NaN detection in HDF5

Pedro Vicente Nunes, THG

Introduction

This Request For Comments (RFC) document explains a suggested approach in the handling of Not A Number (NaN) quantities in the HDF5 library. More specifically, the handling of the h5diff tool regarding NaNs is discussed.

Readers are invited to provide feedback on this RFC by sending comments to help@hdfgroup.org. Current plans are to distribute the implementation of this design with HDF5 release 1.8.1.

1 Special quantities

On some floating-point hardware every bit pattern represents a valid floating-point number. The IBM System/370 is an example of this. On the other hand, the VAX reserves some bit patterns to represent special numbers called *reserved operands* . This idea goes back to the CDC 6600, which had bit patterns for special quantities like infinity [Gol 91]. Without these special quantities, there is no good way to handle exceptional situations like taking the square root of a negative number other than aborting computation [Gol 91].

1.1 IEEE 754

Programmers using floating-point computations in the 1960s and 1970s had to cope with a wide variety of configurations, with each computer supporting a different range and accuracy for floating-point numbers [Sev 98]. In 1976, Intel began to plan for a floating-point coprocessor for the i8086 microprocessor. In 1977, an IEEE working group (IEEE 754) regarding a floating-point arithmetic standard was initiated by Intel consultants that were helping to design the 8087 processor. The group included also DEC VAX, CDC, Cray and IBM. In the end most of Intel's design was adopted [Sev 98].

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) was published in 1985 [IEEE 85] and defines a binary arithmetic standard for floating point calculations. The standard encourages the development of portable numerical programs [Doo 03] and its goal was to end over a dozen commercially significant arithmetics available at the time that boasted diverse wordsizes, precisions, rounding procedures and over/underflow behaviors [Kah 97].

The IEEE standard defines the special quantities NaNs (Not a Number) and infinities. IEEE 754 specifies how floating point numbers are encoded into bits. IEEE 754 defines four floating point formats in two groups, basic and extended, each having two widths, single (32-bit) and

double precision (64-bit) [IEEE 85]. Only the basic single format is required by the format, the other formats being optional for implementation.

Table 1. Correspondence between the values of the three constituent fields *s*, *e* and *f*, of formula (1) and the value represented by the single format bit pattern [Sun 00]

Range for *e* and *f* and *s* value	**Value**
0 < e < 255	$(-1)^s \times 2^{e-127} \times 1.f$ (normal[1] numbers)
e = 0; f not 0 (at least one bit in f is nonzero)	$(-1)^s \times 2^{-126} \times 0.f$ (subnormal numbers)
e = 0; f = 0 (all bits in f are zero)	$(-1)^s \times 0.0$ (signed zero)
s = 0; e = 255; f = 0 (all bits in f are zero)	+INF (positive infinity)
s = 1; e = 255; f = 0 (all bits in f are zero)	-INF (negative infinity)
e = 255; f not 0 (at least one bit in f is nonzero)	NaN (Not-a-Number)

The IEEE single format consists of three fields: a 23-bit fraction, f; an 8-bit biased exponent, e; and a 1-bit sign, s. These fields are stored contiguously in one 32-bit word. Figure 1 shows an example of such a number

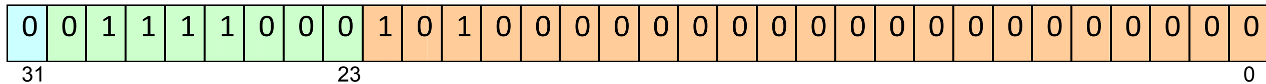

Figure 1. The IEEE 754 format for a 32-bit floating point number. Bits 0:22 (orange) contain the 23-bit fraction, *f,* with bit 0 being the least significant bit[2] of the fraction and bit 22 being the most significant; bits 23:30 (green) contain the 8-bit biased exponent, *e*, with bit 23 being the least significant bit of the biased exponent and bit 30 being the most significant; and the highest-order bit 31 (blue) contains the sign bit, *s*. The number represented is obtained according to formula (1): *e* is 120 decimal ($0x2^7 + 1x2^6 + 1x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 0x2^0$), giving an exponent of -7 after biasing. The significand is 1.101 binary, which is 1.625 decimal ($1x2^{-1} + 0x2^{-2} + 1x2^{-3}$). So, the decimal value represented by this bit pattern is $-1^0 2^{-7}$ 1.625 or 0.012695313.

