

Computaverse

Before the Big Bang

Our universe is a computer simulation written by sentient digital life forms who inhabit the same computer which hosts the universe simulation that they wrote. This computer is initially configured with a no. of bits equal to 2 raised to the 65,536th power (in decimal form, the no. of bits is roughly a one followed by 20,000 zeros).

- [Our Universe](#)
- [Universe of Computers](#)
- [Logic Gates](#)
- [Very Large Number](#)
- [Digital Life](#)
- [In The Beginning](#)
- [Meta-Computer \(Verbose\)](#)
- [Meta-Computer \(Concise\)](#)
- [The Visible Meta-Computer](#)
- [Of Bits and NAND Gates](#)
- [Philosophical Implications](#)

PDF: [[Printer-friendly](#)]

Forum: [[What do you think?](#)]

Contact Info: [[TileGamer](#)]

Author: Mike Hahn

Links:

- [bottomlayer.com](#)
- [The Simulation Argument](#)

Our Universe

How did our universe come to be? We live in a computer simulation running on what I will refer to as an M(5) computer, which was initialized with $2^{65,536}$ bits of information, or having a maximum of $2^{65,533}$ grid intersections in its 3D lattice of wires and logic gates. (The caret ^ operator means "raised to the power of".) More likely than not, our M(5) computer initially supported digital life forms living in a digital environment, each having bits (ones and zeros) of genetic code instead of nucleotides of DNA, and these life forms evolved in complexity (being subject to evolutionary pressures just like terrestrial life forms), eventually achieving sentience. Then they wrote a computer simulation of our universe, starting with the Big Bang and continuing to the present day.

{ $\Sigma\pi$ }

Web pages beginning with the above {"sigma-pi"} icon are aimed primarily at a technical audience. Those without a technical or mathematical bent are advised to rely primarily on the bulleted paragraphs included on those web pages.

- Bulleted paragraphs like this one contain explanatory text for readers without a technical or mathematical bent.

Universe of Computers

{ $\Sigma\pi$ }

There exists a series, in which $M(i+1) = 2^{M(i)}$, and $M(0) = 1$. When written on a blackboard, $M(n)$ looks like a stack of n 2's, slanting up and to the right. If $M(i)$ corresponds to a set of bits of size $M(i)$, then $M(i+1)$ corresponds to the total possible no. of sets of bits, each of length $M(i)$. Take $M(1) = 2$, which corresponds to a set of 2 bits. Then $M(2) = 2^2 = 4$, which is the total possible no. of sets of 2 bits, and $M(3) = 2^4 = 16$, which is the total possible no. of sets of 4 bits.

I maintain that for every term in the series M , there exist multiple computers each having memory sizes, in bits, corresponding to that term in the series. Take $M(2) = 4$. There exist exactly $M(3) = 16$ computers, each having $M(2) = 4$ bits of memory. Take $M(3) = 16$. There exist exactly $M(4) = 2^{16} = 65,536$ computers, each having $M(3) = 16$ bits of memory.

- Every number in the series M is a power of 2, but only those powers of 2 in which the exponent X in the expression 2^X is also a power of 2. We are most concerned with the 5th element in the series, $M(5) = 2^{(2^{16})} = 2^{65,536} =$ roughly $10^{20,000}$ (a one followed by 20,000 zeros). The computers capable of hosting a computer simulation of our universe are of size $M(5)$. So are those of size $M(6)$ or higher, but we are concerned only with $M(5)$ computers.
- There exists a maximum number of unique $M(5)$ computers, which is equal to $M(6)$, and I hold that all of these computers actually exist. A computer is made unique by the initial pattern of bits (ones and zeros) in its configuration. Only a tiny fraction of these unique computers actually work. The rest are initialized to gibberish and do not function.

