SSRG International Journal of Computer Science and Engineering
ISSN: 2348–8387 / https://doi.org/10.14445/23488387/IJCSE-V10I11P107

Volume 10 Issue 11, 49-62, November 2023

Original Article

Developing a New Software Library for Super Calculations on Infinity Numbers while Providing Infinite Persistent Precision in the Technical Context of P versus NP whereas Programming on Artificial Intelligence

Yassine Larbaoui

Department of Electrical Engineering, University Hassan 1er, Settat, Morocco.

[1]*Corresponding Author : yassine.larbaoui.uh1@gmail.com*

Received: 21 September 2023 Revised: 29 October 2023 Accepted: 12 November 2023 Published: 30 November 2023

Abstract *- This paper presents a new computation software library, which we developed for super arithmetic calculations and massive binary operations on infinity numbers that require more than 64 bits to be expressed by manifesting the possibility of expressing these numbers on infinite quantities of virtual bites reassembled in subgroups while providing infinite persistent precision. Then, we integrated this library into an application, which we named Super Infinity Calculator. We reinforced this software library with an Artificial Intelligence entity, which we programmed to manage the used hardware resources of processors and data storing memories during the executed calculations to handle infinity numbers while executing calculations on them in a short time. This Artificial Intelligence entity is incharge of forwarding computation on infinite subgroups of virtual bites in parallel and recursively when necessary while monitoring their results. The programmed functionalities of this Artificial Intelligence entity play a principal role in supporting the execution of computation calculations on super numbers with lengths exceeding Gigabytes and Terabytes while providing infinite persistent precision for each digit of them, including the ones after the floating point. As a result, these developed software resources allow us to shift various super calculations on infinity numbers from NP to P by executing them in the linear dimension of time instead of consuming exponential time.*

Keywords *- Artificial Intelligence, Infinite persistent precision, Parallel computation, P Versus NP, Super computational calculations, Super infinity number.*

1. Introduction

Throughout history, human beings have always been seeking improvements that may allow them to simplify activities or have certain gains at levels that may be of interest to their existence or their ways of life, such as time, space, and ownership. Then, the computation age came, starting by manufacturing the first computer machine [1] in the forties of 19^{th} century, which led to revolutionizing all aspects of humane life and science.

It all started with the conception of a model of a machine, which was a theoretical computation model developed by Turing in 1936 [2]. He based his model on how he perceived mathematical thinking. As digital computers were developed in the forties and fifties of the nineteenth century, the Turing machine proved itself to be the convenient theoretical model for computation at that time.

Quickly, though, it was discovered that the basic Turing machine model fails to account for the needed amount of time or memory by a computer, which was a critical issue over the years. The key idea to measure time and space as a function of the input length came in 1960 by Hartmanis and Stearns, which gave birth to computation complexity [3].

After decades of development, there were still many published articles with content inspired by the Turing machine. As an example, in 2010, Ya.D. Sergeyev and A. Garro presented a refinement of the theory of computation based on the model of the Turing machine in terms of computational calculation [4].

Computer machines have massively evolved in shape and capacities over more than 80 years of technological evolution, starting from the forties of 19^{th} century until our nowadays.

These evolvements have been always leaned on improvements of relevant resources of hardware and software, which lead computers to support the conduction of enormous mathematical calculations and binary bitwise operations in parallel and at gigantic rates of execution for each process.

Even though computer processing capacities have reached outstanding pics [5], conducting computation calculations on infinity numbers with lengths exceeding Gigabytes and Terabytes is still problematic because numbers are based on defined lengths in the computational concept of processing. For example, in the technical aspect of computation using programming languages of C++, Java and Python, numbers are structured according to specific objects with defined maximum lengths, such as Integers, Doubles and BigIntegers. In contrast, massive numbers with infinite lengths are expressed by approximation with precision limitations and needed time for processing. However, there are improvements in computational calculations where the lengths of resulted numbers may depend only on the size of RAM memory (Random Access Memory).

Theoretically, BigIntegers are featured to support massive numbers limited by the size of RAM. However, a BigInteger has official limits of ($Max = 2^{(Integer.Max_{Value})} - 1; Min = -2^{(Integer.Max_{Value})}$) while being physically limited by the RAM size of Java Virtual Machine (JVM).

There is a methodology allowing the execution of numerical computations with finite, infinite, and infinitesimal numbers [6, 7, 8] on a type of computer named "the Infinity Computer" [9]. This methodology allows writing down these numbers by using a finite amount of symbols. This methodology [7,8] evolves the ideas of Cantor and Levi-Civita in a more applied way. It introduces infinite integers that possess both cardinal and ordinal properties as usual finite numbers, which are also used nowadays in programming languages and software frameworks to handle high numbers with finite values.

This methodology is based on a concept of number approximation named "Arithmetic of Infinity", introduced by Y.D. Sergeyev aimed to devise a new coherent computation environment able to handle finite, infinite, and infinitesimal quantities and execute arithmetic operations with them. This concept is based on a positional numeral system with an infinite radix called "grossone", which represents the number of elements of the set of natural numbers $\mathbb{N}$ [11, 12].

This methodology is limited by precision lengths for digits displayed even on Infinity Computer because instead of displaying infinity numbers, they display only approximated expressions them presented on finite lengths of digits and symbols. Therefore, this methodology is a shortcut enabling the display of massive numbers by expressing them on limited lengths of bytes. This is due to sacrificing the precision of preceding digits with inferior densities to express digits with superior densities [7, 8, 10].

Integer objects are expressed on 32 bites, supporting 2^{32} possible values of different integer numbers, whereas Double objects are expressed on 64 bites, supporting 2^{64} possible values of different double numbers. Then, the BigInteger object was developed to manifest the possibility of expressing higher values limited by RAM size, whereas approximation methodologies were developed to handle massive numbers by presenting them on limited lengths of bites [13, 14].

Therefore, expressing infinity numbers of integers and doubles on infinite lengths of bytes exceeding Gigabytes and Terabytes and executing calculations on them was still problematic for us before developing our software. This problem was principally due to the dependencies on the supported lengths of bites by computers for the execution of calculations, the dependencies on the RAM size, and the need to have Infinity Computers handle massive numbers in short times while providing high levels of approximation precision.

There has been a lot of research on how to support computational calculations on numbers that may need to be expressed on more than 64 bites [15, 16], such as by using grosson [17, 18], which led to significant improvements in the supported lengths of numbers during these computational calculations and during digital display of results. However, these improvements did not manifest the mathematical fancy of supporting calculations on infinity numbers that may be expressed on Gigabytes and even on Terabytes with infinite persistent precision and in less time while using ordinary computers.

Furthermore, many of these improvements are based on favoring digits of high density during calculations and numbers displaying to users while neglecting digits with lower densities starting from specific limits [19] to handle processing on further numbers with higher lengths. Therefore, these improvements may not give an absolute precision for each digit of a resulting number during these calculations and during digital display if the length of this number exceeds defined limits.

Compared to ordinary computers, Super Computers [20] and Infinity Computers [21] are enabling to handle massive numbers of infinity by either processing them entirely or presenting them in the form of high-level approximations in terms of used amounts of digits for their expressions and their visual display. In addition, they execute necessary processing operations on these numbers in extremely short times. Nevertheless, they may still need to sacrifice the precision of low-density digits when numbers exceed the limits of their supported lengths of bites in order to process further quantities of bites and display them to users.