For normal numbers, a value *v* is obtained according to formula (1)

$$v = (-1)^s 2^{e-bias} 1.f \quad (1)$$

[1] A normal number has the leading digit of the significand in formula (1) non zero. Non-zero numbers smaller in magnitude than the smallest normal number are called denormal (or subnormal) numbers. Zero is neither normal nor subnormal. Normalization is the act of shifting the fractional part in order to make the left bit of the fractional point one. During this shift the exponent is incremented.

[2] the least significant bit (lsb) is the bit position in a binary integer giving the units value, that is, determining whether the number is even or odd

Where *s* is the sign bit, *e* is the biased exponent, and *f* is the fraction. Only *s*, *e*, and *f* need to be stored to fully specify the number. Because the implicit leading bit of the significand is defined to be 1 for normal numbers, it need not be stored. The *bias* value is 127 for single precision values.

Table 1, reproduced here from [Sun 00], shows the correspondence between the values of the three constituent fields *s*, *e* and *f*, on the one hand, and the value represented by the single-format bit pattern on the other.

1.2 NaNs

Traditionally, the computation of 0/0 or $\sqrt{-1}$ has been treated as an unrecoverable error that causes a computation to halt. There are, however, examples for which it makes sense for a computation to continue in such a situation [Gol 91]. In IEEE 754, expressions like 0/0 or $\sqrt{-1}$ produce NaN rather than halting. IEEE 754 specifies that seven invalid arithmetic operations shall deliver a NaN, listed in Table 2.

Table 2. IEEE 754 invalid arithmetic operations that cause a NaN (reprinted from [Kah 97] and [Gol 91])

Operation	**NaN produced by**
$\sqrt{\ }$	$\sqrt{x}$, when $x < 0$
x	0 x ∞
/	0/0
/	∞ / ∞
REMAINDER	x REMAINDER 0
REMAINDER	∞ REMAINDER y
+	$\infty - \infty$ when signs agree (but $\infty + \infty = \infty$ when signs agree)

In IEEE 754, NaNs are represented as floating-point numbers with the exponent of 255 and non zero significands. Implementations are free to put system dependent information into the significand. Thus, there is not a unique NaN but rather a whole family of NaNs. When an NaN and an ordinary floating-point number are combined, the result is a NaN [Gol 91]. Figure 2 shows the bit pattern for a NaN

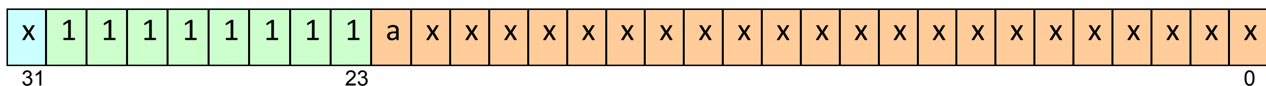

Figure 2. bit pattern for a NaN. x means undefined. If a is 1, the quantity it is a quiet NaN (non trapping), otherwise it is a signalling (trapping) NaN.

1.2.1 Trapping and non trapping NaNs

IEEE 754 defines two different kinds of NaNs: trapping and non trapping, also mentioned as signaling and quiet, respectively (their bit pattern is shown in Figure 2). Trapping NaNs precipitate an invalid operation exception. Non trapping NaNs afford retrospective diagnostic information inherited from invalid or unavailable data and results [IEEE 85].

Trapping NaNs are rarely used [Kah 97]. Similar concepts regarding NaNs were tried previously. Seymour Cray built *Indefinites* into the CDC 6600 in 1963; then DEC put *Reserved Operands* into their PDP-11 and VAX. But nobody used them because they trap when touched. NaNs do not trap (unless they are signaling SNaNs). NaNs propagate through most computations. Consequently they do get used [Kah 97].