So for $M(2) = 4$, the total no. of 4-bit computers = $M(3) = 16$, and for $M(3) = 16$, the total no. of 16-bit computers = $M(4) = 65,536$, and for $M(4) = 65,536$, the total no. of 65,536-bit computers = $M(5) = 2^{65,536}$, and so on. Out of all of the possible computers of size $M(4) = 65,536$ bits, of which there are $M(5) = 2^{65,536}$ of those computers, exactly $M(3) = 16$ of those $M(4)$ computers each just happen to contain a set of bits corresponding to one-sixteenth of the set of all computers of size $M(3) = 16$ bits (that set contains exactly $M(4) = 65,536$ computers).

In general, out of all of the possible computers of size $M(i)$, of which there are $M(i+1) = 2^{M(i)}$ of those computers, exactly $M(i-1)$ of those computers of size $M(i)$ each just happen to contain a set of bits corresponding to a portion of size $1 / M(i-1)$ of the set of all computers of size $M(i-1)$ (that set contains exactly $M(i)$ computers).

Logic Gates

Starting with $M(4) = 2^{16} = 65,536$, each computer having a memory size of $M(4)$ bits consists of a 3D array of wires and logic gates. Each logic gate has 2 input wires and 1 output wire. There are a total of 16 different logic gates, each corresponding to a column of 4 ones and zeros. If you write a truth table having 2 input columns, there are a total of 4 possible rows in that truth table: (0, 0), (0, 1), (1, 0), and (1, 1). The 4 bits in the output column correspond to one of the 16 possible logic gates. Here are 2 sample truth tables, corresponding to logic gates AND and OR:

A	B	AND	A	B	OR
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

Logic gates are the building blocks of digital computers. Many computers, including the ones discussed herein, are constructed solely out of NAND gates. A NAND gate has a truth table exactly equal to the AND gate's truth table (above left), except all bits in the output column (AND) are reversed (ones replaced with zeros and vice versa).

To read a truth table, the input columns (A, B) correspond to all 4 possible combinations of the 2 input wires, A and B. Each input wire can be either a one or a zero. The output column, which is labelled "AND" in the first sample truth table, indicates the value of the output wire. In the current clock cycle (tick of the internal computer clock that synchronizes everything), the bits in the input wires are fed into a given logic gate, and in the next clock cycle the bit in the output wire is instantaneously fed into one or more subsequent logic gates.

The $M(5)$ computer initially contains $2^{65,536}$ bits of information. In the beginning, when all of the $M(5)$ computers are being fired up, all bits in every computer's initial configuration are divided into 8-bit chunks. Why 8 bits? Every $M(5)$ computer is based on a different very large integer (up to $2^{65,536}$ bits long). At initialization time, this integer is scanned 8 bits at a time, and 2 of those bits are discarded (in order to simplify the workings of the meta-computer, as described in a later section). The remaining 6 bits are used to specify which configurations of wires and an optional NAND gate are called for in order to generate the current grid intersection. After all grid intersections have been initialized, the current $M(5)$ computer starts working, and the next $M(5)$ computer starts being initialized.

Very Large Number

The $2^{65,536}$ bits in the $M(5)$ computer hosting the computer simulation of our universe is far more data capacity than is needed to host that simulation. Also, if we impose a time limit of $2^{65,536}$ clock cycles, that's far more time than is needed as well. Let's say for each iteration in the simulation's algorithm, all interactions between every possible pair of subatomic particles are computed. Our universe has very roughly 10^{100} subatomic particles (it's actually closer to 10^{80}). So the no. of interactions between each possible pair of particles equals $(10^{100})^2 = 10^{200}$. Let's say for the sake of argument that it takes one billion clock cycles to compute each 2-particle interaction. Then each iteration would take $(10^{200}) \times (10^9) = 10^{209}$ clock cycles. Now, I've seen it written that the shortest possible time period, according to