Even though supporting computational calculation on numbers with high lengths on account of precision enables approximated results, this precision is actually crucial in many fields that rely on mathematical calculations, such as Numbers Theory [2], Quantum Mechanic [23], etc. For example, this precision is crucial to identifying infinity prime numbers that may be expressed on high lengths of bits or factorizing numbers consisting of millions of digits in the function of prime numbers, whereas this mathematical precision is a vital key to calculating paths of nanoparticles while being moving in vast spaces such as the case in Atlas experiment [24].

In this paper, we present our developed software library to support calculations on infinity numbers that need more than 64 bits to be expressed by providing the possibility of expressing these numbers on infinite amounts of virtual bites reassembled in subgroups, which enables to express them on Gigabytes and even on Terabytes while using ordinary computers. This software library is based on two extendable objects in terms of length, which we named respectfully "SuperInfinityInterger" and "SuperInfinityDouble".

When executing computational calculations on Infinity numbers, allocated resource capacities of memory and processors definitely play a major role in the success of executed processes. Therefore, we have developed an Artificial Intelligence entity to manage these resources, conduct calculations in parallel and forward the results of calculations recursively from one subgroup of virtual bites to another.

These developed resources of software libraries and Artificial Intelligence entities enable structuring numbers on infinite lengths of bits, which means that they can structure numbers with lengths of Gigabytes and Terabytes by relying on the memory space of computers to store them. In addition, these resources handle the computational processing of calculations in parallel to speed up their execution. Furthermore, they can structure the resulting numbers at the end of calculations on thousands of Gigabytes. Therefore, the only limitation that may be encountered by these developed resources is the available memory space in used computers.

In order to overcome the memory space limitation of computers, we programmed the Artificial Intelligence entity to access outside resources of memory space by aggregation, exploiting them during calculations.

We also programmed the Artificial Intelligence entity to analyze variables of numbers and written codes for calculation to find independencies between potential processes to execute these calculations in parallel and minimize the time needed for processing. Furthermore, the Artificial Intelligence entity is programmed to reduce amounts of calculations exponentially by detecting patterns of redundant calculations and converging them into reduced processes without redundancies. As a result, this Artificial Intelligence entity can handle numbers with lengths of Gigabytes in seconds and execute composed calculations on them in a few dozen seconds when using ordinary computers.

The logic of developed objects of "SuperInfinityInteger" and "SuperInfinityDouble" is based on expressing them in the form of subgroups of virtual bites while relying on the Artificial Intelligence entity to forward results from one subgroup to others during arithmetic calculations and binary bitwise operations without neglecting any digit of any number. Therefore, the developed software library and Artificial Intelligence entity provide an infinite persistent precision for each digit of calculated results even when surpassing lengths of Gigabytes and Terabytes. Furthermore, the Artificial Intelligence entity is programmed to optimize calculations that may need exponential time for processing by executing them in linear time, which is manifested by shifting infinity calculations from NP to P, detecting patterns of redundant calculations, converging them into reduced processes and executing these processes in linear time and parallel.

On the one hand, the presented software library and Artificial Intelligence entity in this paper are currently programmed to function on any ordinary computer with at least one processor, 4 Gigabytes of RAM and 10 Gigabytes of EEPROM (Electronically Erasable Programmable Read-Only Memory). On the other hand, these resources can also function in reduced environments with less space for RAM and EEPROM while consuming seconds during calculations on numbers expressed in Gigabytes. However, to support numbers with lengths of Terabytes, there is a need to have more storage space on EEPROM to be virtualized by the Artificial Intelligence entity and then used along with RAM.

The technological trends of big data [26, 27] enable the handling of massive amounts of information in terms of storage and processing either by distributing these data on systems or by processing them on a super-computer [28, 29]. However, in our results, we developed the Artificial Intelligence entity to augment the processing capacities of ordinary computers by virtualizing memory spaces of EEPROM and using them along with RAM. We programmed algorithms within this entity to handle big digits data as infinity numbers. Then, we programmed this entity to optimize the needed times for calculations (especially multiplication, division, roots, exponentials, etc.) by shifting infinity operations from NP to P in order to be executed only in dozens of seconds at maximum when the lengths of input variables and results exceed Gigabytes.

After developing this software library and reinforcing it by developing the Artificial Intelligence entity, we integrated them within an application, which we named Super Infinity Calculator.

$$SuperInfinityInteger\ A\ = new\ SuperInfinityInteger(boolean\ P_3, SuperInfinityInteger\ P_2, BigInteger\ P_1) \quad (1)$$

This paper is structured as follows: Section 2 presents the developed software library. Section 3 presents the programed Artificial Intelligence entity for super calculations on infinity numbers. Section 4 presents the exploit of programmed AI entity in the technical context of shifting infinity calculations from NP to P. Section 5 presents the technical context of shifting infinity operations with infinite quantities of interdependencies from NP to P. Section 6 presents the implementation of developed resources within Super Infinity Calculator along with statistics of its processing capacities. Finally, section 7 for conclusion.

2. Developed Software Library

2.1. Technical Description of Software Library

Our developed software library to support super arithmetic calculations and binary bitwise operations on infinity numbers is based on two new types of objects: the "SuperInfinityInteger" object and the "SuperInfinityDouble" object. We have programmed two versions of this software library, one version in Java and another version in C++.

We have programmed these two objects to be extendable in terms of length, where each one of them can be expressed on infinite lengths of virtual bites reassembled into subgroups of bits with flexible lengths.

Each one of these two objects is independent of predefined limits of length, and their sizes are extendable to exceed Gigabytes and Terabytes. Therefore, we developed an Artificial Intelligence entity to manage the required memory space by infinity numbers while supporting their storage on EEPROM.

This Artificial Intelligence entity is responsible for virtualizing memory spaces of EEPROM to be exploited during calculations along with RAM to reinforce the processing capacities of used computers.

2.2. Developed object of super infinity integer

We programmed the object of "SuperInfinityInteger" to have consisted of three parts (variables), which are as shown in (Eq 1). The first part on the right (P_1) is a BigInteger, the second part in the middle (P_2) is a child-object with type "SuperInfinityInteger", and the third part (P_3) in the left is a Boolean condition to determine whether the second part is null or not. Therefore, instead of relying on defined lengths to execute calculations and binary operations on a "SuperInfinityInteger", we rely on using the Boolean condition to determine whether to forward calculations furthermore to include the child-object or to stop operations at the level of contained BigInteger.

We programmed two categories of the object "SuperInfinityInteger". The first category is a parent-object, and the second is a child-object. The parent-object cannot be a child-object of any "SuperInfinityInteger", whereas a child-object can be a child-object of a "SuperInfinityInteger" and at the same time being a parent of another child-object.

Relying on these two categories, a "SuperInfinityInteger" can be expressed as an infinite tree of nodes, as shown in (Fig. 1), where the first node is from the category parent-object and the other nodes are from the category child-object.

2.3. Developed Object of Super Infinity Double

We programmed the object of "SuperInfinityDouble" to have consisted of two parts, which are as shown in (Eq. 2). The first part (P_1) in the right of the object of "SuperInfinityDouble" is a "SuperInfinityInteger" expressed after the floating point at the right side of zero, whereas the second part (P_2). On the left of this object of "SuperInfinityDouble" is a "SuperInfinityInteger" expressed at the left side before the floating point.

2.4. Developed Infinity for Loop

We programmed an object named "InfintyForLoop", which is expressed as shown in (Eq. 3), in order to conduct infinity loops that may be automatically distributed on parallel segments initiated by "ParallelismAgent" and controlled by "ResourceManagmentAgent" in the Artificial Intelligent entity (Fig. 2). The cursor of the loop is named as "$SuperInfinityInteger(C_r)$".

