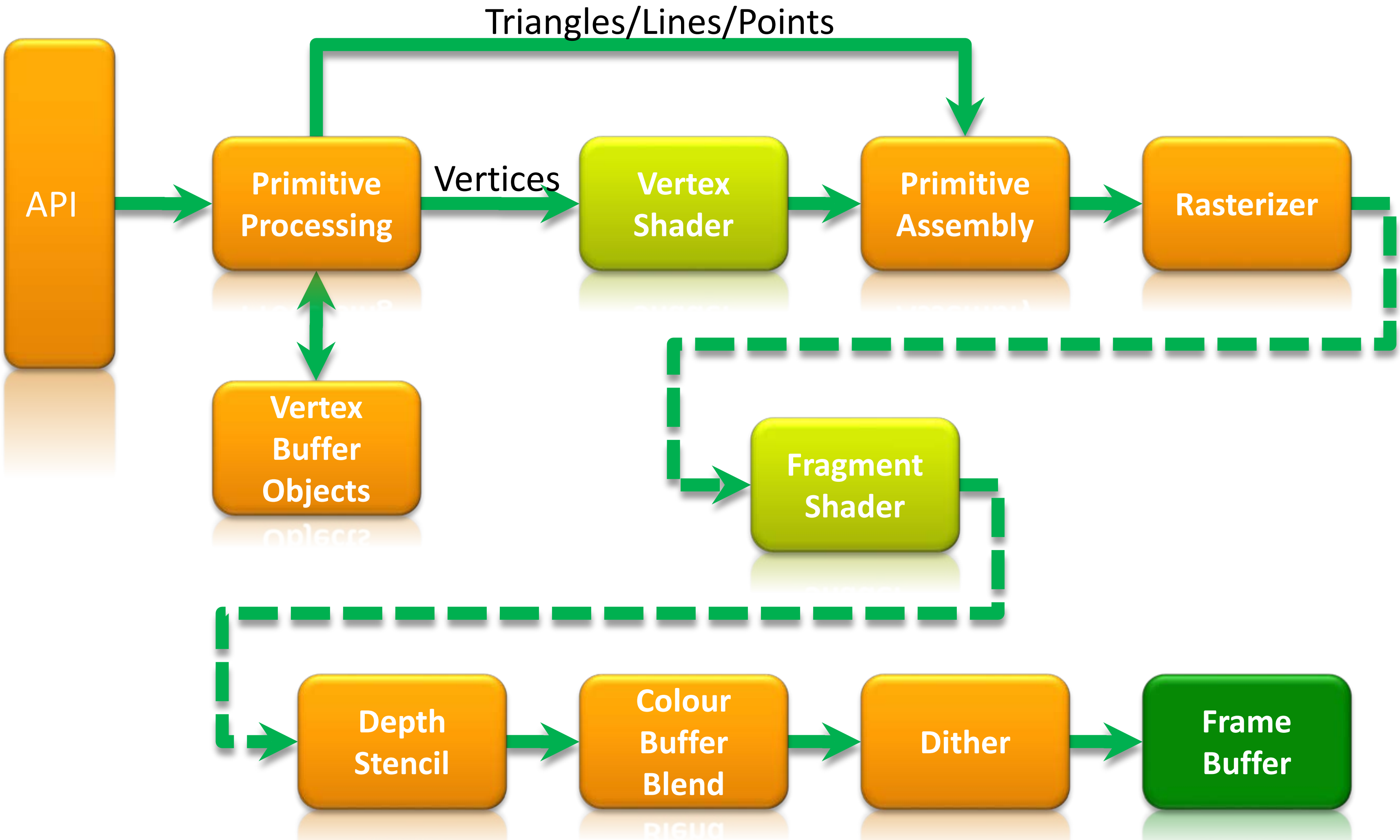
CONSTRAINING THE REAL PROBLEM OF FLOATING POINT NUMBER DISTRIBUTION

Floating point number

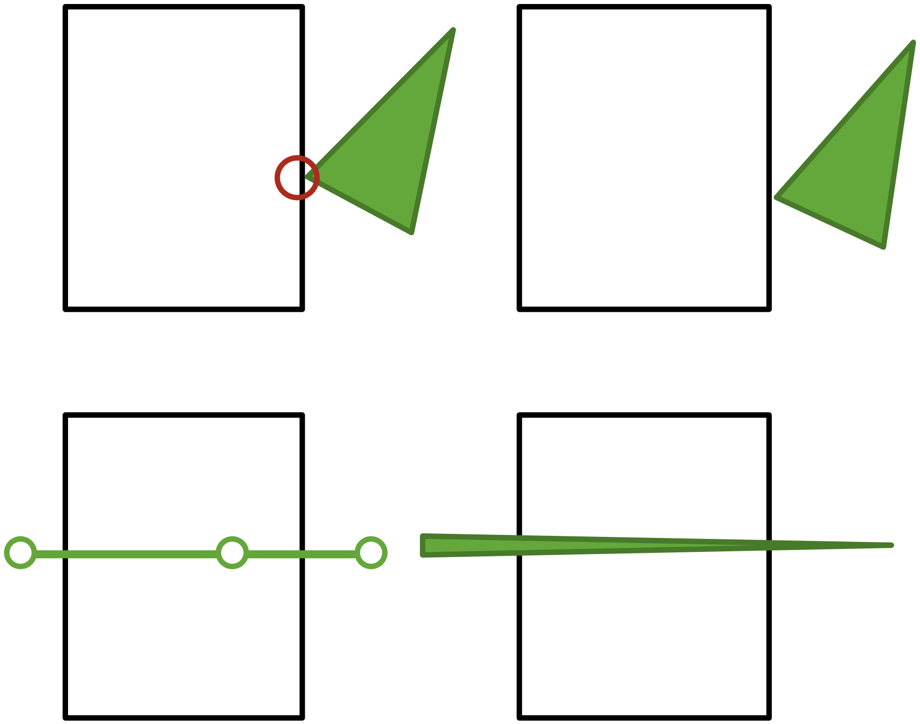
- Explanation what the floating point number is.

GPU and Floating Point Numbers



Examples of Floating Point Arithmetic Tests in GPU Verification

- Functionality of the module performs floating point arithmetic to:
  - Exactly detect if primitive is inside or outside drawing area
  - Remove primitives, which have zero area, and not remove triangles, which have close to zero area
  - Remove primitives based on facing



Floating Point Number Class Library for Verification

- Verification requirements:
  - Good distribution of random floating point numbers
  - Detailed control to hit interesting cases
  - Increase probability to hit values like +/-0, infinity, NaN, smallest non-zero, largest non-zero, etc.
  - Support floating point number arithmetic operations like multiply, divide, add, subtract and comparison
  - Reusable and shareable over multiple test benches

- **Solution:**
  - A practical solution is to encapsulate floating point representation into a class

Floating Point Number Class Features: Formats

- IEEE754 defines three binary base2 formats 32bit, 64bit, 128bit, and two decimal base10 formats with length 64bit and 128bit
- There are custom formats

- **Solution:**
  - Parameterizable base-class with parameters for different field widths
  - Base-class implements interface common to all formats
  - Sub-classes implement behavior for specific formats

Floating Point Number Class Features: Readability

- The main factor affecting usability is the conversion from the number value to the bit vector representation of a floating point number

- **Solution:**
  - Use strings to set values of floating point numbers
  - For log messages, provide floating point number to string conversion

Floating Point Number Class Features: Coarse Ranges

- Need a way to increase probability to hit special values:
  - NaN and infinity are valid test cases in hardware verification
  - Values interesting in special cases like 0, smallest non-zero value, largest possible value, etc. are rare to hit without increased probability

- **Solution:**
  - Random field to choose coarse range that defines what constraints to use