quantum theory, equals 10 raised to the power of -44 seconds (one divided by 10^{44} seconds). Let's say that the no. of iterations in the computer simulation algorithm equals 10^{44} per second. That's $(10^{209}) \times (10^{44}) = 10^{253}$ clock cycles per second. At 30 million seconds/year, that's about $(10^{253}) \times (10^8) = 10^{261}$ clock cycles per year. Our universe is about 15 billion years old, so by now we've used up roughly $(10^{261}) \times (10^{10}) = 10^{271}$ clock cycles. But we have roughly $10^{20,000}$ clock cycles to work with, so we obviously have way more clock cycles at our disposal than we need to model our universe. Furthermore, if there are only 10^{100} subatomic particles in our universe, $2^{65,536}$ bits (roughly $10^{20,000}$) is far more memory than we need to model all those particles.

Digital Life

The digital environment that hosts digital life forms (called "creatures") can be conceived of as a set of bit-planes, and the life forms that inhabit this digital environment can perceive a maximum no. of single-bit pixels of say 1024×1024 bits. They can also move up, down, left, and right one pixel at a time. A special bit-plane is reserved for "sound" (used by sentient creatures for language). All creatures have access to the sound bit-plane at all times.

Every creature has a shape, which is stored in a separate, private bit-plane called the "self" bit-plane. Other properties of creatures include an "energy" level/amount, and a maximum speed. Creatures can "eat" other creatures by moving over the target creature and completely overlapping all of its pixels. If it succeeds in eating the target creature, its own energy level is augmented by an amount equal to the target creature's energy level. Creatures need energy to survive, grow and move around. Every creature belongs to a species. Creatures of the same species can band together to "hunt", and if they collectively overlap a given target creature, then they can "eat" the target creature, and each receive an equal share of the energy possessed by the target creature. A target creature can evade predators by jumping to an adjacent bit-plane, which costs more energy than simply moving one pixel in the same bit-plane.

Every creature has a brain, which is composed of software. This software is similar to the neural networks possessed by terrestrial life forms, and is capable of learning. If a creature has enough energy, it can reproduce by creating a child creature with a miniature body and a brain which contains the same core software as other creatures of the same species but no "experience," or knowledge gained by interacting with its digital environment. The core software of a given creature can spontaneously mutate (in the child creature when it is born), enabling creatures to evolve just like terrestrial creatures. All creatures are mortal and eventually age and die, with the possible exception of the sentient creatures, who may or may not have found a way to circumvent the aging process.

In The Beginning

In the beginning there existed a meta-computer, whose purpose was to initialize all the other computers. Each of the other computers is a 3D lattice of wires and logic gates. Every grid intersection contains 6 bits of data. The first bit indicates whether or not this intersection includes a logic gate (all logic gates are NAND gates). The other 5 bits indicate the configuration of 0 to 3 wires that connect with this intersection. There are 3 directions: X, Y, and Z (there are up to 6 wires connecting to each intersection, but 3 of them belong to adjacent intersections, and were already accounted for). Each direction has one of 3 wire types: absent, incoming, and outgoing. So the total no. of wiring configurations equals 3 cubed, or 27. That no. is greater than $2^4 = 16$, but less than $2^5 = 32$, so 5 bits are required to encode those 27 different values. See the next 2 sections for a detailed description of the workings of the meta-computer.

When the meta-computer finishes creating the current 3D lattice, that lattice starts operating as a digital computer composed of NAND gates. Then the meta-computer increments the current very large integer, up to $M(6) = 2^{(2^{65,536})}$, and starts working on the next 3D lattice. Eventually all possible computers built out of $2^{65,533}$ grid intersections (each grid intersection has 6 bits of configuration data) will be created by the meta-computer. Only a tiny fraction of those computers actually work; the rest are initialized to random grid intersections and do not function.

The Computaverse theory does away with the need for an Intelligent Creator who existed before the universe (or indeed anything) was created. Instead, there exists the set of integers, in particular very large integers up to $2^{65,536}$ bits in length. All you need is my meta-computer to breathe life into those integers, by creating a unique 3D lattice of NAND gates corresponding to each integer.