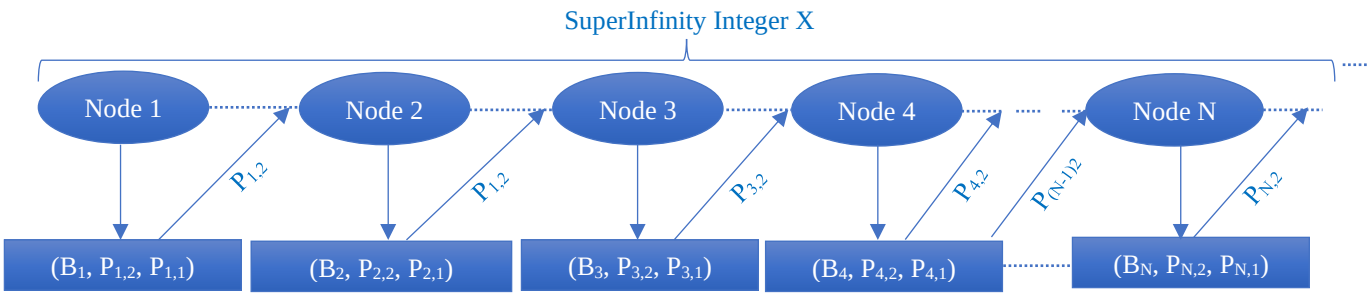

Fig. 1 Developed nodes tree structure for SuperInfinityIntegers

$$InfinityForLoop(SuperInfinityInteger\ A_1, SuperInfinityInteger\ A_2, SuperInfinityInteger\ A_3, int\ A_4, boolean\ B) \quad (2)$$

$$SuperInfinityDouble\ B = new\ SuperInfinityDouble(SuperInfinityInteger\ P_2, SuperInfinityInteger\ P_1) \quad (3)$$

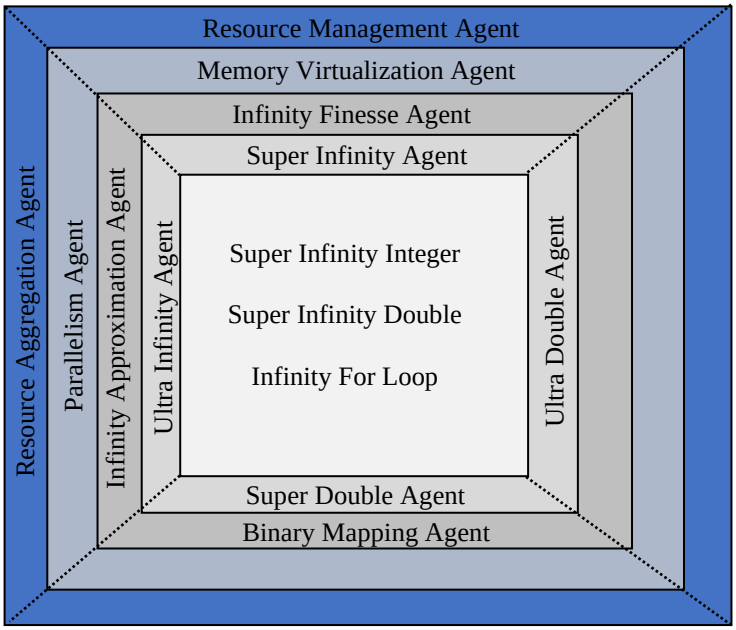

Fig. 2 Developed software library and developed agents of Artificial Intelligence to conduct computational operations on infinity numbers

The $SuperInfinityInteger(A_1)$ shown in (Eq. 3) defines the start value of the loop's cursor where $(C_r = A_1)$. The $SuperInfinityInteger(A_2)$ defines the end value of the loop's cursor where $(C_r \leq A_2)$. The $SuperInfinityInteger(A_3)$ defines the step of augmenting the value of the cursor where $(C_r = C_r.AddYL(A_3))$. The integer (A_4) defines the maximum segments of calculations to be conducted in parallel.

The Boolean B defines whether to allow "ResourceManagmentAgent" to conduct automatic distributions of processes on resilient numbers of parallel segments that may exceed the value of (A_4) depending on the conducted analysis by "ParallelismAgent" and also depending on accessible resources of processors, RAM, virtualized memory, EEPROM and aggregated spaces of memory by "ResourceAggregationAgent" in the AI entity (Fig. 2).

3. Programmed Artificial Intelligence Entity

3.1. Technical Description of Artificial Intelligence Entity

We programmed the Artificial Intelligence entity to conduct calculations in parallel on each of the two objects of "SuperInfinityInteger".

We also programmed this AI entity to be responsible for forwarding processes recursively from a parent-object to a child-object of "SuperInfinityInteger". In addition, forwarding them from any child-object to its successive child-object.

As a result, this Artificial Intelligence entity is responsible for conducting processes in parallel while forwarding the results of calculations and binary operations during computations.

The programmed AI entity consists of multiple software agent entities responsible for managing calculations, optimizing the exploitation of hardware resources, and reducing the time of computation conduction by relying on parallelism while shifting calculations from the exponential dimension of execution time to the linear dimension.

The Artificial Intelligence entity is responsible for the following functionalities:

1) Optimizing the exploit of RAM.
2) Virtualizing storage memory to be used along with RAM.
3) Managing the exploit of integrated hardware resources.
4) Managing the access and exploitation of external hardware resources such as hard drives and distributed computers over the network. (Distributed Computation)
5) Conducting calculations in parallel (Parallel Computation).
6) Detecting patterns of redundant calculations and converging them into unified segments of computation.
7) Regrouping infinite numbers in the form of subgroups of bits.
8) Storing subgroups of bits expressing infinite numbers as files during calculations.

9) Predicting the necessary time to execute segments of calculations.
10) Compressing files of subgroups of bits representing infinite numbers when these numbers are not used during significant amounts of time.
11) Switching from the mode of infinite persistent precision to the mode of "grosson expressions" when there is no more enough space in RAM and Virtualized memory.

The developed software library and developed agents of Artificial Intelligence entities to conduct operations on infinity numbers are shown in (Fig. 2).

3.2. Super Infinity Agent and Ultra Infinity Agent

The Artificial Intelligence entity is programmed to have consisted of two types of agents: "SuperInfinityAgent" and "UltraInfinityAgent".

The developed "SuperInfinityAgent" is responsible for fetching through the parent-object and child-objects of any "SuperInfinityInteger". In addition, any instant of "SuperInfinityAgent" can be connected to any other instant of the same type to conduct synchronous recursive operations of comparison through parent-objects and child-objects of "SuperInfinityIntegers". Furthermore, we programmed the "SuperInfinityAgent" to support the conduction of certain arithmetic operations and binary operations on its handled objects during its connections with other instants of "SuperInfinityAgents".

The "SuperInfinityAgent" is also programmed to be incharge of deleting or copying any object of "SuperInfintyIntegers" whereas being able to create new ones, in the condition of handling one object of "SuperInfintyInteger" at a time.

The developed "UltraInfinityAgent" is responsible for initiating more than one "SuperInfinityAgent" in parallel to handle simultaneous calculations and execute complex processes of multiplication, division, roots, etc. These calculations are complicated because they may generate results expressed on either Gigabytes or Terabytes, depending on the sizes of input numbers. Therefore, we programmed the Artificial Intelligence entity to store any "SuperInfinityInteger" on the EEPROM as files when its size exceeds a specific limit instead of being expressed on RAM and virtualized memory.

The Artificial Intelligence entity is programmed to store any infinity number with a length exceeding Gigabytes in the form of separated files on the EEPROM, where the size of each file may exceed one Gigabyte, and then handle these files during calculations recursively.

The programmed "SuperInfinityAgent" is incharge of shown operations in Table 1, whereas the "UltraInfinityAgent" is programmed to be incharge of shown operations in Table 2.