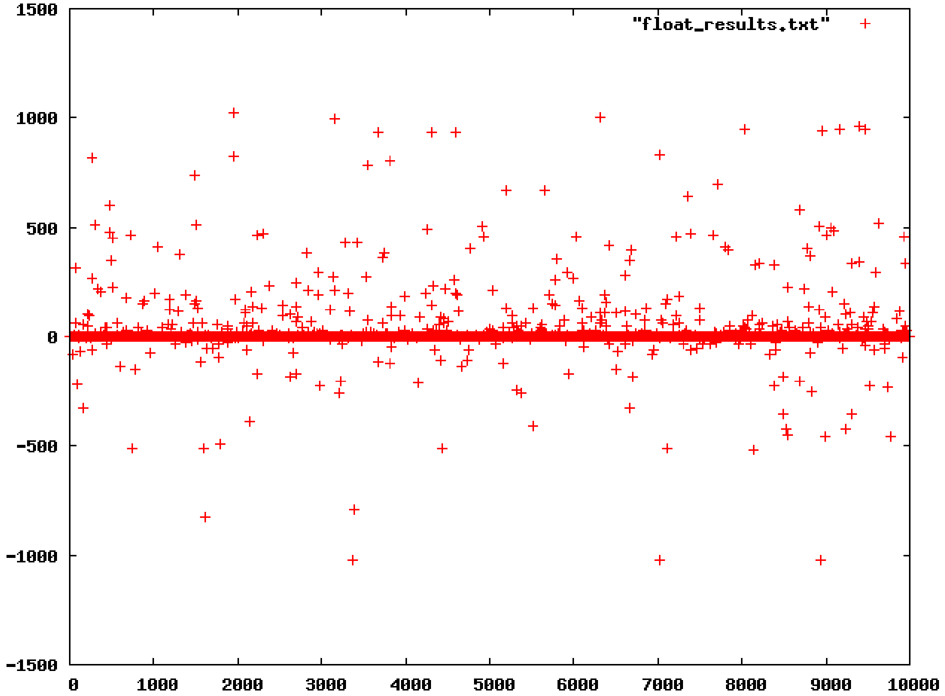
Floating Point Number Class Features: Limits

- Need to be able to define limits
- Need to be able to use random floating point number as a limit for other random floating point number
- Need to complete randomization during randomize phase

- **Solution:**
  - Use random fields to define fields for upper and lower limits
  - Use fields in constraints to limit the actual value

Floating Point Number Class Features: Distribution

- Unlike integers – where the values are uniformly distributed over the variable's legal range – the distribution of floating point values is exponential

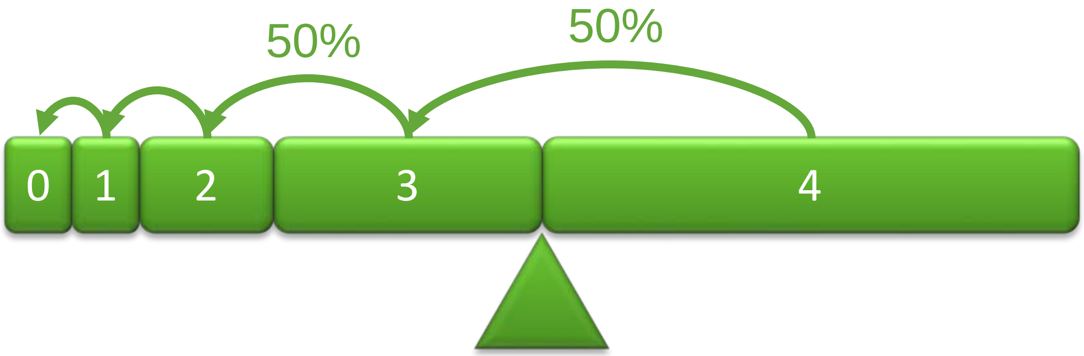


Floating Point Number Class Features: Distribution

- Solving exponential distribution:
  - Floating point values constructed from a binary representation have a 50% probability to be in the highest exponent range
    (Downey, A. (2007, July 25). Generating Pseudo-random Floating-Point Values.)