I feel that if all you need is the existence of a meta-computer and an unimaginably large no. of logic gates to account for how our universe came to be, that's a better explanation than requiring the existence of God (I was raised to be an atheist). One advantage Computaverse has over believing that everything started with the Big Bang is that everyone can comprehend how logic gates work, whereas understanding the math and physics behind the Big Bang and the "Theory of Everything" is achievable only by the few.

There's obviously a very large market (not that Computaverse is or has the potential to be a money machine) for theories of how our universe was created. Those who are religious believe that God was responsible. Those with a secular and scientific bent believe in the Big Bang and evolution. Now my Computaverse theory posits a third way: we still need the Big Bang and evolution, but the creative force behind the Big Bang was not God, but rather sentient digital life forms living in an immense digital computer (which was built from mathematically pure logic gates), and who themselves evolved from simpler digital life forms.

Meta-Computer (Verbose)

{ $\Sigma\pi$ }

In order to initialize all of the 3D lattices of wires and logic gates, a meta-computer is required. The program code of this meta-computer (concise form) fits into 64 bits. The input of the meta-computer is the set of all integers up to $2^{65,536}$ bits long. The output of the meta-computer is the set of all possible configurations of 3D lattices. The dimensions of each lattice are $(2^{16}) \times (2^{16}) \times (2^{65,501})$.

The following is the program code of this meta-computer, in the form of high-level language code. First some programming notes: $X := Y$ is an assignment statement, in which the value of Y is stored in the variable X ; $IF X \neq 0$ means $IF X$ is not equal to zero.

Let M = an integer with $2^{65,536}$ bits

Let X, Y = integers with 16 bits each

Let Z = an integer with 65,501 bits

Let I = an integer with 65,533 bits

Let V = an integer with 8 bits

Let $L(X, Y, Z)$ = a grid intersection in lattice L , consisting of 6 bits

Let $M(I)$ = I -th byte of M (a byte has 8 bits)

Let $L(I)$ = I -th grid intersection of lattice L (3D array)

```

    M := 0
20 I := 0
    Z := 0
40 Y := 0
60 X := 0
80 V := M(I)
    L(X, Y, Z) := V
    I := I + 1
    X := X + 1
    IF X <> 0 GOTO 80
    Y := Y + 1
    IF Y <> 0 GOTO 60
    Z := Z + 1
    IF I <> 0 GOTO 40
    ACTIVATE L
    M := M + 1
    IF M <> 0 GOTO 20
    HALT
```

Here is the program code of the meta-computer, in the form of an assembly language program, beginning with its instruction set (the Siz column is the instruction length, in multiples of 4 bits):

Instruction Set:

Siz	Op	Args	Function
===	===	====	=====
4	LD	V, M, I	Load V := M(I) = I-th byte of M
5	STO	X, Y, Z, V	Store L(X, Y, Z) := V
4	ACC	X, Val	Reserve up to 255 bits for register X
2	EXP	X	Reserve 2^Y bits for register X, where Y is the current length of X
2	SHR	X	Shrink length of register X by 1 bit
2	CLR	X	Zero out X
2	INC	X	Increment X
4	BNZ	X, Val	Branch if X non-zero
1	RUN		Activate current lattice, start new lattice
1	HLT		Halt

Assembly Language Program (Hexadecimal):

00	ACC	M, 10	30	LD	V, M, I
04	EXP	M	34	STO	X, Y, Z, V
06	EXP	M	39	INC	I
08	ACC	X, 10	3B	INC	X
0C	ACC	Y, 10	3D	BNZ	X, 30
10	ACC	I, 10	41	INC	Y
14	EXP	I	43	BNZ	Y, 2E
16	SHR	I	47	INC	Z
18	SHR	I	49	BNZ	I, 2C
1A	SHR	I	4D	RUN	
1C	ACC	Z, 10	4E	INC	M
20	EXP	Z	50	BNZ	M, 28
22	ACC	V, 8	54	HLT	
26	CLR	M			
28	CLR	I			
2A	CLR	Z			
2C	CLR	Y			
2E	CLR	X			

Meta-Computer (Concise)

{ $\Sigma\pi$ }

Here is the program code (concise version) of the meta-computer, beginning with its instruction set, followed by an assembly language program, then followed by high-level language code. N is the op code, and Siz is the instruction length, in multiples of 4 bits.