The developed agent of "SuperInfinityAgent" is responsible for creating "SuperInfinityInteger" objects from either String variables, BigInteger variables, integer variables or sequences of bytes. However, the results of operations are expressed only as String variables, arrays of String variables or sequences of bytes.

3.3. Resource Management Agent and Memory Virtualization Agent

In order to optimize the exploit of computational resources of RAM, EEPROM and processors, we programmed the Artificial Intelligence entity to consist of another agent, which we named "ResourceManagmentAgent". In addition, we programmed an agent named "MemoryVirtualizationAgent", the responsible agent for virtualizing memory spaces of EEPROM and using them along with the RAM.

The "ResourceManagementAgent" is responsible for managing the distribution of calculations and binary operations on processors, RAM, and virtualized memory space in resilient manners that enable their executions to run in parallel. In addition, this agent is responsible for transiting the content of any "SuperInfinityInteger" from RAM to EEPROM in order to be stored as files in the case where its size exceeds 128 Megabytes. Furthermore, it is responsible for storing all new infinity numbers as files instead of being held on the RAM or the virtualized memory space, starting from any moment the free space of RAM drops under 256 Megabytes.

This "ResourceManagementAgent" is also programmed to store infinity numbers as compressed zip files to optimize the use of memory space in cases where these numbers are not used in further calculations for considered amounts of time. Therefore, the "ResourceManagementAgent" is also programmed to predict time amounts of processing based on the involvement of variables and then decide whether to store any of these infinity numbers as zip files or not.

3.4. Resource Parallelism Agent and Resource Aggregation Agent

We programmed an agent named "ParallelismAgent", which is incharge of code analysis and processes analysis to define the operations that may be executed in parallel due to the independence of their involved variables from each other. Then, this "ParallelismAgent" converges the outputs of parallel processes to provide finalized results. In addition, this agent detects redundant patterns of calculations and converges them into reduced processes.

We programmed an agent named "ResourceAggregationAgent" to define outside resources of memory space that may be accessible by used computer in order to exploit them by the "ResourceManagementAgent", such as USB, hard drive, different computers on local network or accessible servers.

Table 1. Operations handled by SuperInfinityAgent on SuperInfinityIntegers

Operations executed by SuperInfinityAgent	Description
SuperInfinityInteger.NegativeLY()	-X.
SuperInfinityInteger.AddLY(SuperInfinityInteger Y)	X+Y.
SuperInfinityInteger.SubtractLY(SuperInfinityInteger Y)	X-Y.
SuperInfinityInteger.NOTLY(SuperInfinityInteger Y)	Complement of X (NOT X).
SuperInfinityInteger.ANDLY(SuperInfinityInteger Y)	X AND Y.
SuperInfinityInteger.ORLY(SuperInfinityInteger Y)	X OR Y.
SuperInfinityInteger.XORLY(SuperInfinityInteger Y)	X XOR Y.
SuperInfinityInteger.ANDNOTLY(SuperInfinityInteger Y)	X AND(NOT Y).
SuperInfinityInteger.NOTXORLY(SuperInfinityInteger Y)	NOT(X XOR Y).
SuperInfinityInteger.ShiftRightLY(SuperInfinityInteger N)	Shifting bites of X to the right by N bites.
SuperInfinityInteger.ShiftLeftLY(SuperInfinityInteger N)	Shifting bites of X to the left by N bites while keeping the bite indicating that it is a signed infinity number at its original place.
SuperInfinityInteger.ShiftLeftLY2(SuperInfinityInteger N)	Shifting bites of X to the left by N bites without keeping the bite indicating that it is a signed infinity number at its original place.
SuperInfinityInteger.IsEqualLY(SuperInfinityInteger Y)	Is (X==Y)
SuperInfinityInteger.MaxLY(SuperInfinityInteger Y)	Maximum among X and Y.
SuperInfinityInteger.MinLY(SuperInfinityInteger Y)	Minimum among X and Y.
SuperInfinityInteger.CompareToLY(SuperInfinityInteger Y)	Comparison between X and Y. The result is 1 if $X > Y$. The result is -1 if $X < Y$, whereas the result is 0 if X=Y.
SuperInfinityInteger.ValueOfStringLY(String s)	Creating SuperInfinityInteger from String s.
SuperInfinityInteger.ValueOfStringArrayLY(String[][] S)	Creating SuperInfinityInteger from String array S.
SuperInfinityInteger.ValueOfBigIntegerLY(BigInteger y)	Creating SuperInfinityInteger from BigInteger y.
SuperInfinityInteger.ValueOfIntegerLY(int x)	Creating SuperInfinityInteger from integer x.
SuperInfinityInteger.ValueOfBytesLY(Byte[] x)	Creating SuperInfinityInteger from Bytes sequence.
SuperInfinityInteger.toStringLY()	Converting SuperInfinityInteger to String.
SuperInfinityInteger.toStringArrayLY()	Converting SuperInfinityInteger to a String Array of two dimensions.
SuperInfinityInteger.toBytesLY()	Converting SuperInfinityInteger to sequence of bytes.
SuperInfinityInteger.toBytesArrayLY()	Converting SuperInfinityInteger to an array of sequences of bytes.
SuperInfinityInteger.ZEROLY ()	Value zero in type of SuperInfinityInteger.
SuperInfinityInteger.ONELY()	Value one in type of SuperInfinityInteger.

Table 2. Operations handled by UltraInfinityAgent on SuperInfinityIntegers

Operations executed by UltraInfinityAgent	Description
SuperInfinityInteger.MultiplyLY(SuperInfinityInteger Y)	X*Y.
SuperInfinityInteger.DivideLY(SuperInfinityInteger Y)	X/Y without expressing values at the right of floating point after zero.
SuperInfinityInteger.ModLY(SuperInfinityInteger Y)	Returning X%Y.
SuperInfinityInteger.DivideLY2(SuperInfinityInteger Y)	Returning two results A and B where A=X/Y and B=X%Y.
SuperInfinityInteger.PowLY(SuperInfinityInteger Y)	X^Y.

3.5. Infinity Approximation Agent

In the case of calculating (X^Y) using two infinity numbers, X and Y, where the length of each one of them surpasses 1 Megabyte, the result Z may be expressed on more than 1024 Gigabytes, which may exceed the storage capacities of ordinary computers even when using data compression techniques. Therefore, we developed an agent named "InfinityApproximationAgent" to express Z as ($Z = N * e^M$) where M is a "SuperInfinityInteger" and N has a length of Gigabytes exceeding the length of X. As a result, Z is handled by storing N and M on only a few Gigabytes instead of being stored on more than a thousand Gigabytes.

Table 3. Operations handled by SuperDoubleAgent on SuperInfinityDoubles while exploiting SuperInfinityAgents.