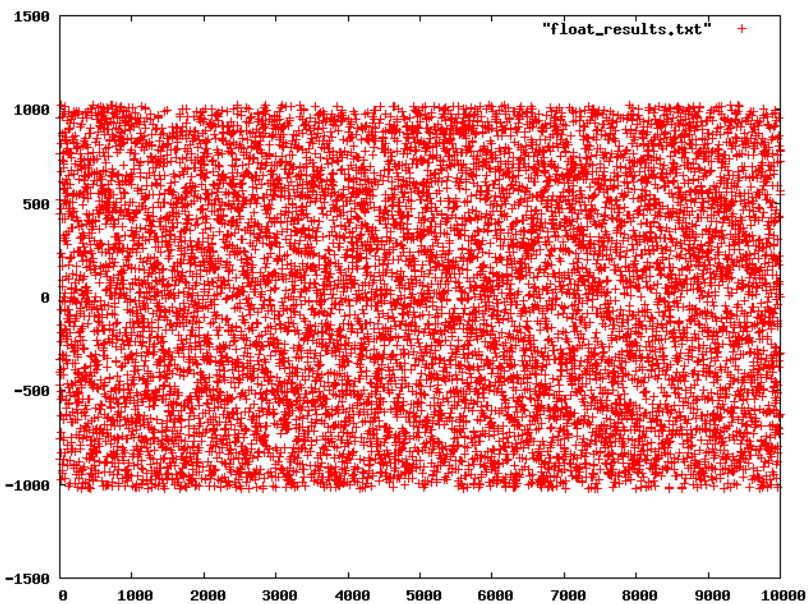
Floating Point Number Class Features: Distribution

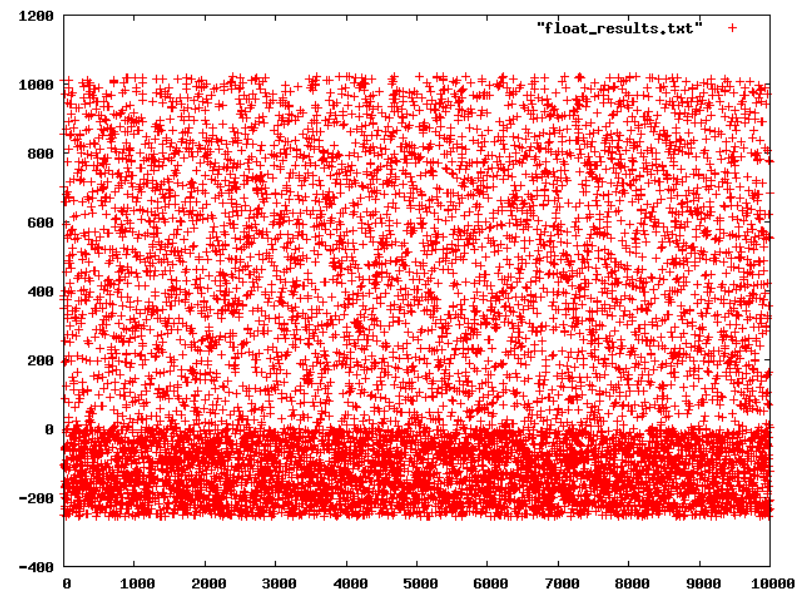
- The algorithm:
  1. Randomize a bit vector that has same width as the exponent
     - Each bit should have 50% probability to be 1
  2. Loop through randomized bit vector to find the first bit set to 1
     - Index of the first bit defines the exponent
  3. The mantissa is chosen freely, but constraints must ensure the legal range of mantissa will not be exceeded