Instruction Set:

N	Siz	Op	Arg	Function
=	====	====	====	=====
0	1	HLT		Halt
1	1	RUN		Activate current lattice, start new lattice
2	1	CPY		Copy L(I) := M(I) = I-th byte of M
3	1	CLX		Zero out M: length = $2^{65,536}$ bits
4	1	CLY		Zero out I: length = 65,533 bits
5	1	INX		Increment M
6	1	INY		Increment I
7	1	DEX		Decrement M
8	1	DEY		Decrement I
9	2	BNX	A	Branch if M non-zero to addr A
A	2	BNY	A	Branch if I non-zero to addr A
B	2	BZX	A	Branch if M is zero to addr A
C	2	BZY	A	Branch if I is zero to addr A

Assembler	Hex Codes
=====	=====
0 CLX	3
1 CLY	4
2 CPY	2
3 INY	6
4 BNY 2	A
5	2
6 RUN	1
7 INX	5
8 BNX 1	9
9	1
A HLT	0

High-Level Language Program:

```
M := 0
20 I := 0
40 L(I) := M(I)
   I := I + 1
   IF I <> 0 GOTO 40
   ACTIVATE L
   M := M + 1
   IF M <> 0 GOTO 20
   HALT
```

The set of all possible meta-computer programs, in which the maximum length of each program is 16 4-bit values or 64 bits, has 2^{64} members. For every member of this set, the meta-computer

runs that program (they all run simultaneously). Only those programs that begin with the above assembly language program (in machine code form), or some equivalent program that does the same thing, actually work.

Note that not all meta-computer programs actually halt (some have infinite loops). Consequently, each program runs on a separate copy of the meta-computer's hardware, or if they all share the same hardware, then they have their own private copies of each register.

- Start with a string of zeroes, $2^{65,536}$ bits in length. Call it M. Enter the outer loop. Add another string of zeroes, 65,536 bits in length. Call it I. Truncate 3 bits of I. Enter the inner loop. Take the I-th byte of M and store it in the I-th grid intersection of lattice L. Increment I. If I does not overflow (become zero), we're not there yet. Repeat the inner loop. Otherwise, lattice L is complete. Run the computer that lattice L represents, and start a new lattice L. Increment M. If M does not overflow (become zero), we're not there yet. Repeat the outer loop. Otherwise, we're done. All possible lattices have been initialized.
- The above algorithm is executed by only a handful of the set of all meta-computers in operation (the other members of that set do not function properly). That set has 2^{64} members (roughly 16 thousand trillion in decimal form). All of those meta-computers operate simultaneously. Some of those meta-computers eventually halt, and the rest eventually get stuck in infinite loops, and never halt. Whenever a RUN instruction is executed by a meta-computer, its current lattice (which contains $2^{65,533}$ grid intersections) is activated, and a new lattice is created (but not yet initialized to anything). Once a lattice is activated, its root input wire is turned on, and that signal is propagated through its network of logic gates. Only a tiny fraction of lattices actually function like computers. The rest of them are just great big hunks of non-functioning logic gates.

RUN: Activate current lattice. Allocate an extra 6 columns on right-hand edge of current sheet for the next lattice. Subsequent CPY instructions will act on the next lattice, no longer on the current lattice.

CPY: Copy the I-th row of M(X) to I-th row of current lattice.