Operations executed by SuperDoubleAgent	Description
SuperInfinityDouble.NegativeLY()	-X.
SuperInfinityDouble.AddLY(SuperInfinityInteger Y)	X+Y.
SuperInfinityDouble.AddLY2(SuperInfinityDouble DY)	X+DY.
SuperInfinityDouble.SubtractLY(SuperInfinityInteger Y)	X-Y.
SuperInfinityDouble.SubtractLY2(SuperInfinityDouble DY)	X-DY.
SuperInfinityDouble.IsEqualLY(SuperInfinityDouble DY)	Is (X==DY)
SuperInfinityDouble.MaxLY(SuperInfinityInteger Y)	Maximum among X and Y.
SuperInfinityDouble.MaxLY2(SuperInfinityDouble DY)	Maximum among X and DY.
SuperInfinityDouble.MinLY(SuperInfinityInteger Y)	Minimum among X and Y.
SuperInfinityDouble.MinLY2(SuperInfinityDouble DY)	Minimum among X and DY.
SuperInfinityDouble.CompareToLY(SuperInfinityInteger Y)	Comparison between X and Y. The result is 1 if X>Y. The result is -1 if X<Y, whereas the result is 0 if X=Y.
SuperInfinityDouble.CompareToLY2(SuperInfinityDouble DY)	Comparison between X and DY. The result is 1 if $X > DY$. The result is -1 if $X < DY$, whereas the result is 0 if X=DY.
SuperInfinityDouble.ValueOfStringLY(String s)	Creating SuperInfinityDouble from String s.
SuperInfinityDouble.ValueOfStringArrayLY(String[][] S)	Creating SuperInfinityDouble from String array S.
SuperInfinityDouble.ValueOfDoubleLY(double dy)	Creating SuperInfinityDouble from double dy.
SuperInfinityDouble.ValueOfSuperIntegerLY(SuperInfinityInteger X)	Creating SuperInfinityDouble from SuperInfinityInteger X.
SuperInfinityDouble.ValueOfBytesLY(Byte[] x)	Creating SuperInfinityDouble from Bytes sequence.
SuperInfinityDouble.toStringLY()	Converting SuperInfinityDouble to String.
SuperInfinityDouble.toStringArrayLY()	Converting SuperInfinityDouble to String array.
SuperInfinityDouble.toBytesLY()	Converting SuperInfinityDouble to sequence of bytes.
SuperInfinityDouble.toBytesArrayLY()	Converting SuperInfinityDouble to array of sequences of bytes.

We also rely on executing this concept of approximation on the results of complicated operations that include different arithmetic calculations where these results are expected to surpass capacities of RAM, virtualized memory, EEPROM and aggregated spaces of memory.

The “InfinityApproximationAgent” works in coherence with “ResourceManagmentAgent” and “ResourceAggregationAgent” to optimize the exploit of processors, RAM, virtualized memory, EEPROM and other accessible resources of memory space.

Therefore, in case when have sufficient resources to conduct the necessary calculations and store infinity numbers, the “InfinityAproximationAgent” does not modify involved variables, and we continue having an infinite persistent precision for each digit of involved infinity numbers, including the digits at the right side of floating point.

3.6. Super Double Agent and Ultra Double Agent

In order to handle the programmed object of “SuperInfintyDouble”, we developed two different agents within the Artificial Intelligence entity: “SuperDoubleAgent” and “UltraDoubleAgent”.

The programmed “SuperDoubleAgent” is responsible for handling one object of “SuperInfinityDouble” at a time by initiating two instants of “SuperInfinityAgents”, where the first instant is incharge of handling the right subobject of the corresponding “SuperInfinityDouble” object. The second instant is incharge of handling its left subobject (Eq. 2). At the end of executed operations by the first created instant, the “SuperDoubleAgent” is responsible for forwarding relevant results of its operations to the second created instant if they are of interest to the requested operations on the left side of corresponding object of “SuperInfinityDouble”.

The programmed “UltraDoubleAgent” is responsible for handling two objects of “SuperInfinityDoubles” at a time, “SuperInfinityDouble” X and “SuperInfinityDouble” Y, by initiating four parallel instants of “UltraInfinityAgents” to execute its shown operations in Table 4. The first instance is incharge of handling operations between the right subobject of “SuperInfinityDouble” X and the right subobject of “SuperInfinityDouble” Y. The second instant is incharge of handling operations between the right subobject of “SuperInfinityDouble” X and the left subobject of “SuperInfinityDouble” Y. The Third instant is incharge of handling operations between the left subobject of

"SuperInfinityDouble" X and the right subobject of "SuperInfinityDouble" Y. The fourth instant is incharge of handling operations between the left subobject of "SuperInfinityDouble" X and the left subobject of "SuperInfinityDouble" Y.

At the end of executed operations by these initiated instants by "UltraDoubleAgent", this agent initiates four objects of "SuperInfintyDoubles" to contain the results of these preceding instants of "UltraInfinityAgents". Then, the "UltraDoubleAgent" initiates two instants of "SuperDoubleAgents" to handle these four objects of "SuperInfintyDoubles" by executing necessary operations on them.

The programmed "SuperDoubleAgent" is incharge of shown operations in Table 3, whereas the "UltraDoubleAgent" is programmed to be incharge of shown operations in Table 4.

Table 4. Operations handled by UltraDoubleAgent on SuperInfinityDoubles while exploiting UltraInfinityAgents and SuperInfinityAgents.

Operations executed by UltraDoubleAgent	Description
SuperInfinityDouble.MultiplyLY(SuperInfinityInteger Y)	X*Y.
SuperInfinityDouble.MultiplyLY2(SuperInfinityDouble DY)	X*DY.
SuperInfinityDouble.InverseLY()	1/X whereas expressing the value at the right of floating point after zero if it exists.
SuperInfinityDouble.DivideLY(SuperInfinityInteger Y)	X/Y whereas expressing the value at the right of floating point after zero if it exists.
SuperInfinityDouble.DivideLY2(SuperInfinityDouble DY)	X/DY whereas expressing the value at the right of floating point after zero if it exists.
SuperInfinityDouble.PowLY(SuperInfinityInteger Y)	X^Y.
SuperInfinityDouble.SqrtLY(SuperInfinityInteger Y)	$\sqrt[Y]{X}$.

$$X = 2;\ Y = 2^{512} + 2^{256} + 2^{140} + 2^{32};\ Z = X^Y;\ Operations\_Reduction\_Factor = 515/Y = 3{,}84e^{-152} \quad (4)$$

$$Y = \sum_{\{i=0\}}^{\{i=N\}} \Gamma_{(Y,i)} \quad (5)$$

$$\Gamma_{(Y,i)} = Y\ AND\ 2^i \quad (6)$$

$$X = 2;\ Y = 2^{256} + 2^{128} + 2^{32};\ Z = X^Y;\ Operations\_Reduction\_Factor = 418/Y = 3{,}6e^{-75} \quad (7)$$

3.7. Infinity Finesse Agent

The results of calculations and binary operations on infinity numbers may have many digits with the value zero at the left edges of "SuperInfinityIntegers" or the right edges of "SuperInfinityDoubles" after floating point, which is a waste of memory space. Therefore, we programmed an agent named "InfinityFinesseAgent" within the Artificial Intelligence entity to be responsible for eliminating these digits with value zero and optimizing the used memory space by each infinity number.

3.8. Binary Mapping Agent

We programmed an agent named "BinaryMappingAgent" to express infinity numbers as subgroups of virtual bites and use each subgroup to guide the conduction of composed arithmetic calculations such as the processes of SuperInfinityInteger.MultiplyLY(Y), SuperInfinityInteger.PowLY(Y), SuperInfinityDouble.MultiplyLY(Y), SuperInfinityDouble.PowLY(Y), etc.

This "BinaryMappingAgent" can reduce an ordinary total amount of calculations T to be only $3{,}47Te^{-18}$. Furthermore, this agent is relied on to reduce massive amounts of calculations exponentially to reach reduction levels under the value $3{,}47e^{-18}$, which shifts infinity calculations from NP to P.

4. The Exploit of Programmed AI Entity in the Technical Context of P Versus NP

The programmed Artificial Intelligence entity relies on parallel computation to conduct calculations quickly. In addition, it relies on shifting arithmetic operations and binary operations from the exponential dimension of execution time (T^N) to the linear dimension of time ($N * ln(T)$) by relying on the Binary Mapping Agent, Parallelism Agent and Resource Management Agent, which are programmed in the technical context of shifting infinity calculations from NP to P. Furthermore, this AI entity detects patterns of redundant calculations and then converges them into unified segments to avoid overloading during resource exploits whereas minimizing consumed time in total.