Floating Point Number Class Features: Distribution

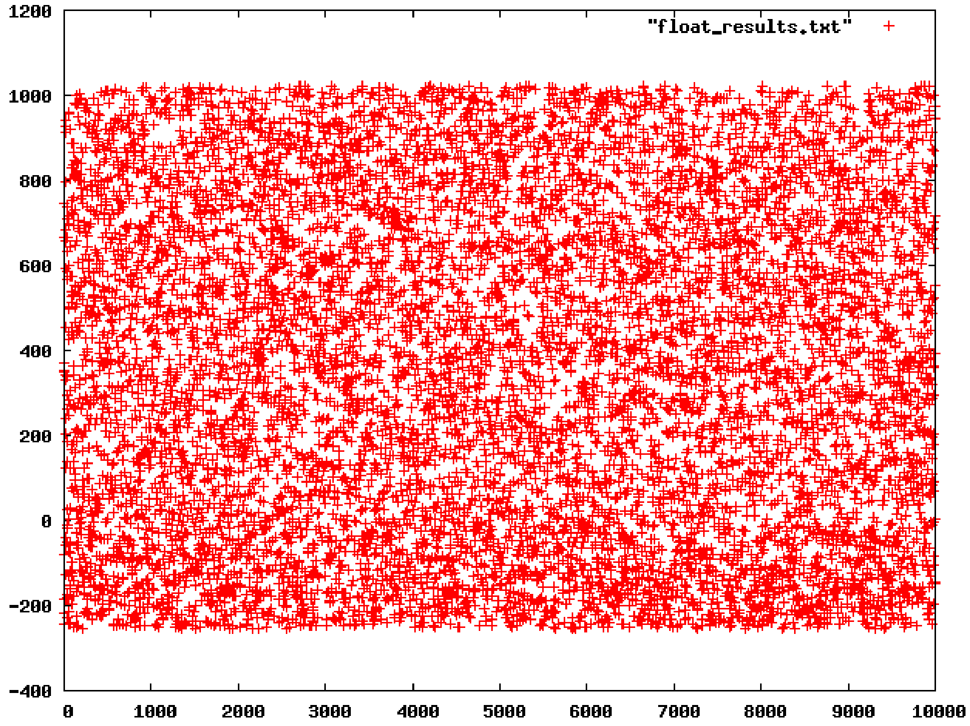
- Distribution using algorithm:
  - Uniform distribution when ranges over and below zero are equal
  - Non-balanced distribution when ranges over and below zero are not equal





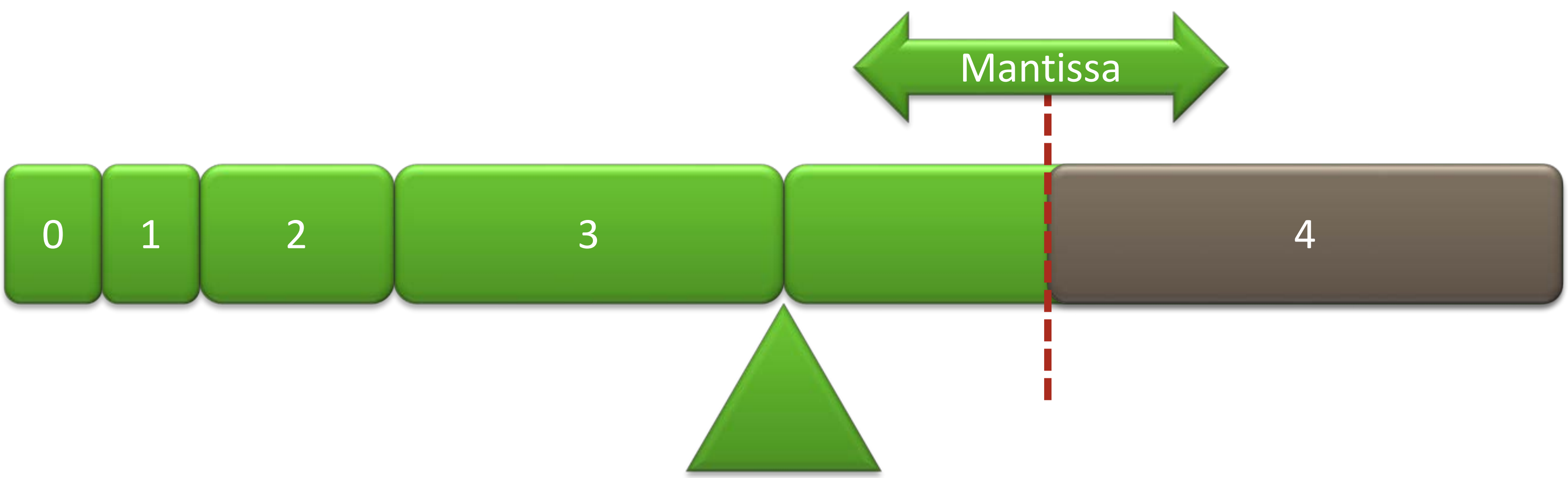
Floating Point Number Class Features: Distribution