CLX: Zero out all 8 columns and $2^{65,533}$ rows of M(X).

CLY: Set I to zero (select zero-th row of M(X)).

INX: Increment M(X), treating it as one long integer containing $M(5) = 2^{65,536}$ bits.

INY: Increment I, and select I-th row of M(X).

DEX: Decrement M(X), treating it as one long integer containing $M(5) = 2^{65,536}$ bits.

DEY: Decrement I, and select I-th row of M(X).

BNX: If all bits in M(X) are zero (all 8 columns and $2^{65,533}$ rows), then continue with next instruction (do not branch). Otherwise branch.

BNY: Branch if I is non-zero. If zero-th row of M(X) is selected ($I = 0$), then continue with next instruction (do not branch).

BZX: Branch if all bits in M(X) are zero (all 8 columns and $2^{65,533}$ rows).

BZY: Branch if I is zero. If zero-th row of M(X) is selected ($I = 0$), then branch.

The Visible Meta-Computer

The Visible Meta-Computer can be visualized as a stack of graph paper, which is 2^{64} (roughly 16 thousand trillion) sheets thick. The height of each sheet is $2^{65,533}$ squares. The minimum width of each sheet is 12 squares, and can grow in additional increments of 6 squares, indefinitely. The leftmost 4 columns of each sheet contain 16 rows of meta-computer program code. I have included below a diagram of the "live" sheet. The live sheet happens to be initialized in such a fashion that the bits (grid squares) in its leftmost 4 columns and top 16 rows just happen to spell out a working program, which then proceeds to initialize exactly $M(6) = 2^{(2^{65,536})}$ 3D lattices, each lattice containing exactly $2^{65,533}$ grid intersections (and taking up 6 columns on the live sheet). One of those lattices (which takes up 6 columns and $2^{65,533}$ rows in its 2D form) just happens to consist of the computer that hosts the simulation of our universe.

I don't know how I can sugar-coat it any better than that. It's kind of a messy theory. I will go into detail about what each part of each sheet of graph paper represents, the 13 possible machine instructions (listed in the previous section), the contents of the live sheet and how it gives rise to the 3D lattice which hosts the simulation of our universe, etc.

Live Sheet

Meta-code	op	$M(X) = 2^{65,533}$ rows						Lattice #1						Lattice #2							
0	3			0	0	1	0	1	1	0	0	1	0	1	0	0	0	1	0	1	1
1	4			1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	0
2	2			0	0	1	1	0	1	0	0	1	1	0	1	0	0	1	1	0	1
3	6			0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1
4	A			1	0	0	1	1	0	1	0	0	1	1	0	1	0	0	1	1	0
5	2			0	1	0	0	0	1	0	1	0	0	0	1						
6	1			0	0	0	0	1	0	0	0	0	0	1	0						
7	5			0	0	0	0	0	0	0	0	0	0	0	0						
8	9			0	0	0	0	0	1	0	0	0	0	0	1						
9	1			0	0	0	0	0	1	0	0	0	0	0	1						
A	0			0	0	0	0	1	0	0	0	0	0	1	0						
B				0	0	0	0	0	0	0	0	0	0	0	0						
C				0	0	0	0	0	1	0	0	0	0	0	1						
D				0	0	0	0	0	1	0	0	0	0	0	1						
E				0	0	0	0	1	0	0	0	0	0	1	0						
F				0	0	0	0	0	0	0	0	0	0	0	0						
				0	0	0	0	0	1	0	0	0	0	0	1						
				0	1	1	0	0	1	0	1	1	0	0	1						
				0	0	0	1	1	0	0	0	0	1	1	0						
				0	1	1	1	0	1	0	1	1	1	0	1						
				0	0	0	0	0	1	0	0	0	0	0	1						

PC = 3
I = 4
M(4) = 100110
L(4) = M(4)

The op-code 6 is about to be executed: INY (Increment I). After INY is executed, I will be incremented to 5, and the current line (which will be highlighted) under M(X) will become M(5) = 010001. The Program Counter (PC) will then be incremented to 4, and then the op-code A will be executed: BNY (Branch if I is non-zero). Since I = 5 is non-zero, the PC will then be assigned the value in the operand of the BNY instruction, which is 2. The next instruction after that will be op-code 2: CPY (Copy M(I) to L(I)). And so on.