As an example of the use of the "BinaryMappingAgent" to shift operations from the exponential dimension of time execution to the linear dimension, we suppose the case when having two numbers ($X = 2$) and ($Y = 2^{64}$) whereas we seek to calculate ($Z = X^Y$). Instead of conducting the multiplication process over a total amount of (2^{64}) processes, the "BinaryMappingAgent" guides the calculation process to be conducted only over 64 processes by multiplying the result of each process by itself in the following multiplication process. As a result, we reduce the total amount of calculations massively, which may be quantified as follows: $Operations\_Reduction\_Factor = 64/2^{64} = 3{,}47e^{-18}$.

As another example of the use of the "BinaryMappingAgent", we suppose the case of having two numbers ($X = 2$) and ($Y = 2^{256} + 2^{128} + 2^{32}$) whereas we seek to calculate ($Z = X^Y$), and we suppose ($y_1 = 2^{256}$), ($y_2 = 2^{128}$) and ($y_3 = 2^{32}$). Instead of conducting the multiplication process over a total amount equals Y, the calculations are distributed on 3 parallel segments where the first segment calculates (X^{y_1}) over 256 processes of multiplication, the second segment calculates (X^{y_2}) over 128 processes of multiplication, and the third segment calculates (X^{y_3}) over 32 processes of multiplication. In parallel with these three segments, a fourth segment is initiated to calculate ((X^{y_1}) $*$ (X^{y_2}) $*$ (X^{y_3})) over two processes of multiplication. As a result, there are only 418 conducted processes of multiplication instead of conducting an amount of Y multiplications, which induce a reduction that may be expressed as shown in (Eq. 4).

The "BinaryMappingAgent" relies on the mathematical operator (Γ) shown in (Eq. 6), which we programmed to concretize the binary mapping of infinity numbers mathematically.

By using the operator (Γ) on Y, the "BinaryMappingAgent" expresses the value of Y as shown in (Eq. 7), where (N) is a "SuperInfinityInteger". Therefore, the parallelism agent launches multiple processes to calculate (X^Y) where each process with index (j) calculates ($\Gamma_{(Y,i)}$). However, the role of the parallelism agent does not stop only on executing processes in parallel but also includes detecting repeated patterns of calculations among parallel processes that may be categorized as redundancies.

As a simple example of redundancies reduction by "ParallelismAgent" and "ResourceManagementAgent", we suppose the case of having two numbers ($X = 2$) and ($Y = 2^{512} + 2^{256} + 2^{140} + 2^{32}$) whereas we seek to calculate ($Z = X^Y$), and we suppose ($y_1 = 2^{512}$), ($y_2 = 2^{256}$), ($y_3 = 2^{140}$) and ($y_4 = 2^{32}$). By relying on "BinaryMappingAgent" and "ParallelismAgent", the calculations are distributed on 4 parallel segments. The first segment calculates (X^{y_1}) over 512 processes of multiplication, the second segment calculates (X^{y_2}) over 256 processes of multiplication, the third segment calculates (X^{y_3}) over 140 processes of multiplication, and the fourth segment calculates (X^{y_4}) over 32 processes of multiplication.

However, the "ParallelismAgent" interferes by detecting that there are redundancies of calculation among (X^{y_1}), (X^{y_2}), (X^{y_3}) and (X^{y_4}). Therefore, "ResourceManagementAgent" holds their processes in order to not consume processing resources, and then "ParallelismAgent" converges their calculations into a fifth segment calculating (X^{y_1}) and extracting the values of (X^{y_2}), (X^{y_3}) and (X^{y_4}) during this process to be provided to their corresponding segments. A sixth segment of calculation, in parallel with others, is incharge of calculating (X^{y_1}) $*$ (X^{y_2}) $*$ (X^{y_3}) $*$ (X^{y_4}) over 3 processes of multiplication. As a result, there are only 515 conducted processes of multiplication instead of conducting an amount of Y multiplications, which induce a reduction that may be expressed as shown in (Eq. 7).

The "ParallelismAgent" is programmed to detect multiple redundancies of calculation among parallel processes with multiple variables exceeding the previous case of having only X and Y. After redundancies detection, "ParallelismAgent" is responsible for the following processes: holding the calculations of these redundancies with collaboration with "ResourceManagementAgent" in order to not consume processing resources, converging these redundant calculations into simplified processes and then providing results to other processes where redundancies were detected.

The principle of relying on the use of "ParallelismAgent" along with "ResourceManagementAgent" is to optimize the performance of the Binary Mapping Agent during the conduction of massive calculations by transferring their executions from exponential dimensions to linear dimensions. Therefore, instead of executing infinity calculations over exponential amounts of time [30, 31], they are developed to be executed over linear time, shifting them from NP to P.

As a result, the developed AI entity is capable of shifting infinity calculations from NP to P. However, currently, it can handle only infinity calculations with redundancies and common operation patterns where there is no "infinite quantity of interdependencies" between variables and processes.

5. Technical Context of P versus NP for Infinity Operations while having Infinite Quantities of Interdependencies

The developed techniques in the presented AI entity are a showcase on adapting infinity calculations from NP to P, in condition of having redundant operations, having common patterns between variables and processes, and relying on parallel computations. However, when encountering infinite quantities of interdependencies between variables and

processes, the complexity can augment itself from the dimension of $a^{(b)^{(c)}}$ to the dimension $a^{(b)^{(c^d)}}$ and even higher levels of nested exponentials.

As an example of the complexity of having infinite quantities of interdependencies, the case when having an infinite matrix map of variables where each variable is dependent on too many neighbours of variables, such as the case of having an infinite structure of Sudoku.

When having infinity operations characterised by infinite quantities of interdependencies, we should re-express the operations at the mathematical level to neutralize these interdependencies into independencies as much as possible while merging operations and formulas. Then, we should re-express the problem at the algorithmic level of computation while relying on parallel computation, distributed computation, redundancies elimination, and pattern converging.

As a result, in order to be able to shift all infinity operations that include having infinite quantities of interdependencies between variables and processes, we identified 12 technical axes as shown in Table 5.

Table 5. Identified technical axes to shift infinity operations and infinite quantities of interdependencies from NP to P.

ID	Axe	Description
1	Capacities of RAM.	Available memory space of RAM.
2	Capacities of GPU and DPU.	Available GPU and DPU.
3	Capacities of processors.	Available number of processors.
4	Capacities of frequency.	Available frequency of processing.
5	Capacities of storage and memory virtualization.	Available memory spaces of storage and access rate to virtualized memory.
6	Capacities of parallel computation.	Maximum limit of possible parallel processes to execute.
7	Capacities of distributed computation.	Distributing computation processes over processers, computers, networks and cloud services.
8	Capacities of eliminating redundant operations.	Eliminating redundant operations that are based on using the same operators or the same logic of calculation.
9	Capacities of eliminating redundant variables.	Eliminating redundant variables to free memory space while sharing access to one original version of each variable among processes.
10	Capacities of converging common patterns of processing into unified segments.	Detecting common patterns of processing and converging them into unified segments to free hardware resources and minimize consumed time.
11	Capacities to re-express operations on infinite quantities of interdependencies between variables.	Re-express operations when having infinite matrix structures of variables with infinite quantities of interdependencies between variables, in order to reduce the quantity of processes.
12	Capacities to re-express operations on infinite quantities of interdependencies between processes.	Re-express operations when having infinite matrix structures of processes with infinite quantities of interdependencies between processes, in order to reduce the quantity of processes.