1.2.2 NaN detection

Were there no way to get rid of NaNs, they would be as useless as *Indefinites* on CRAYs; as soon as one were encountered, computation would be best stopped rather than continued for an indefinite time to an *Indefinite* conclusion. That is why some operations upon NaNs must deliver non-NaN results [Kah 97].

Table 3. IEEE 754 relational expressions involving NaN, in C language syntax. x or y or both are NaN. An invalid operation must signal ([Kah 97])

Relational expression	Result	Operation
x ==x	false	valid
x !=x	true	valid
x < y	false	invalid
x <= y	false	invalid
x >= y	false	invalid
x > y	false	invalid

The exceptions are C programming language predicates " x == x " and " x != x ", which are respectively 1 and 0 for every infinite or finite number x but reverse if x is NaN; these provide the only simple unexceptional distinction between NaNs and numbers in languages that lack a word for NaN and a predicate IsNaN(x). Overoptimizing compilers that substitute 1 for x == x violate IEEE 754 [Kah 97]. IEEE 754 assigns values to all relational expressions involving NaN, listed in Table 3.

1.2.3 Display of NaNs

Different software libraries, operating systems and programming languages will have different string representations of NaN, since the IEEE specification does not mandate such a representation. Examples of such string representations are nan, NaN, NaN%, NAN, NaNQ, NaNS, qNaN, sNaN, 1.#SNAN and 1.#QNAN

1.3 Infinity

Just as NaNs provide a way to continue a computation when expressions like 0/0 or $\sqrt{-1}$ are encountered, infinities provide a way to continue when an overflow occurs. This is much safer than simply returning to the largest representable number [Gol 91].

Table 4. Platforms were this study was conducted

Platform	CPU	C compiler
Microsoft Windows XP	Intel	Visual Studio 6
Linux 2.6.9	Intel	gcc 3.4.6
SunOS	SPARC	cc
FreeBSD 6.2	Intel	gcc 3.4.6
CRAY		
VMS		

2 NaNs and HDF5

NaNs in HDF5 are treated like any other floating-point quantity, i.e, they are stored as any other floating-point quantity is stored. They are displayed by the string representation present on any operating system or compiler version HDF5 supports. This is the case with the h5dump tool, the HDF5 tool that displays information about an HDF5 file. The study presented in this RFC was applied to six different platforms, listed in Table 4, and Appendix 1 list the h5dump output of the use case used in these tests. However, when comparison of NaNs is involved, some behavior has to be programmatically set, due to the special behavior of these quantities regarding relational expressions (Table 3). One HDF5 tool that compares quantities is the h5diff tool, and that is the subject of this RFC.

2.1 h5diff

h5diff is a tool that compares two HDF5 files and reports the differences, that is, each object in the file is compared datum element by datum element at a time. For floating-point values the comparison criteria is a relative error formula, but for simplicity purposes in this RFC we will assume that the inequality operator is used.

The behavior we would like to propose regarding the comparison of NaNs in h5diff is shown in Table 5.

Table 5. NaN detection in h5diff. x and y represent two floating-point quantities to be compared

x and y are not NaNs, compare as usual
one is a NaN, do not compare but assume difference
both are NaN, consider them equal

So, a way to detect if a quantity is a NaN is needed before trying to compare *x* and *y* in Table 5. The test for NaN we use is the return value of the expression

(*x*!=*x*)

that should be TRUE if *x* is a NaN. However, in some platforms, this expression does not evaluate to TRUE, so in these cases, we do a string comparison between the value represented by *x* and one of the possible string representations of NaN presented in section 1.2.3. In all our test platform cases there was a correct evaluation of the quantity used as a NaN. The results are shown in Appendix 2.

3 References

[Gol 91] D. Goldberg, "What a computer scientist should know about floating point arithmetic," ACM Computing Surveys, vol. 23, no. 1, pp. 5--48, 1991.

[IEEE 85] Institute of Electrical and Electronics Engineers, 1985. “IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985”. Reprinted in SIGPLAN Notices, 22(2):9-25, 1987.

