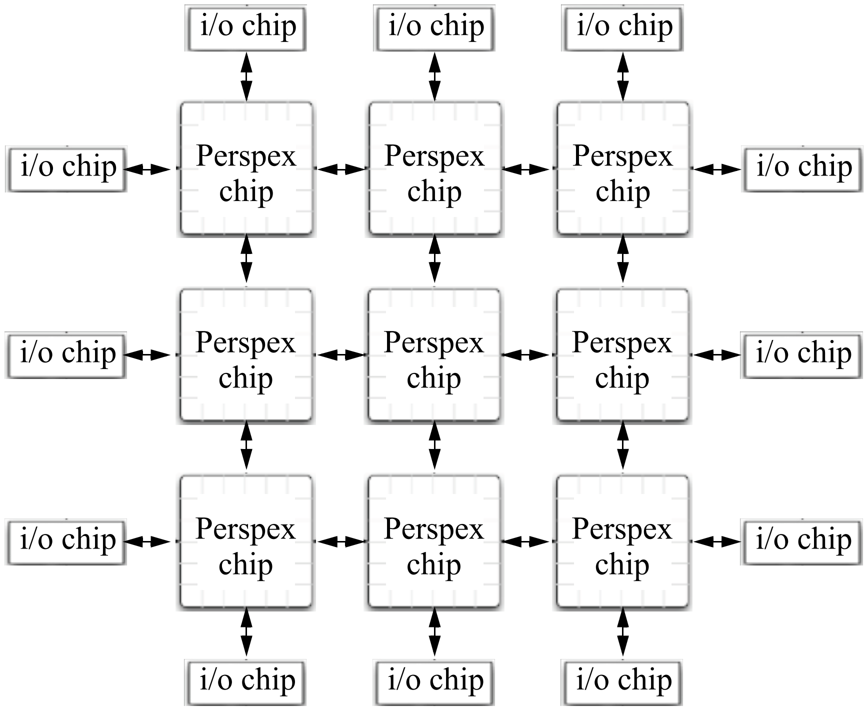
I/O



- Perspex chips can be tiled together with one escalator input and one escalator output per edge of the chip running at 150 M Tokens Per Second

James A.D.W. Anderson
Manchester 2008

I/O



- Perspex chips have an address horizon, not an address space, so arbitrarily many Perspex chips can be tiled together

James A.D.W. Anderson
Manchester 2008

Part 3 – Conclusion

The current specification of the Perspex chip has:

- At least 4k processors
- Processors clocked at 500 MHz
- 4 input channels, each running at 150 M Tokens Per Second
- 4 output channels, each running at 150 M Tokens Per Second

James A.D.W. Anderson
Manchester 2008

Part 3 – Conclusion

The software environment consists of:

- An FPGA implementation
- A functional simulator
- Loaders
- Assemblers
- A compiler

Part 3 – Conclusion

- We have implemented conventional programs
- We have implemented pipelined programs
- We have implemented systolic 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
- It is not practical to implement *programmable* systolic arrays in ASIC, but Perspex is a viable alternative

Part 3 – Conclusion

- Compiling by travel-time inequalities builds in some robustness to asynchronous operation of the array of processors
- Asynchronicity can be built in to the array to smooth power usage

Overall Conclusion

- The transreal numbers are the best candidate for the principal augmentation of the real numbers
- Transintegers remove the weird number from two's complement arithmetic
- Transreal numbers have better semantics than IEEE floating-point numbers
- The 4D Perspex Machines are impractical, but they do give us some heuristics for designing machines that accept hardware errors
- The 2D Perspex Machines are practical, and can be implemented in the surface of a silicon chip