Table 6. Examples showing precisions of calculation and digits display when using developed software library and Artificial Intelligence entity.

Variable	Integer Value	Total amount of digits
X	1123 654889568 789993399 999999789 777789336 954789999 999986333 331112225 548879666 321455666 698788963 333211111 236547899 963334445 568799833 356498999 654478965 554789666 321477897 777777899 963214586 599632144 789666325 558558884 444999321	220
Y	9 999999988 882222000 000000003 336458123 641112223 654782222 111145632 221111000 000236488 000222369 800004478 522233311 999998452 200000006 333331112 225548879 666321455 666698788 963333211 111236547 899963334 445568799 833356498 999654478 965554789 666321477 897777777 899963214 111122220 000003336 999888844 789666325 558558884 444999321	307
Z=X*Y	11236 548883195 354323159 677126404 140180177 597994453 531409142 029266691 252217065 355607950 617099685 527952949 207582997 514334618 961753124 363192431 106380573 311045065 764721117 424179846 861361296 593235819 987808212 573468584 980973642 601083931 593908178 075613216 495340694 336502583 196362740 654492997 097272808 998073831 767413253 607534807 543814687 643171283 891013531 426980095 638811153 193027549 547047138 065579033 957394139 204032845 664617307 499119910 261788269 726723297 712640222 628919743 618532752 252898232 689830016 048801732 275572754 435059923 690461041	527

6. Implementation of Super Infinity Calculator and Statistics of its Use

We integrated the developed software library and Artificial Intelligence entity within an application to be used for super arithmetic calculations and binary operations on infinity numbers. We named this application as Super Infinity Calculator. It is as shown in Fig. 3. We integrated a command user interface in this application to support code writing of complex operations on "SuperInfinityIntegers" and "SuperInfinityDoubles" while being able to use the programmed "InfintyForLoop" and other usual computational variables such as integers, doubles, strings, chars, etc.

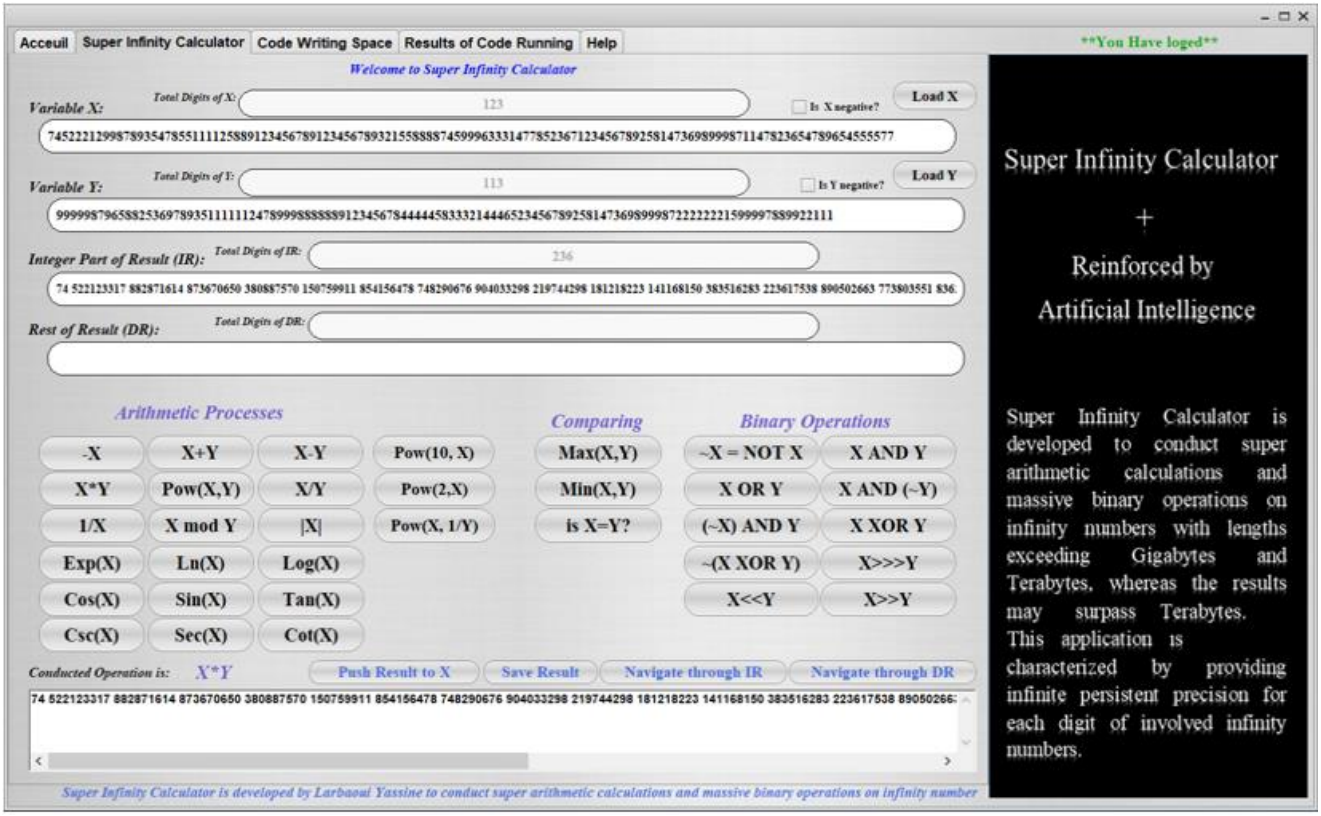

Fig. 3 Developed Application of Super Infinity Calculator

Table 7. Statistics of consumed times in Nanoseconds by Super Infinity Calculator during arithmetic operation (X^Y).

Operation	Average Consumed Time in Nanoseconds	Total Amount of digits	Length of Result in bits
$2^{(1\,000\,000\,000)}$	500 000 nS	300 010 300	2 400 082 400
$2^{(2\,000\,000\,000)}$	1 000 000 nS	600 020 600	4 800 164 800
$2^{(3\,000\,000\,000)}$	2 020 000 nS	900 030 900	7 200 247 200
$2^{(4\,000\,000\,000)}$	3 040 000 nS	1 200 041 200	9 600 329 600
$2^{(5\,000\,000\,000)}$	4 070 000 nS	1 500 051 500	12 000 412 000
$2^{(6\,000\,000\,000)}$	5 120 000 nS	1 800 061 800	14 400 494 400
$2^{(7\,000\,000\,000)}$	6 180 000 nS	2 100 072 100	16 800 576 800
$2^{(8\,000\,000\,000)}$	7 270 000 nS	2 400 082 400	19 200 659 200
$2^{(9\,000\,000\,000)}$	8 420 000 nS	2 700 092 700	21 600 741 600
$2^{(10\,000\,000\,000)}$	9 660 000 nS	3 000 103 000	24 000 824 000

We used this programmed application to conduct calculations on infinity numbers resulting from composed arithmetic calculations to prove the potentials of our developed objects of "SuperInfinityIntegers" and "SuperInfinityDoubles". Table 6 presents examples of using the Super Infinity Calculator to highlight the precision of provided calculations and digits display, using variables with high values.

Recently, we deployed the Super Infinity Calculator on a virtual machine with one processor, 4 Gigabytes of RAM, 10 Gigabytes of EEPROM and 2,84 GHz of processing frequency in order to provide statistics of needed time to execute calculations on infinity numbers while providing infinite persistent precision for each digit of these numbers. The results of these statistics are shown in Table 7.