- The amount of hits needs to be balanced and related to the size of the range



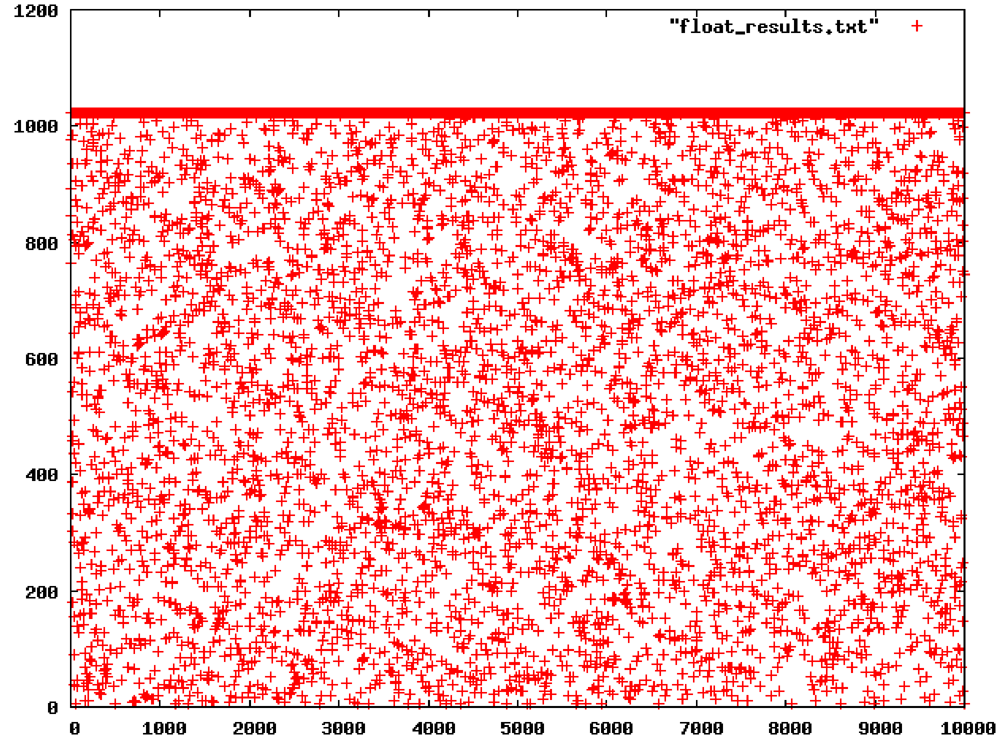
Issues With the Implemented Algorithm: Non-full Range Mantissa

- The algorithm assumes that full range of mantissa will be used
- Limiting the range of mantissa reduces the amount of numbers on highest exponent



Issues With the Implemented Algorithm: Non-full Range Mantissa

- If non-full range of mantissa is used, it shows as a higher density of hits on highest exponent



Issues with the Implemented Algorithm: Performance

- Using C to convert strings to float causes a drop in performance during randomization due to an increase in simulation DPI calls
- The use of random fields for limits can cause a large chain of constraints, and poor performance on constraint solver
- Random size array of classes requires a large enough array to be created first, but creating maximum number of classes has a negative affect on performance and causes unnecessarily large memory consumption

Conclusion

- It is possible to create a library for handling floating point numbers in verification, but it is not easy - issues are partially due to the implementation, and partially due to the functionality of the SystemVerilog
- With the feedback received from Synopsys, the library improved its performance dramatically, but at some point there must be a trade-off between required features and usability
- The industry should recognize the need and requirements for random floating point numbers and add them as part of the standard library to be implemented natively within tools and languages

Thank You