[Kah 97] W. Kahan. “Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic.” Elect. Eng. & Computer Science. University of California Berkeley, 1997

[Doo 03] P. Van Dooren. “Algorithmique Numerique”. Universite catholique de Louvain. Faculte des Sciences Appliquees. Departement d'Ingenierie Mathematique, 2003

[Sev 98] C. Severance. “IEEE 754: An Interview with William Kahan”. IEEE Computer, March 1998.

[Sun 00] Sun Microsystems Inc. “Numerical Computation Guide”. May 2000.

Appendix 1

This section shows the h5dump output of the dataset used in the tests. To generate a NaN the operation SQRT(-1) was used. In all the platforms tested the result of this operation was a non trapping NaN. Two datasets were generated, of 6 elements each, of HDF5 type corresponding to 32-bit floating point. At some positions in the arrays a NaN was written. Table A1.0 shows these 2 datasets. Tables A1.1 to A1.4 show the h5dump output of the first dataset.

Table A1.0. The 2 datasets used in the tests

Array element	0	1	2	3	4	5
Dataset 1	SQRT(-1)	1	SQRT(-1)	1	1	1
Dataset 2	SQRT(-1)	SQRT(-1)	1	1	1	1

Table A1.1. Windows XP h5dump output of the first test dataset

```
HDF5 "h5diff_basic1.h5" {
DATASET "/g1/fp15" {
  DATATYPE  H5T_IEEE_F32LE
  DATASPACE  SIMPLE { ( 6 ) / ( 6 ) }
  DATA {
  (0): -1.#IND, 1, -1.#IND, 1, 1, 1
  }
}
}
```

Table A1.2. Linux 2.6.9 h5dump output of the first test dataset

```
HDF5 "/home1/pvn/hdf5/tools/testfiles/h5diff_basic1.h5" {
DATASET "/g1/fp15" {
  DATATYPE  H5T_IEEE_F32LE
  DATASPACE  SIMPLE { ( 6 ) / ( 6 ) }
  DATA {
  (0): nan, 1, nan, 1, 1, 1
  }
}
```

```
}
```

Table A1.3. FreeBSD 6.2 h5dump output of the first test dataset

```
HDF5 "/home1/pvn/hdf5/tools/testfiles/h5diff_basic1.h5" {
DATASET "/g1/fp15" {
   DATATYPE  H5T_IEEE_F32LE
   DATASPACE  SIMPLE { ( 6 ) / ( 6 ) }
   DATA {
   (0): nan, 1, nan, 1, 1, 1
   }
}
}
```

Table A1.4. SunOS 5.10 h5dump output of the first test dataset

```
HDF5 "/home1/pvn/hdf5/tools/testfiles/h5diff_basic1.h5" {
DATASET "/g1/fp15" {
   DATATYPE  H5T_IEEE_F32LE
   DATASPACE  SIMPLE { ( 6 ) / ( 6 ) }
   DATA {
   (0): -NaN, 1, -NaN, 1, 1, 1
   }
}
}
```

Appendix 2

This section shows the h5diff output regarding the difference between the 2 datasets. Shown here is one example of such output, other systems the output being similar except for the NaN string representation

Table A2.0. Linux 2.6.9 h5diff output regarding the difference of the 2 datasets of Table A1.0

```
dataset: </g1/fp15> and </g1/fp16>
size:        [6]        [6]
position     fp15         fp16       difference
------------------------------------------------------------
[ 1 ]        1            nan        nan
[ 2 ]        nan          1          nan
2 differences found
```

Acknowledgements

The design presented in this document regarding the detection of NaNs resulted from a function originally written by Robb Mattzke (LLNL). His contribution was greatly appreciated.

Author Biography

Pedro Vicente Nunes is a senior staff software engineer with The HDF Group in Champaign, Illinois. He holds a B.Sc. degree in environmental engineering from the New University of Lisbon, Portugal and a M.Sc. degree in numerical geophysical modeling from the Technical University of Lisbon, Portugal. His e-mail is pvn@hdfgroup.org.