We used the limited capacities of virtual machines to provide a general vision of the performance of the developed software library and Artificial Intelligence entity on limited hardware devices and ordinary computers. Using integer variables in programming languages enables calculations to support (2^{32}) values, whereas using BigInteger variables may enable calculations to reach high values depending on the available size of RAM while having official limits of ($Max = 2^{(Integer.Max_{Value})} - 1; Min = -2^{(Integer.Max_{Value})}$).

Therefore, we used the Super Infinity Calculator to conduct operations surpassing the limitations of integers and BigIntegers.

7. Conclusion

The presented software library and Artificial Intelligence entity in this paper can conduct arithmetic calculations and binary bitwise operations on infinity numbers with lengths exceeding Gigabytes and Terabytes in short times while reinforcing processing capacities of used computers by virtualizing memory spaces of EEPROM to be used along with the RAM. In addition, these presented resources provide infinite persistent precision for each digit of involved numbers even when exceeding lengths of Gigabytes where the Artificial Intelligence entity may store the results in outside memory spaces if RAM, virtualized space of memory and EEPROM were satirized. Furthermore, these software and AI resources rely on parallel computation, distributed computation and shifting calculations from the exponential dimension of execution time to the linear dimension. Therefore, this software library and Artificial Intelligence entity enable computers to reach supreme levels of exploit in terms of computational calculations and binary operations on infinity numbers while providing infinite persistent precision for each digit during calculations and values display, including the digits after floating point.

References

[1] M. Friedewald, "The First Computers-History and Architectures [Review]," *IEEE Annals of the History of Computing*, vol. 23, no. 2, pp. 75-76, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[2] Akira Maruoka, *Concise Guide to Computation Theory*, Springer, London, pp. 205–421, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[3] J. Hartmanis, and R.E. Stearns, "On the Computational Complexity of Algorithms," *Transactions of the American Mathematical Society*, vol. 117, no. 1, pp. 285-285, 1965. [Google Scholar] [Publisher Link]

[4] Yaroslav D. Sergeyev, and Garro Alfredo, "Observability of Turing Machines: A Refinement of the Theory of Computation," *Informatica*, vol. 21, no. 3, pp. 425–454, 2010. [Google Scholar] [Publisher Link]

[5] Sven Ove Hansson, "Technology and Mathematics," *Philosophy & Technology*, *Springer*, vol. 33, pp. 117–13, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[6] Yaroslav D. Sergeyev, *Arithmetic of Infinity*, Edizioni Orizzonti Meridionali, vol. 103, pp.51-57, 2003. [Publisher Link]

[7] Yaroslav D. Sergeyev, "A New Applied Approach for Executing Computations with Infinite and Infinitesimal Quantities," *Informatica*, vol. 19, no. 4, pp. 567–59, 2008. [Google Scholar] [Publisher Link]

[8] Yaroslav D. Sergeyev, "Methodology of Numerical Computations with Infinities and Infinitesimals," *Rendiconti del Seminario Matematico dell'Universit e del Politecnico di Torino*, vol. 68, no. 2, pp. 95–113, 2010.] [Google Scholar] [Publisher Link]

[9] Yaroslav D. Sergeyev, "*Computer System for Storing Infinite, Infinitesimal, and finite Quantities and Executing Arithmetical Operations with Them*," EU patent 1728149, issued 03.06.2009; RF patent 2395111, issued 20.07.2010; USA patent 7,860,914 issued 28.12.2010. [Google Scholar] [Publisher Link]

[10] Yaroslav D. Sergeyev, "Numerical Point of View on Calculus for Functions Assuming Finite, Infinite and Infinitesimal Values Over Finite, Infinite, and Infinitesimal Domains," *Nonlinear Analysis Series A: Theory, Methods & Applications*, vol. 71, no. 12, pp. 1688–1707, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[11] Gabriele Lolli, "Infinitesimals and Infinites in the History of Mathematics: A Brief Survey," *Applied Mathematics and Computation*, vol. 218, no. 16, pp. 7979–7988, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[12] M. Margenstern, "Using Grossone to Count the Number of Elements of Infinite Sets and the Connection with Bijections," *P-Adic Numbers, Ultrametric Analysis and Applications*, vol. 3, no. 2, pp. 196–204, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[13] Louis D'Alotto, "Cellular Automata Using Infinite Computations," *Applied Mathematics and Computation*, vol. 218, no. 16, pp. 8077–8082, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[14] Sonia De Cosmis, and Renato De Leone, "The Use of Grossone in Mathematical Programming and Operations Research," *Applied Mathematics and Computation*, vol. 218, no. 16, pp. 8029–8038, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[15] D.I. Iudin, Ya. D. Sergeyev, and M. Hayakawa, "Interpretation of Percolation in Terms of Infinity Computations," *Applied Mathematics and Computation*, vol. 218, no. 16, pp. 8099–8111, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[16] Yaroslav D. Sergeyev, and Alfredo Garro, "The Grossone Methodology Perspective on Turing Machines," *Automata, Universality, Computation,* vol. 12, pp. 139-169, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[17] Yaroslav D. Sergeyev, "Numerical Computations and Mathematical Modelling with Infinite and Infinitesimal Numbers," *Applied Mathematics and Computation*, vol. 29, pp. 177–195, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[18] Anatoly Zhigljavsky, "Computing Sums of Conditionally Convergent and Divergent Series Using the Concept of Grossone," *Applied Mathematics and Computation*, vol. 218, no. 16, pp. 8064–8076, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[19] Jean-Michel Muller et al., *Handbook of Floating-Point Arithmetic*, Birkhäuser Boston, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[20] Yaroslav D. Sergeyev, "Mathematical Foundations of the Infinity Computer," *Annales UMCS Informatica AI*, vol. 4, pp. 20-33, 2006. [CrossRef] [Google Scholar] [Publisher Link]

[21] Manish Agrawal, and Grandon Gill, "Infinity Computer Solutions: Ramping Up," *Journal of Information Technology Education Discussion Cases*, vol. 1, no. 1, pp. 1-22, 2012. [Google Scholar] [Publisher Link]

[22] R. Gioiosa, *Resilience for Extreme Scale Computing*, Rugged Embedded Systems, pp. 123-148, 2017. [Google Scholar]

[23] Andrej Dujella, *Number Theory*, University of Zagreb textbooks, 2019. [Google Scholar] [Publisher Link]

[24] David Sager, "Quantum Mechanics Model," *Quantum Mechanics and Nanoptics*, 2019. [Google Scholar]

[25] George Panagiotopoulos et al., "The Underground Atlas Project: Can We Really Crowdsource the Underground Space?," *Procedia Engineering*, vol. 165, pp. 233-241, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[26] Stephen Bonner et al., "Exploring the Evolution of Big Data Technologies," *Software Architecture for Big Data and the Cloud*, pp. 253-283, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[27] D. Ramesh et al., "Python Based Implementation towards Cloud and Big Data Analytics for high Performance Applications," *Materials Today: Proceeding*, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[28] Krish Krishnan, *Building Big Data Applications*, Academic Press, 2020. [Google Scholar] [Publisher Link]

[29] Philipp Neumann, and Julian Kunkel, "High-Performance Techniques for Big Data Processing," *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, pp. 137-158, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[30] Burt Kaliski, "Exponential Time," *Encyclopedia of Cryptography and Security, Springer, Boston*, pp. 434, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[31] Vincent T'kindt, Lei Shang, and Federico Della Croce, "Exponential Time Algorithms for Just-in-Time Scheduling Problems with Common due Date and Symmetric Weights," *Journal of Combinatorial Optimization*, vol. 39, pp. 764–775, 2020. [CrossRef] [Google Scholar] [Publisher Link]