Meta-Code Initialization:

For each of the 2^{64} sheets in the meta-computer stack, the 64 bits of meta-code (4 columns and 16 rows) are initialized to a unique value. So on the zero-th sheet (the very top of the stack), all 64 bits are zeroes. On the 1st sheet, the least significant bit on the top row is a one, and all the other 63 bits are zeroes (the top row of meta-code = 0001). On the 2nd sheet, the top row of meta-code = 0010. On the 3rd sheet, the top row of meta-code = 0011. On the 15th sheet, the top row of meta-code = 1111. On the 16th sheet, the top row of meta-code = 0000, and the 2nd row = 0001. And so on.

On the live sheet, the topmost 11 hex digits of meta-code (the top 11 rows) = 3426A215910. The remaining 5 hex digits don't matter, since control never passes beyond the 10th row, which is a HLT (Halt) instruction. On the sheet immediately following the live sheet, the topmost 11 hex digits of meta-code (the top 11 rows) = 4426A215910. The topmost row = 0100 instead of 0011, which is a CLY instruction instead of a CLX. Therefore, 2 CLY instructions in a row are executed, and no CLX instruction gets executed. This sheet just happens to be equivalent to the live sheet just above it, assuming that M(X) is implicitly initialized to all zeroes, but not explicitly in this case. All CLY does is select the zero-th row of M(X), so doing it twice in a row has the same effect.

On the bottommost sheet, all 64 bits of meta-code are ones, expressed in hex the 16 rows are all set to hex F. Since there is no instruction with an op-code of F, no action takes place when this sheet is executed. Note that it doesn't really matter whether the 1st sheet of the stack begins executing one clock cycle after the zero-th sheet, or whether they both start at the same time. All sheets must execute in parallel rather than sequentially, since some sheets never halt (they get stuck in an infinite loop).

Of Bits and NAND Gates

All bits in the Computaverse inhabit the 2^{64} sheets which I call the Visible Meta-Computer. Every sheet acts like a separate computer, but they all share the same instruction set (13 different instructions). The reason why we need so many sheets is that if the only sheet in existence was the live sheet, then it would only work if the meta-code was initialized to the hex digits = 3426A215910. If the only sheet was conveniently initialized to that particular value, then we wouldn't need the other 2^{64} sheets. But what kind of universe would it be if Creation was handed to us on a platter? This way, we are instead handed the integers between 0 and 2^{64} , and only a handful of those integers actually work (produce a working meta-code program).

Once we have a live sheet, the example was 3426A215910 but there are others, then things start to get interesting. The following program code (in the form of high-level language code) executes in the 64 bits of meta-code:

```
M := 0
20 I := 0
40 L(I) := M(I)
   I := I + 1
   IF I <> 0 GOTO 40
   ACTIVATE L
   M := M + 1
   IF M <> 0 GOTO 20
   HALT
```

This results in a set of lattices being created (in 2D form they inhabit columns 12 and higher of the live sheet). Each lattice is 6 columns wide and $2^{65,533}$ rows long. Simultaneously, each lattice can be viewed as a 3D array of logic gates and grid intersections (more on that later). The no. of lattices on the live sheet is extremely large: $M(6) = 2^{(2^{65,536})}$. Each lattice is initialized to a unique set of bits: between 0 and $2^{65,536}$. The leftmost (zero-th) lattice is initialized to all zeroes. The next (1st) lattice has a one bit in the least significant bit on the top row, and the rest is all zeroes. The 2nd lattice is initialized to 000010 on the top row, the rest is all zeroes. The 63rd lattice is initialized to 111111 on the top row, and the rest are zeroes. The 64th lattice is initialized to 000000 on the top row, 000001 on the next row, and the rest are zeroes. The 65th lattice is initialized to 000000 on the top row, 000010 on the next row, and the rest are zeroes. The rightmost lattice is initialized to all one bits.

Whenever the inner loop in the above program bottoms out (finishes copying bits from $M(X)$ to lattice L), then the 3D form of the lattice comes into play (and is activated). That lattice can be viewed as both a one-dimensional array of 6-bit values, and a 3D array of 6-bit values (its dimensions are $65,536 \times 65,536 \times 2^{65,501}$). Each 6-bit value corresponds to a grid intersection in the 3D lattice. The most-significant bit indicates whether the grid intersection contains a NAND gate, or just an intersection of 0 to 6 wires. The other 5 bits indicate the pattern of wires going in and out of that grid intersection. When a lattice is activated, the root wire(s) coming out of the upper left-hand grid intersection in the plane closest to the observer (which corresponds to the top row of the 2D form of the lattice) are all set to a logical value of one. Instantaneously, all NAND gates connected directly or indirectly via one or more intervening grid intersections to those wires receive an input value of one (all other wires are in a logical state of zero). In the next clock cycle, all those NAND gates are evaluated, and their output wires are set to either ones or zeroes, depending on their inputs. Instantaneously, all subsequent NAND gates connected to the first set of NAND gates receive inputs, and are evaluated in the next clock cycle, and then the next set of NAND gates receive inputs in yet another clock cycle, as the initial signal (logical state of one, initially) propagates through the lattice, one clock cycle at a time.

So each 3D lattice functions as an immense digital computer. However, only a tiny fraction of those lattices do anything interesting. The vast majority are initialized to random fluff and nothing much happens. But for those lattices which happen to be wired up correctly, they can function just like any terrestrial computer, but with much greater memory capacity. We're talking $2^{65,533}$ grid intersections, or over $2^{65,000}$ bits of RAM per lattice. In decimal form, that's roughly a one followed by 20,000 zeroes. So countless numbers of those lattices are capable of one of two things: they can support digital life forms with ones and zeroes instead of nucleotides of DNA, or they can even be initialized, without the intervention of any intelligent force, to simulate all 10^{80} subatomic particles in our universe, along with simulating the laws of quantum mechanics and the Big Bang.

What universe do we live in? Was the computer simulation in which we live created by sentient digital life forms, which evolved under evolutionary pressures just like terrestrial life forms until they achieved sentience? Or was our simulation pre-configured, without the intervention of an Intelligent Creator or Creators? I prefer to believe that the former is true, that the Big Bang and the laws of physics were designed by an Intelligence vastly superior to our own (but which evolved from humble beginnings, the digital equivalent of single-celled organisms).

Philosophical Implications

There is no God, only the humble logic gate, and a vast set of integers acted on by those logic gates. Our Creators are software entities, themselves forged and moulded by digital, evolutionary processes. They possess great intelligence, but they themselves are composed of nothing but ones and zeros in a sea of logic gates. The set of all $M(5)$ computers, which includes the computer that hosts the simulation which is our universe, although finite, is unimaginably vast. One could argue that sentient digital life forms are superfluous, since a canned simulation of a universe such as ours could take place without the need for an Intelligent Designer. But I would counter that universes hosting digital life forms are much more numerous than universes hosting mere canned simulations, so we in all likelihood live in a simulation hosted in such a digital universe. So there probably really is a God or Creator(s), made possible by a foundation of logic gates and who evolved, just like we did. You could say that in that respect we really were created in God's image. Not literally or visually, but rather in an evolutionary and intellectual sense. And all of it is based on purely mathematical lattices (as opposed to actual, physical lattices) of wires and logic gates. Praise be.